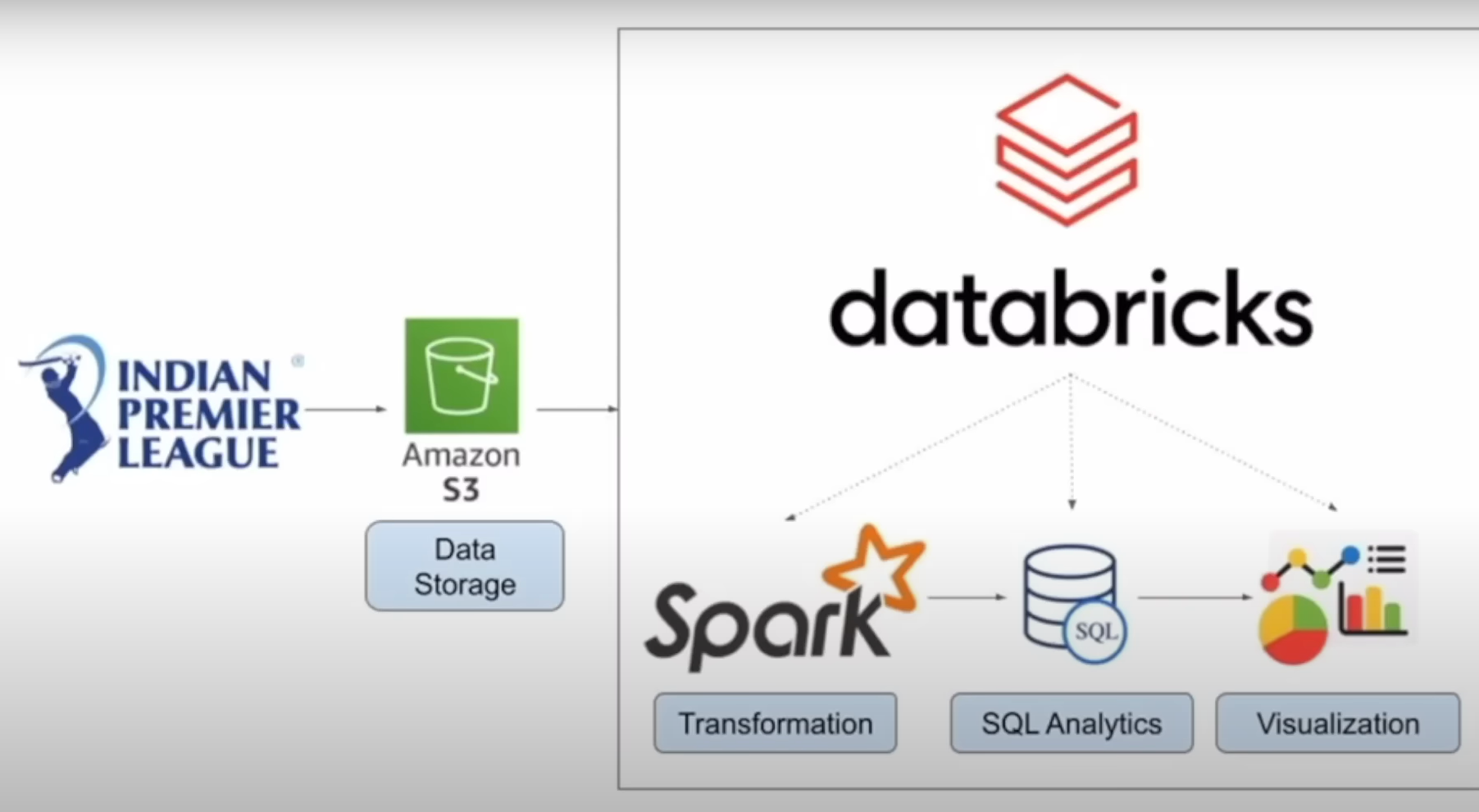
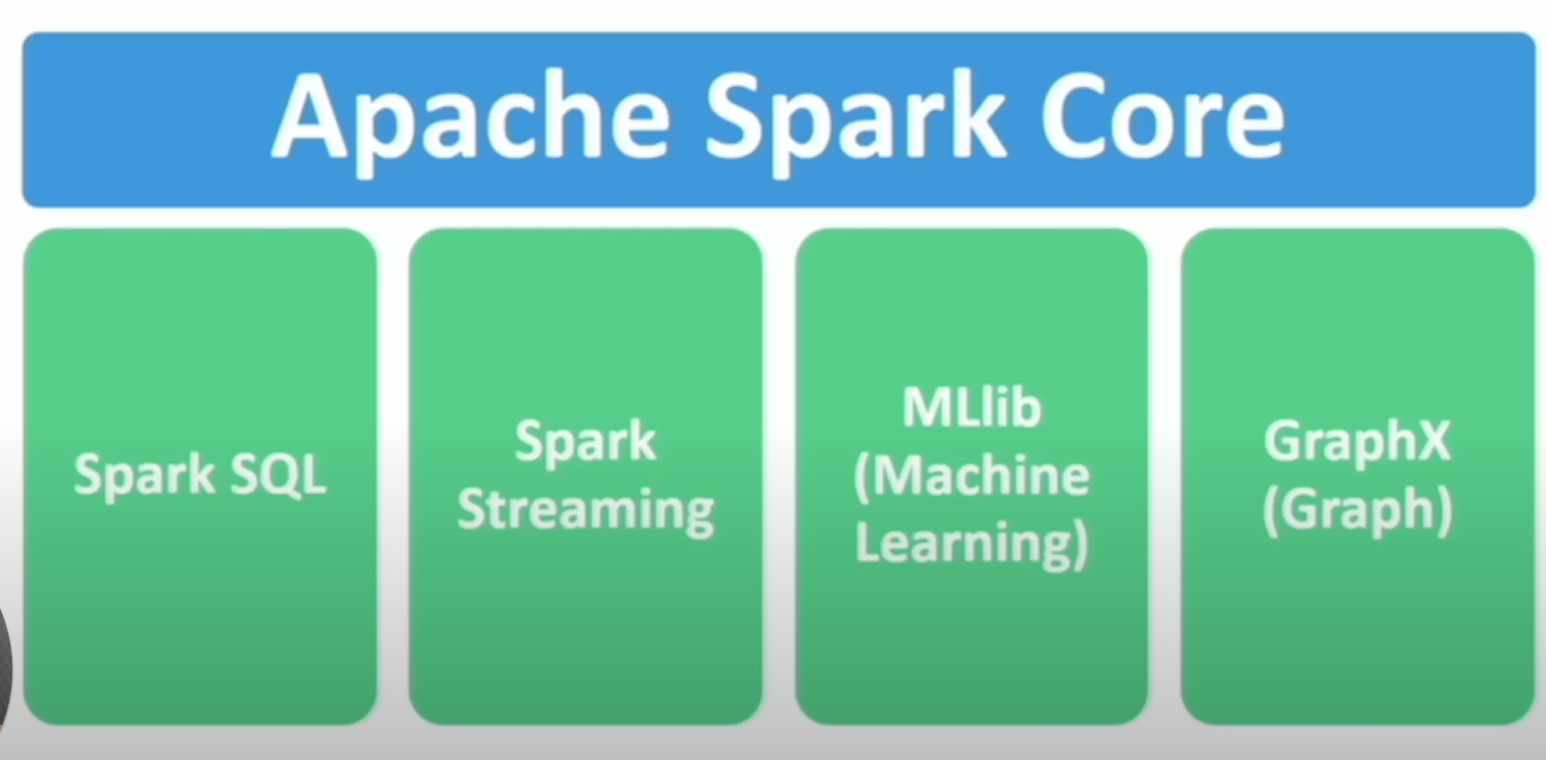
**IPL Data Analysis | Apache Spark End-To-End Data Engineering Project**



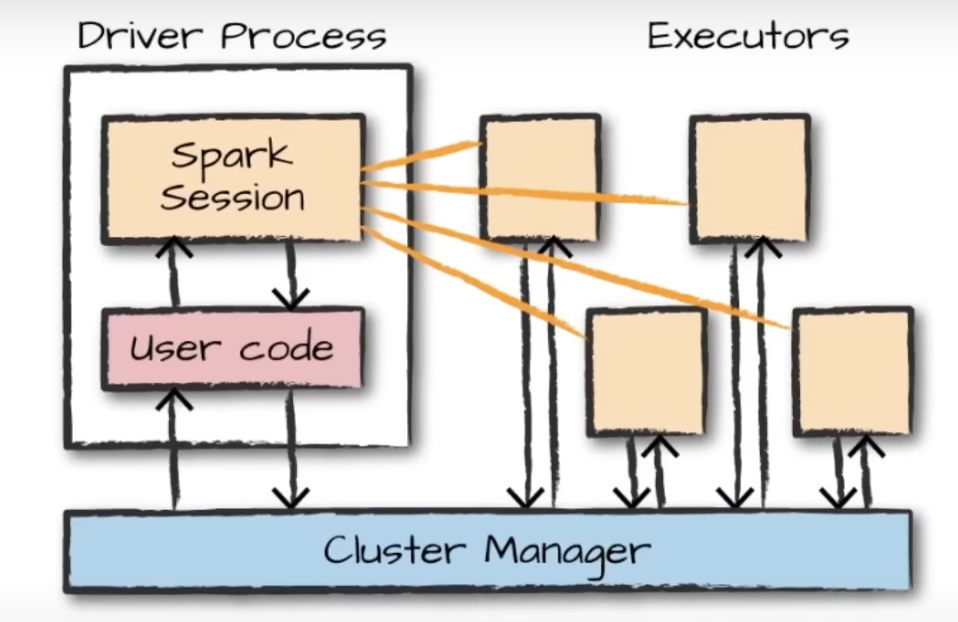
Here we will be using IPL dataset.

* We will be using IPL dataset.
* We will be uploading this data on Amazon S3 (Simple Storage Service).
* We will use DataBrick for to create it.
* We will use Apache Spark for Transformation and to build our logic.
* We will use SQL to analyze our data.

This will majorly focus on **“How Spark works and to write code in it”**.



Apache Spark Core is the Heart of the Spark which is responsible to execute code you submit to spark.



DataBricks is basically a software which supports Spark Environment. If you want to configure Apache in your local system you need to get JVM, packages, Env path. This is kind of On-premisconfiguration.

So, DataBricks says do worry about anything(infrastructure) we will do everything for you, just focus on coding and business.

Upload the file into DataBrick:

Local system location - C:\Users\negig\Downloads\raghu543-ipl-data-till-2017

Databrick location -dbfs:/FileStore/sample\_data/SPARK/raghu543-ipl-data-till-2017/Ball\_By\_Ball.csv

Step 1: Starting Spark session:

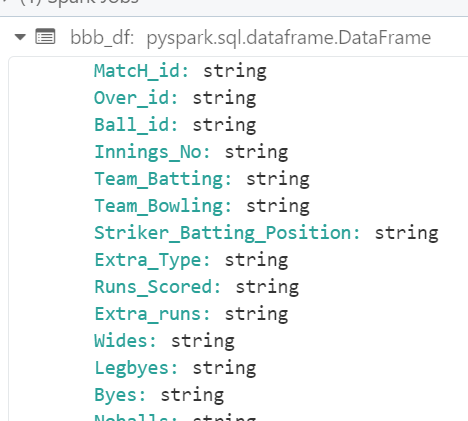
#But let's follow standard practice

from pyspark.sql import SparkSession

spark= SparkSession.builder.appName('IPL Data Analysis').getOrCreate()

Step 2: Readingfile to create DataFrame

But if you see the data type all the Boolean values column is having string datatype



bbb\_df=spark.read.format('csv').option('header','true').load('dbfs:/FileStore/sample\_data/SPARK/raghu543-ipl-data-till-2017/Ball\_By\_Ball.csv')

It is best practice to create your own schema to fill all the data with proper datatype.

Step 3: Importing other functions

To create new schema we need to import some other functions

from pyspark.sql.types import StructField, StructType,StringType, IntegerType, BooleanType, DateType,DecimalType

Step 4: Creating StructField with the help of chat GPT

Copy pastes schema of the file from below link:

<https://data.world/raghu543/ipl-data-till-2017/workspace/data-dictionary>

Give it to chat GPT to create StructField and StructType

Step 5: Unsing Schem object to give it in configuration while creating DF

bbb\_df=spark.read.schema(ball\_by\_ball\_schema).format('csv').option('header','true').load('dbfs:/FileStore/sample\_data/SPARK/raghu543-ipl-data-till-2017/Ball\_By\_Ball.csv')

Step 6: Similarly create DF for all the remaining CSV files.

So, with the help of chat GPT we have created DF for all the 5 CSV files.

**Working with ball\_by\_ball\_df (bbb\_df)**

Step 7: Transformation

This is the final logic you apply on the dataset.

* It can be removing null values.
* It can be adding 2 columns to create 3rd column.
* Filtering data to filter out all the data that you do not need and only pass the values which is required.

# Filter to include only valid deliveries (Excluding extra like wides and no balls for specific analyses)

from pyspark.sql.functions import col, when , sum, avg, row\_number

bbb\_df=bbb\_df.filter((col('wides')==0) & (col('noballs')==0))

# Aggregations: Calculate the total and average run scored in each match

total\_and\_average\_runs=bbb\_df.groupBy('match\_id','innings\_no').agg(

    sum('runs\_scored').alias('Total\_runs'),

    avg('runs\_scored').alias('Average\_runs')

)

Step 8: Writing Windows function:

from pyspark.sql.window import Window

# Window Functions: Calculating running total of runs in each match for each over

windowSpec= Window.partitionBy('match\_id','innings\_no').orderBy('over\_id')

#Adding that column in the Df

bbb\_df=bbb\_df.withColumn(

    'running\_total\_runs',

    sum('runs\_scored').over(windowSpec)

)

bbb\_df.show(3)

Step 9: Conditional Formating

# Conditional Columns : Flag for high impact balls (either a wicket or more than 6 runs including extras)

bbb\_df=bbb\_df.withColumn(

    'High\_impact',

    when((col('runs\_scored')+col('extra\_runs')>6) | (col('bowler\_wicket')==True),True).otherwise(False)

)

**Working with match\_df**

Step 10: Doing transformation and generating new columns

from pyspark.sql.functions import year,month, dayofmonth, when

# Extracting year,month, day from the match date for more details time-based analysis

match\_df=match\_df.withColumn('year',year('match\_date'))

match\_df=match\_df.withColumnb('month',month('match\_date'))

match\_df=match\_df.withColumn('day',dayofmonth('match\_date'))

# high margin win: Categorizing win margin into 'high', 'medium' and 'low'

match\_df=match\_df.withColumn(

    'win\_margin\_category',

    when(col('win\_margin')>=100,'High')

    .when((col('win\_margin')<100) & (col('win\_margin')>=50),'Medium')

    .otherwise('Low')

)

#Analyze the impact of toss: who wins the toss and match

match\_df=match\_df.withColumn(

    'toss\_match\_winner',

    when(col('toss\_winner')==col('match\_winner'),'Yes')

    .otherwise('No')

)

#Show the enhanced DF

match\_df.show()

**Working on player\_df**

Step 11: Doing transformation and generating new columns

from pyspark.sql.functions import lower, regexp\_replace

# Normalize and clean player names

player\_df = player\_df.withColumn("player\_name", lower(regexp\_replace("player\_name", "[^a-zA-Z0-9 ]", "")))

# Handle missing values in 'batting\_hand' and 'bowling\_skill' with a default 'unknown'

player\_df = player\_df.na.fill({"batting\_hand": "unknown", "bowling\_skill": "unknown"})

# Categorizing players based on batting hand

player\_df = player\_df.withColumn(

    "batting\_style",

    when(col("batting\_hand").contains("left"), "Left-Handed").otherwise("Right-Handed")

)

# Show the modified player DataFrame

player\_df.show(2)

**Working on player\_match**

Step 12:

from pyspark.sql.functions import col, when, current\_date, expr

# Add a 'veteran\_status' column based on player age

player\_match\_df = player\_match\_df.withColumn(

    "veteran\_status",

    when(col("age\_as\_on\_match") >= 35, "Veteran").otherwise("Non-Veteran")

)

# Dynamic column to calculate years since debut

player\_match\_df = player\_match\_df.withColumn(

    "years\_since\_debut",

    (year(current\_date()) - col("season\_year"))

)

# Show the enriched DataFrame

player\_match\_df.show()

So far we have wrote basic transformation on the dataframe

Step 13: We will try to do some analysis using SQL queries. But unfortunately we can not perform SQL queries on DataFrame so we need to create Views with the help of DF.

Syntax- <df>.createOrReplaceGlobalTempView(‘<View name>’)

<https://community.cloud.databricks.com/?o=3393600450228831#notebook/937684163359520/command/1954453707280103>