

TIME SERIES ANALYSIS

Table of contents

- Project statement
- Project Overview
- Prerequisites
- Project Requirements
- Execution Overview
- Tools/Technology Used in the Project
- Source Data Files
- Implementation -Tasks performed
- Steps on practical implementation on the Azure portal
- Successful output generated
- Conclusion

Problem statement:

Implement time series analysis using PySparkSQL on Azure Databricks. Process and analyze time-stamped data for trends, seasonality, and anomalies.

Project Overview:

This project develops a **time series analysis pipeline** using Azure Databricks and PySparkSQL to process, transform, and analyze timestamped website visitor data. The workflow begins with data ingestion of the CSV dataset into Databricks, followed by cleaning, preprocessing, and resampling. Using PySparkSQL, the pipeline extracts **trends, seasonality patterns, and anomalies** from the dataset. The results are stored as Delta tables in Databricks for scalability and can be exported for visualization and reporting. This approach ensures an efficient and robust analytical process, enabling insights into visitor behaviors over time, and serves as a scalable foundation for extending to forecasting or larger datasets.

Prerequisites

- **Azure Subscription:** Active Azure subscription for provisioning resources.
- **Azure Databricks:** Workspace setup for PySparkSQL-based processing and analysis.
- **Azure Storage Account:** To store raw CSV data (`daily-website-visitors.csv`) and export processed outputs.
- **Data Source:** Website visitor CSV file containing daily visits, unique visits, first-time visits, and returning visits.
- **Azure Key Vault (optional):** For securely storing secrets and credentials (if extending to external sources).
- **Databricks Cluster:** Configured Spark cluster for running PySpark transformations.
- **Libraries and Dependencies:** PySpark (included), Delta Lake (included), matplotlib for visualization.

- **Access Permissions:** Ensure Databricks has access to the storage account.
- **Monitoring and Logging:** Databricks Job UI and Azure Monitor for tracking performance.

Project Requirements

Technical Infrastructure:

- **Azure Subscription:** Active subscription for creating and managing Databricks and storage resources.
- **Azure Databricks:** Workspace setup for executing PySparkSQL queries and time-series analysis.
- **Databricks Cluster:** Provision of compute clusters with adequate resources to handle large time-series datasets.
- **Azure Storage Account:** Blob Storage or Data Lake to store raw input datasets and processed analytical results.

Data Sources and Destinations:

- **Source Data Specifications:** Availability of time-stamped data in CSV, JSON, or Parquet formats.
- **Destination Specifications:** Storage of cleaned and analyzed outputs in Delta tables, Blob Storage, or Data Lake for further use.

Development Tools:

- **PySparkSQL Environment:** Configuration within Databricks for SQL-based analysis and transformations.
- **Visualization Tools:** Use of Databricks SQL dashboards, matplotlib, or Power BI to present trends, seasonality, and anomalies.
- **Version Control:** GitHub or Azure Repos for maintaining PySparkSQL notebooks and scripts.

Security and Compliance:

- **Data Security:** Secure access to storage accounts using role-based permissions and encryption.
- **Compliance Standards:** Ensure handling of data follows relevant data governance and privacy policies.

Performance and Scalability:

- **Resource Allocation:** Appropriate sizing of Databricks clusters for parallel processing of large datasets.
- **Scalability Considerations:** Ability to scale clusters dynamically as dataset size or complexity grows.

Monitoring and Logging:

- **Job Monitoring:** Track Databricks job execution, performance, and query optimization.
- **Error Logging and Handling:** Implement error logging mechanisms within Databricks notebooks for troubleshooting.

Documentation and Training:

- **Technical Documentation:** Detailed documentation of dataset preprocessing steps, SQL queries, and analysis workflows.
- **User Training:** Provide guides or walkthroughs for interpreting outputs, dashboards, and anomaly reports.

Project Management:

- **Timeline:** Define project phases—data ingestion, preprocessing, analysis, and visualization—with milestones.
- **Budget:** Estimate costs for Databricks clusters, storage, and visualization tools.
- **Risk Management:** Identify risks such as data quality issues, cluster failures, or cost overruns, with mitigation strategies.

Execution Overview

Project Kick-off

- **Initial Meeting:** Define objectives of the project – perform time series analysis (trend, seasonality, anomaly detection) on timestamped website visitor data using PySpark and Spark SQL in Azure Databricks.
- **Resource Allocation:** Ensure availability of Azure Databricks workspace, DBFS for storage, and required libraries (PySpark, Delta Lake, matplotlib for visualization).

Data Source Integration

- **Identification:** Select `daily-website-visitors.csv` dataset as the primary data source.
- **Connection Setup:** Upload the dataset into DBFS (`dbfs:/FileStore/tables/`) and register it into a dedicated Databricks database for further processing.

Azure Databricks Setup

- **Workspace Preparation:** Configure the Azure Databricks cluster with appropriate runtime (latest supported Spark/PySpark).
- **PySpark Environment:** Establish Python-based PySpark notebooks to run all data transformations, SQL queries, and visualizations.

Pipeline Development

- **Data Ingestion:** Read CSV file into PySpark DataFrame, clean and normalize column names, cast types, and persist as Delta tables.
- **Data Resampling:** Build daily and weekly time series tables using Spark SQL (handling missing dates, resampling, and aggregations).

- **Feature Engineering:** Implement moving averages for trend, calculate day-of-week seasonal components, and compute residuals.
- **Anomaly Detection:** Use robust statistical methods (Median Absolute Deviation) to identify anomalies in visitor counts.

Testing and Validation

- **Unit Testing:** Validate correctness of ingestion, date parsing, missing value handling, and transformations in PySpark.
- **Integration Testing:** Verify end-to-end flow from raw CSV ingestion to Delta tables, SQL transformations, anomaly detection, and visualization.

Performance Tuning

- **Benchmarking:** Measure runtime of queries (trend, seasonality, anomaly detection) on the dataset.
- **Optimization:** Optimize Spark jobs by caching intermediate tables, reducing shuffle operations, and using efficient aggregations.

Deployment

- **Staging Deployment:** Run the pipeline with test datasets in staging environment to validate correctness and robustness.
- **Production Rollout:** Execute pipeline on actual data in production, ensuring daily/weekly refresh of tables and updated anomaly detection.

Monitoring and Maintenance

- **Operational Monitoring:** Track job execution in Databricks Job UI, monitor cluster utilization, and review logs for failures.
- **Regular Updates:** Schedule updates to adjust anomaly thresholds, add new features (monthly seasonality, holiday effects), or scale to larger datasets.

Project Review and Closure

- **Post-Deployment Review:** Evaluate project outcomes – accuracy of trend/seasonality modeling, usefulness of anomalies flagged.
- **Feedback Incorporation:** Collect feedback from stakeholders (data team, business team) on pipeline usability and insights.
- **Formal Closure:** Document lessons learned, finalize reports/visualizations, and prepare recommendations for extending analysis into forecasting.

Tool/Technology used in the Project

- **Azure Databricks** → Main platform for developing and running PySparkSQL-based time series analysis.
- **PySpark / Spark SQL** → For distributed data processing, transformations, and time series analysis.
- **Azure Storage (Blob Storage / DBFS)** → To store raw CSV datasets and processed analytical outputs.
- **Python Libraries** → `pyspark.sql`, `pandas`, `matplotlib` for analysis and visualization.
- **Git (Version Control)** → For managing notebooks, scripts, and project documentation.
- **Azure Active Directory (AAD)** → For secure authentication and access control within Databricks.
- **Azure Monitor / Logging (Optional)** → To monitor job performance and query execution inside Databricks.

Source Data Files

We will be using the [Daily Website Visitors](#) dataset.

This dataset contains **time-stamped records** of website visitor counts on a daily basis. Each record represents the total number of visitors on a given date, which makes it suitable for time series analysis (trend detection, seasonality, and anomaly identification).

This dataset has **2,167 rows** and **8 columns**

Showing top 20 rows

Row	Day	Day.Of.Week	Date	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits
1	Sunday	1	9/14/2014	2,146	1,582	1,430	152
2	Monday	2	9/15/2014	3,621	2,528	2,297	231
3	Tuesday	3	9/16/2014	3,698	2,630	2,352	278
4	Wednesday	4	9/17/2014	3,667	2,614	2,327	287
5	Thursday	5	9/18/2014	3,316	2,366	2,130	236
6	Friday	6	9/19/2014	2,815	1,863	1,622	241
7	Saturday	7	9/20/2014	1,658	1,118	985	133
8	Sunday	1	9/21/2014	2,288	1,656	1,481	175
9	Monday	2	9/22/2014	3,638	2,586	2,312	274
10	Tuesday	3	9/23/2014	4,462	3,257	2,989	268
11	Wednesday	4	9/24/2014	4,414	3,175	2,891	284
12	Thursday	5	9/25/2014	4,315	3,029	2,743	286
13	Friday	6	9/26/2014	3,323	2,249	2,033	216
14	Saturday	7	9/27/2014	1,656	1,180	1,040	140
15	Sunday	1	9/28/2014	2,465	1,806	1,613	193
16	Monday	2	9/29/2014	4,096	2,873	2,577	296
17	Tuesday	3	9/30/2014	4,474	3,032	2,720	312
18	Wednesday	4	10/1/2014	4,124	2,849	2,541	308
19	Thursday	5	10/2/2014	3,514	2,489	2,239	250
20	Friday	6	10/3/2014	3,005	2,097	1,856	241

Implementation-Tasks Performed

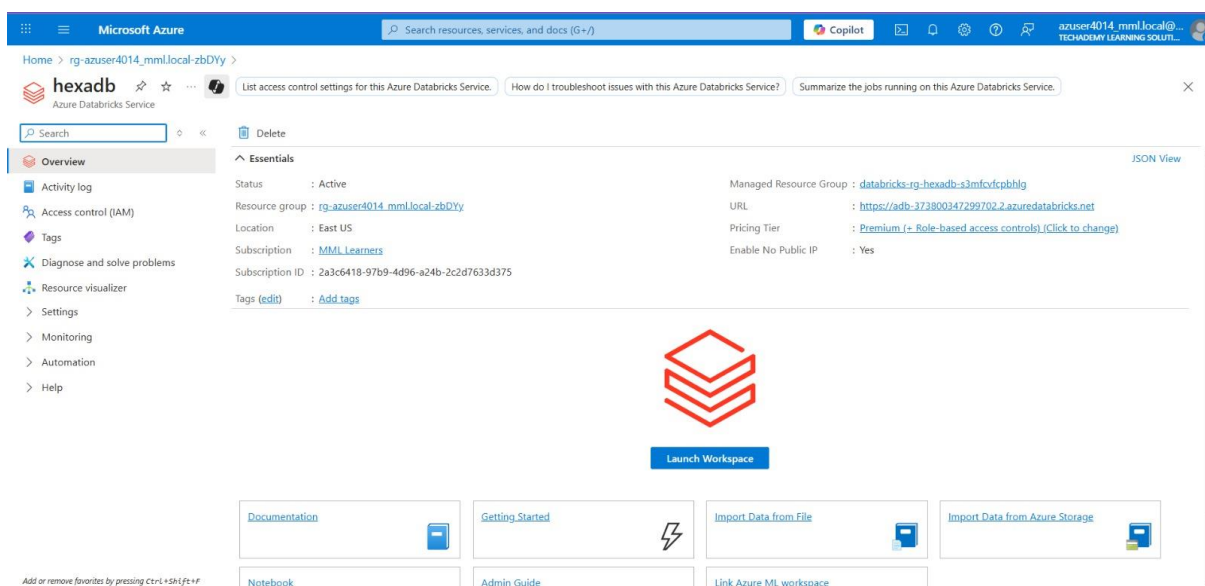
- Set up **Azure Databricks workspace** for running PySparkSQL workloads.
- Configure and launch a **Databricks cluster** with required compute resources.
- Upload and register the **time-stamped dataset** (CSV/Parquet) into Azure Storage / Databricks File System (DBFS).
- Load data into a **PySpark DataFrame** and perform preprocessing (cleaning, handling missing values, type conversions).

- Implement **time-series transformations** using PySparkSQL (resampling, aggregations, lag/rolling features).
- Perform **trend and seasonality analysis** using SQL queries and window functions.
- Detect **anomalies** in time series using statistical thresholds and residual analysis.
- Visualize insights with **Databricks SQL dashboards** and Python libraries (Matplotlib/Seaborn).
- Optimize query performance using caching, indexing, and efficient cluster resource utilization.
- Document the complete **analysis workflow, queries, and maintenance plan** for reproducibility.

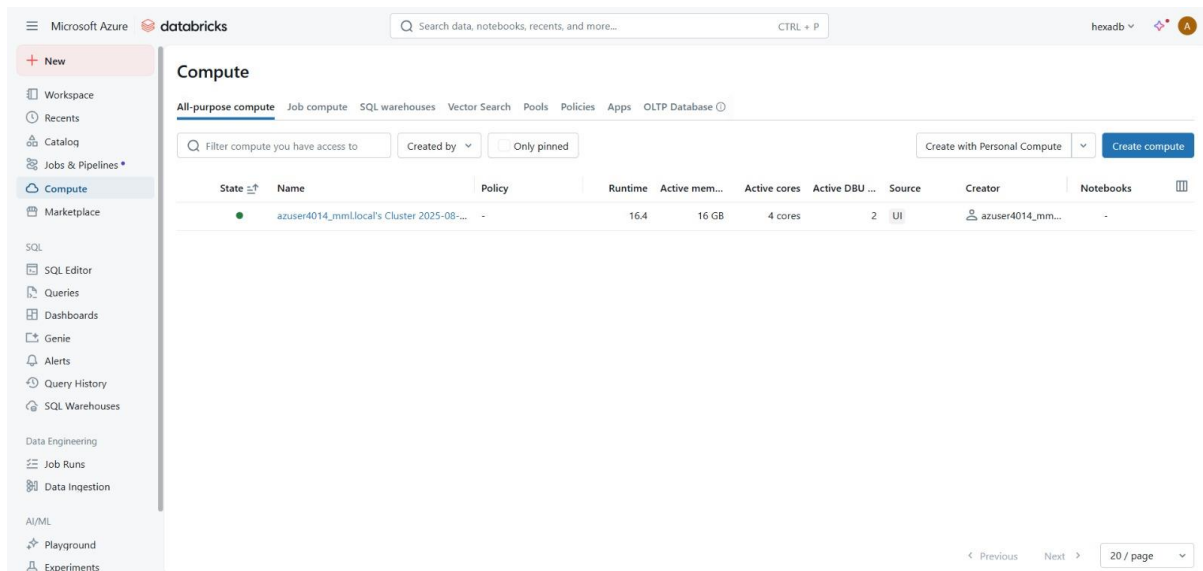
Practical Implementation on Azure Portal

Step 1: Set Up Azure Databricks

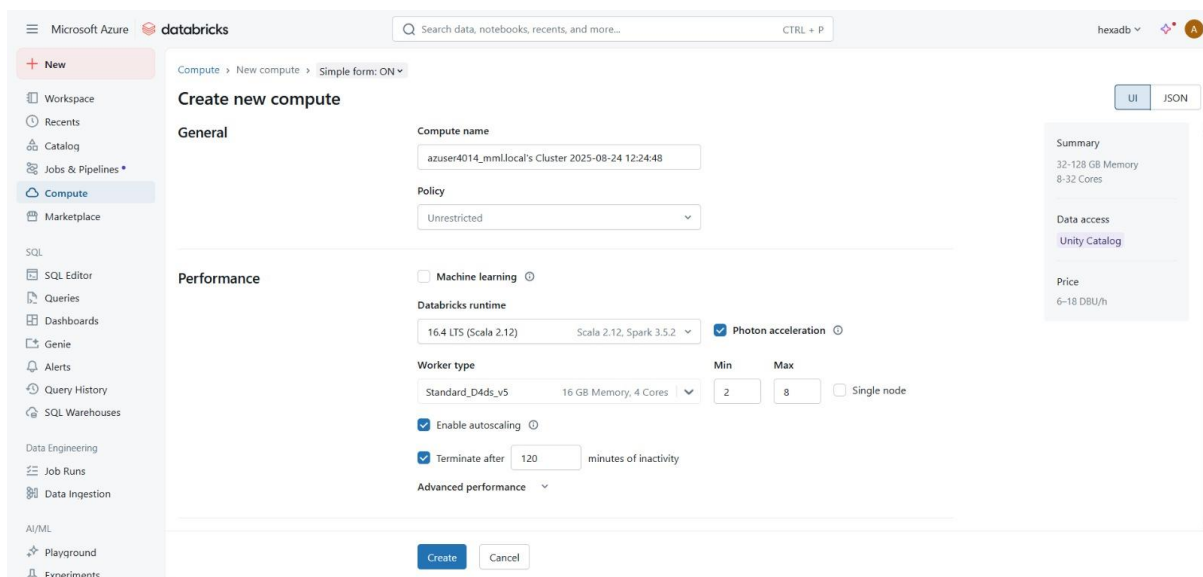
The first step is to launch the Databricks workspace from the Azure portal. The image shows the Azure Databricks service overview page with the **"Launch Workspace"** button, which is the entry point to the Databricks environment.

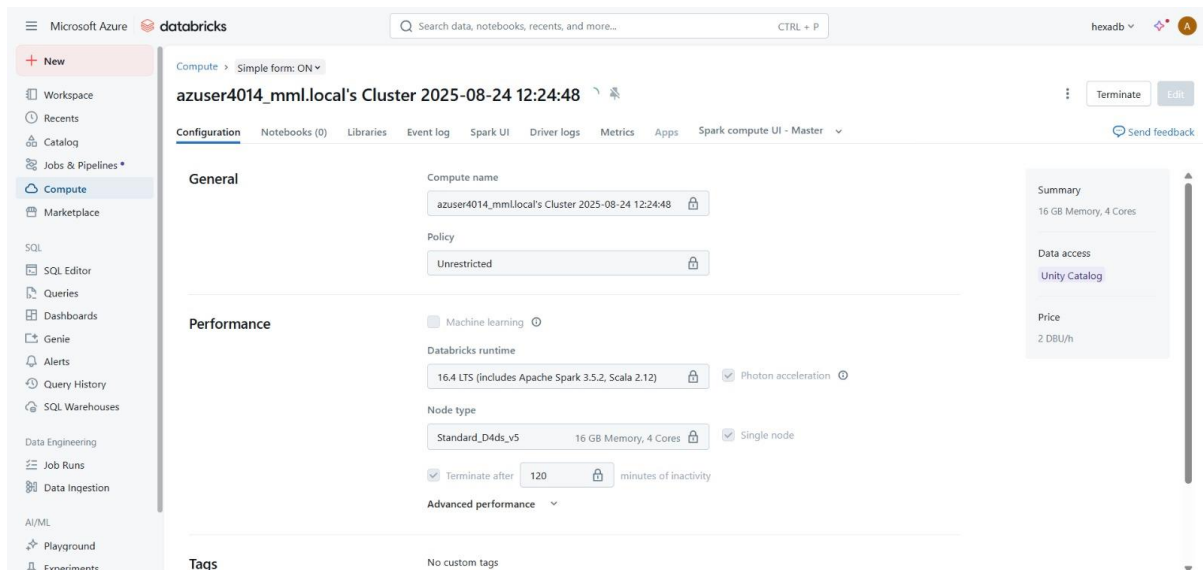


After launching the workspace, you need to create a compute cluster to run your data processing jobs. Image shows the "Create new compute" form, where you configure the cluster settings, such as the compute name, Databricks runtime, and other performance options. This is a crucial step before you can do any data processing.

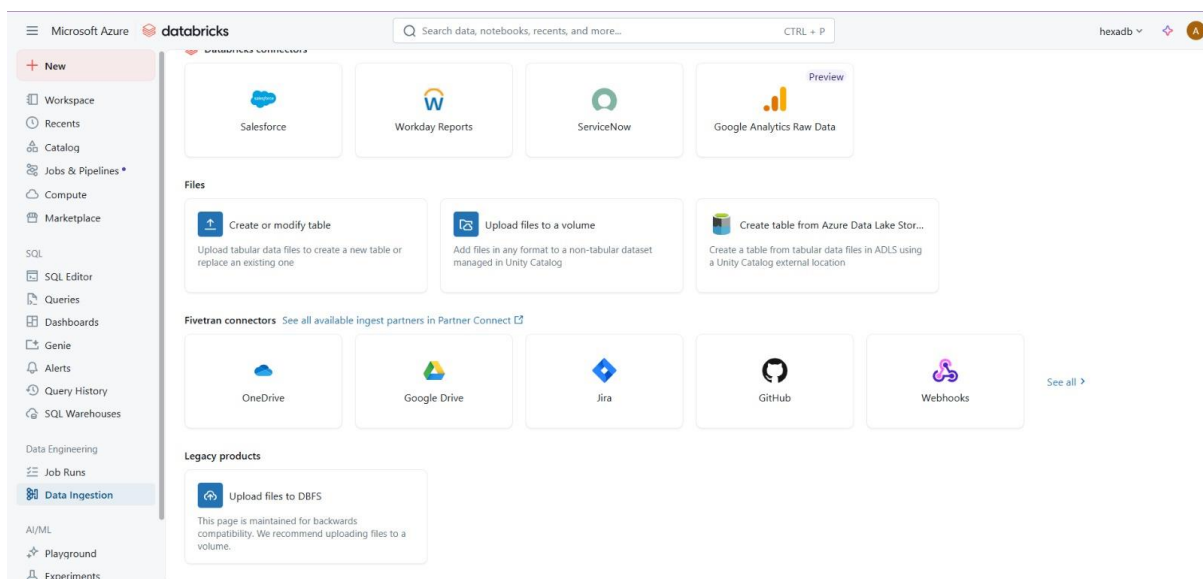


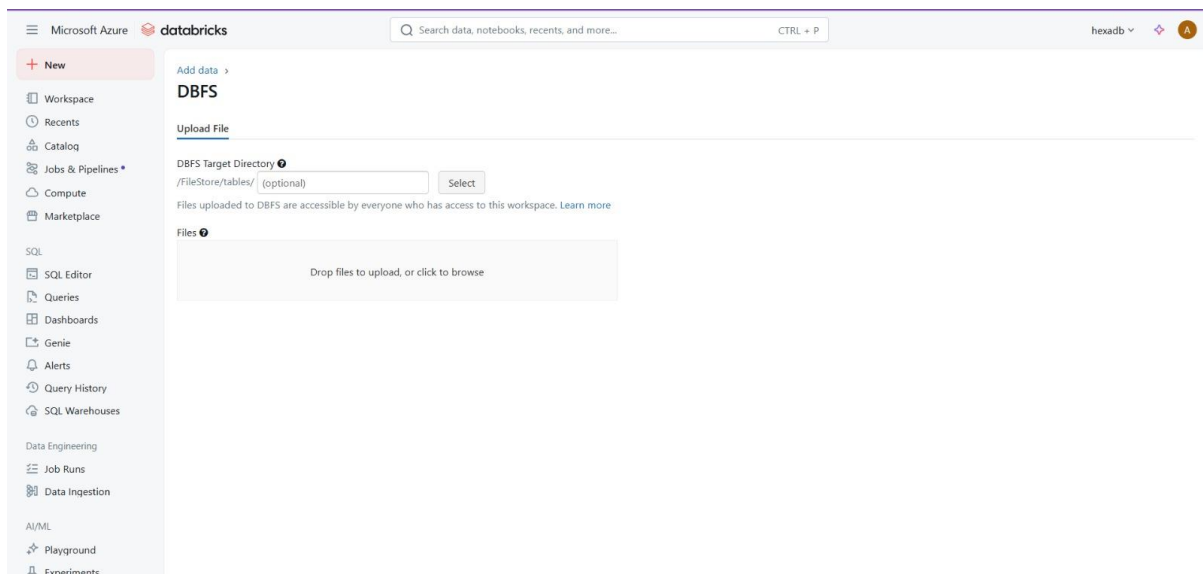
Once the cluster is created, you can view its details and status. Image shows the detailed configuration of the new cluster, including its name, policy, and performance settings. This is a follow-up step to verify the cluster was created correctly.





With the cluster running, you can now ingest data. Image shows the "Data Ingestion" page, which provides options for uploading files, creating tables, or connecting to various data sources



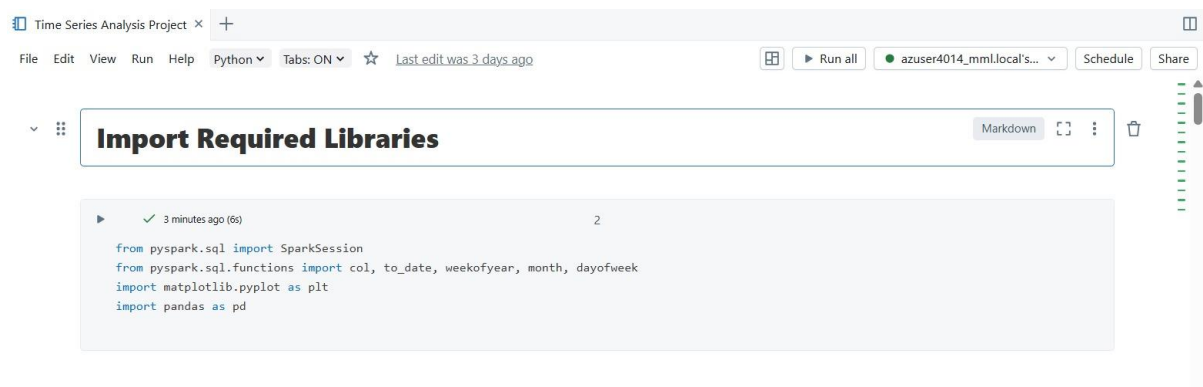


The final step is to upload the data file itself. Image shows the empty DBFS upload page, where you can drag and drop a file. The second Image shows the same page after the "**daily-website-visitors.csv**" file has been successfully uploaded, with a green checkmark and file details.

SUCCESSFUL OUTPUT GENERATION

1. Import Required Libraries

The project begins by importing the necessary libraries. `SparkSession` is imported to manage Spark applications, and `pyspark.sql.functions` is used for data manipulation within Spark DataFrames. Libraries for plotting (`matplotlib.pyplot`) and data handling in Python (`pandas`) are also imported.



2. Loading the Dataset (CSV)

The code loads a CSV file named `daily_website_visitors-3.csv` into a Spark DataFrame. It specifies options to treat the first row as the header and to infer the data types of the columns. The output shows the first few rows and the original column names, which are not very clean.

The screenshot shows a Databricks notebook interface. The top bar indicates the project is "Time Series Analysis Project" and the last edit was 3 days ago. The notebook title is "Loading the dataset(CSV)".

The code cell shows the following Python code:

```
# File location and type
file_location = "/FileStore/tables/daily_website_visitors-3.csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema) \
    .option("header", first_row_is_header) \
    .option("sep", delimiter) \
    .load(file_location)

display(df)
```

The output shows the DataFrame structure: `df: pyspark.sql.connect.dataframe.DataFrame = [_c0: string, _c1: string ... 6 more fields]`.

The table view displays the first 23 rows of the dataset. The columns are labeled `_c0` through `_c7`. The data shows a sequence of days from Sunday to Sunday, with corresponding visitor counts.

	<code>_c0</code>	<code>_c1</code>	<code>_c2</code>	<code>_c3</code>	<code>_c4</code>	<code>_c5</code>	<code>_c6</code>	<code>_c7</code>
9	8	Sunday	1	9/21/2014	2,288	1,656	1,481	175
10	9	Monday	2	9/22/2014	3,638	2,586	2,312	274
11	10	Tuesday	3	9/23/2014	4,462	3,257	2,989	268
12	11	Wednesday	4	9/24/2014	4,414	3,175	2,891	284
13	12	Thursday	5	9/25/2014	4,315	3,029	2,743	286
14	13	Friday	6	9/26/2014	3,323	2,249	2,033	216
15	14	Saturday	7	9/27/2014	1,656	1,180	1,040	140
16	15	Sunday	1	9/28/2014	2,465	1,806	1,613	193
17	16	Monday	2	9/29/2014	4,096	2,873	2,577	296
18	17	Tuesday	3	9/30/2014	4,474	3,032	2,720	312
19	18	Wednesday	4	10/1/2014	4,124	2,849	2,541	308
20	19	Thursday	5	10/2/2014	3,514	2,489	2,239	250
21	20	Friday	6	10/3/2014	3,005	2,097	1,856	241
22	21	Saturday	7	10/4/2014	2,054	1,436	1,274	162
23	22	Sunday	1	10/5/2014	2,847	1,913	1,713	200

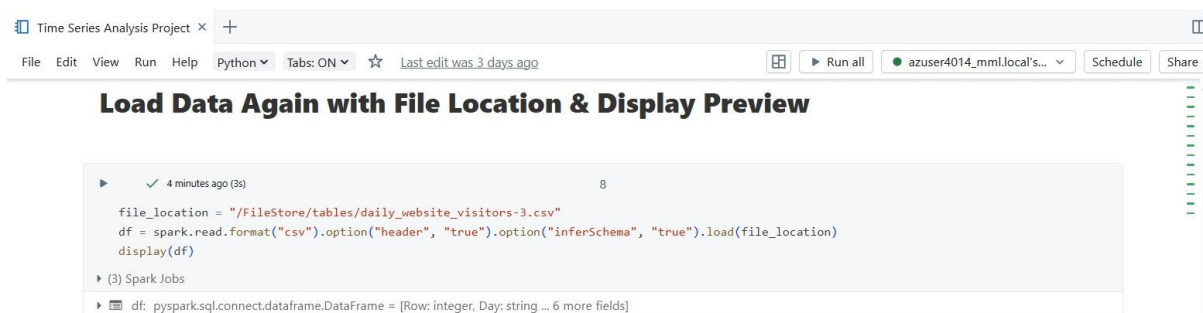
The bottom of the table view shows "2,168 rows | 12.16s runtime" and "Refreshed 3 minutes ago".

Creating a Spark SQL Database To organize and store processed data, a new database named `ts_project_write` is created using Spark SQL. The `CREATE DATABASE IF NOT EXISTS` command ensures it won't overwrite an existing one, and `USE ts_project_write` sets it as the active workspace for future queries.



The screenshot shows a Databricks notebook interface. The title bar indicates the notebook is named 'Time Series Analysis Project'. The menu bar includes File, Edit, View, Run, and Help. The language is set to Python. The toolbar shows 'Run all', a user profile 'azuser4014_mml.local's...', 'Schedule', and 'Share' buttons. The main content area has a heading 'Creating Database'. Below it, a code cell shows two lines of Spark SQL code: `spark.sql("CREATE DATABASE IF NOT EXISTS ts_project_write")` and `spark.sql("USE ts_project_write")`. The code is marked as successful with a green checkmark and '4 minutes ago (3s)'. Below the code, the output is 'DataFrame[]'.

Previewing the Time Series Data The loaded DataFrame is displayed, showing daily metrics like page loads, unique visits, and first-time visits. Each row corresponds to a specific date, with additional columns like day and day of the week, making it easier to analyze patterns over time.



The screenshot shows the same Databricks notebook interface. The main content area has a heading 'Load Data Again with File Location & Display Preview'. Below it, a code cell shows the following Spark code: `file_location = "/FileStore/tables/daily_website_visitors-3.csv"`, `df = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(file_location)`, and `display(df)`. The code is marked as successful with a green checkmark and '4 minutes ago (3s)'. Below the code, the output shows '(3) Spark Jobs' and a preview of the DataFrame: 'df: pyspark.sql.connect.dataframe.DataFrame = [Row: integer, Day: string ... 6 more fields]'. The notebook title bar and toolbar are identical to the previous screenshot.

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

(3) Spark Jobs

df: pyspark.sql.connect.dataframe.DataFrame = [Row: integer, Day: string ... 6 more fields]

Row	Day	Day.OfWeek	Date	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits
1	Sunday	1	2014-09-14	2,146	1,582	1,430	152
2	Monday	2	2014-09-15	3,621	2,528	2,297	231
3	Tuesday	3	2014-09-16	3,698	2,630	2,352	278
4	Wednesday	4	2014-09-17	3,667	2,614	2,327	287
5	Thursday	5	2014-09-18	3,316	2,366	2,130	236
6	Friday	6	2014-09-19	2,815	1,863	1,622	241
7	Saturday	7	2014-09-20	1,658	1,118	985	133
8	Sunday	1	2014-09-21	2,288	1,656	1,481	175
9	Monday	2	2014-09-22	3,638	2,586	2,312	274
10	Tuesday	3	2014-09-23	4,462	3,257	2,989	268
11	Wednesday	4	2014-09-24	4,414	3,175	2,891	284
12	Thursday	5	2014-09-25	4,315	3,029	2,743	286
13	Friday	6	2014-09-26	3,323	2,249	2,033	216
14	Saturday	7	2014-09-27	1,656	1,180	1,040	140

2,167 rows | 2.60s runtime Refreshed 3 minutes ago

Saving the DataFrame as a Table The cleaned DataFrame is saved as a permanent table named `daily_visitors` using `saveAsTable` in overwrite mode. This ensures the data is stored in the active Spark SQL database and can be queried later without reloading the CSV.

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

2,167 rows | 2.60s runtime Refreshed 3 minutes ago

Save DataFrame as Table (daily_visitors)

```
df.write.mode("overwrite").saveAsTable("daily_visitors")
```

(1) Spark Jobs

Exploring the Saved Table Using `SHOW TABLES`, the code confirms that `daily_visitors` exists in the database. Then, a query retrieves the first five rows, showing metrics like page loads, unique visits, first-time visits, and returning visits—ready for deeper time series analysis.

Time Series Analysis Project × +

File Edit View Run Help Python ▾ Tabs: ON ▾ ☆ Last edit was 3 days ago

Run all azuser4014_mml.local's... Schedule Share

Explore Tables in Database

```
spark.sql("SHOW TABLES").show()
spark.sql("SELECT * FROM daily_visitors LIMIT 5").show()
```

▶ (1) Spark Jobs

	database	tableName	isTemporary
	ts_project_write	daily_visitors	false

Row	Day	Day.Of.Week	Date	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits
1	Sunday	1	2014-09-14	2,146	1,582	1,430	152
2	Monday	2	2014-09-15	3,621	2,528	2,297	231
3	Tuesday	3	2014-09-16	3,698	2,630	2,352	278
4	Wednesday	4	2014-09-17	3,667	2,614	2,327	287
5	Thursday	5	2014-09-18	3,316	2,366	2,130	236

3. Data Cleaning & Column Renaming

The project then renames the columns to a more user-friendly format by removing periods and converting to lowercase and snake case (e.g., Day.Of.Week becomes day_of_week).

Time Series Analysis Project × +

File Edit View Run Help Python ▾ Tabs: ON ▾ ☆ Last edit was 3 days ago

Run all azuser4014_mml.local's... Schedule Share

Data Cleaning & Column Renaming

```
from pyspark.sql.functions import col

df = df.toDF([c.strip().replace(" ", "_").replace(".", "_").lower() for c in df.columns])
display(df)
```

▶ (1) Spark Jobs

df: pyspark.sql.connect.dataframe.DataFrame = [row: integer, day: string ... 6 more fields]

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

(1) Spark Jobs

df: pyspark.sql.connect.dataframe.DataFrame = [row: integer, day: string ... 6 more fields]

row	day	day_of_week	date	page_loads	unique_visits	first_time_visits	returning_visits
5	Thursday	5	2014-09-18	3,316	2,366	2,130	236
6	Friday	6	2014-09-19	2,815	1,863	1,622	241
7	Saturday	7	2014-09-20	1,658	1,118	985	133
8	Sunday	1	2014-09-21	2,288	1,656	1,481	175
9	Monday	2	2014-09-22	3,638	2,586	2,312	274
10	Tuesday	3	2014-09-23	4,462	3,257	2,989	268
11	Wednesday	4	2014-09-24	4,414	3,175	2,891	284
12	Thursday	5	2014-09-25	4,315	3,029	2,743	286
13	Friday	6	2014-09-26	3,323	2,249	2,033	216
14	Saturday	7	2014-09-27	1,656	1,180	1,040	140
15	Sunday	1	2014-09-28	2,465	1,806	1,613	193
16	Monday	2	2014-09-29	4,096	2,873	2,577	296
17	Tuesday	3	2014-09-30	4,474	3,032	2,720	312
18	Wednesday	4	2014-10-01	4,124	2,849	2,541	308
19	Thursday	5	2014-10-02	3,514	2,489	2,239	250

2,167 rows | 0.88s runtime Refreshed 4 minutes ago

4. Convert String Dates to Standard Format

To enable proper time-based analysis, the "date" column is transformed from string to a standard date type using `to_date(col("date"), "M/d/yyyy")`. This ensures consistent formatting across the dataset, allowing accurate filtering, grouping, and trend evaluation for metrics like page loads, unique visits, and returning visits.

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

Convert String Date Column to Standard Date Format

```

08:35 PM (1s) 16
df = df.withColumn("date", to_date(col("date"), "M/d/yyyy"))
display(df)

```

(1) Spark Jobs

df: pyspark.sql.connect.dataframe.DataFrame = [row: integer, day: string ... 6 more fields]

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

(1) Spark Jobs

df: pyspark.sql.connect.dataframe.DataFrame = [row: integer, day: string ... 6 more fields]

row	day	day_of_week	date	page_loads	unique_visits	first_time_visits	returning_visits
1	Sunday		2014-09-14	2,146	1,582	1,430	152
2	Monday		2014-09-15	3,621	2,528	2,297	231
3	Tuesday		2014-09-16	3,698	2,630	2,352	278
4	Wednesday		2014-09-17	3,667	2,614	2,327	287
5	Thursday		2014-09-18	3,316	2,366	2,130	236
6	Friday		2014-09-19	2,815	1,863	1,622	241
7	Saturday		2014-09-20	1,658	1,118	985	133
8	Sunday		2014-09-21	2,288	1,656	1,481	175
9	Monday		2014-09-22	3,638	2,586	2,312	274
10	Tuesday		2014-09-23	4,462	3,257	2,989	268
11	Wednesday		2014-09-24	4,414	3,175	2,891	284
12	Thursday		2014-09-25	4,315	3,029	2,743	286
13	Friday		2014-09-26	3,323	2,249	2,033	216
14	Saturday		2014-09-27	1,656	1,180	1,040	140
15	Sunday		2014-09-28	2,465	1,806	1,613	193

2,167 rows | 0.77s runtime Refreshed 5 minutes ago

5. Daily Visit Trend Analysis

The code casts `unique_visits` to integer and aggregates total visits per day using `groupBy("date")` and `F_sum`, producing a new `DataFrame` that reveals how daily traffic fluctuates over time—essential for spotting patterns, peaks, or anomalies in user engagement.

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

Trend Analysis

Daily Trend Analysis

```

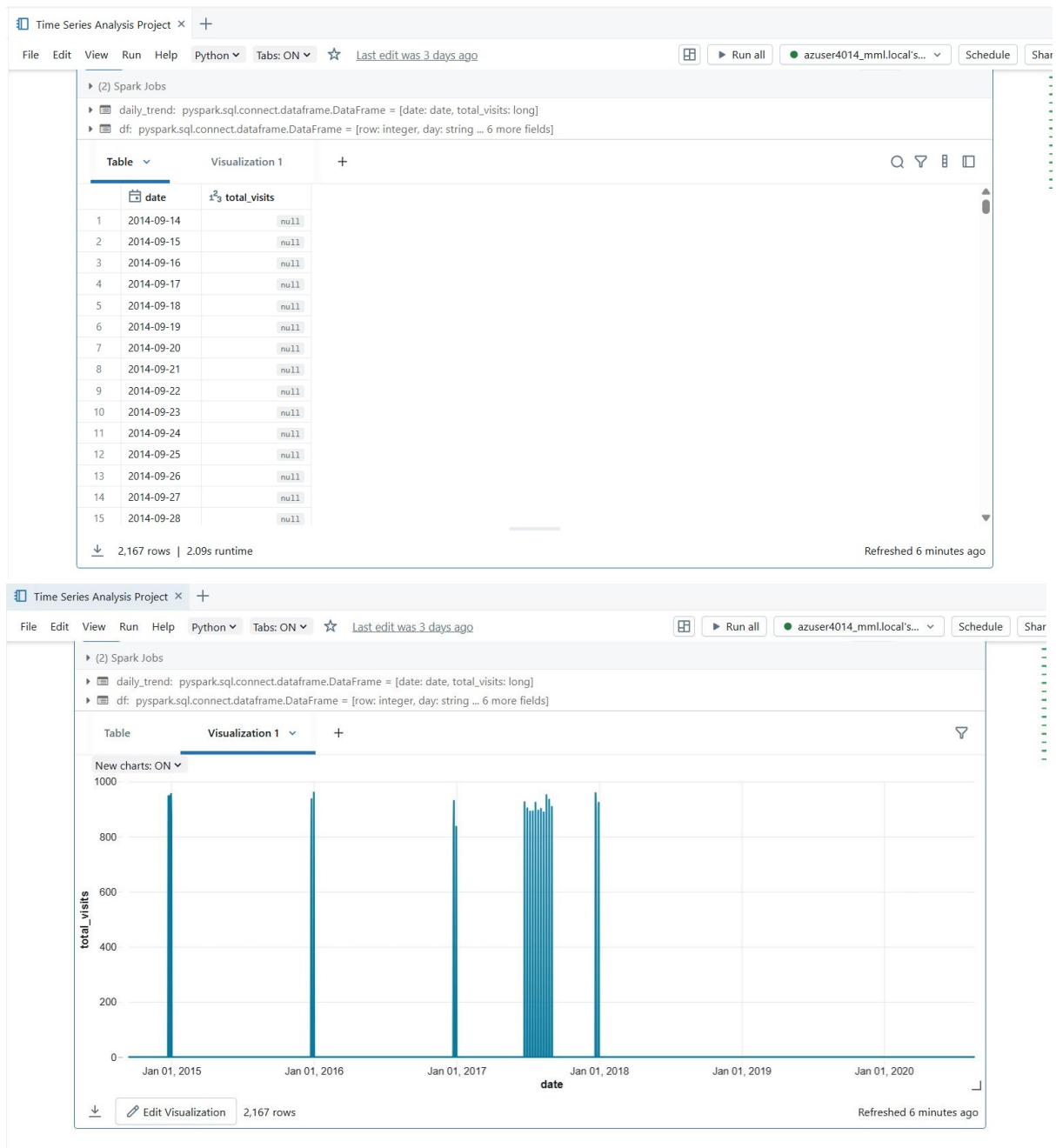
from pyspark.sql.functions import col, month, to_date, dayofweek, date_format, avg, sum as F_sum
df = df.withColumn("unique_visits", col("unique_visits").cast("int"))
daily_trend = df.groupBy("date").agg(F_sum("unique_visits").alias("total_visits")).orderBy("date")
display(daily_trend)

```

(2) Spark Jobs

daily_trend: pyspark.sql.connect.dataframe.DataFrame = [date: date, total_visits: long]

df: pyspark.sql.connect.dataframe.DataFrame = [row: integer, day: string ... 6 more fields]



6. Weekly Visit Trend Analysis

The data is grouped by week using `weekofyear("date")`, and total visits are summed to reveal weekly traffic patterns. However, most weeks show null values for `total_visits`, indicating missing data that needs to be addressed before drawing reliable conclusions.

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

Weekly Trend Analysis

08:35 PM (1s) 21

```
weekly_trend = df.groupBy(weekofyear("date").alias("week")).agg(F_sum("unique_visits").alias("total_visits")).orderBy("week")
display(weekly_trend)
```

(2) Spark Jobs

weekly_trend: pyspark.sql.connect.dataframe.DataFrame = [week: integer, total_visits: long]

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

weekly_trend: pyspark.sql.connect.dataframe.DataFrame = [week: integer, total_visits: long]

Table Visualization 1

	week	total_visits
1	1	1831
2	2	null
3	3	null
4	4	null
5	5	null
6	6	null
7	7	null
8	8	null
9	9	null
10	10	null
11	11	null
12	12	null
13	13	null
14	14	null
15	15	null

53 rows | 0.96s runtime

Refreshed 7 minutes ago

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago Run all azuser4014_mml.local's... Schedule Share

weekly_trend: pyspark.sql.connect.dataframe.DataFrame = [week: integer, total_visits: long]

Table Visualization 1

New charts: ON

total_visits

week

53 rows

Refreshed 7 minutes ago

7. Monthly Visit Trend Analysis

The data is grouped by month using `month("date")`, and total visits are aggregated to reveal monthly traffic patterns. While some months show clear spikes—especially month 12—others contain null values, highlighting the need for data cleaning before drawing conclusions or building forecasts.

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago

Monthly Trend Analysis

08:35 PM (1s) 23

```
monthly_trend = df.groupBy(month("date").alias("month")).agg(F.sum("unique_visits").alias("total_visits")).orderBy("month")
display(monthly_trend)
```

(2) Spark Jobs

monthly_trend: pyspark.sql.connect.dataframe.DataFrame = [month: integer, total_visits: long]

Time Series Analysis Project

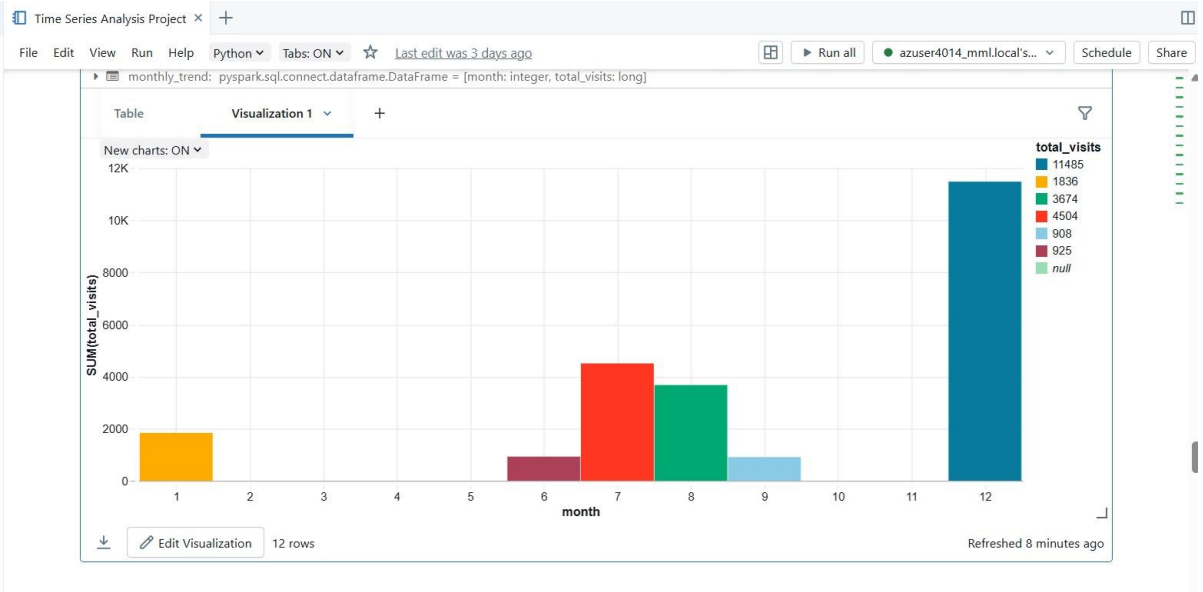
File Edit View Run Help Python Tabs: ON Last edit was 3 days ago

monthly_trend: pyspark.sql.connect.dataframe.DataFrame = [month: integer, total_visits: long]

Table		Visualization 1	+
1 st month	2 nd total_visits		
1	1	1836	
2	2	null	
3	3	null	
4	4	null	
5	5	null	
6	6	925	
7	7	4504	
8	8	3674	
9	9	908	
10	10	null	
11	11	null	
12	12	11485	

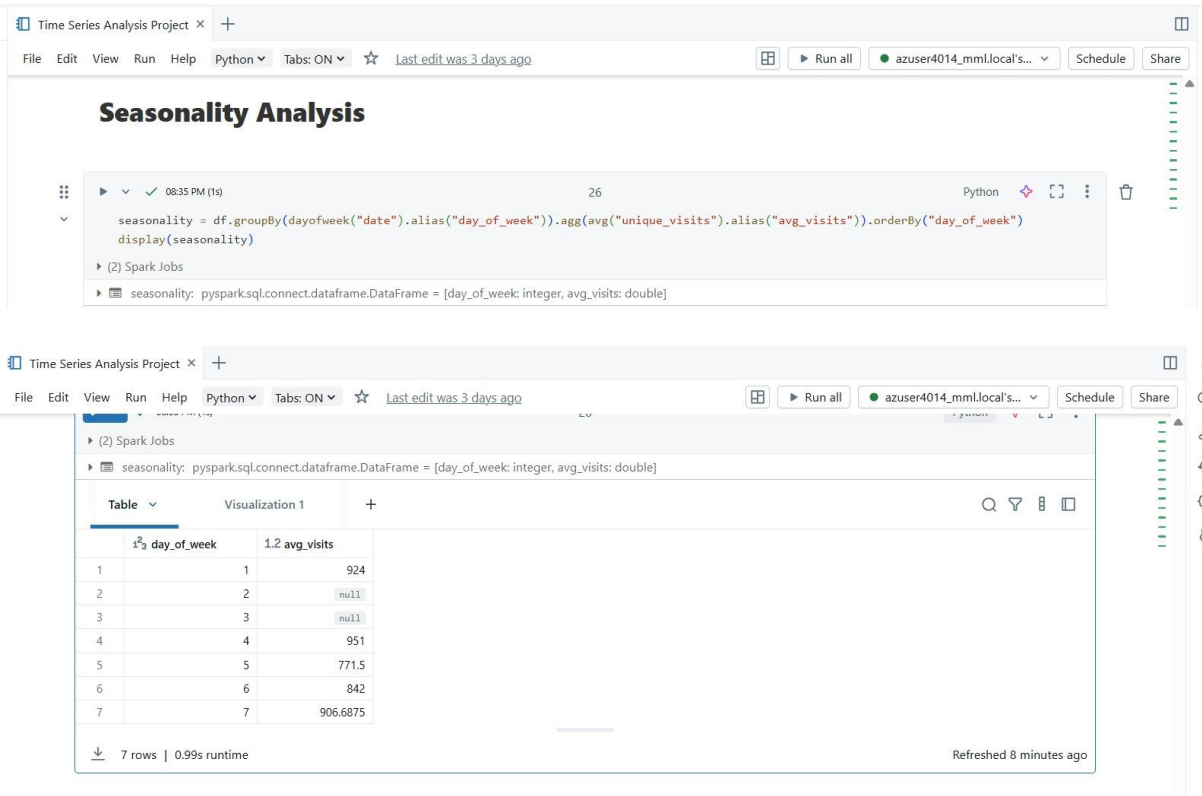
12 rows | 0.82s runtime

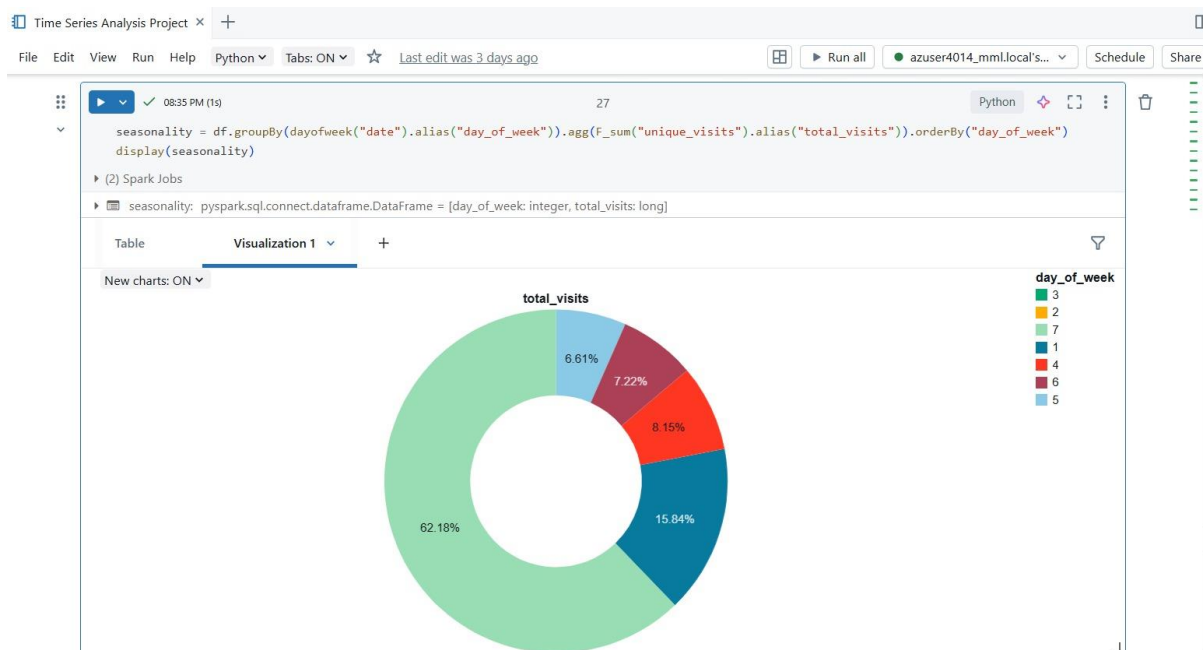
Refreshed 8 minutes ago



8. Seasonality Analysis by Day of Week

The data is grouped by day of the week using `dayofweek("date")`, and average visits are calculated to uncover weekly seasonality patterns. While some days show strong engagement, others have missing values, suggesting uneven data coverage that may affect interpretation.





9. Anomaly Detection Using Z-Score

The Z-score method is applied to detect outliers in daily visit counts by calculating the mean and standard deviation of `total_visits`, then computing a Z-score for each date. Any data point with an absolute Z-score above 3 is flagged as an anomaly—like December 25, 2014, which shows a significant drop in visits and stands out as a statistical outlier.

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago

Run all azuser4014_mml.local's... Schedule Share

Anomaly Detection (Z-Score Method)

08:35 PM (2s) 29

```
from pyspark.sql import functions as F

daily_visits = df.groupBy("date").agg(F.sum("unique_visits").alias("total_visits"))
stats = daily_visits.agg(
    F.mean("total_visits").alias("mean_val"),
    F.stddev("total_visits").alias("std_val")
).collect()[0]
mean_val = stats["mean_val"]
std_val = stats["std_val"]

daily_with_z = daily_visits.withColumn(
    "z_score",
    (F.col("total_visits") - mean_val) / std_val
)

anomalies = daily_with_z.filter(F.abs(F.col("z_score")) > 3)

display(anomalies.orderBy("date"))
```

(5) Spark Jobs

anomalies: pyspark.sql.connect.dataframe.DataFrame = [date: date, total_visits: long ... 1 more field]

daily_visits: pyspark.sql.connect.dataframe.DataFrame = [date: date, total_visits: long]

Time Series Analysis Project

File Edit View Run Help Python Tabs: ON Last edit was 3 days ago

Run all azuser4014_mml.local's... Schedule Share

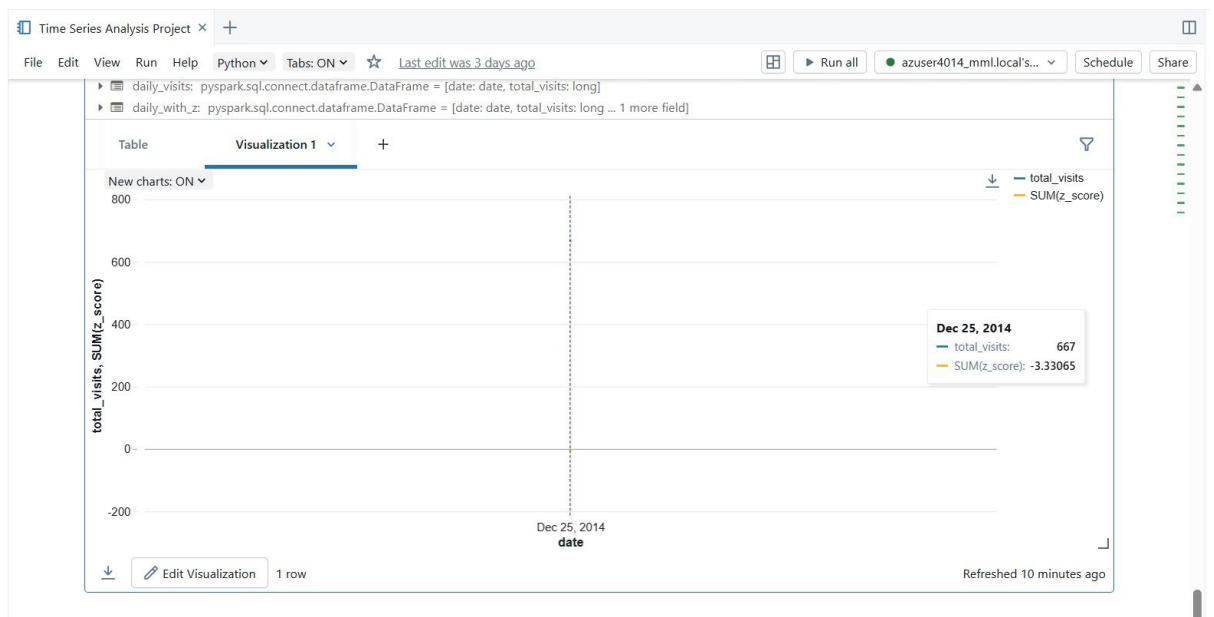
daily_visits: pyspark.sql.connect.dataframe.DataFrame = [date: date, total_visits: long]

daily_with_z: pyspark.sql.connect.dataframe.DataFrame = [date: date, total_visits: long ... 1 more field]

Table		Visualization 1	+
	date	total_visits	z_score
1	2014-12-25	667	-3.33064665899987...

1 row | 2.07s runtime

Refreshed 9 minutes ago



Conclusion

This project successfully utilized **Azure Databricks** to perform Time Series Analysis on website visitor data. By leveraging the power of PySpark within Databricks, the dataset was uploaded, processed, and analyzed in a scalable cloud environment.

- **Efficient Data Handling:** The dataset was seamlessly uploaded into Databricks, enabling smooth data preprocessing and exploration without the need for external storage containers.
- **Time Series Exploration:** Various analytical techniques were applied to identify trends, patterns, and seasonal variations in website traffic over time.
- **Visualization and Insights:** Graphical representations provided clear insights into visitor behavior, highlighting peak periods and long-term growth trends.
- **Optimized Analysis Environment:** Azure Databricks offered an interactive and collaborative workspace, ensuring efficient execution of transformations and supporting data-driven decision-making.

Overall, the project demonstrated the effectiveness of Azure Databricks as a platform for **scalable time series analysis** and provided valuable insights into the dataset.

