

# IPL Data Analytics

## 1. Introduction

We need to build a solution that processes IPL Dataset from 2008 to 2022 and visualize analytics on Power BI Dashboard(s). We need to assume that we are receiving updated data set at specific interval. That dataset raw files are uploaded in data lake. Once we have the new file in data lake, we process the data through a data pipeline and produce modelled data in data warehouse. We will be able to analyse the data in Power BI through pre-defined dashboards.



## 1.1 Project Objective

Our Project Objective is to perform analytics on given IPL dataset and carry out some of the insights from it and to perform queries on the cleaned dataset. Our Project must be suitable for any upcoming real-time data.

## 1.2 Problem Definition

We have a Indian Premier League dataset from the year 2008-2020. This dataset is not in ready to use format and not good enough to give better visualisation of data. We have to extract, load and transform our data. we need to explore of several technologies to build a solution .

## 1.3 Resource & Requirements

### 1. Resource:

**Data sets:** CSV file: IPL matches 2008 to 2020

### 2. Requirements:

- Azure Portal Access
  1. Resource Group
  2. Storage account (Blob/ ADLS Gen2)
  3. Azure Data Factory
  4. Azure Logic App
  5. Azure Key Vault
- Snowflake & Snowpark
- Power BI

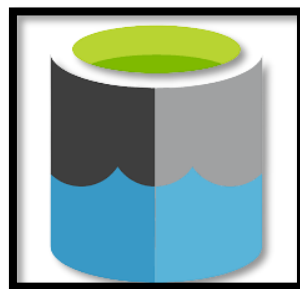
### 3. Services and Tools to be used:

- Azure Data Lake Storage Gen2
- Azure Blob Storage
- Azure Data Factory
  1. Pipeline

2. Linked Service
  3. Dataset
  4. Script Activity
  5. Copy data Activity
  6. Delete Activity
  7. Web Activity
- Azure Key Vault
  - Snowflake & Snowpark
  - Power BI Desktop

## 2. Azure Data Lake Storage Gen2:

Azure Data Lake Storage Gen2 is a set of capabilities dedicated to big data analytics, built on Azure Blob Storage. Data Lake Storage Gen2 converges the capabilities of Azure Data Lake Storage Gen1 with Azure Blob Storage. For example, Data Lake Storage Gen2 provides file system semantics, file-level security, and scale. Because these capabilities are built on Blob storage, we'll also get low-cost, tiered storage, with high availability/disaster recovery capabilities. Data Lake Storage Gen2 builds on Blob storage and enhances performance, management, and security in the following ways.



ADLS Gen2

### 3.2 Azure Blob Storage

Azure Blob storage allows users to store and manage unstructured data such as images, videos, documents, and logs in the cloud. Blob Storage provides three types of blobs: block blobs, append blobs, and page blobs. Block blobs are optimized for storing large amounts of data and are used for streaming and backup

scenarios, while append blobs are optimized for append operations, such as logging data. Page blobs are optimized for random read/write operations and are used for storing virtual hard drive (VHD) files and other structured data. Blob Storage also provides features such as automatic tiering, data lifecycle management, and security options such as encryption and access control.



Azure Blob Storage

### **3.3 Azure Data Factory**

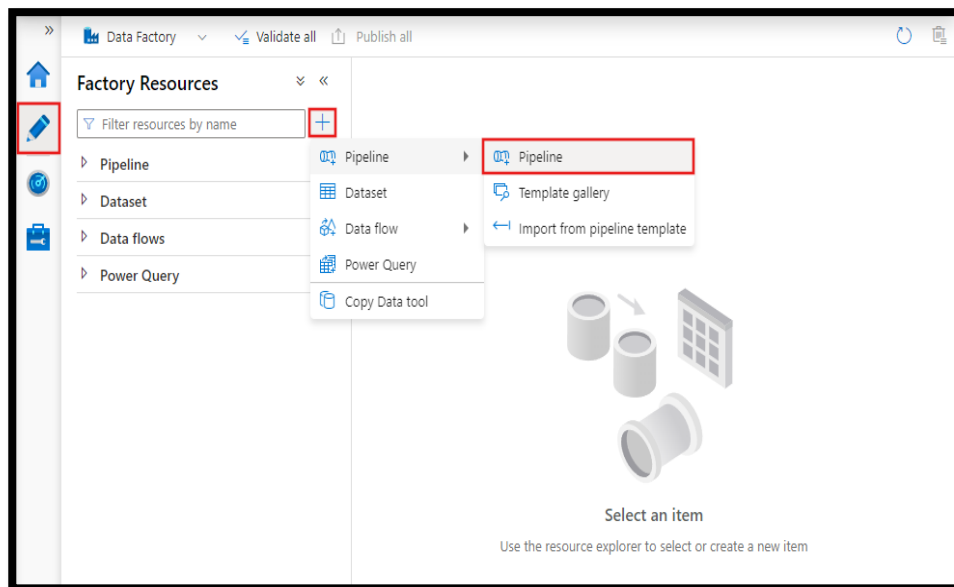
A fully managed, server less data integration solution for ingesting, preparing and transforming all our data at scale. Easily construct ETL and ELT processes code-free in an intuitive environment or write our own code. Then deliver integrated data to Azure Synapse Analytics to unlock business insights. Rehost SQL Server Integration Services (SSIS) in a few clicks and build ETL and ELT pipelines code-free, with built-in Git and CI/CD support. Ingest all our on-premises and software as a service (SaaS) data with more than 90 built-in connectors. Orchestrate and monitor at scale. Use autonomous ETL to unlock operational efficiencies and enable citizen integrators.



Azure Data Factory

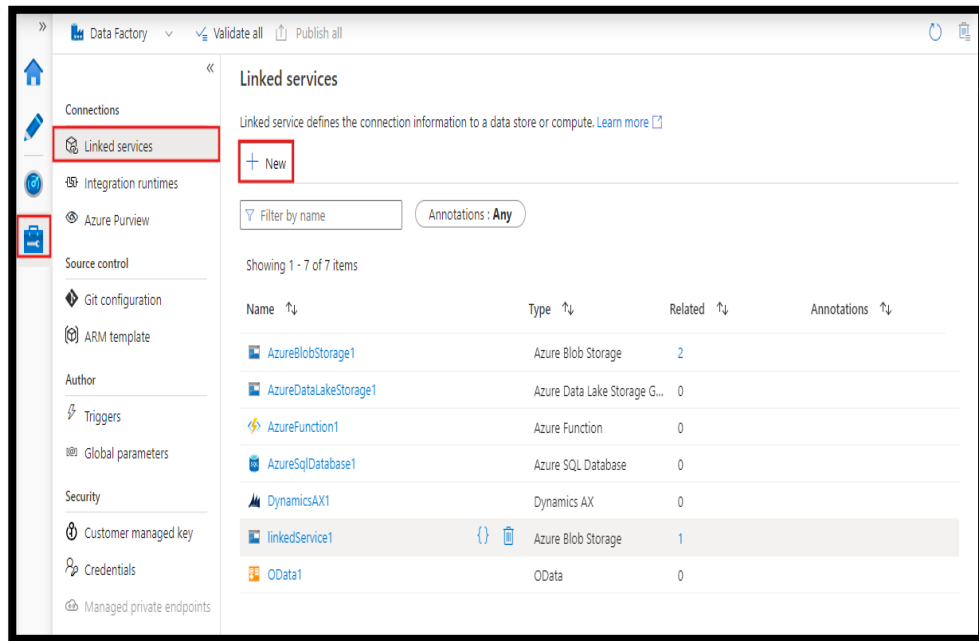
#### **a. Pipelines:**

A pipeline is a logical grouping of activities that together perform a task. A pipeline could contain a set of activities that ingest and clean log data, and then kick off a mapping data flow to analyse the log data. The pipeline allows us to manage the activities as a set instead of each one individually. We deploy and schedule the pipeline instead of the activities independently. A data factory can have one or more pipelines.



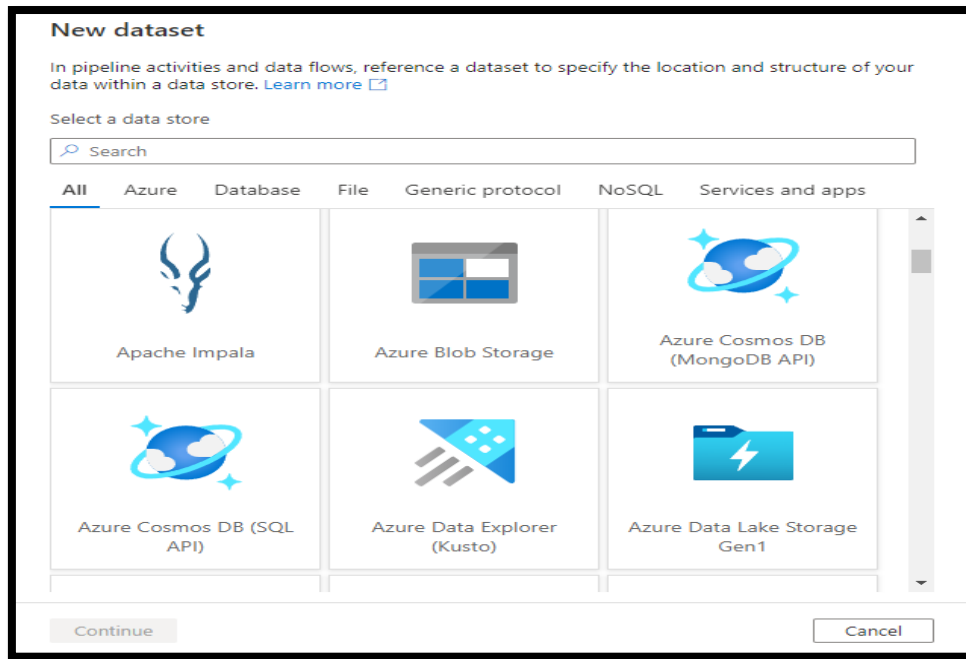
## b. Linked Services:

Linked services are much like connection strings, which define the connection information needed for Data Factory to connect to external resources. Before we create a dataset, we must create a linked service to link our data store to the data factory.



### c. Datasets:

Dataset is a named view of data that simply points or references the data we want to use in our activities as inputs and outputs. Datasets identify data within different data stores, such as tables, files, folders, and documents. The data set represents the structure of the data within the linked data stores, and the linked service defines the connection to the data source. For example, an Azure Storage linked service links a storage account to the data factory. An Azure Blob dataset represents the blob container and the folder within that Azure Storage account that contains the input blobs to be processed.

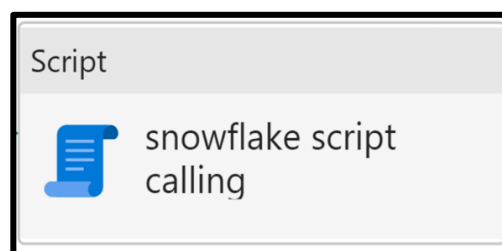


#### d. Script Activity:

The script activity can be used to execute code written in languages like Python, R, or PowerShell. It can be helpful when you need to perform complex data transformations that are not supported by the built-in data transformation activities provided by Azure Data Factory.

The script activity can also be used to perform activities such as calling external APIs, integrating with external systems, and performing data validation or cleansing operations.

To use the script activity, you need to define the script code and any input and output datasets in the activity settings. Once the script activity is executed, the code is run on a compute environment, and the output is written to the specified output dataset.



General
Settings
User properties

Linked service \* ⓘ
SnowflakeConfService

Script
☒ Query ⓘ
☐ NonQuery ⓘ

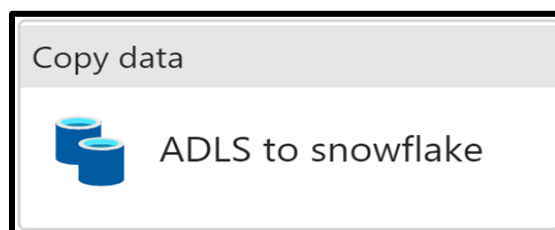
insert into project.config (filename) val...

Script parameters ⓘ

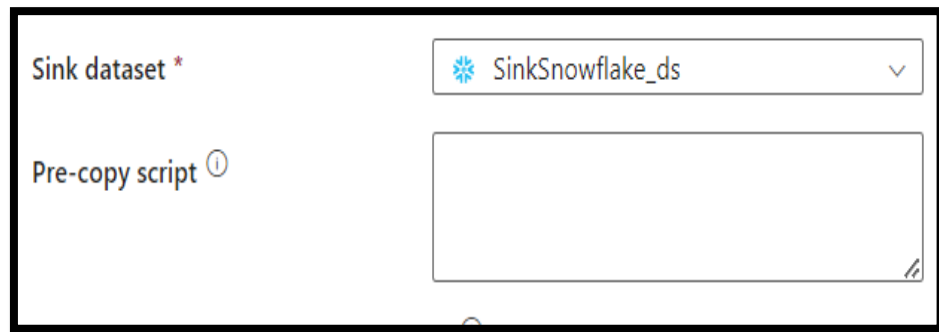
+ New

### e. Copy data Activity

The Copy activity in Azure Data Factory is used to copy data between the different data stores that are located on-premises and in the cloud, in order to use the copied data in other transformation or analysis tasks or copy the transformed or analysed data to the final store to be visualized. Azure Data Factory supports reading from and writing to different file formats, including Avro, Binary, Delimited text, Excel, JSON, ORC, Parquet and XML file formats. For a complete list of the supported data sources (called sources), and data targets (called sinks), in the Azure Data Factory copy activity, check the Supported Data Stores and Formats.







Sink dataset \*

SinkSnowflake\_ds

Pre-copy script ①

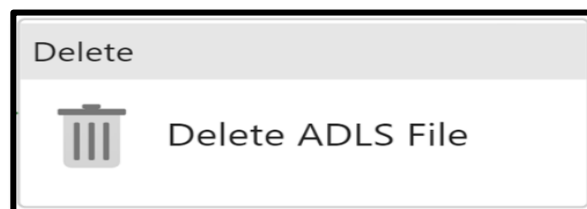
## f. Delete Activity

The "Delete" activity in Azure Data Factory (ADF) pipeline is used to delete files or directories in a specified location such as Azure Blob storage, Azure Data Lake Storage, or on-premises file system.

The "Delete" activity can be useful in various scenarios such as cleaning up the storage before processing new data, deleting old files that are no longer needed, or removing intermediate files generated during a pipeline run.

To use the "Delete" activity in an ADF pipeline, you need to specify the location and path of the file or directory you want to delete. You can also use dynamic content to specify the location and path based on the values of parameters or variables.

Note that the "Delete" activity permanently deletes the specified files or directories, so it's important to use it with caution and ensure that you're deleting the correct files. You may want to consider using a backup or versioning system for important files that you're deleting.



General **Source** Logging settings User properties

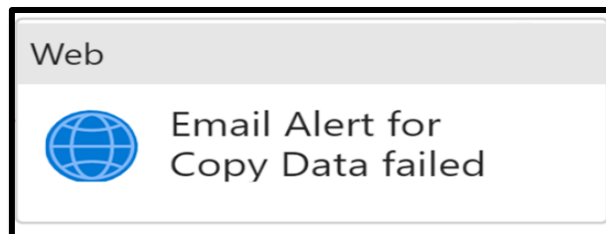
Dataset \* ⓘ Maindir\_ds

File path type ☐ File path in dataset ☒ Wildcard file path

Wildcard file name

## g. Web Activity

Web Activity can be used to call a custom REST endpoint from an Azure Data Factory or Synapse pipeline. We can pass datasets and linked services to be consumed and accessed by the activity.



General **Settings** User properties

URL \* ⓘ   
 ⚠ Information will be sent to the URL specified. Please ensure you trust the URL entered.

Method \* ⓘ POST

Body

Authentication ⓘ None

## 3.3 Azure Key Vault:

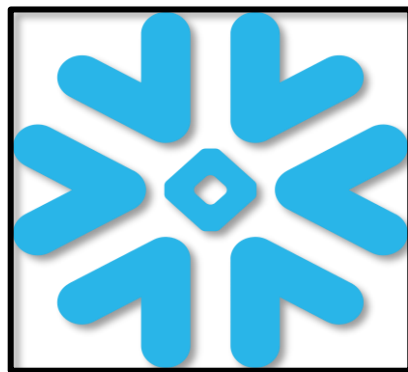
Azure Key Vault is a secure storage platform for managing cryptographic keys, secrets, and certificates used by our applications and services. We can use Azure Key Vault in Azure Data Factory (ADF) to store sensitive information such as connection strings, passwords, and other credentials required by our data integration and ETL pipelines.



Azure Key Vault

### 3.4 Snowflake & Snowpark

Snowflake enables data storage, processing, and analytic solutions that are faster, easier to use, and far more flexible than traditional offerings where Snowpark is a new developer experience for Snowflake that allows developers to write code in their preferred language and run that code directly on Snowflake. It exposes new interfaces for development in Python, Scala, or Java to supplement Snowflake's original SQL interface.



Snowflake

### 3.5 Power BI Desktop

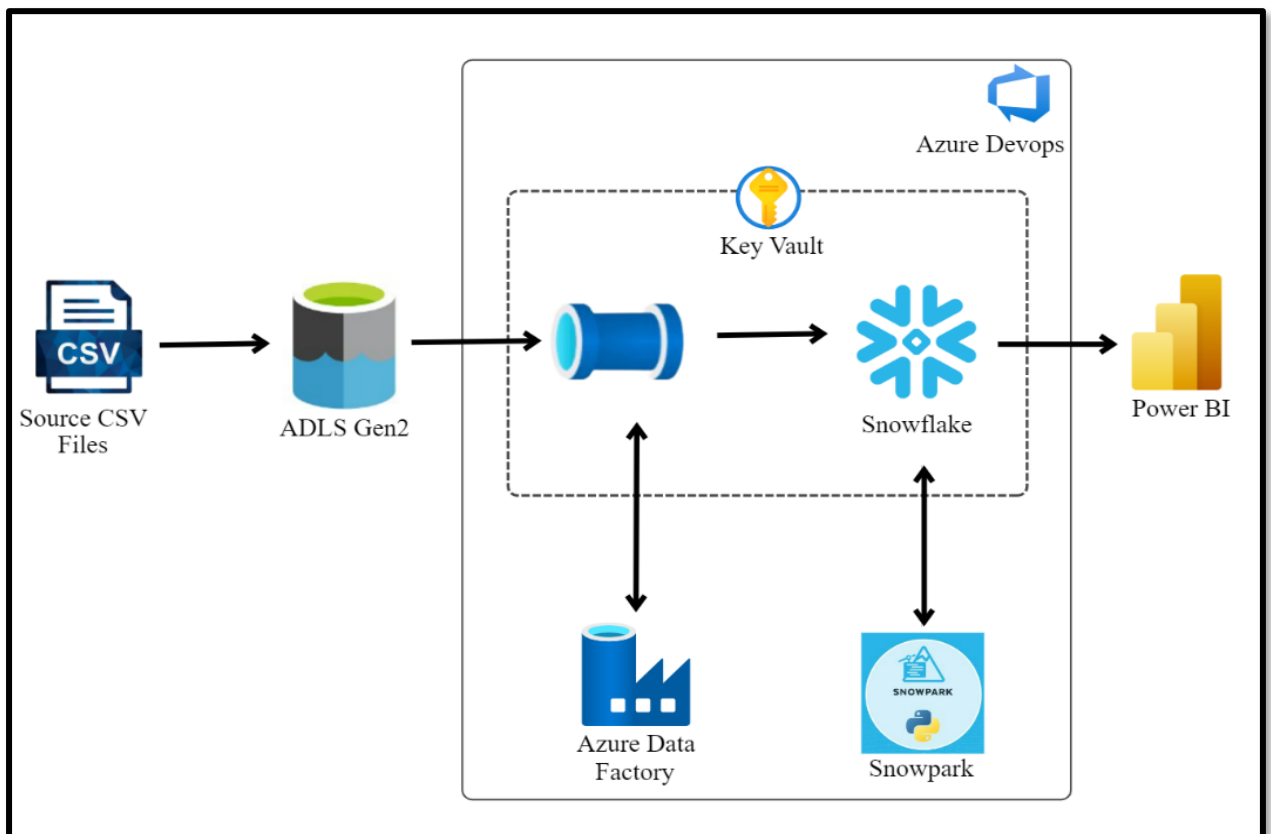
Power BI is Microsoft's interactive data visualization and analytics tool for business intelligence (BI). We can use it to pull data from a wide range of systems in the cloud and on premises and create dashboards that track the metrics we care about the most, or drill in and (literally) ask questions about our data.



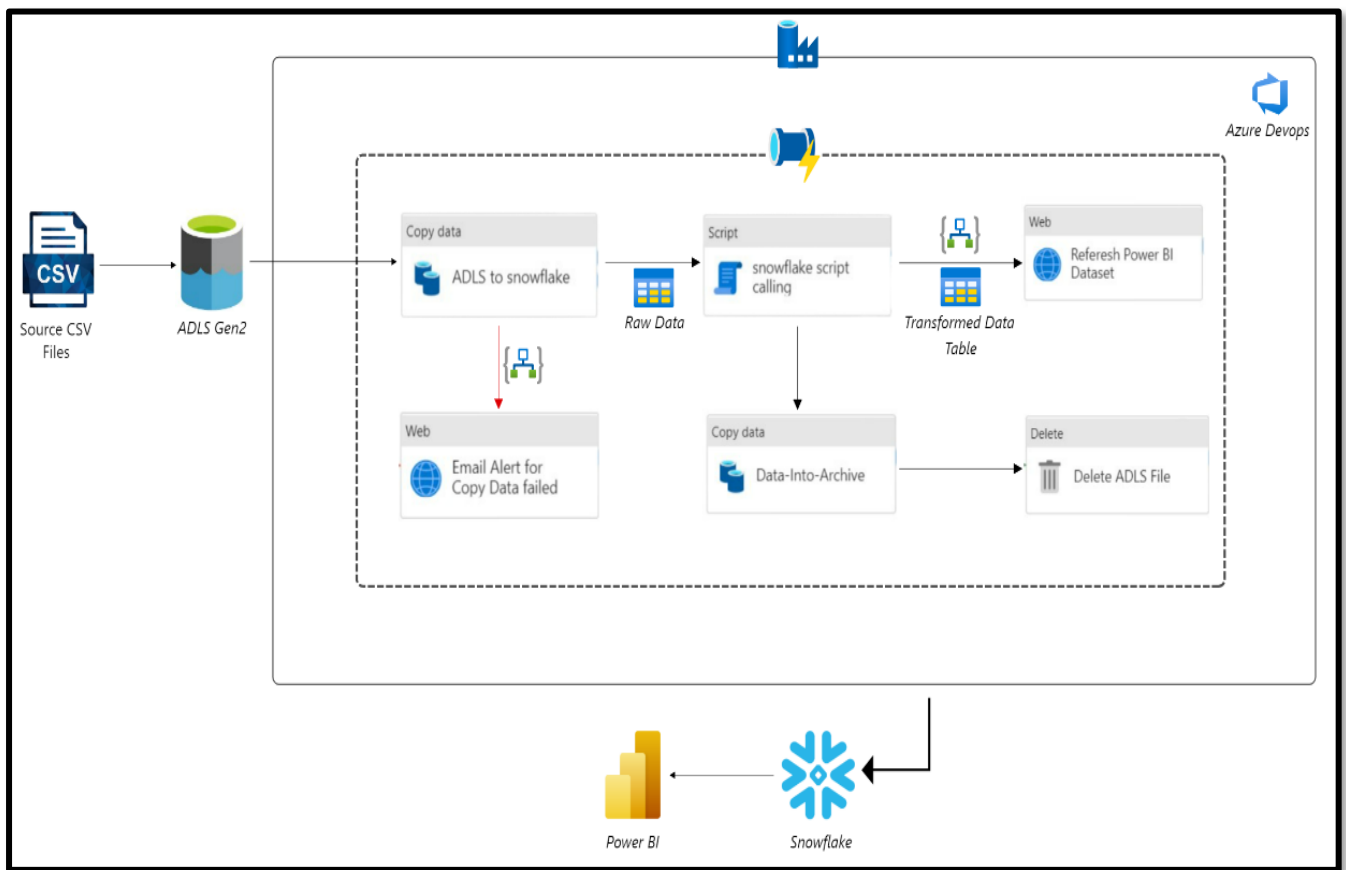
Power BI

## 4. Solution Approach:

### 4.1 Architecture:

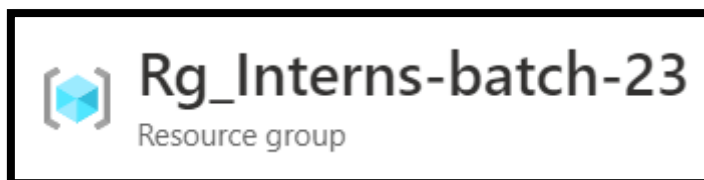


## 4.2 Workflow Diagram:

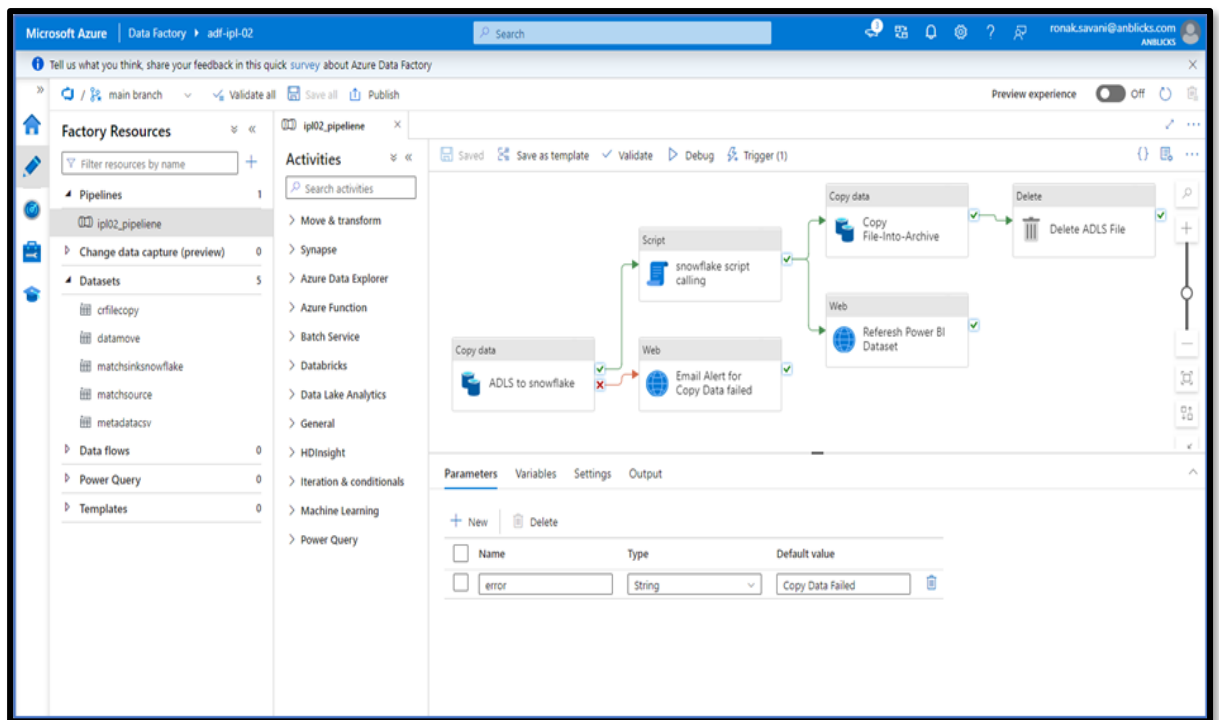


## 5. Implementation:

- We used Rg\_Interns-batch-23 resource group. From here, begin to create all the components within this resource group.



- First of all, we are uploading all the CSVs into the Data Lake with creating a container and then a directory.
- This is our main pipeline which consists of multiple activities and used for orchestrating the data.



- First of all, the raw dataset will be loaded into ADLS Gen2. After loading it, we need to copy into snowflake with the help of CopyData Activity.
- After loading into snowflake, we need to clean the raw data with the help of Script Activity in which we have written stored procedure snowpark script.
- After completion of script activity, we are copying all the raw data and move it to archive container.
- At the same time we are also refreshing the cleaned dataset in Power BI with the help of logic App service and using web activity in ADF pipeline.
- After completion of copy activity, we are deleting all the raw data files from the main container using delete activity.
- In any case over first copydata activity fails then we will get an email alert in which we will get message that your pipeline have failed. All this happens with the help of web activity.

## 6. Snowflake-Snowpark Script:

```
CREATE OR REPLACE PROCEDURE SP_IPL(frm_table string,to_table
string)
```

```
RETURNS STRING
```

```
LANGUAGE PYTHON
```

```
RUNTIME_VERSION = '3.8'
```

```
PACKAGES = ('snowflake-snowpark-python')
```

```
HANDLER = 'run'
```

```
AS
```

```
$$
```

```
from snowflake.snowpark import Session
```

```
from snowflake.snowpark.functions import *
```

```
def run(session,frm_table,to_table):
```

```
    #Get data from raw table and transformed table , without get duplicate
    from raw table
```

```
    ipl_table=session.table("IPL_TABLE") # raw table
```

```
    ipl_table=ipl_table.dropDuplicates(["ID"])
```

```
    clean_table=session.table("clean_table") #Transformed table
```

```
    #select only MatchID column from transformed table for comparing id of
    both table
```

```
    id_of_clean_table=clean_table.select("ID")
```

# Find the data having same MatchID in both the table using ".in\_" function

```
similar_col_df=ipl_table.filter(ipl_table["ID"].in_(id_of_clean_table))
```

#Get unique MatchID column from raw table by removing data we get using ".in\_" function

```
filter_ipl_dataframe=ipl_table.except_(similar_col_df)
```

#DROP RESULT\_MARGIN, POM AND ELIMINATOR

```
filter_ipl_dataframe =  
filter_ipl_dataframe.filter(col("RESULT_MARGIN")!="NA")
```

```
filter_ipl_dataframe =  
filter_ipl_dataframe.filter(col("PLAYER_OF_MATCH") != "NA")
```

```
filter_ipl_dataframe = filter_ipl_dataframe.filter(col("ELIMINATOR") !=  
"NA")
```

#change neutral vanue and result\_margin to int

```
filter_ipl_dataframe=filter_ipl_dataframe.with_column("RESULT_MARGI  
N", col("RESULT_MARGIN").cast("int"))
```

```
filter_ipl_dataframe=filter_ipl_dataframe.with_column("NEUTRAL_VENU  
E", col("NEUTRAL_VENUE").cast("int"))
```

#replace replace 'NA' value with usual method in METHOD

```
filter_ipl_dataframe = filter_ipl_dataframe.withColumn("method",  
replace(col("method"), "NA", "Usual method"))
```



#REPLACE CITY VALUE 'NA TO DUBAI AND SARJAH

```
filter_ipl_dataframe =  
filter_ipl_dataframe.withColumn("CITY",when(col("VENUE") == 'Sharjah  
Cricket Stadium', replace(col("CITY"), "NA",  
"Sharjah")).otherwise(replace(col("CITY"), "NA", "Dubai")))
```

#REPLACE BENGALURU WITH BANGALORE

```
filter_ipl_dataframe = filter_ipl_dataframe.withColumn("CITY",  
replace(col("CITY"), "Bengaluru","Bengalore"))
```

#REPLACE STADIUM NAME WHICH IS DEDICATED TO ONE

```
filter_ipl_dataframe=filter_ipl_dataframe.withColumn("VENUE",  
replace(col("VENUE"), "M Chinnaswamy Stadium", "M.Chinnaswamy  
Stadium"))
```

#REPLACE TEAMS NAME

```
filter_ipl_dataframe=filter_ipl_dataframe.replace(["Rising Pune  
Supergiant","Kings XI Punjab", "Delhi Daredevils"], ["Rising Pune  
Supergiants","Punjab Kings", "Delhi Capitals"])
```

#Write and save table in new session table in snowflake

```
filter_ipl_dataframe.write.mode("append").save_as_table("clean_table")
```

#Create view for neccessary data coulmns

```
session.sql("SELECT ID, CITY, DATE, PLAYER_OF_MATCH,  
VENUE, TEAM1, TEAM2, TOSS_WINNER,TOSS_DECISION,  
WINNER, RESULT, RESULT_MARGIN FROM  
clean_table").createOrReplaceView("transformed_data_view")
```

```

#Delete all the raw data from IPL_TABLE

session.table("IPL_TABLE").delete()

# Predictive Analysis Machine learning


match_df=session.table("clean_table")
match_data=match_df.to_pandas()


#Toss affecting the win dataframe
toss_win_result =
match_data.groupby('TOSS_WINNER').WINNER.value_counts().reset_index(name="count")
    toss_win_result['RESULT']=np.where(toss_win_result.WINNER==toss_win_result.TOSS_WINNER,'won','lost')
    toss_win_result_df =
toss_win_result.groupby(['TOSS_WINNER','RESULT'])['count'].sum().reset_index()


plot = sns.barplot(x="TOSS_WINNER", y="count", hue="RESULT",
data=toss_win_result_df)
plot.set_title('Matches won/lost by teams winning toss ')
plot.set_xticklabels(toss_win_result_df['TOSS_WINNER'].unique(),rotation=90)
plt.show()


#Winning stats of teams bat/field first by venues
venue_toss_decision_result=match_data[["TOSS_WINNER","TOSS_DECISION","WINNER","VENUE"]]
    venue_toss_decision_result["TOSS_DECISION"]=np.where((venue_toss_decision_result.TOSS_WINNER ==
venue_toss_decision_result.WINNER)(venue_toss_decision_result.TOSS_D

```

```

ECISION=="field"),"field_won","bat_won")
venue_result=venue_toss_decision_result.groupby(["VENUE"]).TOSS_D
ECISION.value_counts().reset_index(name="count")

```

```

sns.set(rc={'figure.figsize':(11.7,8.27)})
plot = sns.barplot(x="VENUE", y="count", hue="TOSS_DECISION",
data=venue_result)
plot.set_title('Teams bat/field first results on venues')
plot.set_xticklabels(venue_result['VENUE'].unique(),rotation=90)
plt.show()

```

```

#Get the percentage of wins for each team in all the venues
for i in match_data["WINNER"].unique(): # Get the each team from
winner column
    print("WINNER: ", i)
    plt.figure(figsize=(6,10)) # Adjusting the figure size
    plt.pie(match_data[match_data["WINNER"] ==
i]["VENUE"].value_counts()[:10], autopct = "%1.1f%% ")
    plt.legend(match_data[match_data["WINNER"] ==
i]["VENUE"].value_counts()[:10].index, bbox_to_anchor = (2,1), loc =
"upper right")
    plt.show()

```

```

#Team win when they play at there homwtown
homeground={ }
for i in match_data.WINNER.unique():
    homeground[i]=match_data[match_data.WINNER==i].VENUE.value_
counts().head(1).index[0]

```

```

matches_played_homeground={ }
for i in homeground:
    matches_played_homeground[i]=len(match_data[match_data.VENUE=
=homeground[i]])

```

```

# Gives the number of matches won by each team on their homeground
matches_won_homeground={}
for i in homeground:
    matches_won_homeground[i]=len(match_data.loc[(match_data['VENU
E']==homeground[i]) & (match_data['WINNER']==i)])

#Success percentage of each team on their homeground
success_percentage={}
for i in homeground:
    success_percentage[i]=(matches_won_homeground[i]/matches_played_
homeground [i])*100
    success_percentage

teams = list(success_percentage.keys())
success_percentage = list(success_percentage.values())

# plotting barplot of Success_percentage with above values
sns.barplot(teams,success_percentage)
plt.xticks(rotation=90)
plt.xlabel("Teams", fontsize = 14)
plt.ylabel("Success Percentage", fontsize = 14)
plt.title("Success percentage of each team at thier home grounds", fontsize
= 15)
plt.show()

return "SUCCESS"

$$;

```

```

CALL IPL_SCHEMA.SP_IPL('IPL_TABLE','clean_table');

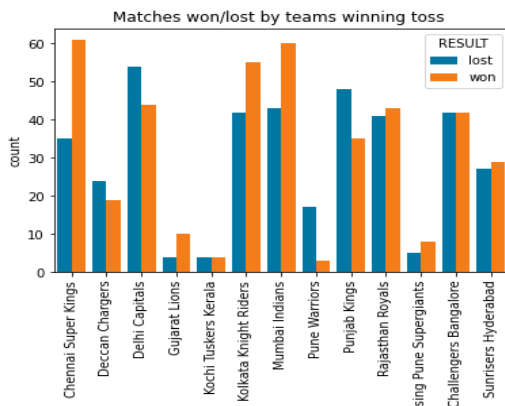
```

## • Predictive Analysis

```
match_data=pd.read_csv("/content/clean_table.csv")
```

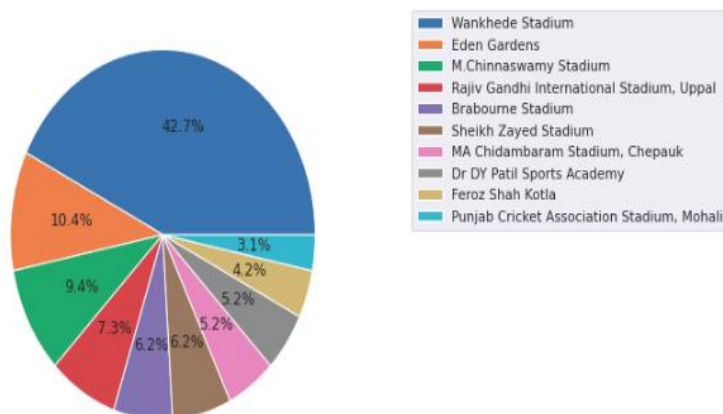
```
[14] #Toss affecting the win dataframe
toss_win_result = match_data.groupby('TOSS_WINNER').WINNER.value_counts().reset_index(name="count")
toss_win_result['RESULT']=np.where(toss_win_result.WINNER==toss_win_result.TOSS_WINNER, 'won', 'lost')
toss_win_result_df = toss_win_result.groupby(['TOSS_WINNER', 'RESULT'])['count'].sum().reset_index()
```

```
[ ] plot = sns.barplot(x="TOSS_WINNER", y="count", hue="RESULT", data=toss_win_result_df)
plot.set_title('Matches won/lost by teams winning toss ')
plot.set_xticklabels(toss_win_result_df['TOSS_WINNER'].unique(),rotation=90)
plt.show()
```



```
#Get the percentage of wins for each team in all the venues
for i in match_data["WINNER"].unique(): # Get the each team from winner column
    print("WINNER: ", i)
    plt.figure(figsize=(6,10)) # Adjusting the figure size
    plt.pie(match_data[match_data["WINNER"] == i]["VENUE"].value_counts()[:10], autopct = "%1.1f%%")
    plt.legend(match_data[match_data["WINNER"] == i]["VENUE"].value_counts()[:10].index, bbox_to_anchor = (2,1), loc = "upper right")
    plt.show()
```

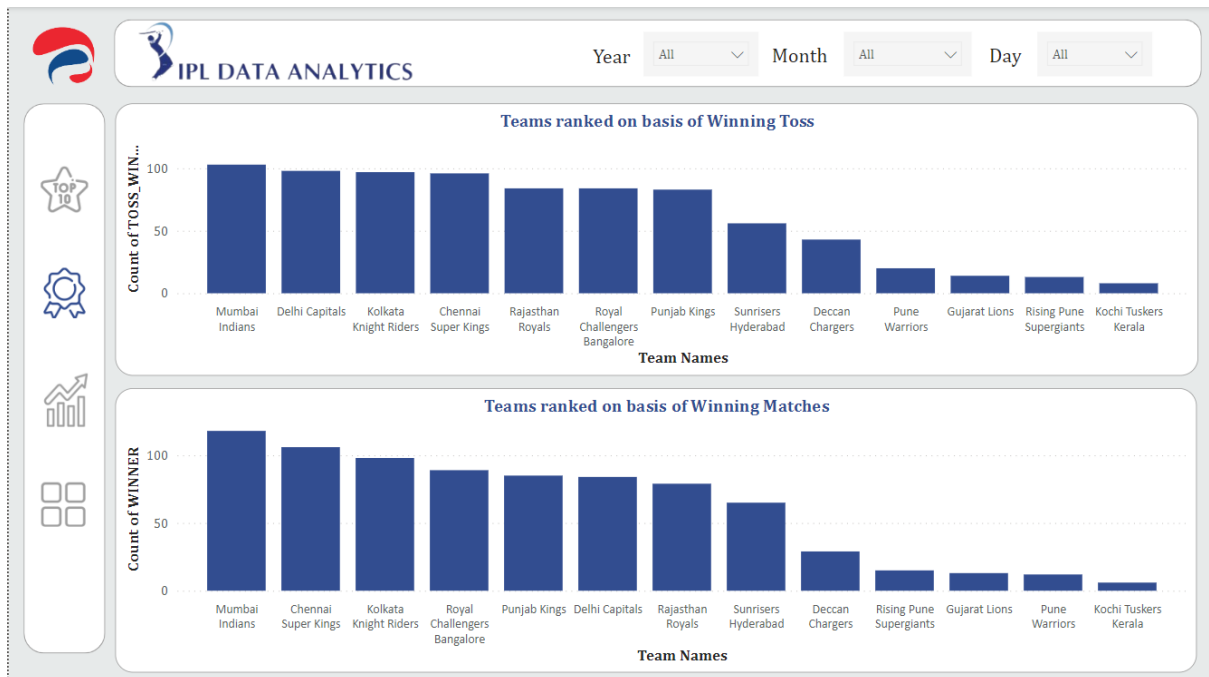
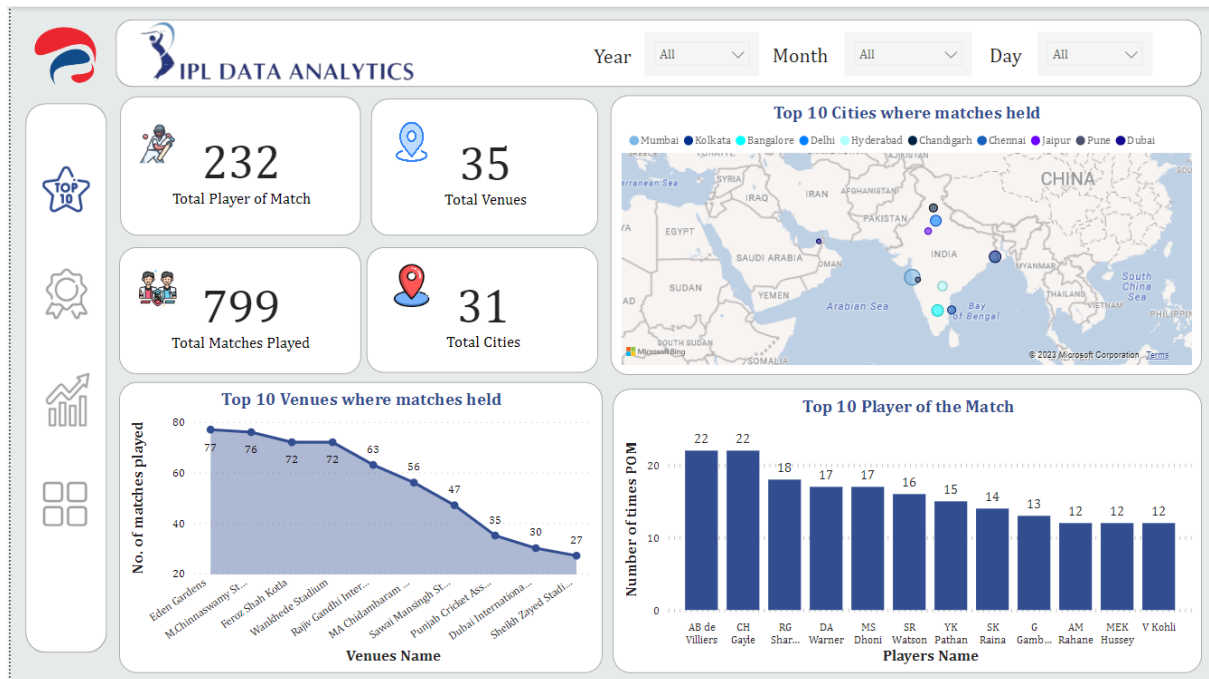
WINNER: Mumbai Indians

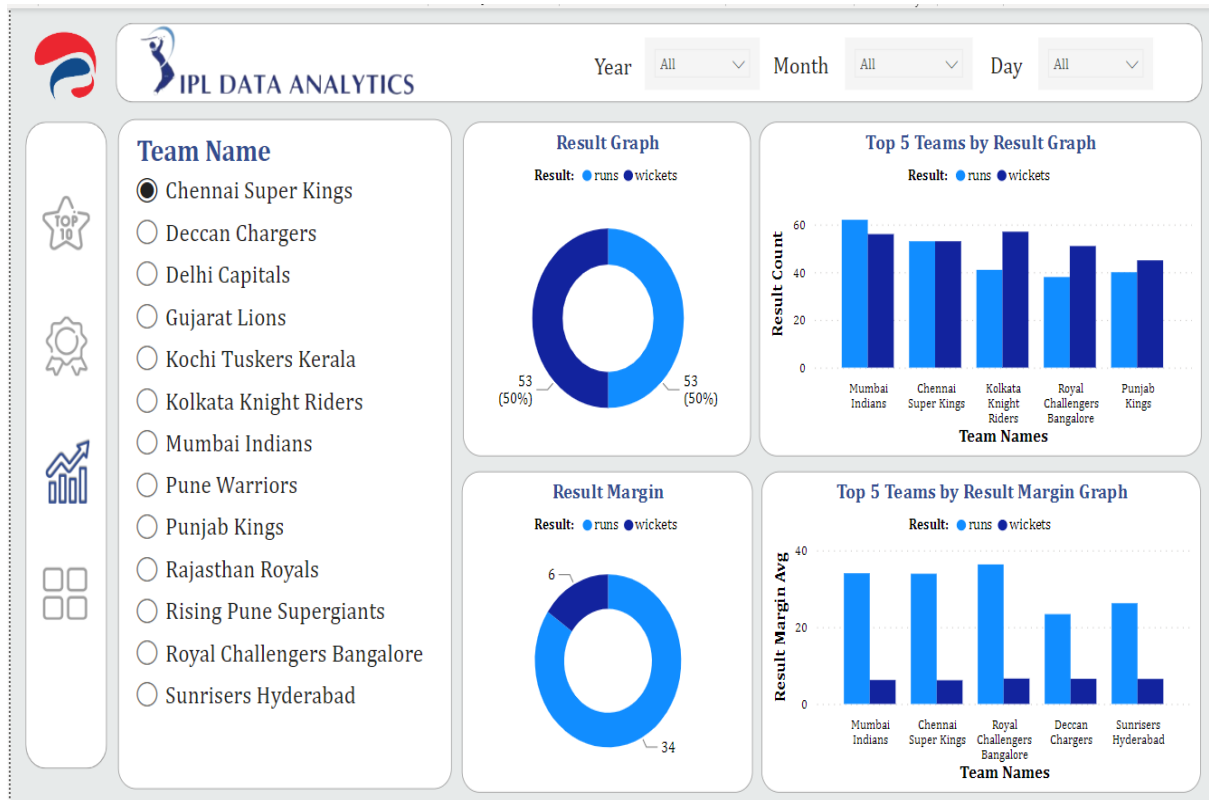


# 7. Output

We can see the Data Insight on Power BI.

## 1.1 Home Page





Result Graph

Result: 

runs

wickets



Top 5 Teams by Result Graph

Result: 

runs

wickets



Team Names	runs	wickets
Mumbai Indians	60	55
Chennai Super Kings	55	55
Kolkata Knight Riders	40	58
Royal Challengers Bangalore	38	52
Punjab Kings	40	45

Result Margin

Result: 

runs

wickets



Top 5 Teams by Result Margin Graph

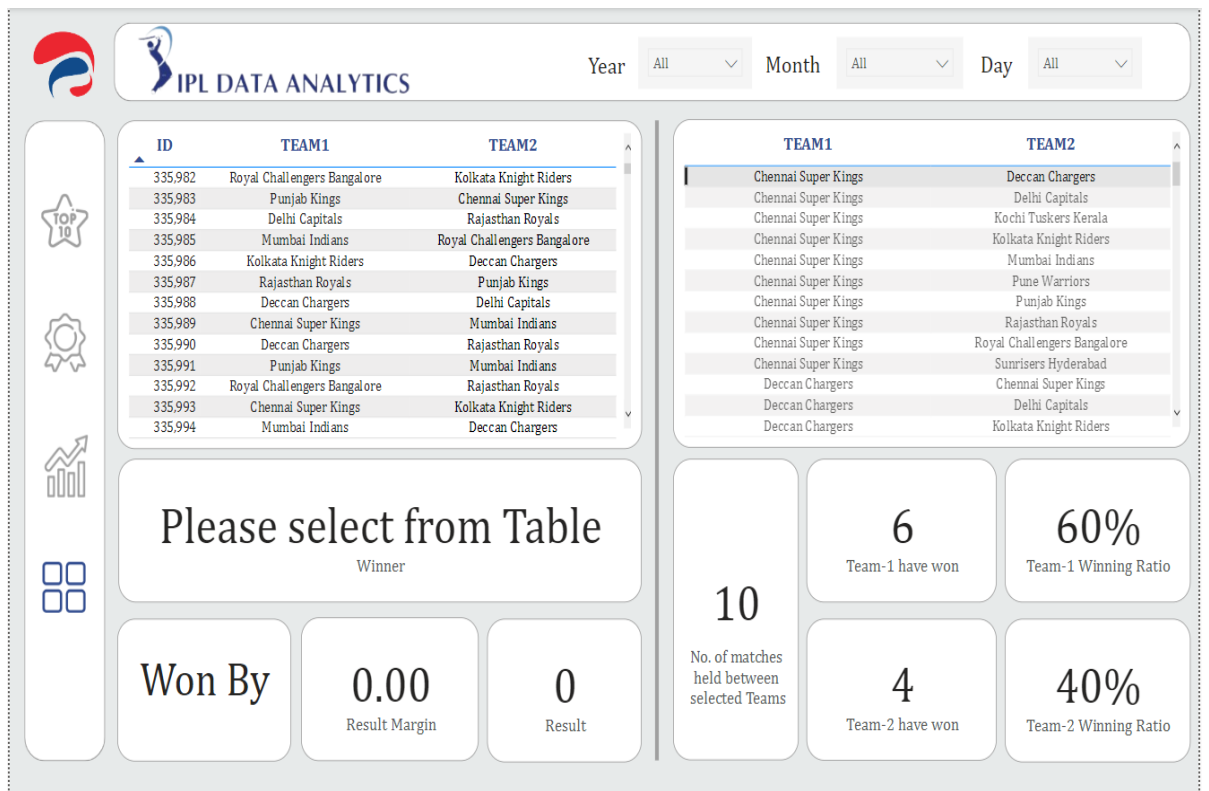
Result: 

runs

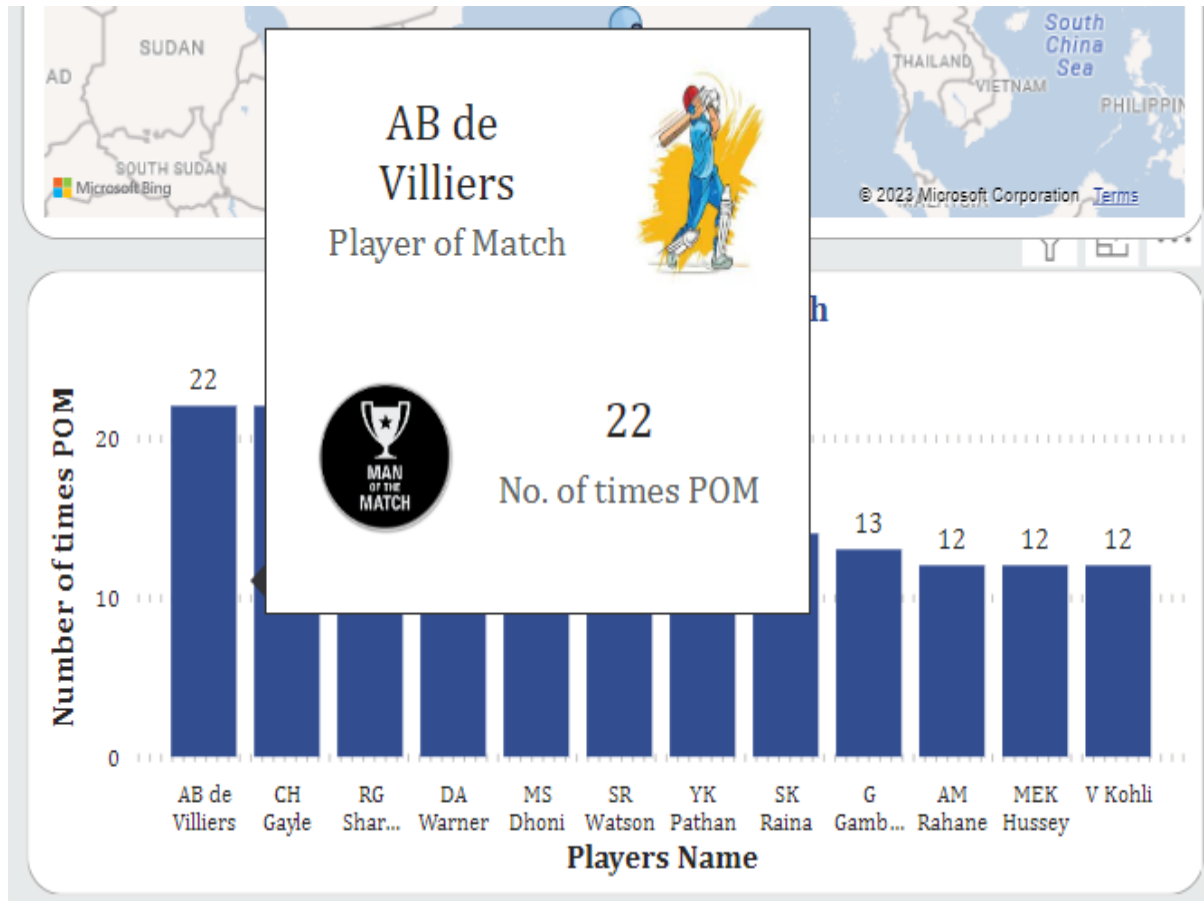
wickets



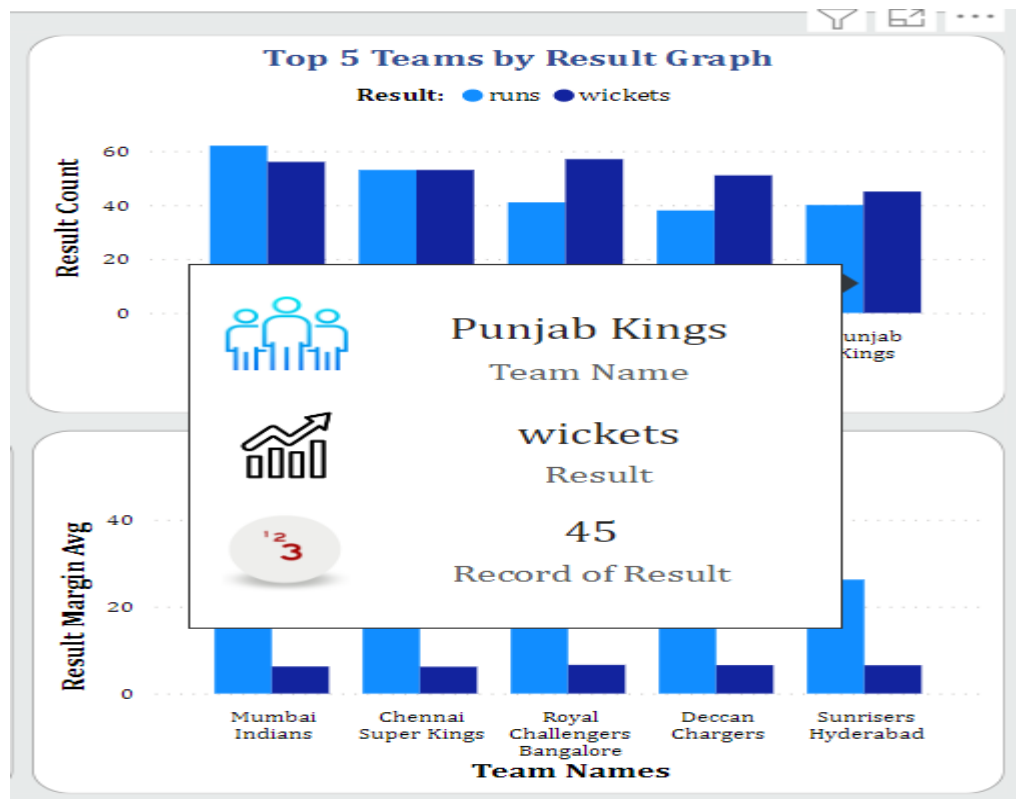
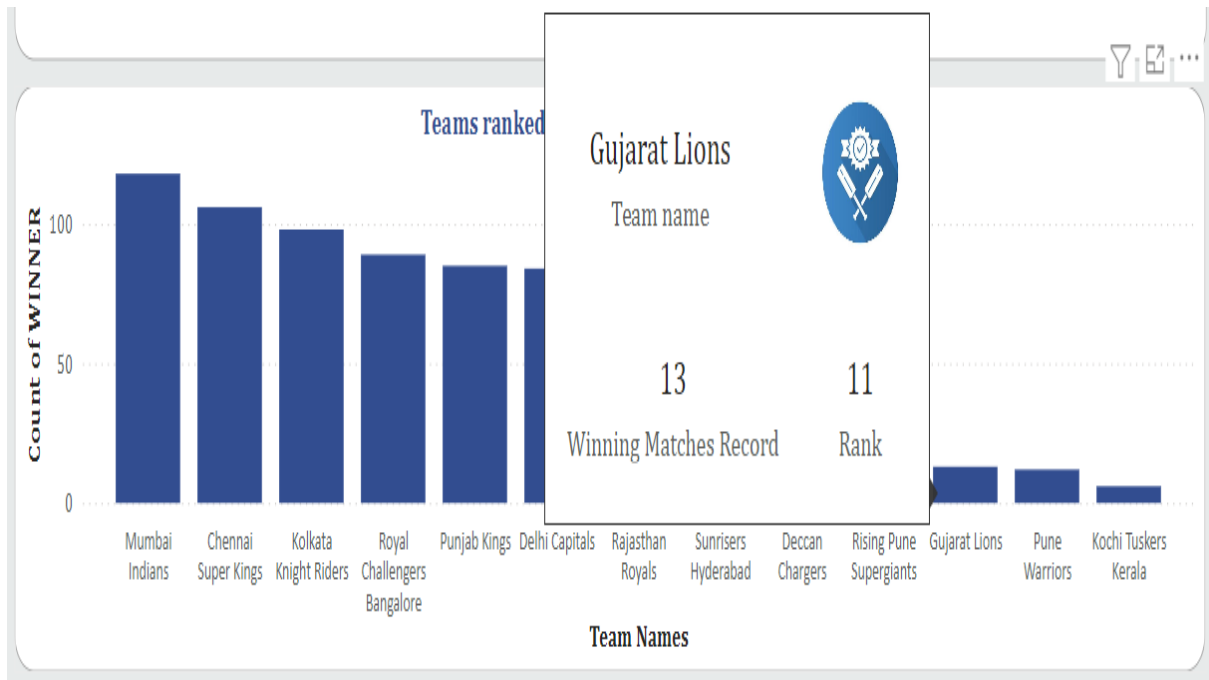
Team Names	runs	wickets
Mumbai Indians	35	5
Chennai Super Kings	35	5
Royal Challengers Bangalore	38	5
Deccan Chargers	25	5
Sunrisers Hyderabad	28	5



## 8. Observation





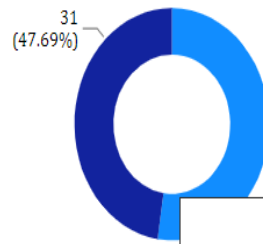


## Team Name

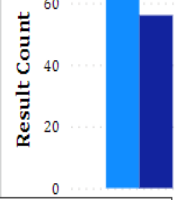
- ☐ Chennai Super Kings
- ☐ Deccan Chargers
- ☐ Delhi Capitals
- ☐ Gujarat Lions
- ☐ Kochi Tuskers Kerala
- ☐ Kolkata Knight Riders
- ☐ Mumbai Indians
- ☐ Pune Warriors
- ☐ Punjab Kings
- ☐ Rajasthan Royals
- ☐ Rising Pune Supergiants
- ☐ Royal Challengers Bangalore
- ☒ Sunrisers Hyderabad

## Result Graph

Result: ● runs ● wickets



## Top !



## Sunrisers Hyderabad

Team Name

wickets 7

Result Result Margin Average

## Result M

Result: ● runs ● wickets

