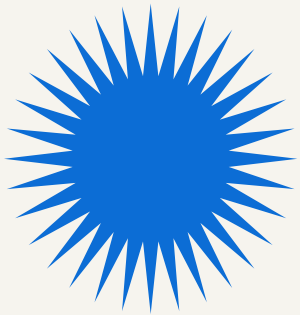
March 1, 2025

ROCKET LEAGUE DATA PIPELINE PROJECT



AGENDA ✶

Introduction to Rocket League Data Pipeline

Phase 1: Data Exploration

Loading Multiple CSV Files

Exploratory Data Analysis (EDA)

Cleaning Considerations

Star Schema Model

Data Wrangling and Feature Engineering

Visualization and Insights

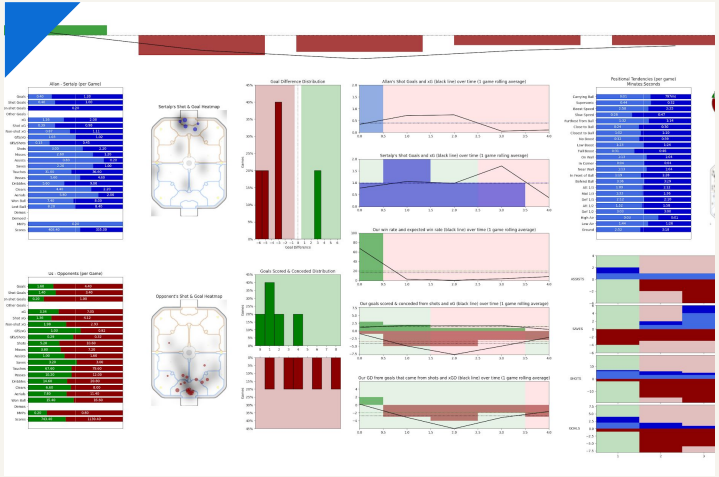
3D Scatter Plots of Player Positions

Final Notes and Documentation

INTRODUCTION TO ROCKET LEAGUE DATA PIPELINE

OVERVIEW OF THE DATA PIPELINE AND EDA

- The Rocket League data pipeline consists of a series of steps to collect, process, and analyze match data, focusing on player performance and game dynamics.
- Exploratory Data Analysis (EDA) is the initial phase in this pipeline, aimed at summarizing the main characteristics of the dataset, often using visual methods.
- EDA helps identify patterns, anomalies, and relationships within the data that can inform further analysis and modeling efforts, ensuring that the data is clean and relevant for downstream use.



Plus tip:

Consider adding specific examples of the types of data collected in Rocket League matches, such as player positions, ball velocity, and scoring events, to enhance the overview.

PHASE 1: DATA EXPLORATION

Plus tip:

Customize this slide by adding specific examples of data exploration techniques you used or insights gained from the initial data review.

• The initial phase focuses on understanding the dataset's structure and contents using libraries like pandas, numpy, matplotlib, and seaborn.

LOADING MULTIPLE CSV FILES

Objective: Understand dataset structure and contents.

Libraries Used: `pandas`, `numpy`, `matplotlib.pyplot`, `seaborn`.

First Step: Load multiple CSV files into a single Pandas DataFrame.

- Facilitates efficient handling of split datasets.

Function: `load_multiple_csv` to read relevant CSV files in a folder.
Dataset Preview: Use `rocket_league_df.head()` to assess the first few entries.

Goal: Identify inconsistencies or issues before deeper analysis.

EXPLORATORY DATA ANALYSIS (EDA)

Plus tip:

Consider adding examples of the code used for each step to provide clarity on the EDA process.

Checking DataFrame Info

The preliminary step involved printing the DataFrame's information using 'df.info()' to obtain basic details about the dataset, including the range index and total columns.

Identifying Missing Values

The presence of missing values was assessed by using 'df.isnull().sum()', which provided a count of non-null entries for each column.

Analyzing Data Types

Data types for each column were examined with 'df.dtypes' to ensure homogeneity, allowing for effective analysis and operations on the data.

CLEANING CONSIDERATIONS

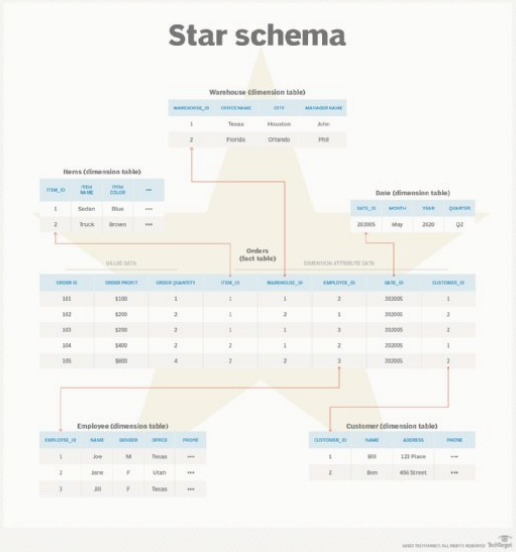
- **Importance**: Ensures dataset accuracy and usability for analytics.
- **Usage**: Supports analytics and modeling for player performance, match dynamics, and esports insights.
- **Goal**: Make data clean and easy to query.
- **Storage Format**: Use **Parquet** for:
  - Efficient compression and encoding to reduce storage costs.
  - Optimization for read-heavy operations.
  - Support for predicate pushdown to accelerate query performance.
- **Benefit**: Enhances storage efficiency and query speed for Rocket League match data.

4o

Plus tip:

Consider including specific examples of how data cleaning has improved model performance in past projects.

STAR SCHEMA MODEL



OVERVIEW OF THE STAR SCHEMA MODEL

- The Star Schema model is optimized for analytical queries, particularly in environments that require fast aggregations and reporting.
- **FACT_GAME_EVENTS Table**: Central table containing event metrics such as ball position, player speed, and event time, linked to dimension tables.
- **DIM_GAME Table**: Stores metadata about games, simplifying the retrieval of game-specific information.
- **DIM_PLAYER Table**: Contains player information, linked directly to the events they participate in.
- **DIM_EVENT Table**: Categorizes event types, allowing for easy filtering and grouping of event data.

Plus tip:

You can customize the visual representation by replacing the diagram with one that matches your branding or by adding specific examples relevant to your analysis.

DATA WRANGLING AND FEATURE ENGINEERING

Plus tip:

Customize the details in each step to reflect the specific data wrangling techniques you used in your analysis, or expand on the significance of each feature in your context.

Calculate Speed

Calculate speed for each player using: `p_speed = np.sqrt(p_vel_x**2 + p_vel_y**2 + p_vel_z**2)` to enrich the dataset with player performance metrics.

Player speed metrics

Distance to Ball

Calculate distance from each player to the ball using: `p_dist_to_ball = np.sqrt((p_pos_x - ball_pos_x)**2 + (p_pos_y - ball_pos_y)**2 + (p_pos_z - ball_pos_z)**2)` for insights on positioning.

Distance metrics to ball

Adjust Data Types

Ensure appropriate data types for each column. Round ball positions using: `ball_pos_x = round(ball_pos_x)` to enhance precision for analysis.

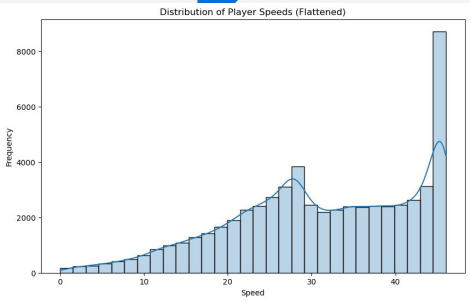
Adjusted data types and rounded values

Feature Extraction

Extract features for better insights, such as event frequency statistics. This aids in predictive modeling and deeper analysis.

Enriched dataset with extracted features

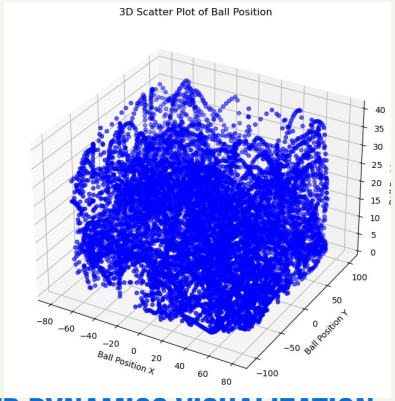
VISUALIZATION AND INSIGHTS



DISTRIBUTION OF PLAYER SPEEDS

Visualizations reveal the distribution of player speeds across all players. The histogram shows how player speeds vary with key insights into performance dynamics.

3D SCATTER PLOTS OF PLAYER POSITIONS



PLAYER DYNAMICS VISUALIZATION

3D scatter plots illustrate player positions in Rocket League, revealing spatial relationships and movement patterns during matches.

PLAYER POSITIONING OVER TIME

Each player's position is tracked across time, demonstrating how players navigate the arena in relation to the ball and each other.

INSIGHTS FROM SCATTER PLOTS

The scatter plots enable analysts to identify strategic positioning and movement efficiency, crucial for performance analysis.

FINAL NOTES AND DOCUMENTATION

This notebook has performed exploratory data analysis (EDA) by checking for data cleanliness, evaluating storage formats, and enriching the dataset for future consumption.

The next steps involve transforming and loading the cleaned and enriched data into a database or data warehouse, following the ETL pipeline. Ensuring that the data is clean and easily queryable is crucial for effective downstream analytics and model building.

Scaling the Rocket League Data Pipeline in Azure Databricks

01 **Original Prototype Limitations**

Developed with pandas, limited to single machine, not suitable for large datasets. Relied on local storage and manual logging.

02 **Migration to Azure Databricks**

Utilized PySpark for distributed processing, deployed on Azure Spark clusters, enabling scalable cloud access and processing.

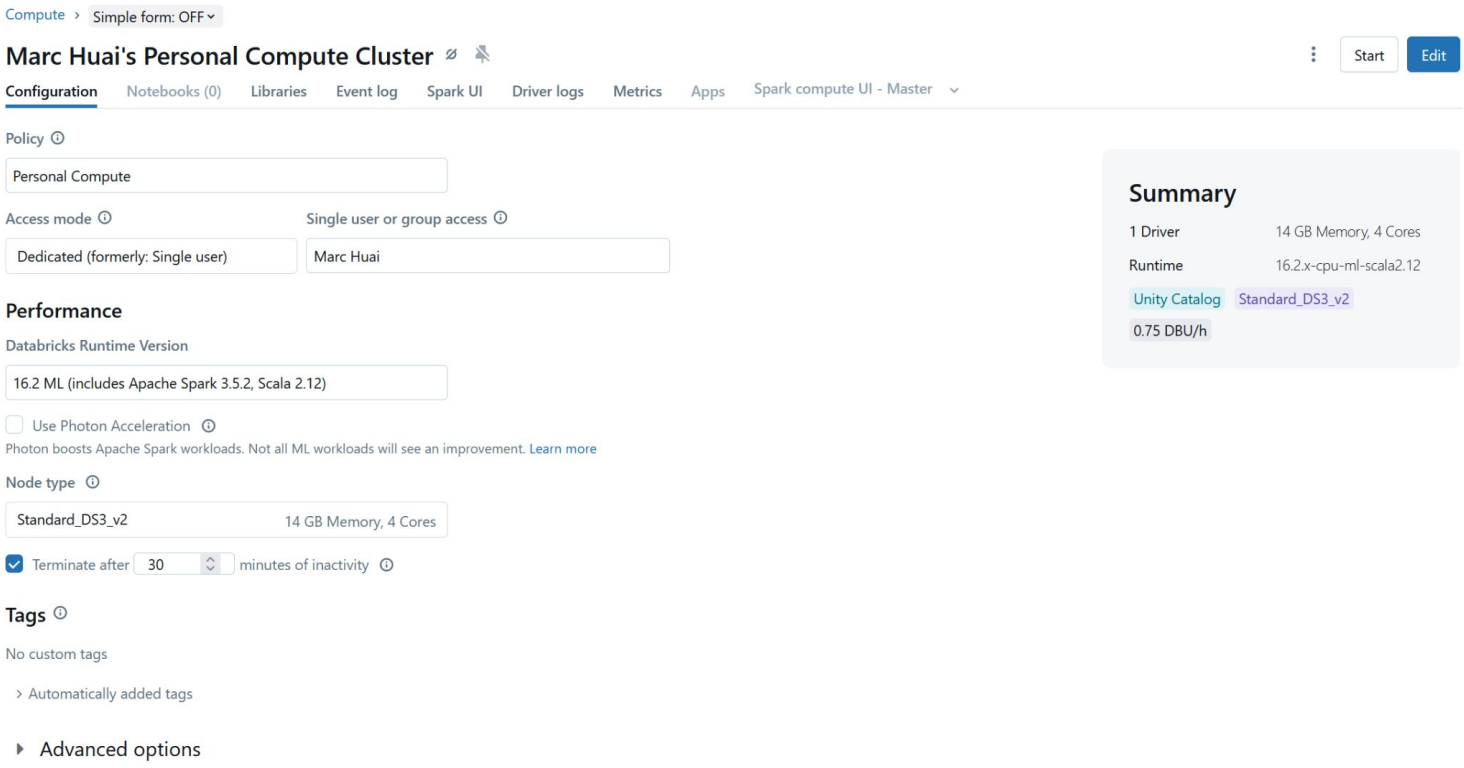
03 **Performance Improvements**

Achieved faster processing with parallel operations, handling millions of rows efficiently. Enhanced logging and modularity simplify monitoring.

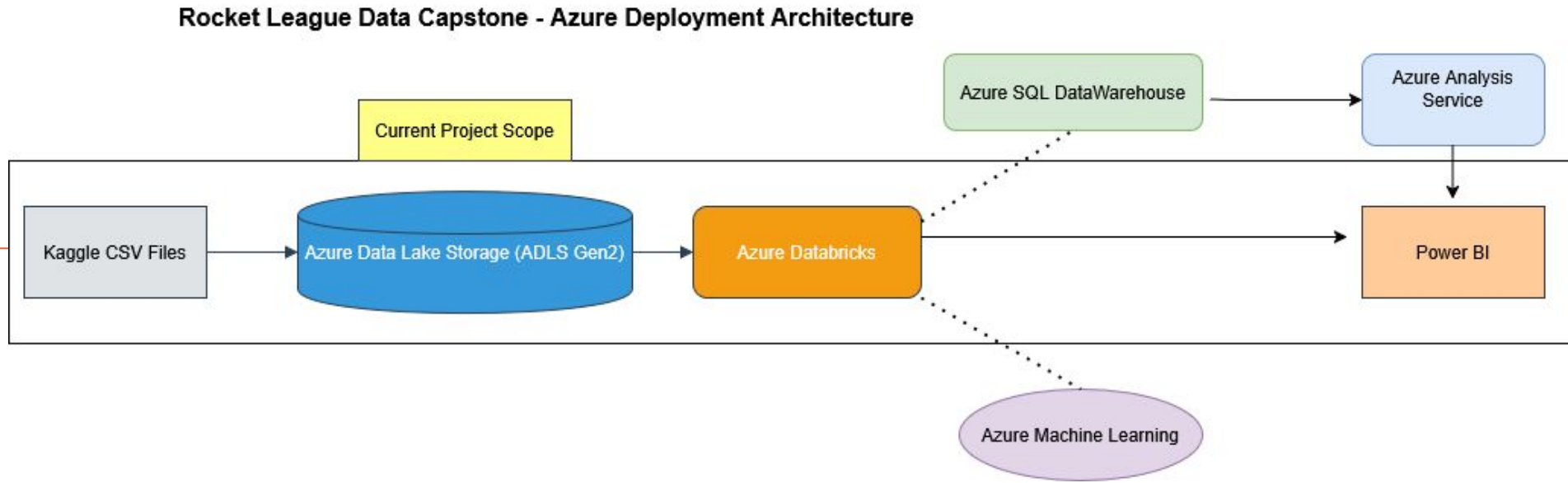
04 **Production-Ready Features**

Horizontally scalable, supports real-time and batch processing. Cloud architecture is fault-tolerant and testable, suitable for production.

Cluster Creation Snapshot



Architecture Diagram



Architecture Rationale

Project Overview

The architecture processes Rocket League game data using Azure services, integrating storage, analytics, and visualization tools for streamlined workflow.

Key Components

Utilizes Azure Data Lake Storage, Databricks, SQL Data Warehouse, Power BI, and Azure ML for comprehensive data handling and analysis.

Trade-offs Made

Excluded Azure Data Factory due to manual file uploads, opted for SQL Data Warehouse over Synapse Analytics for smaller datasets.

Rationale Benefits

This setup offers scalability, efficient data processing, and ease of use, crucial for analyzing Rocket League data effectively.

Testing & Deployment Enhancements in Rocket League Pipeline

Comprehensive Test Suite	Implemented using Python's unittest framework, focusing on validating core pipeline methods like load_data(), clean_data(), and store_as_parquet().
Edge Case Handling	Tested null-handling logic with rows of None values, verified type casting to TimestampType, and ensured schema consistency for Parquet files.
Codebase Modifications for Testing	Introduced Configuration.max_rows for sample limiting, modularized Spark operations, and adjusted logging for Azure blob resilience.
Deployment Architecture Adjustments	Validated Azure Blob permissions, ensured compatibility in test and full runs, and used DBFS paths for sandboxed validations.

Unit Testing Results Snapshot

```
.INFO:__main__:Attempting to load data from wasbs://rocketleague-data@trial25.blob.core.windows.net/train_1_subset.csv
INFO:__main__:Data loaded successfully with 2 records
.INFO:__main__:Attempting to load data from wasbs://rocketleague-data@trial25.blob.core.windows.net/train_1_subset.csv
Wrote 59840 bytes.
Logged metrics for stage 'Loading Stage' to:
wasbs://rocketleague-data@trial25.blob.core.windows.net/Pipeline Metrics Log/pipeline_metrics.log
INFO:__main__:Data loaded successfully with 2 records
Wrote 60627 bytes.
Logged metrics for stage 'Loading Stage' to:
wasbs://rocketleague-data@trial25.blob.core.windows.net/Pipeline Metrics Log/pipeline_metrics.log
INFO:__main__:Parquet file written to: wasbs://rocketleague-data@trial25.blob.core.windows.net/sample_test_parquet
.
----------------------------------------------------------------------
Ran 5 tests in 27.116s

OK
```

Migration from Test Code to Final Code

Secret Management Improvement	Replaced hardcoded Azure Storage account key with .env file loaded via python-dotenv, improving security and modularity.
Unit Tests Stripped for Production	Removed test-specific code, such as mocks and sample data creation, to streamline the final pipeline and reduce overhead.
Simplified Code Structure	Refactored configuration and data pipeline classes, removing test dependencies and unnecessary parameterization for cleaner, more efficient code.
Enhanced Logging and Error Handling	Improved log messaging using logger.info() and logger.error() in key functions for better traceability and maintenance.

Finalized Production Deployment Architecture & Code

Azure Integration

Utilizes Azure Data Lake Storage, Databricks, and SQL Data Warehouse for efficient data handling and analysis.

Scalability & Efficiency

Architecture supports both real-time and batch processing, ensuring scalable and efficient data analysis.

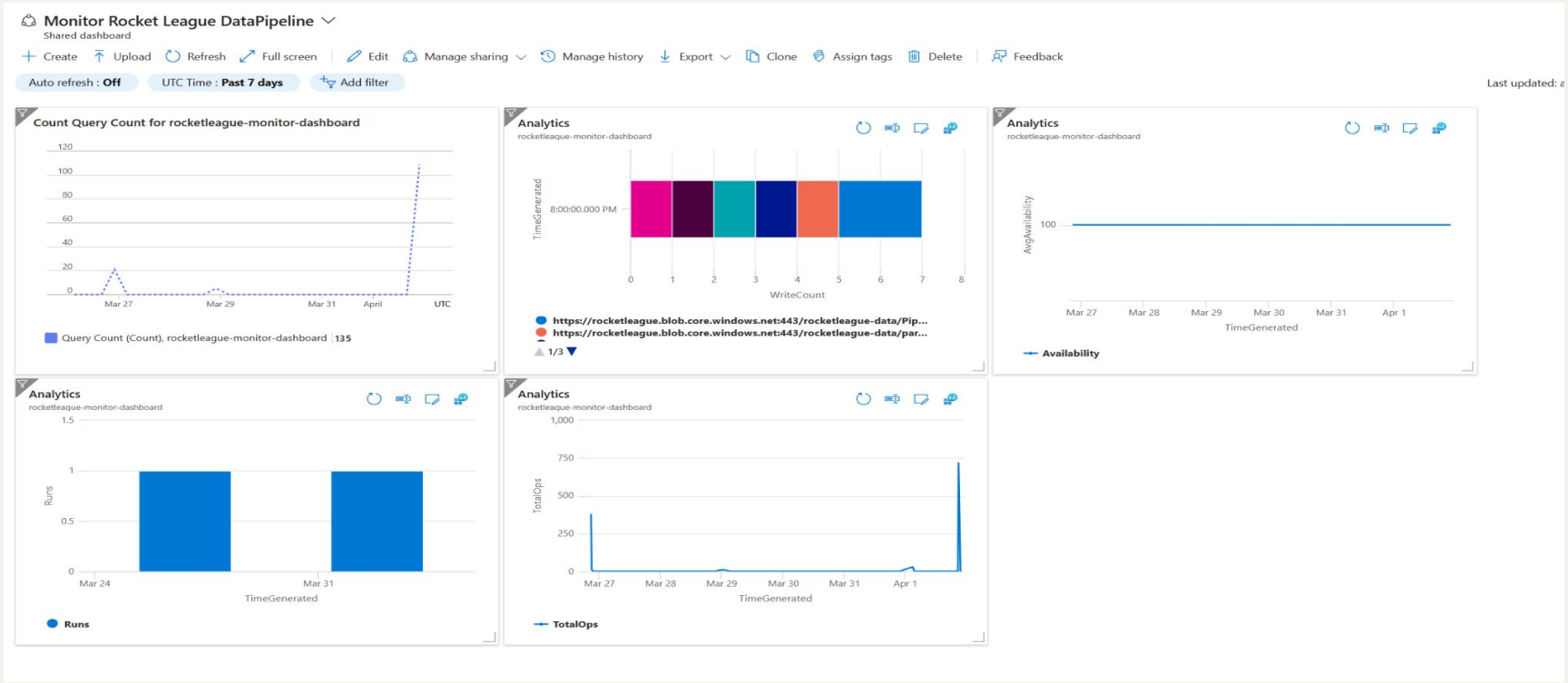
Security Enhancements

Implements .env files for secure storage access, enhancing security by avoiding hardcoded credentials.

Logging & Monitoring

Enhanced logging with Azure Blob resilience, providing better traceability and system maintenance.

Dashboard Snapshot for Rocket League Pipeline



Dashboard Metrics

Why These Metrics Were Chosen

- Operational Health: Availability and Blob Operations help monitor the pipeline infrastructure
- Pipeline Output Monitoring: Write count per URI shows which outputs are being produced and when
- Pipeline Execution Validation: Notebook Runs ensure the ETL process is executing on schedule
- Observability: Query Count reflects how and how often the dashboard is used (manual or automated)

What the Metrics Mean

Tile	What It Shows	Why It Matters
- Query Count - Write Count by URI - Availability (%) - Notebook Runs - Total Blob Operations	- Number of log queries made to monitor the dashboard - Number of writes (PutBlob) to files (.parquet, .log, etc.) - Storage availability over time - Timestamped runs of the Rocket League pipeline - Total PUT/GET/DELETE operations on storage	- Tracks dashboard usage and automation activity - Identifies most active pipeline outputs - Helps detect service downtime or write failures - Validates scheduled pipeline execution - Measures data throughput and system workload

Final Dashboard Outcome

- The dashboard provides a comprehensive view of the Rocket League data pipeline's performance metrics and operational status.
- Real-time monitoring of pipeline executions and file outputs ensures that data processes are running smoothly and efficiently.
- Key metrics include latency, uptime, and the success rate of data processing steps, allowing for immediate identification and resolution of issues.
- The dashboard's design supports data engineering goals by making the pipeline observable, measurable, and auditable.
- Users can proactively detect and address potential issues, such as missing writes or pipeline failures, minimizing downtime and enhancing reliability.

THANK YOU