IPL Team Analytics Project Documentation

-Suraj Anil Badchikar
-INT-22

Project Overview

The project aims to analyze IPL team performance, player contributions, and venue impact using structured pipeline design and **Medallion Architecture** using :

- **PySpark** for Data Preprocessing and ETL,
- **SQL Server** for structured data storage and aggregation,
- **Power BI** for visualization,
- **Azure Data Lake Gen2** and **ADF** for cloud storage and orchestration.

Architecture: Medallion Approach

Layer	Purpose	Format
Bronze	Raw data storage	Parquet
Silver	Cleaned, enriched, joined data	Parquet + SQL Server (SSMS)
Gold	Aggregated analytical model for Power BI	SQL Server (SSMS)

Data Sources

1. team.csv

Column	Description
team_id	Unique team identifier
team_name	Name of IPL team
home_ground	Stadium name
captain	Team captain

2. player_team.csv

Column	Description
player_id	Unique player ID
player_name	Name of the player
team_id	Foreign key to team
player_role	Role (Batsman/Bowler)

3. match_performance.csv

Column	Description
match_id	Match identifier
team_id	Team that played
opponent_team_id	Opponent
runs_scored	Runs scored
wickets_taken	Wickets taken
match_result	Win/Loss/Tie

4. stadium.csv

Column	Description
stadium_id	Unique stadium ID
stadium_name	Stadium name
city	Location
capacity	Seating capacity

Utility Functions - Used Notebook workflows to make use of common functions like , read , write , list etc.

The file is common_functions and its functions are as follows :-

list_files_in_blob –
This function helps user to list all files in a given path and return a list of file paths

read_file –
This function helps uer to read a file from a given path and return a Spark DataFrame and the source file name

write_file –
This function helps user to write a Spark DataFrame to a given path and return the source file name while adding audit columns(ingestion_date, source_file)

delete_file_from_blob –
Deletes a file or folder from Azure Blob Storage using full abfss path.

null_counts –
Counts the number of null values in each column of a Spark DataFrame and returns a new DataFrame

write_file_to_silver –
Writes a Spark DataFrame to the 'ipl-silver' container on 'ipldataadlsg2' storage account. Ensures it creates a folder, not a file.

write_df_to_sql –
This function helps user to write a Spark DataFrame to a given table name in a SQL database

I have also set the configurations for adlsg2 , I have used access tokens to get access to the data .

I have used adls gen 2 due to its feature of hierarchical namespacing allowing me to have traditional file system-like features .

1)Data Ingestion convert_csv_to_parquet.ipynb

Steps:-

1)Imports Common Functions
The notebook begins by importing utility functions from a different notebook as mentioned above in utility functions (common_functions) using %run. These functions handle reading, writing, and deleting files from ADLS Gen2, while also adding audit metadata like ingestion_date and source_file.

2)Defines Storage Paths
It sets up parameters for:

- **src_container**: The container where raw CSV files reside (ipl-bronze)
- **storage_acc**: The name of the Azure storage account
- **output_container**: A full path used to write transformed Parquet files back to the same Bronze container

3)CSV to Parquet Conversion Loop
The notebook:

- Uses **list_files_in_blob()** to retrieve all file paths from the Bronze container.
- Iterates over each file path and:
    - Reads the CSV file using read_file()
    - Displays the DataFrame for visual verification (Databricks only)
    - Logs the file being processed
    - Writes the DataFrame to the Bronze layer as a **Parquet** file using write_file(), which also adds metadata columns
    - Deletes the original CSV file using delete_file_from_blob() to keep the storage clean and avoid duplication

4)Optional Testing Block
A commented-out section at the bottom provides a quick test to read and display one of the Parquet files (team_bronze) to confirm successful conversion.

2)Data Cleaning, Transformation & Silver Layer Ingestion

This notebook contains the core data cleaning and transformation step in the Medallion Architecture pipeline. It operates on data already converted into Parquet format in the Bronze Layer and prepares it for the Silver Layer, where data is cleaned, normalized, and made query-ready.

Steps:-

1)Imports Common Functions

The notebook begins by importing utility functions from a different notebook as mentioned above in utility functions (common_functions) using %run. These functions handle reading, writing, and deleting files from ADLS Gen2, while also adding audit metadata like ingestion_date and source_file.

2)Reading All Parquet Files from Bronze

It dynamically lists and reads all datasets stored in the ipl-bronze container using a loop:

- Each file path is converted into a Spark DataFrame.
- The DataFrame is stored with a dynameic variable name that reflects the file name (e.g., team_bronze_df).
- All variable names are stored in a list for easy iteration and identification later on.

3)Null Value Detection

Using the null_counts() utility, the notebook prints null counts for every column in each dataset. This gives an overview of which datasets require cleaning.

4)Each dataset is cleaned based on its unique structure and business logic:

match_performance_bronze_df

- Drops rows with nulls in runs_scored or wickets_taken.
- Filters only valid match results: "Win", "Loss", "Tie".
- Casts ingestion_date to TimestampType.
- Selected relevant columns.

player_team_bronze_df

- Drops null player_name.
- Replaces null player_role with "Unknown".
- Casts ingestion_date to TimestampType.

stadium_bronze_df

- Replaces null capacity values with the column's average.
- Replaces null city with "Unknown".
- Casts ingestion_date to TimestampType.

team_bronze_df

- Replaces null home_ground with "Unknown Homeground".
- Replaces null captain with "Unknown Captain".
- Casts ingestion_date to TimestampType.

match_stadium_bronze_df & player_performance_bronze_df

- Casts ingestion_date to TimestampType.
- Selected relevant columns for performance tracking.

5)Silver Layer Ingestion (ADLS Gen2)

Each cleaned DataFrame is then saved to the **ipl-silver container** using the write_file_to_silver() function. This organizes the cleaned data in a well-structured Silver Layer.

6)Silver Layer Ingestion (Azure SQL Database)

Finally, all cleaned DataFrames are written to an **Azure SQL Database (ipl-db)** under the schema silver_db. The write_df_to_sql() function is reused for this operation.

3)Gold Layer SQL Aggregations

This SQL script represents the final layer of the Medallion Architecture (Gold Layer) where cleaned and structured Silver data is aggregated and transformed analytical models for Power BI dashboards.. These outputs are designed for direct consumption by BI dashboards, data analysts, and stakeholders.

1)Reading Silver Tables

The notebook begins by reading all relevant tables from the **Silver Layer** (silver_db). These include:

- Match performance
- Match-stadium mapping
- Player performance
- Player-team mapping
- Stadium details
- Team details

gold_db.ipl_team_performance_summary

- Shows total matches, wins, win percentage, and home ground for each team.
- Uses match_result from match_performance_silver and joins with team_silver.
- Ideal for comparing overall team strength and consistency.

Insights Provided:

- Best-performing teams by win ratio.
- Home ground influence and usage.

gold_db.venue_performance_summary

- Uses a CTE (match_details) to determine whether each match was a Home or Away match.
- Calculates team performance based on match location.
- Helps evaluate home advantage or away dominance.

Insights Provided:

- Do teams win more often at home?
- How do away stats compare for each team?

gold_db.team_player_performance_summary

- Aggregates stats per player **within their team**.
- Includes total runs, wickets, and matches played.
- Uses player_performance_silver joined with player-team and team details.

Insights Provided:

- Top players by team.
- Player contributions in both batting and bowling.

gold_db.player_efficiency_summary

- Focuses on **individual player efficiency**, regardless of team.
- Calculates:
    - **Average Strike Rate**
    - **Total Batting Runs**
    - **Total Balls Faced**
    - **Innings Played**
    - **Average Runs per Innings**
    - **Wickets Taken**

Insights Provided:

- Best batters and bowlers across all teams.
- Balanced players with dual roles (bat & bowl).
- Efficiency metrics like strike rate and avg runs.

gold_db.player_match_venue_performance

- Detailed breakdown of each player's performance **per match and stadium**.
- Identifies whether it was a home match.
- Captures granular data: runs, wickets, balls faced.

Insights Provided:

- How specific players perform in certain stadiums.
- Venue-wise adaptability and consistency.

The entire Databricks workflow has been seamlessly orchestrated through Azure Data Factory (ADF), enabling end-to-end automation of the Bronze → Silver → Gold data pipeline. This ensures scalable, scheduled, and monitored data processing across all transformation stages.

The final Gold layer outputs have been **visualized in Power BI**, providing interactive dashboards and insightful analytics on team performance, player efficiency, and venue-wise match outcomes for effective decision-making.

I have attached the Power BI dashboard to the Github.

Other Factors I have explored during the project are -

1)SHIR - Self hosted integration run time , I explored this to directly fetch files from the system and reduce the loop of converting files from csv to parquet in the ADLS , it did work but the parquet format it came in was not similar to the Spark pyspark and a quick search told me it lacked some features important to parquet files.

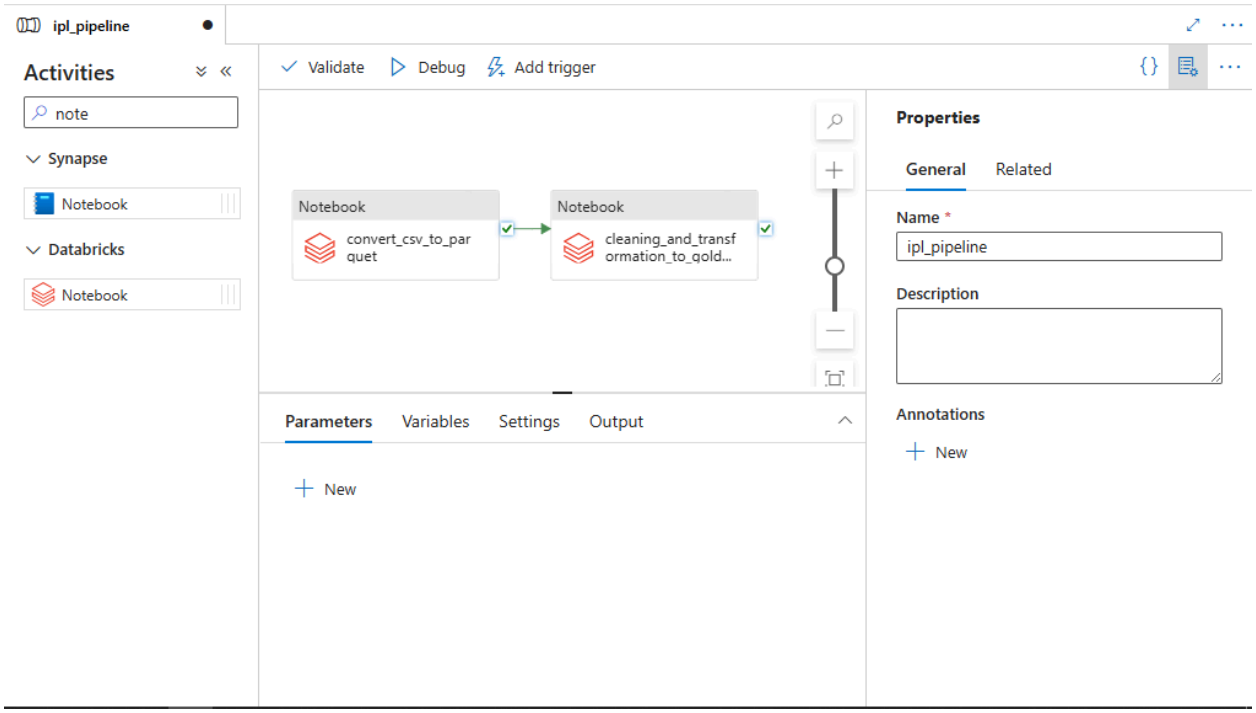
2)Azure key Vault - I have explore this option to get a similar behaviour like .env file used in developing applications , to avoid displaying the keys in my notebook . I did get successful in achieving this but removed it as the project did not actually demand it , and the cost factor the key vault carries .

I'll summarize how I did below :-

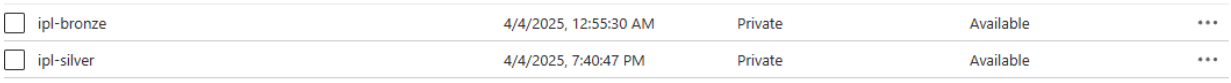
1) Create the Azure key vault service : - and then define secret keys in it .
2) Go to the databricks resource , click on the databricks logo on the left side , type #createScope in the search bar in front of the link and then it takes you to the page related to secret scope
3) Fill the required details there and then,
4) Start accessing secrets in notebooks

I have uploaded the relevant screenshots below :-

1)ADF Pipeline



2)ADLS Gen2 Screenshots :-
Containers



Ipl-bronze parquet files -

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
match_performance_...						-
match_stadium_bron...						-
player_performance_...						-
player_team_bronze						-
stadium_bronze						-
team_bronze						-

n_correlationId=92954877-12...

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: ipl-bronze / match_performance_bronze

Search blobs by prefix (case-sensitive) Show deleted blobs

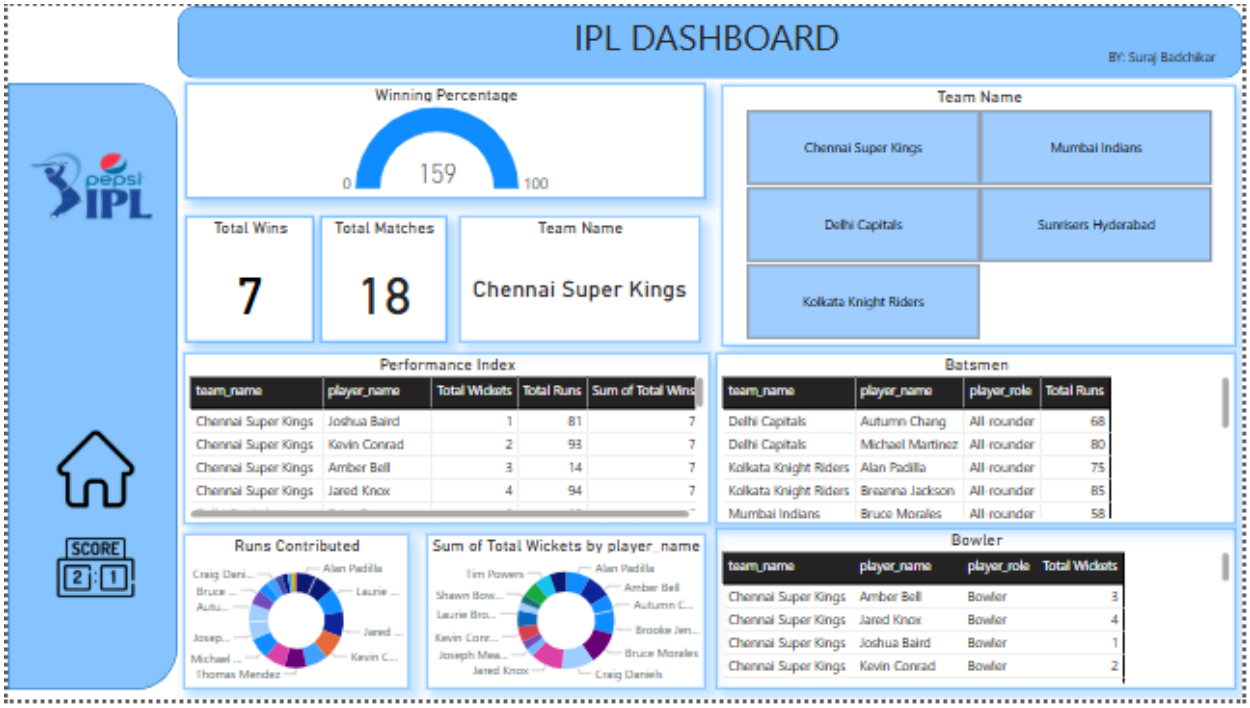
Add filter

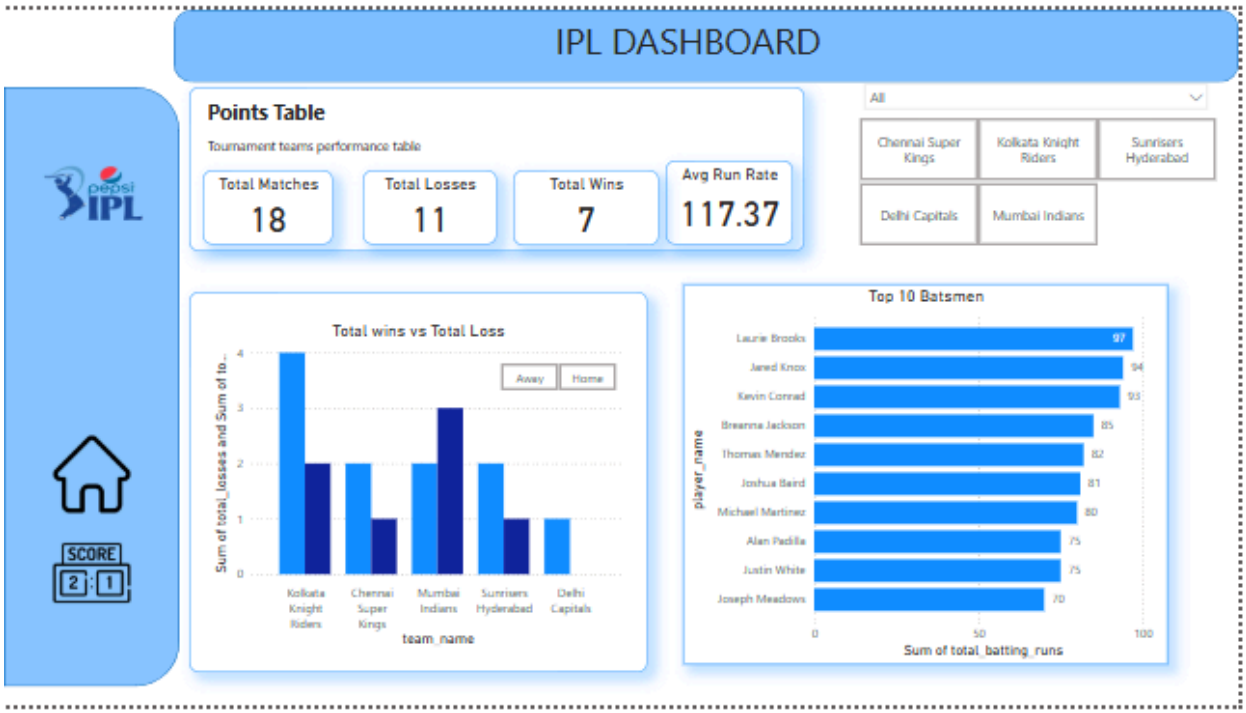
Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
[..]						
_committed_347999...	4/4/2025, 11:50:26 AM	Hot (Inferred)		Block blob	123 B	Available
_started_3479996120...	4/4/2025, 11:50:24 AM	Hot (Inferred)		Block blob	0 B	Available
_SUCCESS	4/4/2025, 11:50:26 AM	Hot (Inferred)		Block blob	0 B	Available
part-00000-tid-3479...	4/4/2025, 11:50:25 AM	Hot (Inferred)		Block blob	3.09 KiB	Available

Ipl-silver container -

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
match_performance-...						-
match_stadium-silver						-
player_performance-...						-
player_team-silver						-
stadium-silver						-
team-silver						-

Power BI Dashboards-





Snapshot of tables in Database : -