

Cognizant Technology Solutions



A Project Report on

“FORMULA 1 PROJECT USING PYSPARK”

Prepared By

Ankita Shivane	2320390
Harshit Yadav	2320382
Mimansa Yadav	2320956
Nandhini T	2320539
Rajanya Kamilya	2320828
Shambhavi Shukla	2320529

INTRODUCTION

FORMULA RACING

Formula racing, exemplified by Formula 1 (F1), stands as the pinnacle of motorsport, blending cutting-edge technology with unparalleled athleticism and strategy. Originating in the post-war era, Formula racing has evolved into a global phenomenon, captivating audiences with its fusion of speed, precision engineering, and human skill.

At its heart, Formula racing features a series of high-profile races held on circuits around the world, with the Formula 1 World Championship being the most prestigious. This championship comprises a calendar of races held across diverse tracks, from iconic street circuits like Monaco to purpose-built facilities like Silverstone and Suzuka.

Behind the scenes, Formula racing is a relentless pursuit of perfection for constructors, who design and build the cars that compete on the track. Teams like Mercedes, Ferrari, and Red Bull invest heavily in research, development, and innovation, striving to create the fastest and most reliable machines possible.

Formula racing is also a stage for remarkable achievements, with records continually being shattered and history being made. From historic race wins to championship triumphs, the sport is a tapestry of unforgettable moments that etch themselves into the annals of motorsport history.

In essence, Formula racing is a captivating blend of technology, athleticism, and strategy, where races, drivers, constructors, tracks, and achievements converge to create an electrifying spectacle that captivates audiences around the globe.

PYSPARK

PySpark is a Python API to support Python with Apache Spark. PySpark provides Py4j library, with the help of this library, Python can be easily integrated with Apache Spark. PySpark plays an essential role when it needs to work with a vast dataset or analyze them. This feature of PySpark makes it a very demanding tool among data engineers.



A large amount of data is generated offline and online. It is necessary to extract valuable information from the raw data. We require a more efficient tool to perform different types of operations on the big data. There are various tools to perform the multiple tasks on the huge dataset, but these tools are not so appealing anymore. It is needed some scalable and flexible tools to crack big data and gain benefit from it.

BUSINESS REQUIREMENTS

Description:

Analyze and process Formula 1 data files to extract insights about dominant drivers, teams and their performances.

REQUIREMENTS:

Data Ingestion Requirements:

- Ingest the data from various source files
- Ingested data must have the schema applied
- Ingested data must have audit columns
- Ingested data must be stored in a columnar format

Data Transformation Requirements:

- Join the key information required for reporting to create a new table.
- Join the key information required for Analysis to create a new table.
- Transformed tables must have audit columns
- Must be able to analyze the transformed data via SQL
- Transformed data must be stored in columnar format (i.e. Parquet)

Reporting Requirements:

- Driver Standings
- Constructor Standings

Analysis Requirements:

- Dominant Drivers
- Dominant Teams
- Visualize the Outputs

F1 DATA SET

circuits table

Field	Type	Null	Key
circuitId	int(11)	NO	PRI
circuitRef	varchar(255)	NO	
name	varchar(255)	NO	
location	varchar(255)	YES	
country	varchar(255)	YES	
lat	float	YES	
lng	float	YES	
alt	int(11)	YES	
url	varchar(255)	NO	UNI

drivers table

Field	Type	Null	Key
driverId	int(11)	NO	PRI
driverRef	varchar(255)	NO	
number	int(11)	YES	
code	varchar(3)	YES	
forename	varchar(255)	NO	
surname	varchar(255)	NO	
dob	date	YES	
nationality	varchar(255)	YES	
url	varchar(255)	NO	UNI

constructors table

Field	Type	Null	Key
constructorId	int(11)	NO	PRI
constructorRef	varchar(255)	NO	
name	varchar(255)	NO	UNI
nationality	varchar(255)	YES	
url	varchar(255)	NO	

races table

Field	Type	Null	Key
raceId	int(11)	NO	PRI
year	int(11)	NO	
round	int(11)	NO	
circuitId	int(11)	NO	
name	varchar(255)	NO	UNI
date	date	NO	
time	time	YES	
url	varchar(255)	YES	
fp1_date	date	YES	
fp1_time	time	YES	
fp2_date	date	YES	
fp2_time	time	YES	
fp3_date	date	YES	
fp3_time	time	YES	
quali_date	date	YES	
quali_time	time	YES	
sprint_date	date	YES	
sprint_time	time	YES	

lap_times table

Field	Type	Null	Key
raceId	int(11)	NO	FK
driverId	int(11)	NO	FK
lap	int(11)	NO	PRI
position	int(11)	YES	
time	varchar(255)	YES	
milliseconds	int(11)	YES	

pit_stops table

Field	Type	Null	Key
raceId	int(11)	NO	FK
driverId	int(11)	NO	FK
stop	int(11)	NO	PRI
lap	int(11)	NO	
time	time	NO	
duration	varchar(255)	YES	
milliseconds	int(11)	YES	

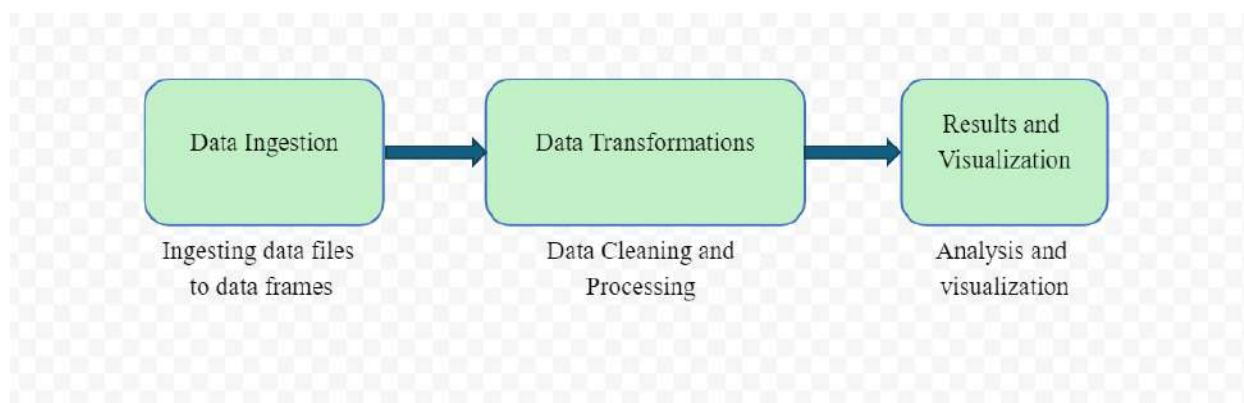
qualifying table

Field	Type	Null	Key
qualifyId	int(11)	NO	PRI
raceId	int(11)	NO	
driverId	int(11)	NO	
constructorId	int(11)	NO	
number	int(11)	NO	
position	int(11)	YES	
q1	varchar(255)	YES	
q2	varchar(255)	YES	
q3	varchar(255)	YES	

sprint_results table

Field	Type	Null	Key
sprintResultId	int(11)	NO	PRI
raceId	int(11)	NO	
driverId	int(11)	NO	
constructorId	int(11)	NO	
number	int(11)	YES	
grid	int(11)	NO	
position	int(11)	YES	
positionText	varchar(255)	NO	
positionOrder	int(11)	NO	
points	float	NO	
laps	int(11)	NO	
time	varchar(255)	YES	
milliseconds	int(11)	YES	
fastestLap	int(11)	YES	
fastestLapTime	varchar(255)	YES	
statusId	int(11)	NO	

PICTORIAL FLOWCHART:



IMPLEMENTATION:

- **REQUIREMENT 1:**

DATA INGESTION

Data ingestion is the process of importing and loading data into a system. It's a critical step in any data-centric workflow, ensuring that the correct information is available at the right time for analysis and decision-making.

To ingest a dataset into Databricks, you can follow these steps:

1. Create a Compute engine:

- Start by creating a Databricks cluster.
- This cluster will provide the compute resources needed to run your commands.

2. Enable DBFS in Databricks:

- Use Databricks features to explore your raw dataset.

3. Ingest the Raw Data:

- Load the raw data into a table to make it available for further processing.

Here, in this project we have 8 datafiles that we need to ingest into DBFS system.

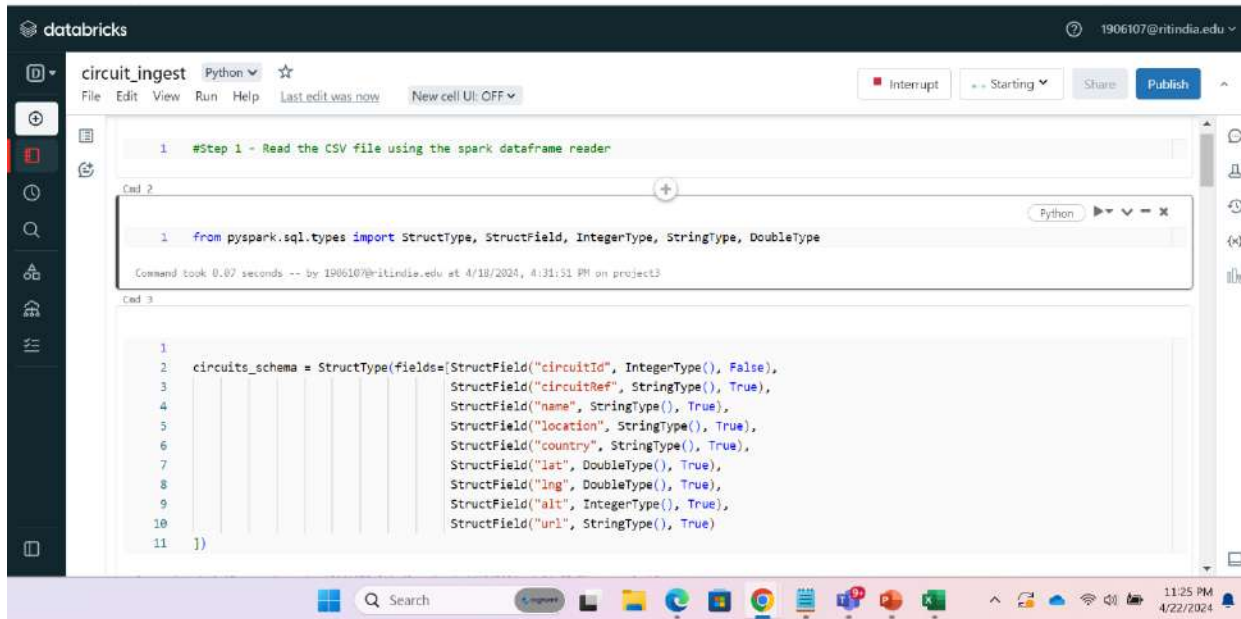
1. Circuit file ingestion:

Step 1: Open Databricks community and login to your account.

Create a compute engine for analysis.

Create new notebook in workspace and write the above syntax

Step 2: Create the schema for Circuit file using StructType



The screenshot shows a Databricks notebook titled 'circuit_ingest' with a Python environment. The notebook contains two code cells. Cell 2 defines the schema for the circuit file using StructType and StructField. Cell 3 defines the schema for the circuit file using StructType and StructField.

```

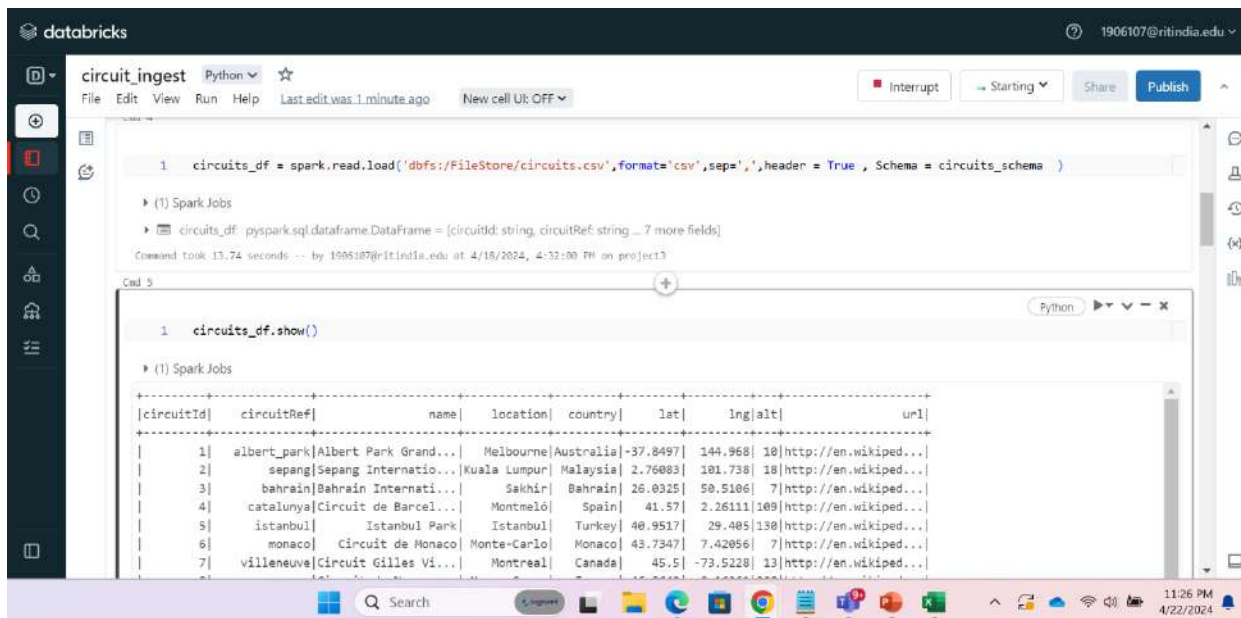
1 #Step 1 - Read the CSV file using the spark dataframe reader

Cell 2
1 from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType

Command took 0.07 seconds -- by 1906107@nitindia.edu at 4/18/2024, 4:31:51 PM on project13

Cell 3
1
2 circuits_schema = StructType(fields=[StructField("circuitId", IntegerType(), False),
3     StructField("circuitRef", StringType(), True),
4     StructField("name", StringType(), True),
5     StructField("location", StringType(), True),
6     StructField("country", StringType(), True),
7     StructField("lat", DoubleType(), True),
8     StructField("lng", DoubleType(), True),
9     StructField("alt", IntegerType(), True),
10    StructField("url", StringType(), True)
11 ])
  
```

Step 3: We have created a dataframe by reading the data from circuit csv file loaded in DBFS system



The screenshot shows a Databricks notebook titled 'circuit_ingest' with a Python environment. The notebook contains two code cells. Cell 1 reads the CSV file from DBFS and creates a dataframe. Cell 2 shows the dataframe content.

```

1 circuits_df = spark.read.load('dbfs:/FileStore/circuits.csv',format='csv',sep=',',header = True , Schema = circuits_schema )

(1) Spark Jobs
↳ circuits_df: pyspark.sql.dataframe.DataFrame = [circuitId: string, circuitRef: string ... 7 more fields]
Command took 13.74 seconds -- by 1906107@nitindia.edu at 4/18/2024, 4:32:00 PM on project13

Cell 2
1 circuits_df.show()

(1) Spark Jobs
↳ (1) Spark Jobs
|circuitId| circuitRef|      name| location| country| lat| lng|alt| url|
|-----|-----|-----|-----|-----|----|----|----|----|
|1| albert_park|Albert Park Grand...| Melbourne|Australia|37.8497| 144.968| 18|http://en.wikiped...
|2| sepang|Sepang Internatio...| Kuala Lumpur| Malaysia| 2.76083| 101.738| 18|http://en.wikiped...
|3| bahrain|Bahrain Internati...| Sakhir| Bahrain| 26.0325| 50.5106| 7|http://en.wikiped...
|4| catalunya|Circuit de Barcel...| Montmeló| Spain| 41.57| 2.26111|109|http://en.wikiped...
|5| istanbul| Istanbul Park| Istanbul| Turkey| 40.9517| 29.4051|138|http://en.wikiped...
|6| monaco| Circuit de Monaco| Monte-Carlo| Monaco| 43.7347| 7.42056| 7|http://en.wikiped...
|7| villeneuve|Circuit Gilles Vi...| Montreal| Canada| 45.5| -73.5228| 13|http://en.wikiped...
  
```

Step 4: Selecting only the required columns and creating a new dataframe.

```

1 #Step 2 - Select only the required columns

1 from pyspark.sql.functions import col

1 circuits_selected_df = circuits_df.select(col("circuitId"), col("circuitRef"), col("name"), col("location"), col("country"), col("lat"), col("lng"), col("alt"))
2

```

Command took 0.12 seconds -- by 1906107@ritindia.edu at 4/18/2024, 4:32:40 PM on project3

Step 5: Renaming the columns using withColumnRenamed command to match the naming standards.

```

1 #Step 3 - Rename the columns as required

1 circuits_renamed_df = circuits_selected_df.withColumnRenamed("circuitId", "circuit_id") \
2 .withColumnRenamed("circuitRef", "circuit_ref") \
3 .withColumnRenamed("lat", "latitude") \
4 .withColumnRenamed("lng", "longitude") \
5 .withColumnRenamed("alt", "altitude")
6

```

Command took 0.22 seconds -- by 1906107@ritindia.edu at 4/18/2024, 4:32:53 PM on project3

Step 6: Adding the ingestion_date column with current_timestamp and displaying the dataframe

The screenshot shows a Databricks notebook titled 'circuit_ingest' with a Python environment. The notebook is at cell 12, and the user is adding an ingestion date to the DataFrame. The code in cell 12 is:

```
1 #Step 4 - Add ingestion date to the dataframe
```

The command took 0.09 seconds. In cell 13, the user imports the `current_timestamp` function from `pyspark.sql.functions`:

```
1 from pyspark.sql.functions import current_timestamp
```

The command took 0.06 seconds. In cell 14, the user adds the `ingestion_date` column to the `circuits_renamed_df` DataFrame:

```
1 circuits_final_df = circuits_renamed_df.withColumn("ingestion_date", current_timestamp())
```

The command took 0.18 seconds. In cell 15, the user displays the final DataFrame:

```
1 circuits_final_df.show(truncate=False)
```

The command took 0.18 seconds. The output shows the DataFrame with columns: `circuit_id`, `circuit_ref`, `name`, `location`, `country`, `latitude`, `longitude`, `altitude`, and `ingestion_date`.

The screenshot shows the final output of the DataFrame `circuits_final_df` displayed in a table format. The table has 9 columns: `circuit_id`, `circuit_ref`, `name`, `location`, `country`, `latitude`, `longitude`, `altitude`, and `ingestion_date`. The data is as follows:

circuit_id	circuit_ref	name	location	country	latitude	longitude	altitude	ingestion_date
1	albert_park	Albert Park Grand Prix Circuit	Melbourne	Australia	-37.8497	144.968	10	2024-04-22 17:59:07.918
2	sepang	Sepang International Circuit	Kuala Lumpur	Malaysia	2.76083	101.738	18	2024-04-22 17:59:07.918
3	bahrain	Bahrain International Circuit	Sakhir	Bahrain	26.0325	50.5106	7	2024-04-22 17:59:07.918
4	catalunya	Circuit de Barcelona-Catalunya	Montmeló	Spain	41.57	2.26111	109	2024-04-22 17:59:07.918
5	istanbul	Istanbul Park	Istanbul	Turkey	40.9517	29.405	130	2024-04-22 17:59:07.918
6	monaco	Circuit de Monaco	Monte-Carlo	Monaco	43.7347	7.42056	7	2024-04-22 17:59:07.918
7	villeneuve	Circuit Gilles Villeneuve	Montreal	Canada	45.5	-73.5228	13	2024-04-22 17:59:07.918
8	magny_cours	Circuit de Nevers Magny-Cours	Magny Cours	France	46.8642	3.16361	228	2024-04-22 17:59:07.918
9	silverstone	Silverstone Circuit	Silverstone	UK	52.0786	-1.01694	153	2024-04-22 17:59:07.918
10	hockenheimring	Hockenheimring	Hockenheim	Germany	49.3278	8.56583	103	2024-04-22 17:59:07.918
11	hungaroring	Hungaroring	Budapest	Hungary	47.5789	19.2486	264	2024-04-22 17:59:07.918
12	valencia	Valencia Street Circuit	Valencia	Spain	39.4589	-0.331667	4	2024-04-22 17:59:07.918
13	spa	Circuit de Spa-Francorchamps	Spa	Belgium	50.4372	5.97139	401	2024-04-22 17:59:07.918
14	monza	Autodromo Nazionale di Monza	Monza	Italy	45.6156	9.28111	162	2024-04-22 17:59:07.918
15	marina_bay	Marina Bay Street Circuit	Marina Bay	Singapore	1.2914	103.864	18	2024-04-22 17:59:07.918
16	fuji	Fuji Speedway	Oyama	Japan	35.3717	138.927	583	2024-04-22 17:59:07.918
17	shanghai	Shanghai International Circuit	Shanghai	China	31.3389	121.22	5	2024-04-22 17:59:07.918
18	interlaos	Autódromo José Carlos Pace	São Paulo	Brazil	-23.7036	-46.6997	785	2024-04-22 17:59:07.918

Step 7: Creating new directory in databricks filestore and storing our dataframe in columnar format that is parquet file.

```

1 dbutils.fs.mkdirs("dbfs:/FileStore/Formu1a1/processed")

Out[16]: True

Command took 0.74 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:28:47 PM on My Cluster

Cell 16

1 circuits_final_df.write.parquet("FileStore/Formu1a1/processed/circuits")

AnalysisException: Path dbfs:/FileStore/Formu1a1/processed/circuits already exists.

Command took 3.13 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:28:47 PM on My Cluster

Cell 17

1 %fs ls "FileStore/Formu1a1/processed"

Command skipped

Command took 0.60 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:28:47 PM on My Cluster

Cell 18

```

2. Constructor file ingestion:

Step 1: Create the schema for Constructor file using String format and then read the file from constructor JSON file.

```

1 #Step 1 - Read the JSON file using the spark dataframe reader

Command took 0.00 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:32:14 PM on My Cluster

Cell 2

1 from pyspark.sql.functions import col , current_timestamp

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:32:14 PM on My Cluster

Cell 3

1 constructors_schema = "constructorId INT, constructorRef STRING, name STRING, nationality STRING, url1 STRING"
2

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:32:14 PM on My Cluster

Cell 4

1 constructor_df = spark.read.json("dbfs:/FileStore/constructors.json", schema=constructors_schema)

constructor_df pyspark.sql.dataframe.DataFrame = [constructorId: integer, constructorRef: string ... 3 more fields]

```

Step2: We are dropping the url column by drop command.

The screenshot shows a Databricks notebook interface with the following code cells:

```

1 #Step 2 - Drop unwanted columns from the dataframe

Command took 0.06 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:32:14 PM on My Cluster

Cell 6

1 constructor_dropped_df = constructor_df.drop(col('un1'))

↳ constructor_dropped_df: pyspark.sql.dataframe.DataFrame = [constructorId: integer, constructorRef: string...2 more fields]

Command took 0.19 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:32:14 PM on My Cluster

Cell 7

1 #Step 3 - Rename columns and add ingestion date

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:32:14 PM on My Cluster

Cell 8

1 constructor_renamed_df = constructor_dropped_df.withColumnRenamed("constructorId", "constructor_id") \
2 .withColumnRenamed("constructorRef", "constructor_ref") \
3 .withColumn("ingestion_date", current_timestamp())
  
```

The screenshot shows the output of the command `constructor_renamed_df.show(truncate=False)`. The output is a table with 17 rows and 5 columns: `constructor_id`, `constructor_ref`, `name`, `nationality`, and `ingestion_date`.

constructor_id	constructor_ref	name	nationality	ingestion_date
1	mclaren	McLaren	British	2024-04-22 18:03:52.448
2	bmw_sauber	BMW Sauber	German	2024-04-22 18:03:52.448
3	williams	Williams	British	2024-04-22 18:03:52.448
4	renault	Renault	French	2024-04-22 18:03:52.448
5	toro_rosso	Toro Rosso	Italian	2024-04-22 18:03:52.448
6	ferrari	Ferrari	Italian	2024-04-22 18:03:52.448
7	toyota	Toyota	Japanese	2024-04-22 18:03:52.448
8	super_aguri	Super Aguri	Japanese	2024-04-22 18:03:52.448
9	red_bull	Red Bull	Austrian	2024-04-22 18:03:52.448
10	force_india	Force India	Indian	2024-04-22 18:03:52.448
11	honda	Honda	Japanese	2024-04-22 18:03:52.448
12	spyker	Spyker	Dutch	2024-04-22 18:03:52.448
13	mf1	MF1	Russian	2024-04-22 18:03:52.448
14	spyker_mf1	Spyker MF1	Dutch	2024-04-22 18:03:52.448
15	sauber	Sauber	Swiss	2024-04-22 18:03:52.448
16	bar	BAR	British	2024-04-22 18:03:52.448
17	jordan	Jordan	Irish	2024-04-22 18:03:52.448

Step3: Storing our dataframe in columnar format that is parquet file.

The screenshot shows a Databricks notebook interface. The top section displays a table with 18 rows of constructor data, including names like Honda, Spyker, MF1, and Minardi, along with their nationalities and a timestamp. Below the table, a command to write the data to a Parquet file is shown, but it has failed with an `AnalysisException` stating that the path already exists.

11	honda	Honda	Japanese	2024-04-22 18:03:52.448
12	spyker	Spyker	Dutch	2024-04-22 18:03:52.448
13	mf1	MF1	Russian	2024-04-22 18:03:52.448
14	spyker_mf1	Spyker MF1	Dutch	2024-04-22 18:03:52.448
15	sauber	Sauber	Swiss	2024-04-22 18:03:52.448
16	bar	BAR	British	2024-04-22 18:03:52.448
17	jordan	Jordan	Irish	2024-04-22 18:03:52.448
18	minardi	Minardi	Italian	2024-04-22 18:03:52.448

```

1 #Step 4 Write output to parquet file

1 constructor_renamed_df.write.parquet("dbfs:/FileStore/Formula1/processed/f1_processed.constructors")

```

`AnalysisException: Path dbfs:/FileStore/Formula1/processed/f1_processed.constructors already exists.`

3. Driver file ingestion:

Step 1: Create the schema for the drivers file using StructType

The screenshot shows a Databricks notebook interface. The top section displays a command to read a JSON file using the Spark DataFrame reader API. Below this, a command to import the necessary StructType and StructField classes from pyspark.sql.types is shown. The final command defines a name_schema with fields for 'forename' and 'surname'.

```

1 #Step 1 - Read the JSON file using the spark dataframe reader API

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

1 name_schema = StructType(fields=[StructField("forename", StringType(), True),
2     StructField("surname", StringType(), True)
3 ])

```

Step 3: We have created a dataframe by reading the data from json file loaded in DBFS system

The screenshot shows a Databricks notebook titled "driver processed" with a Python environment. The notebook is running on a cluster named "My Cluster". The code in the first cell defines a schema for driver data:

```
1 drivers_schema = StructType(fields=[StructField("driverId", IntegerType(), False),
2                                     StructField("driverRef", StringType(), True),
3                                     StructField("number", IntegerType(), True),
4                                     StructField("code", StringType(), True),
5                                     StructField("name", name_schema),
6                                     StructField("dob", DateType(), True),
7                                     StructField("nationality", StringType(), True),
8                                     StructField("un1", StringType(), True)
9                                     ])
10 ])
```

The command took 0.00 seconds to execute. The second cell shows the data being read from a JSON file:

```
1 drivers_df = spark.read.json("dbfs:/FileStore/drivers.json", schema = drivers_schema)
```

The command took 0.00 seconds to execute. The notebook interface includes a sidebar with navigation icons and a top bar with the Databricks logo and user information.

Step 4: Renaming the columns using withColumnRenamed command to match the naming standards.

The screenshot shows the same Databricks notebook, now with the second cell executed. The code in the third cell shows the renaming of columns and adding a new column:

```
1 #Step 2 - Rename columns and add new columns
2 #driverId renamed to driver_id
3 #driverRef renamed to driver_ref
4 #ingestion date added
5 #name added with concatenation of forename and surname
```

The command took 0.00 seconds to execute. The fourth cell shows the import of the necessary functions:

```
1 from pyspark.sql.functions import col, concat, lit, current_timestamp
```

The command took 0.00 seconds to execute. The fifth cell shows the renaming of columns and adding a new column:

```
1 drivers_with_ingestion_date_df = drivers_df.withColumn("ingestion_date", current_timestamp())
```

The command took 0.00 seconds to execute. The notebook interface includes a sidebar with navigation icons and a top bar with the Databricks logo and user information.

```

1 drivers_with_columns_df = drivers_with_ingestion_date_df.withColumnRenamed("driverId", "driver_id") \
2   .withColumnRenamed("driverRef", "driver_ref") \
3   .withColumn("name", concat(col("name.forename"), lit(" "), col("name.surname")))

> drivers_with_columns_df: pyspark.sql.dataframe.DataFrame = [driver_id: integer, driver_ref: string ... 7 more fields]
Command took 0.29 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:36:50 PM on My Cluster

Cell 10
1 #Step 3 - Drop the unwanted columns
2 #name.forename
3 #name.surname
4 #url

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:36:50 PM on My Cluster

Cell 11
1 drivers_final_df = drivers_with_columns_df.drop(col("url"))

> drivers_final_df: pyspark.sql.dataframe.DataFrame = [driver_id: integer, driver_ref: string ... 6 more fields]
Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:36:50 PM on My Cluster

```

Step 5: Storing our dataframe in columnar format that is parquet file.

```

1 #Step 4 - Write to output to processed container in parquet format

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:36:50 PM on My Cluster

Cell 13
1 drivers_final_df.write.parquet("dbfs:/FileStore/Formu1a1/processed/f1_processed.drivers")

AnalysisException: Path dbfs:/FileStore/Formu1a1/processed/f1_processed.drivers already exists.
Command took 3.38 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:36:50 PM on My Cluster

Cell 14
1 drivers_final_df.show(truncate=False)

> (1) Spark Jobs

+-----+-----+-----+-----+-----+-----+-----+
|driver_id|driver_ref|number|code|name|dob|nationality|ingestion_date|
+-----+-----+-----+-----+-----+-----+-----+
|1|hamilton|44|HAM|Lewis Hamilton|1985-01-07|British|2024-04-22 18:07:10.948|
|2|heidfeld|null|HEI|Nick Heidfeld|1977-05-10|German|2024-04-22 18:07:10.948|
|3|rosberg|6|ROS|Nico Rosberg|1985-06-27|German|2024-04-22 18:07:10.948|
|4|alonso|14|ALO|Fernando Alonso|1981-07-29|Spanish|2024-04-22 18:07:10.948|
|5|Kovalainen|null|KOV|Mika Kovalainen|1981-10-10|Finnish|2024-04-22 18:07:10.948|

```

Similarly, we have performed ingestion for all the other files into the DBFS system.

4. LAPTIME INGEST:

The screenshot shows a Databricks notebook titled 'laptimes ingest'. The interface includes a top bar with 'python', 'pyspark', and 'proj' tabs. The notebook has three cells visible:

- Cell 1:** Contains the code `dbutils.fs.mkdirs("dbfs:/FileStore/Formula1/lap_times")`. The output is `Out[1]: True`. A command execution message indicates it took 0.26 seconds.
- Cell 2:** Contains a comment: `#Step 1 - Read the CSV file using the spark dataframe reader API`. A command execution message indicates it took 0.13 seconds.
- Cell 3:** Contains the code `from pyspark.sql.types import *`. A command execution message indicates it took 0.09 seconds.

Cell 4 is partially visible, showing the start of a `StructType` definition for `lap_times_schema`.

This screenshot continues the notebook from the previous one. It shows the completion of the schema definition and the subsequent data ingestion:

- Cell 4:** The `lap_times_schema` is fully defined with fields: `raceId` (Integer, False), `driverId` (Integer, True), `lap` (Integer, True), `position` (Integer, True), `time` (String, True), and `milliseconds` (Integer, True). A command execution message indicates it took 0.09 seconds.
- Cell 5:** Contains the code `lap_times_df = spark.read.csv("dbfs:/FileStore/Formula1/lap_times", schema= lap_times_schema)`. A command execution message indicates it took 0.60 seconds. Below the code, a preview of the DataFrame is shown: `lap_times_df: pyspark.sql.dataframe.DataFrame = [raceId: integer, driverId: integer ... 4 more fields]`.

community.cloud.databricks.com/?o=6796678178271343#notebook/841007799630703/command/641007799630704

databricks 1906107@nitindia.edu

lap_times_ingest Python

File Edit View Run Help Last edit was 4 days ago New cell UI: OFF

Run all My Cluster Share Publish

```

1 #Step 2 - Rename columns and add new columns
2 #Rename driverId and raceId
3 #Add ingestion_date with current timestamp

```

Command took 0.08 seconds -- by 1906107@nitindia.edu at 4/22/2024, 11:40:14 PM on My Cluster

Cell 7

```

1 from pyspark.sql.functions import current_timestamp

```

Command took 0.09 seconds -- by 1906107@nitindia.edu at 4/22/2024, 11:40:14 PM on My Cluster

Cell 8

```

1 final_df = lap_times_df.withColumnRenamed("driverId", "driver_id") \
2 .withColumnRenamed("raceId", "race_id") \
3 .withColumn("ingestion_date", current_timestamp())
4

```

final_df: pyspark.sql.dataframe.DataFrame = (race_id: integer, driver_id: integer ... 5 more fields)

Command took 0.19 seconds -- by 1906107@nitindia.edu at 4/22/2024, 11:40:14 PM on My Cluster

Cell 9

community.cloud.databricks.com/?o=6796678178271343#notebook/841007799630703/command/3271776022846954

databricks 1906107@nitindia.edu

lap_times_ingest Python

File Edit View Run Help Last edit was 1 minute ago New cell UI: OFF

Run all My Cluster Share Publish

```

1 final_df.show(truncate=False)

```

(1) Spark Jobs

race_id	driver_id	lap	position	time	milliseconds	ingestion_date
841	20	1	1	1:38.109	98109	2024-04-22 18:12:25.882
841	20	2	1	1:33.006	93006	2024-04-22 18:12:25.882
841	20	3	1	1:32.713	92713	2024-04-22 18:12:25.882
841	20	4	1	1:32.803	92803	2024-04-22 18:12:25.882
841	20	5	1	1:32.342	92342	2024-04-22 18:12:25.882
841	20	6	1	1:32.605	92605	2024-04-22 18:12:25.882
841	20	7	1	1:32.502	92502	2024-04-22 18:12:25.882
841	20	8	1	1:32.537	92537	2024-04-22 18:12:25.882
841	20	9	1	1:33.240	93240	2024-04-22 18:12:25.882
841	20	10	1	1:32.572	92572	2024-04-22 18:12:25.882
841	20	11	1	1:32.669	92669	2024-04-22 18:12:25.882
841	20	12	1	1:32.902	92902	2024-04-22 18:12:25.882
841	20	13	1	1:33.698	93698	2024-04-22 18:12:25.882
841	20	14	3	1:52.075	112075	2024-04-22 18:12:25.882
841	20	15	4	1:38.385	98385	2024-04-22 18:12:25.882
841	20	16	2	1:31.548	91548	2024-04-22 18:12:25.882
841	20	17	1	1:30.800	90800	2024-04-22 18:12:25.882

lap_times_ingest Python

841	20	14	3	1:52.075	111075	2024-04-22 18:12:25.882
841	20	15	4	1:38.385	98385	2024-04-22 18:12:25.882
841	20	16	2	1:31.548	91548	2024-04-22 18:12:25.882
841	20	17	1	1:30.888	90888	2024-04-22 18:12:25.882
841	20	18	1	1:31.818	91818	2024-04-22 18:12:25.882

```

1 dbutils.fs.mkdirs("dbfs:/FileStore/Formula1/processed/lap_times")

Out[10]: True

1 final_df.write.parquet("dbfs:/FileStore/Formula1/processed/lap_time")

AnalysisException: Path dbfs:/FileStore/Formula1/processed/lap_time already exists.
  
```

5. PIT STOP INGESTION:

pit_stops_processed Python

```

1 #Step 1 - Read the JSON file using the spark dataframe reader API

1 from pyspark.sql.types import StructType, StructField, IntegerType, StringType

1 pit_stops_schema = StructType(fields=[StructField("raceId", IntegerType(), False),
2                                             StructField("driverId", IntegerType(), True),
3                                             StructField("stop", StringType(), True),
4                                             StructField("lap", IntegerType(), True),
5                                             StructField("time", StringType(), True),
6                                             StructField("duration", StringType(), True),
7                                             StructField("milliseconds", IntegerType(), True)
8                                             ])
  
```

The screenshot shows a Databricks notebook titled "pit_stops_processed". The first code cell reads a JSON file from the FileStore and creates a DataFrame. The second code cell displays the DataFrame using `show()`.

```
1 pit_stops_df = spark.read.json("dbfs:/FileStore/pit_stops.json", schema = pit_stops_schema , multiline=True)
```

```
pit_stops_df: pyspark.sql.dataframe.DataFrame = [raceId: integer, driverId: integer ... 5 more fields]
```

Command took 0.39 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:43:46 PM on My Cluster

```
1 pit_stops_df.show()
```

(1) Spark Jobs

raceId	driverId	stop	lap	time	duration	milliseconds
841	153	1	1	17:05:23	26.898	26898
841	30	1	1	17:05:52	25.021	25021
841	17	1	1	17:20:48	23.426	23426
841	4	1	1	17:22:34	23.251	23251
841	13	1	1	17:24:10	23.842	23842
841	22	1	1	17:24:29	23.643	23643
841	20	1	1	17:25:17	22.603	22603
841	814	1	1	17:26:03	24.863	24863
841	816	1	1	17:26:50	25.259	25259

The screenshot shows the next steps in the Databricks notebook. Step 2 involves renaming columns and adding a new column. The code uses `lit` and `current_timestamp` functions.

```
1 #Step 2 - Rename columns and add new columns
2 #Rename driverId and raceId
3 #Add ingestion_date with current timestamp
```

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:47:33 PM on My Cluster

```
1 from pyspark.sql.functions import lit , current_timestamp
```

Command took 0.08 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:47:33 PM on My Cluster

```
1 pit_stops_with_ingestion_date_df=pit_stops_df.withColumn("ingestion_date",current_timestamp())
```

```
pit_stops_with_ingestion_date_df: pyspark.sql.dataframe.DataFrame = [raceId: integer, driverId: integer ... 6 more fields]
```

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:47:33 PM on My Cluster

```
1 final_df = pit_stops_with_ingestion_date_df.withColumnRenamed("driverId", "driver_id") \
2 withColumnRenamed("raceId", "race_id")
```


community.cloud.databricks.com/?o=6796678178271343#notebook/3417363049632130/command/3417363049632143

1906107@ritindia.edu

pit_stops_processed Python

File Edit View Run Help Last edit was 1 minute ago New cell UI: OFF

```

1 final_df = pit_stops_with_ingestion_date_df.withColumnRenamed("driverId", "driver_id") \
2   .withColumnRenamed("raceId", "race_id")

final_df: pyspark.sql.dataframe.DataFrame = [race_id: integer, driver_id: integer ... 6 more fields]
Command took 0.89 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:47:33 PM on My Cluster

```

Cell 10

```

1 final_df.show(truncate=False)

```

(1) Spark Jobs

race_id	driver_id	stop	lap	time	duration	milliseconds	ingestion_date
841	153	1	1	17:05:23	26.898	26898	2024-04-22 18:17:35.691
841	130	1	1	17:05:52	25.021	25021	2024-04-22 18:17:35.691
841	17	1	11	17:20:48	23.426	23426	2024-04-22 18:17:35.691
841	4	1	12	17:22:34	23.251	23251	2024-04-22 18:17:35.691
841	13	1	13	17:24:10	23.842	23842	2024-04-22 18:17:35.691
841	22	1	13	17:24:29	23.643	23643	2024-04-22 18:17:35.691
841	20	1	14	17:25:17	22.603	22603	2024-04-22 18:17:35.691
841	814	1	14	17:26:03	24.863	24863	2024-04-22 18:17:35.691
841	816	1	14	17:26:50	25.259	25259	2024-04-22 18:17:35.691

11:48 PM 4/22/2024

community.cloud.databricks.com/?o=6796678178271343#notebook/3417363049632130/command/3417363049632143

1906107@ritindia.edu

pit_stops_processed Python

File Edit View Run Help Last edit was 2 minutes ago New cell UI: OFF

```

1 final_df.write.mode('overwrite').parquet("dbfs:/FileStore/Formula1/processed/F1_processed/pit_stops")

```

Command took 0.59 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:47:33 PM on My Cluster

Cell 11

```

1 #Step 3 - Write to output to processed container in parquet format

```

Command took 0.29 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:47:33 PM on My Cluster

Cell 12

```

1 final_df.write.mode('overwrite').parquet("dbfs:/FileStore/Formula1/processed/F1_processed/pit_stops")

```

(1) Spark Jobs

Command took 2.10 seconds -- by 1906107@ritindia.edu at 4/22/2024, 11:47:33 PM on My Cluster

Cell 13

```

1

```

11:48 PM 4/22/2024

6. Qualifying ingest:

The screenshot shows a Databricks notebook titled 'qualifying_processed'. The interface includes a top bar with 'python', 'pyspark', and 'pro' tabs. The notebook has a dark theme and a sidebar with navigation icons. The main area displays four code cells:

- Cell 1:** Contains a single line of code: `dbutils.fs.mkdirs("dbfs:/FileStore/Formula1/qualifying")`. The output shows 'Command execution completed' and 'Out[1]: True'.
- Cell 2:** Contains a single line of code: `#Step 1 - Read the JSON file using the spark dataframe reader API`. The output shows 'Command execution completed'.
- Cell 3:** Contains a single line of code: `from pyspark.sql.types import StructType, StructField, IntegerType, StringType`. The output shows 'Command execution completed'.
- Cell 4:** Contains two lines of code: `1` and `2`. Line 2 defines a schema: `qualifying_schema = StructType(fields=[StructField("qualifyId", IntegerType(), False),`.

The bottom of the notebook shows a Windows taskbar with a search bar and various application icons. The system clock indicates 9:28 AM on 4/23/2024.

The screenshot shows the same Databricks notebook, now with a message at the top: 'Stop running all cells in this notebook.' The notebook displays five code cells:

- Cell 4:** Contains a multi-line schema definition: `1` through `11`. The schema includes fields for 'qualifyId', 'raceId', 'driverId', 'constructorId', 'number', 'position', 'q1', 'q2', and 'q3'. The output shows 'Command took 0.08 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:00 AM on compute'.
- Cell 5:** Contains a single line of code: `1` `qualifying_df = spark.read.json("dbfs:/FileStore/Formula1/qualifying", schema = qualifying_schema , multiline= True)`. The output shows 'Running command...' and a preview of the resulting DataFrame: `qualifying_df: pyspark.sql.dataframe.DataFrame = [qualifyId: integer, raceId: integer... 7 more fields]`.

The bottom of the notebook shows the same Windows taskbar with a search bar and various application icons. The system clock indicates 9:28 AM on 4/23/2024.

The screenshot shows a Databricks notebook interface with the following code in Cell 8:

```

1 #Step 2 - Rename columns and add new columns
2 #Rename qualifyingId, driverId, constructorId and raceId
3 #Add ingestion_date with current timestamp

```

Below the code, the output of the command is displayed:

```

Command took 0.22 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:40 AM on compute

Cell 8
1 from pyspark.sql.functions import lit, current_timestamp

Command took 0.16 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:40 AM on compute

Cell 9
1 final_df = qualifying_df.withColumnRenamed("qualifyingId", "qualify_id") \
2 .withColumnRenamed("driverId", "driver_id") \
3 .withColumnRenamed("raceId", "race_id") \
4 .withColumnRenamed("constructorId", "constructor_id") \
5 .withColumn("ingestion_date", current_timestamp())

final_df: pyspark.sql.dataframe.DataFrame = [qualify_id: integer, race_id: integer ... 8 more fields]

Command took 0.88 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:40 AM on compute

Cell 10

```

A notification at the top right states "Last command failed".

The screenshot shows a Databricks notebook interface with the following code in Cell 6:

```

1 qualifying_df.show()

```

The output of the command is displayed as a table:

```

(1) Spark Jobs
+-----+-----+-----+-----+-----+-----+-----+-----+
|qualifyId|raceId|driverId|constructorId|number|position|q1|q2|q3|
+-----+-----+-----+-----+-----+-----+-----+-----+
|1|18|1|1|22|1|1:26.572|1:25.187|1:26.714|
|2|18|9|2|4|2|1:26.103|1:25.315|1:26.869|
|3|18|5|1|23|3|1:25.664|1:25.452|1:27.079|
|4|18|13|6|2|4|1:25.994|1:25.691|1:27.178|
|5|18|2|2|3|5|1:25.960|1:25.518|1:27.236|
|6|18|15|7|11|6|1:26.427|1:26.101|1:28.527|
|7|18|3|3|7|7|1:26.295|1:26.059|1:28.687|
|8|18|14|9|9|8|1:26.381|1:26.063|1:29.041|
|9|18|10|7|12|9|1:26.919|1:26.164|1:29.593|
|10|18|20|5|15|10|1:26.702|1:25.842|\N|
|11|18|22|11|17|11|1:26.369|1:26.173|\N|
|12|18|4|4|5|12|1:26.907|1:26.188|\N|
|13|18|18|11|16|13|1:26.712|1:26.259|\N|
|14|18|6|3|8|14|1:26.891|1:26.413|\N|
|15|18|17|9|10|15|1:26.914|\N|\N|
|16|18|8|6|1|16|1:26.140|\N|\N|

```

community.cloud.databricks.com/?o=6796678178271343#notebook/3417363049632145/command/3417363049632151

databricks

qualifying_processed Python ☆

File Edit View Run Help Last edit was 19 hours ago New cell UI: OFF

1 #Step 2 - Rename columns and add new columns
2 #Rename qualifyingId, driverId, constructorId and raceId
3 #Add ingestion_date with current timestamp

Command took 0.22 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:40 AM on compute

Cmd 8

1 from pyspark.sql.functions import lit, current_timestamp

Command took 0.16 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:40 AM on compute

Cmd 9

1 final_df = qualifying_df.withColumnRenamed("qualifyingId", "qualify_id") \
2 .withColumnRenamed("driverId", "driver_id") \
3 .withColumnRenamed("raceId", "race_id") \
4 .withColumnRenamed("constructorId", "constructor_id") \
5 .withColumn("ingestion_date", current_timestamp())

final_df: pyspark.sql.dataframe.DataFrame = [qualify_id: integer, race_id: integer ... 8 more fields]

Command took 0.88 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:40 AM on compute

Cmd 10

community.cloud.databricks.com/?o=6796678178271343#notebook/3417363049632145/command/3417363049632155

databricks

qualifying_processed Python ☆

File Edit View Run Help Last edit was now New cell UI: OFF

Run all compute Share Publish

Cmd 10

1 final_df.show(truncate=False)

(1) Spark Jobs

	qualify_id	race_id	driver_id	constructor_id	number	position	q1	q2	q3	ingestion_date
1	18	1	1	22	1	1:26.572	1:25.187	1:26.714	2024-04-23 04:00:32.158	
2	18	9	2	4	2	1:26.103	1:25.315	1:26.869	2024-04-23 04:00:32.158	
3	18	5	1	23	3	1:25.664	1:25.452	1:27.079	2024-04-23 04:00:32.158	
4	18	13	6	2	4	1:25.994	1:25.691	1:27.178	2024-04-23 04:00:32.158	
5	18	2	2	3	5	1:25.968	1:25.518	1:27.236	2024-04-23 04:00:32.158	
6	18	15	7	11	6	1:26.427	1:26.101	1:28.527	2024-04-23 04:00:32.158	
7	18	3	3	7	7	1:26.295	1:26.059	1:28.687	2024-04-23 04:00:32.158	
8	18	14	9	9	8	1:26.381	1:26.063	1:29.041	2024-04-23 04:00:32.158	
9	18	10	7	12	9	1:26.919	1:26.164	1:29.593	2024-04-23 04:00:32.158	
10	18	20	5	15	10	1:26.782	1:25.842	\N	2024-04-23 04:00:32.158	
11	18	22	11	17	11	1:26.369	1:26.173	\N	2024-04-23 04:00:32.158	
12	18	4	4	5	12	1:26.907	1:26.188	\N	2024-04-23 04:00:32.158	
13	18	18	11	16	13	1:26.712	1:26.259	\N	2024-04-23 04:00:32.158	
14	18	6	3	8	14	1:26.891	1:26.413	\N	2024-04-23 04:00:32.158	
15	18	17	9	10	15	1:26.914	\N	\N	2024-04-23 04:00:32.158	

qualifying_processed Python

```

1 #Step 3 - Write to output to processed container in parquet format

Command took 0.87 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:41 AM on compute

Cell 12

1 final_df.write.parquet("dbfs:/FileStore/Formu1al/processed/f1_processed.qualifying")

AnalysisException: Path dbfs:/FileStore/Formu1al/processed/f1_processed.qualifying already exists.
Command took 6.01 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:28:41 AM on compute

Cell 13

1 final_df.show(truncate=False)

Python

(1) Spark Jobs
+-----+-----+-----+-----+-----+-----+-----+-----+
|qualify_id|race_id|driver_id|constructor_id|number|position|q1      |q2      |q3      |ingestion_date|
+-----+-----+-----+-----+-----+-----+-----+-----+
|1         |18     |1        |1              |22    |1       |1:26.572|1:25.187|1:26.714|2024-04-23 04:01:10.743|
|2         |18     |9        |2              |4      |2       |1:26.103|1:25.315|1:26.869|2024-04-23 04:01:10.743|
|3         |18     |5        |1              |23    |3       |1:25.664|1:25.452|1:27.079|2024-04-23 04:01:10.743|
|4         |18     |13       |6              |2      |4       |1:25.994|1:25.691|1:27.178|2024-04-23 04:01:10.743|
|5         |18     |2        |2              |3      |5       |1:25.960|1:25.518|1:27.236|2024-04-23 04:01:10.743|
|6         |18     |15       |7              |11     |6       |1:26.427|1:26.101|1:28.527|2024-04-23 04:01:10.743|
|7         |18     |3        |3              |17     |17      |1:26.295|1:26.059|1:28.687|2024-04-23 04:01:10.743|

```

7. Race Ingest

RACES_INGEST Python

```

1 #Step 1 - Read the CSV file using the spark dataframe reader API

Command took 0.57 seconds -- by 1906107@ritindia.edu at 4/18/2024, 10:30:06 PM on PROJECT4

Cell 2

1 from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DateType
2 from pyspark.sql.functions import current_timestamp

Command took 0.14 seconds -- by 1906107@ritindia.edu at 4/18/2024, 10:30:06 PM on PROJECT4

Cell 3

1
2 races_schema = StructType(fields=[StructField("raceId", IntegerType(), False),
3                                     StructField("year", IntegerType(), True),
4                                     StructField("round", IntegerType(), True),
5                                     StructField("circuitId", IntegerType(), True),
6                                     StructField("name", StringType(), True),
7                                     StructField("date", DateType(), True),
8                                     StructField("time", StringType(), True),
9                                     StructField("url", StringType(), True)
10 ])

```

The screenshot shows a Databricks notebook interface with the title "RACES_INGEST". The notebook is running on a cluster. The first three code cells are visible:

```

1 races_df = spark.read.csv("dbfs:/FileStore/races.csv", schema = races_schema )

races_df: pyspark.sql.dataframe.DataFrame = [raceId: integer, year: integer ... 6 more fields]
Command took 2.50 seconds -- by 1906107@ritindia.edu at 4/18/2024, 10:39:06 PM on PROJECT4

Cell 5

1 #Step 2 - Add ingestion date and race_timestamp to the dataframe

Command took 0.16 seconds -- by 1906107@ritindia.edu at 4/18/2024, 10:39:06 PM on PROJECT4

Cell 6

1 from pyspark.sql.functions import to_timestamp, concat, col, lit

Command took 0.00 seconds -- by 1906107@ritindia.edu at 4/18/2024, 10:39:06 PM on PROJECT4

Cell 7

1 races_with_timestamp_df = races_df.withColumn("race_timestamp", to_timestamp(concat(col('date'), lit(' '), col('time')), 'yyyy-MM-dd
HH:mm:ss')).withColumn("ingestion_date", current_timestamp())

races_with_timestamp_df: pyspark.sql.dataframe.DataFrame = [raceId: integer, year: integer ... 8 more fields]
  
```

The screenshot shows the continuation of the Databricks notebook. The next three code cells are visible:

```

Cell 8

1 #Step 3 - Select only the columns required & rename as required

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/18/2024, 10:39:06 PM on PROJECT4

Cell 9

1 races_selected_df = races_with_timestamp_df.select(col('raceId').alias('race_id'), col('year').alias('race_year'),
2 col('round'), col('circuitId').alias('circuit_id'), col('name'), col('ingestion_date'),
col('race_timestamp'))

races_selected_df: pyspark.sql.dataframe.DataFrame = [race_id: integer, race_year: integer ... 5 more fields]
Command took 0.20 seconds -- by 1906107@ritindia.edu at 4/18/2024, 10:39:06 PM on PROJECT4

Cell 10

1 #Write the output to processed container in parquet format

Command took 0.00 seconds -- by 1906107@ritindia.edu at 4/18/2024, 10:39:06 PM on PROJECT4

Cell 11

1 races_selected_df.write.parquet("dbfs:/FileStore/Formula1/processed/fl_processed_races")
  
```

The screenshot shows a Databricks notebook interface with the title 'RACES_INGEST'. The notebook is in Python mode. The first cell contains a comment: '#Write the output to processed container in parquet format'. The second cell contains the code: `1 races_selected_df.write.mode("overwrite").parquet("dbfs:/FileStore/Formu1a1/processed/f1_processed_races")`. The third cell contains the code: `1 races_selected_df.show(truncate=False)`. The output of the third cell is a table with 7 columns: race_id, race_year, round, circuit_id, name, ingestion_date, and race_timestamp. The table contains 5 rows of data.

race_id	race_year	round	circuit_id	name	ingestion_date	race_timestamp
1	2009	1	1	Australian Grand Prix	2024-04-23 04:06:46.352	2009-03-29 06:00:00
2	2009	2	2	Malaysian Grand Prix	2024-04-23 04:06:46.352	2009-04-05 09:00:00
3	2009	3	17	Chinese Grand Prix	2024-04-23 04:06:46.352	2009-04-19 07:00:00
4	2009	4	3	Bahrain Grand Prix	2024-04-23 04:06:46.352	2009-04-26 12:00:00
5	2009	5	16	Spanish Grand Prix	2024-04-23 04:06:46.352	2009-05-03 00:00:00

8. Result file ingestion

The screenshot shows a Databricks notebook interface with the title 'result processed'. The notebook is in Python mode. The first cell contains a comment: '#Step 1 - Read the JSON file using the spark dataframe reader API'. The second cell contains the code: `1 from pyspark.sql.types import StructType, StructField, IntegerType, StringType, FloatType`. The third cell contains the code: `1 results_schema = StructType(fields=[StructField("resultId", IntegerType(), False), StructField("raceId", IntegerType(), True), StructField("driverId", IntegerType(), True), StructField("constructorId", IntegerType(), True), StructField("number", IntegerType(), True), StructField("grid", IntegerType(), True), StructField("position", IntegerType(), True), StructField("positionText", StringType(), True), StructField("positionOrder", IntegerType(), True), StructField("points", FloatType(), True), StructField("laps", IntegerType(), True), StructField("status", StringType(), True)])`.

The screenshot shows a Databricks notebook interface. The notebook is titled "result processed" and is in Python mode. The code defines a schema for Formula 1 results. The schema includes fields for resultId, raceId, driverId, constructorId, number, grid, position, positionText, positionOrder, points, laps, time, milliseconds, fastestLap, rank, fastestLapTime, fastestLapSpeed, and statusId. The notebook is running on a cluster named "compute".

```

1 results_schema = StructType(fields=[StructField("resultId", IntegerType(), False),
2                                             StructField("raceId", IntegerType(), True),
3                                             StructField("driverId", IntegerType(), True),
4                                             StructField("constructorId", IntegerType(), True),
5                                             StructField("number", IntegerType(), True),
6                                             StructField("grid", IntegerType(), True),
7                                             StructField("position", IntegerType(), True),
8                                             StructField("positionText", StringType(), True),
9                                             StructField("positionOrder", IntegerType(), True),
10                                            StructField("points", FloatType(), True),
11                                            StructField("laps", IntegerType(), True),
12                                            StructField("time", StringType(), True),
13                                            StructField("milliseconds", IntegerType(), True),
14                                            StructField("fastestLap", IntegerType(), True),
15                                            StructField("rank", IntegerType(), True),
16                                            StructField("fastestLapTime", StringType(), True),
17                                            StructField("fastestLapSpeed", FloatType(), True),
18                                            StructField("statusId", StringType(), True)])

```

Command took 0.06 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute

```

1 results_df = spark.read.json("dbfs:/FileStore/results.json", schema = results_schema)

```

The screenshot shows the same Databricks notebook interface, but now showing the next step in the process: renaming columns and adding new columns. The code uses the `withColumnRenamed` method to rename columns and the `lit` function to add a new column for the current timestamp.

```

1 #Step 2 - Rename columns and add new columns

```

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute

```

1 from pyspark.sql.functions import lit, current_timestamp

```

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute

```

1 results_with_columns_df = results_df.withColumnRenamed("resultId", "result_id") \
2                                     .withColumnRenamed("raceId", "race_id") \
3                                     .withColumnRenamed("driverId", "driver_id") \
4                                     .withColumnRenamed("constructorId", "constructor_id") \
5                                     .withColumnRenamed("positionText", "position_text") \
6                                     .withColumnRenamed("positionOrder", "position_order") \
7                                     .withColumnRenamed("fastestLap", "fastest_lap") \
8                                     .withColumnRenamed("fastestLapTime", "fastest_lap_time") \
9                                     .withColumnRenamed("fastestLapSpeed", "fastest_lap_speed")
10

```

results_with_columns_df: pyspark.sql.dataframe.DataFrame = [result_id: integer, race_id: integer ... 16 more fields]


```

1 results_with_ingestion_date_df = results_with_columns_df.withColumn("ingestion_date", current_timestamp())
> results_with_ingestion_date_df: pyspark.sql.dataframe.DataFrame = [result_id: integer, race_id: integer ... 17 more fields]
Command took 0.18 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute

Cell 9

1 #Step 3 - Drop the unwanted column
Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute

Cell 10

1 from pyspark.sql.functions import col
Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute

Cell 11

1 results_final_df = results_with_ingestion_date_df.drop(col("statusId"))
> results_final_df: pyspark.sql.dataframe.DataFrame = [result_id: integer, race_id: integer ... 16 more fields]
Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute
  
```

```

Cell 12

1 results_deduped_df = results_final_df.dropDuplicates(['race_id', 'driver_id'])
> results_deduped_df: pyspark.sql.dataframe.DataFrame = [result_id: integer, race_id: integer ... 16 more fields]
Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute

Cell 13

1 results_deduped_df.show(truncate=False)

(2) Spark Jobs
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|result_id|race_id|driver_id|constructor_id|number|grid|position|position_text|position_order|points|laps|time      |milliseconds|fastest_lap|rank|fastest_lap_time|fastest_lap_speed|ingestion_date|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|7573     |1      |1        |1              |18    |18    |null      |0              |20             |0.0  |58    |\N        |null        |39         |13    |1:29.020       |214.455         |2024-04-23 04:09:58.81|
|7563     |1      |2        |2              |6      |9      |10         |10             |10             |0.0  |58    |+7.085    |5662869     |48         |5         |1:28.283       |216.245         |2024-04-23 04:09:58.81|
  
```

The screenshot shows a Databricks notebook titled "result processed" in Python. The top toolbar includes buttons for "Run all", "compute", "Share", and "Publish". The notebook content is divided into three sections:

- Cell 14:** Displays a table with 11 columns. The first four columns contain numerical data, and the remaining seven columns contain date-time strings in ISO format.
- Cell 15:** Contains a Python code snippet: `#Step 4 - Write to output to processed container in parquet format`. Below the code, it shows the execution time: "Command took 0.00 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute".
- Cell 16:** Contains a Python code snippet: `results_deduped_df.write.parquet("dbfs:/FileStore/Formu1al/processed/f1_processed.result")`. Below the code, it shows the execution time: "Command took 15.37 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:38:15 AM on compute".

The bottom of the notebook shows a Spark job status bar with the message: "AnalysisException: Path dbfs:/FileStore/Formu1al/processed/f1_processed.result already exists." Below this, there are instructions: "[Shift+Enter] to run and move to next cell" and "[Esc H] to see all keyboard shortcuts".

• REQUIREMENT 2: Data Transformations

Step 1: Read all the parquet files from DBFS file store using the spark.read.parquet command

The screenshot shows a Databricks notebook interface. The notebook is titled "transformation_overall_results". The code in the first cell is:

```
1 #Read all the data as required
```

The second cell contains the following code:

```
1 drivers_df = spark.read.parquet("dbfs:/FileStore/Formula1/processed/fl_processed.drivers")\
2 .withColumnRenamed("number", "driver_number") \
3 .withColumnRenamed("name", "driver_name") \
4 .withColumnRenamed("nationality", "driver_nationality")
```

The output of the second cell shows the Spark DataFrame schema: `drivers_df: pyspark.sql.dataframe.DataFrame = [driver_id: integer, driver_ref: string ... 6 more fields]`. The third cell contains the command `drivers_df.show(truncate=False)`, which is partially visible.

The screenshot shows the same Databricks notebook, but the third cell's output is now visible. The command `drivers_df.show(truncate=False)` has been executed, displaying a table of driver information.

driver_id	driver_ref	driver_number	code	driver_name	dob	driver_nationality	ingestion_date
1	hamilton	44	HAM	Lewis Hamilton	1985-01-07	British	2024-04-22 07:05:25.521
2	heidfeld	null	HEI	Nick Heidfeld	1977-05-10	German	2024-04-22 07:05:25.521
3	rosberg	6	ROS	Nico Rosberg	1985-06-27	German	2024-04-22 07:05:25.521
4	alonso	14	ALO	Fernando Alonso	1981-07-29	Spanish	2024-04-22 07:05:25.521
5	kovalainen	null	KOV	Heikki Kovalainen	1981-10-19	Finnish	2024-04-22 07:05:25.521
6	nakajima	null	NAK	Kazuki Nakajima	1985-01-11	Japanese	2024-04-22 07:05:25.521
7	bourdais	null	BOU	Sébastien Bourdais	1979-02-28	French	2024-04-22 07:05:25.521
8	raikkonen	7	RAI	Kimi Räikkönen	1979-10-17	Finnish	2024-04-22 07:05:25.521
9	kubica	88	KUB	Robert Kubica	1984-12-07	Polish	2024-04-22 07:05:25.521
10	glock	null	GLO	Timo Glock	1982-03-18	German	2024-04-22 07:05:25.521
11	sato	null	SAT	Takuma Sato	1977-01-28	Japanese	2024-04-22 07:05:25.521
12	piquet_jr	null	PIQ	Nelson Piquet Jr.	1985-07-25	Brazilian	2024-04-22 07:05:25.521
13	massa	19	MAS	Felipe Massa	1981-04-25	Brazilian	2024-04-22 07:05:25.521
14	coulthard	null	COU	David Coulthard	1971-03-27	British	2024-04-22 07:05:25.521
15	trulli	null	TRU	Jarno Trulli	1974-07-13	Italian	2024-04-22 07:05:25.521
16	sutil	99	SUT	Adrian Sutil	1983-01-11	German	2024-04-22 07:05:25.521

community.cloud.databricks.com/?o=6796678178271343#notebook/489064752877510/command/489064752877515

databricks

transformation_overall_results Python ☆

File Edit View Run Help Last edit was 2 minutes ago New cell UI: OFF

Run all compute Share Publish

Last command failed

Cell 4

```
1 constructors_df = spark.read.parquet("dbfs:/FileStore/Formu1a1/processed/f1_processed.constructors") \
2 .withColumnRenamed("name", "team")
```

▶ (1) Spark Jobs

constructors_df: pyspark.sql.dataframe.DataFrame = [constructor_id: integer, constructor_ref: string ... 3 more fields]

Command took 1.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:44:01 AM on compute

Cell 5

```
1 constructors_df.show(truncate=False)
```

▶ (1) Spark Jobs

constructor_id	constructor_ref	team	nationality	ingestion_date
1	mclaren	McLaren	British	2024-04-18 17:31:45.863
2	bmw_sauber	BMW Sauber	German	2024-04-18 17:31:45.863
3	williams	Williams	British	2024-04-18 17:31:45.863
4	renault	Renault	French	2024-04-18 17:31:45.863
5	toro_rosso	Toro Rosso	Italian	2024-04-18 17:31:45.863

9:45 AM 4/23/2024

community.cloud.databricks.com/?o=6796678178271343#notebook/489064752877510/command/489064752877518

databricks

transformation_overall_results Python ☆

File Edit View Run Help Last edit was 3 minutes ago New cell UI: OFF

Run all compute Share Publish

Last command failed

Cell 6

```
1 circuits_df = spark.read.parquet("dbfs:/FileStore/Formu1a1/processed/circuits") \
2 .withColumnRenamed("location", "circuit_location")
```

▶ (1) Spark Jobs

circuits_df: pyspark.sql.dataframe.DataFrame = [circuit_id: string, circuit_ref: string ... 7 more fields]

Command took 0.89 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:44:41 AM on compute

Cell 7

```
1 circuits_df.show(truncate=False)
```

▶ (1) Spark Jobs

circuit_id	circuit_ref	name	circuit_location	country	latitude	longitude	altitude	ingestion_date
1	albert_park	Albert Park Grand Prix Circuit	Melbourne	Australia	-37.8497	144.968	10	2024-04-18 11:08:04.236
2	sebang	Sepang International Circuit	Kuala Lumpur	Malaysia	2.76083	101.738	18	2024-04-18 11:08:04.236
3	bahrain	Bahrain International Circuit	Sakhir	Bahrain	26.0325	50.5106	7	2024-04-18 11:08:04.236
4	catalunya	Circuit de Barcelona-Catalunya	Montmeló	Spain	41.57	2.26111	109	2024-04-18 11:08:04.236
5	istanbul	Istanbul Park	Istanbul	Turkey	40.9527	29.405	130	2024-04-18 11:08:04.236
6	monaco	Circuit de Monaco	Monte-Carlo	Monaco	43.7347	7.42055	7	2024-04-18 11:08:04.236
7	villeneuve	Circuit Gilles Villeneuve	Montreal	Canada	45.5	-73.5228	13	2024-04-18 11:08:04.236
8	laguna_auca	Circuit de la Laguna Aca	Caia	France	45.843	12.3525	128	2024-04-18 11:08:04.236

9:45 AM 4/23/2024

transformation_overall_results Python

```

1 races_df = spark.read.parquet("dbfs:/FileStore/Formula1/processed/fl_processed_races") \
2   .withColumnRenamed("name", "race_name") \
3   .withColumnRenamed("race_timestamp", "race_date")

```

(1) Spark Jobs

Command took 1.20 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:44:01 AM on compute

```

1 races_df.show(truncate=False)

```

(1) Spark Jobs

race_id	race_year	round	circuit_id	race_name	ingestion_date	race_date
1	2009	1	1	Australian Grand Prix	2024-04-23 04:06:31.534	2009-03-29 06:00:00
2	2009	2	2	Malaysian Grand Prix	2024-04-23 04:06:31.534	2009-04-05 09:00:00
3	2009	3	17	Chinese Grand Prix	2024-04-23 04:06:31.534	2009-04-19 07:00:00
4	2009	4	3	Bahrain Grand Prix	2024-04-23 04:06:31.534	2009-04-26 12:00:00
5	2009	5	4	Spanish Grand Prix	2024-04-23 04:06:31.534	2009-05-10 12:00:00

transformation_overall_results Python

```

1 results_df = spark.read.parquet("dbfs:/FileStore/Formula1/processed/fl_processed.result") \
2   .withColumnRenamed("time", "race_time") \
3   .withColumnRenamed("race_id", "result_race_id") \
4   .withColumnRenamed("file_date", "result_file_date")

```

(1) Spark Jobs

Command took 0.99 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:44:01 AM on compute

```

1 results_df.show(truncate=False)

```

(1) Spark Jobs

result_id	result_race_id	driver_id	constructor_id	number	grid	position	position_text	position_order	points	laps	race_time	milliseconds	fastest_lap	rank	fastest_lap_time	fastest_lap_speed	ingestion_date
7572	1	5	1	2	12	null	R	19	0.0	0	\N	null	null				2024-04-22 07:21:10.804
null	\N																

Step 2: Join the required tables for reporting and analysis requirements with only selected columns.

The screenshot shows a Databricks notebook titled "transformation_overall_results". The code in Cell 13 joins the `circuits_df` and `races_df` DataFrames. The output in Cell 14 shows a preview of the resulting `race_circuits_df` with 3 rows.

```

1 #Join circuits to races

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:44:41 AM on compute

Cell 13

1 race_circuits_df = races_df.join(circuits_df, races_df.circuit_id == circuits_df.circuit_id, "inner") \
2 .select(races_df.race_id, races_df.race_year, races_df.race_name, races_df.race_date, circuits_df.circuit_location)
3

race_circuits_df: pyspark.sql.dataframe.DataFrame = [race_id: integer, race_year: integer ... 3 more fields]
Command took 0.19 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:44:41 AM on compute

Cell 14

1 race_circuits_df.show(truncate=False)

(2) Spark Jobs

+-----+-----+-----+-----+-----+
|race_id|race_year|race_name      |race_date      |circuit_location|
+-----+-----+-----+-----+-----+
|1      |2009     |Australian Grand Prix|2009-03-29 06:00:00|Melbourne      |
|2      |2009     |Malaysian Grand Prix|2009-04-05 09:00:00|Kuala Lumpur    |
|3      |2009     |Chinese Grand Prix  |2009-04-19 07:00:00|Shanghai       |
+-----+-----+-----+-----+-----+

```

The screenshot shows the next step in the Databricks notebook. Cell 16 joins `race_results_df` with `race_circuits_df`, `drivers_df`, and `constructors_df`. The output in Cell 17 shows a preview of the resulting `race_results_df` with 34 fields.

```

1 #Join results to all other dataframes

Command took 0.09 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:44:41 AM on compute

Cell 16

1 race_results_df = results_df.join(race_circuits_df, results_df.result_race_id == race_circuits_df.race_id) \
2 .join(drivers_df, results_df.driver_id == drivers_df.driver_id) \
3 .join(constructors_df, results_df.constructor_id == constructors_df.constructor_id)
4

race_results_df: pyspark.sql.dataframe.DataFrame = [result_id: integer, result_race_id: integer ... 34 more fields]
Command took 0.20 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:44:41 AM on compute

Cell 17

1 race_results_df.show(truncate=False)

(5) Spark Jobs

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|result_id|result_race_id|driver_id|constructor_id|number|grid|position|position_text|position_order|points|laps|race_time |milliseconds|fastest_lap|rank|fastest_lap_time|fastest_lap_speed|ingestion_date|race_id|race_year|race_name      |race_date      |circuit_location|driver_id|driver_ref |driver_number|code|driver_name      |dob      |driver_nationality|ingestion_date|constructor_id|constructor_ref|tea|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

The screenshot shows a Databricks notebook interface with a Python cell. The cell contains the command `race_results_df.show(truncate=False)`. The output displays a table of race results with columns: `result_id`, `result_race_id`, `driver_id`, `constructor_id`, `number`, `grid`, `position`, `position_text`, `position_order`, `points`, `laps`, `race_time`, `milliseconds`, `fastest_lap`, `rank`, `fastest_lap_time`, `fastest_lap_speed`, `ingestion_date`, `race_id`, `race_year`, `race_name`, `race_date`, `circuit_location`, `driver_id`, `driver_ref`, `driver_number`, `code`, `driver_name`, `dob`, `driver_nationality`, `ingestion_date`, `constructor_id`, `constructor_ref`, `team`, `nationality`, `ingestion_date`.

The output shows data for the Australian Grand Prix 2009-03-29 06:00:00 Melbourne. The table is truncated, showing only the first few rows of data.

The screenshot shows a Databricks notebook interface with a Python cell. The cell contains the command `final_df.show(truncate=False)`. The output displays a table of race results with columns: `race_id`, `race_year`, `race_name`, `race_date`, `circuit_location`, `driver_name`, `driver_number`, `driver_nationality`, `team`, `grid`, `fastest_lap`, `race_time`, `points`, `position`.

The output shows data for the Australian Grand Prix 2009-03-29 06:00:00 Melbourne. The table is truncated, showing only the first few rows of data.

The screenshot shows a Databricks notebook titled 'transformation_overall_results'. The code cell contains the command `final_df.show(truncate=False)`, which displays a table of race results. The table has columns for race_id, race_year, race_name, race_date, circuit_location, driver_name, driver_number, driver_nationality, team, and grid. The data shows results for the Australian Grand Prix in 2009 and 2024.

race_id	race_year	race_name	race_date	circuit_location	driver_name	driver_number	driver_nationality	team	grid
1	2009	Australian Grand Prix	2009-03-29 06:00:00	Melbourne	Sébastien Buemi	null	Swiss	Toro Rosso	13
34	2024	Australian Grand Prix	2024-04-23 04:14:56.935	Melbourne	Rubens Barrichello	null	Brazilian	Brawn	2
1	2009	Australian Grand Prix	2009-03-29 06:00:00	Melbourne	Sebastian Vettel	5	German	Red Bull	3
43	2024	Australian Grand Prix	2024-04-23 04:14:56.935	Melbourne	Adrian Sutil	99	German	Force India	16
1	2009	Australian Grand Prix	2009-03-29 06:00:00	Melbourne	Robert Kubica	88	Polish	BMW Sauber	4
36	2024	Australian Grand Prix	2024-04-23 04:14:56.935	Melbourne	Kimi Räikkönen	7	Finnish	Ferrari	7
1	2009	Australian Grand Prix	2009-03-29 06:00:00	Melbourne	Sébastien Bourdais	null	French	Toro Rosso	17

Step 3: Store the joined tables in columnar format (i.e. parquet files)

The screenshot shows a Databricks notebook with the following code cells:

```

1 dbutils.fs.mkdirs("dbfs:/FileStore/Formula1/trans_before_result")
Out[49]: True

1 final_df.write.parquet("dbfs:/FileStore/Formula1/trans_before_result/file_for_standing")

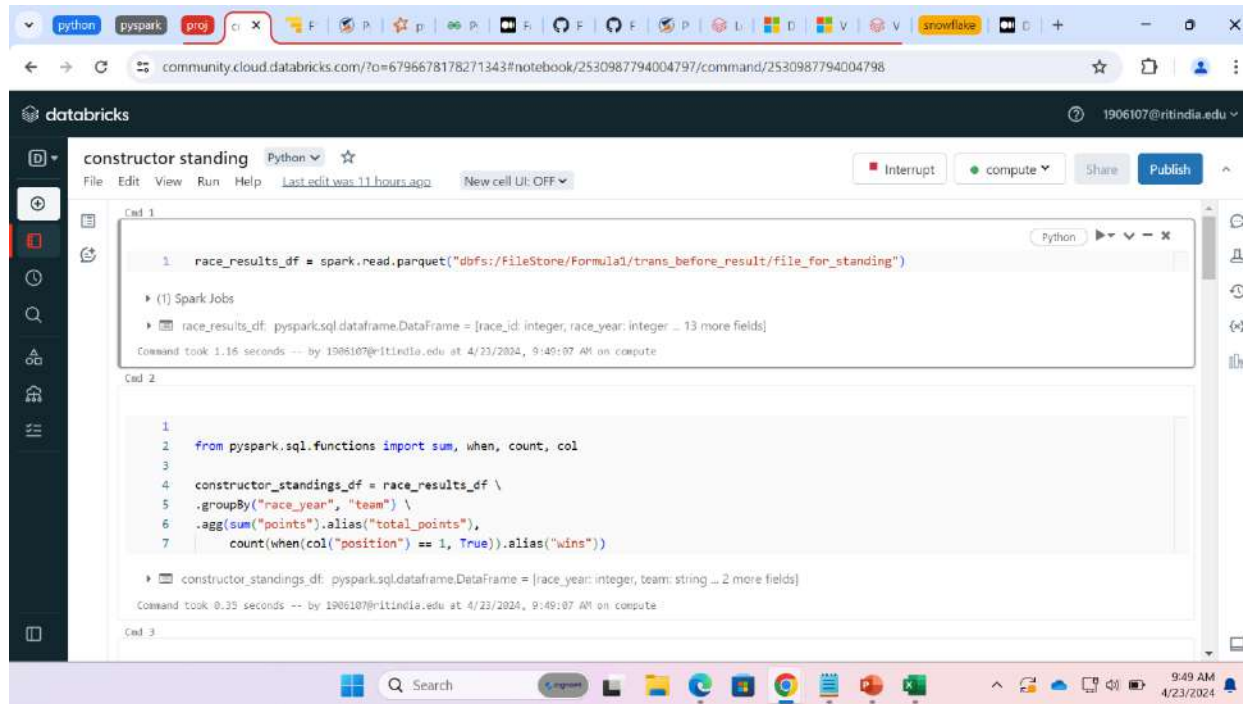
```

The third cell shows an error message: `AnalysisException: Path dbfs:/FileStore/Formula1/trans_before_result/file_for_standing already exists.`

REQUIREMENT 3 & 4:

Reporting, Analysis and Visualization

1. Constructor standings and dominant team.



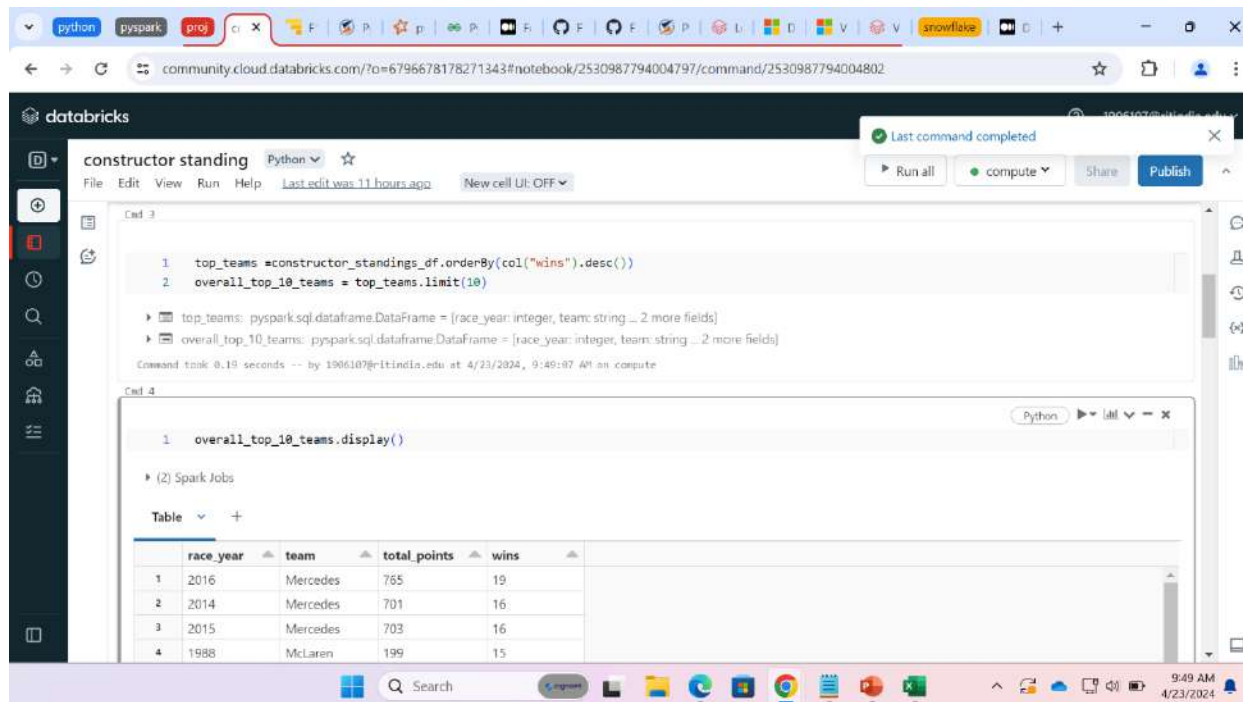
The screenshot shows a Databricks notebook titled "constructor standing". The first cell (Cell 1) contains the following Python code:

```
1 race_results_df = spark.read.parquet("dbfs:/FileStore/Formula1/trans_before_result/file_for_standing")
```

The output for Cell 1 shows a Spark job completed successfully, resulting in a DataFrame with 13 fields: race_id, race_year, and 11 more fields. The second cell (Cell 2) contains the following Python code:

```
1
2 from pyspark.sql.functions import sum, when, count, col
3
4 constructor_standings_df = race_results_df \
5     .groupBy("race_year", "team") \
6     .agg(sum("points").alias("total_points"),
7         count(when(col("position") == 1, True)).alias("wins"))
```

The output for Cell 2 shows a Spark job completed successfully, resulting in a DataFrame with 2 fields: race_year and team.



The screenshot shows the continuation of the Databricks notebook. The third cell (Cell 3) contains the following Python code:

```
1 top_teams = constructor_standings_df.orderBy(col("wins").desc())
2 overall_top_10_teams = top_teams.limit(10)
```

The output for Cell 3 shows a Spark job completed successfully, resulting in a DataFrame with 2 fields: race_year and team. The fourth cell (Cell 4) contains the following Python code:

```
1 overall_top_10_teams.display()
```

The output for Cell 4 shows a table with 5 columns: race_year, team, total_points, and wins. The table displays the top 10 teams by wins.

	race_year	team	total_points	wins
1	2016	Mercedes	765	19
2	2014	Mercedes	701	16
3	2015	Mercedes	703	16
4	1988	McLaren	199	15

The screenshot shows a Databricks notebook titled "constructor standing" with Python code in two cells. Cell 6 contains the logic to calculate constructor rankings based on total points and wins. Cell 7 displays the resulting DataFrame.

```

1 #que 2 constructor standing

Command took 0.06 seconds -- by 1906107@nitindia.edu at 4/23/2024, 9:49:07 AM on compute

Cell 6

1 from pyspark.sql.window import Window
2 from pyspark.sql.functions import desc, rank, asc
3
4 constructor_rank_spec = Window.partitionBy("race_year").orderBy(desc("total_points"), desc("wins"))
5 final_df = constructor_standings_df.withColumn("rank", rank().over(constructor_rank_spec))

final_df: pyspark.sql.dataframe.DataFrame = [race_year: integer, team: string ... 3 more fields]
Command took 0.29 seconds -- by 1906107@nitindia.edu at 4/23/2024, 9:49:07 AM on compute

Cell 7

1 final_df.display()

(3) Spark Jobs

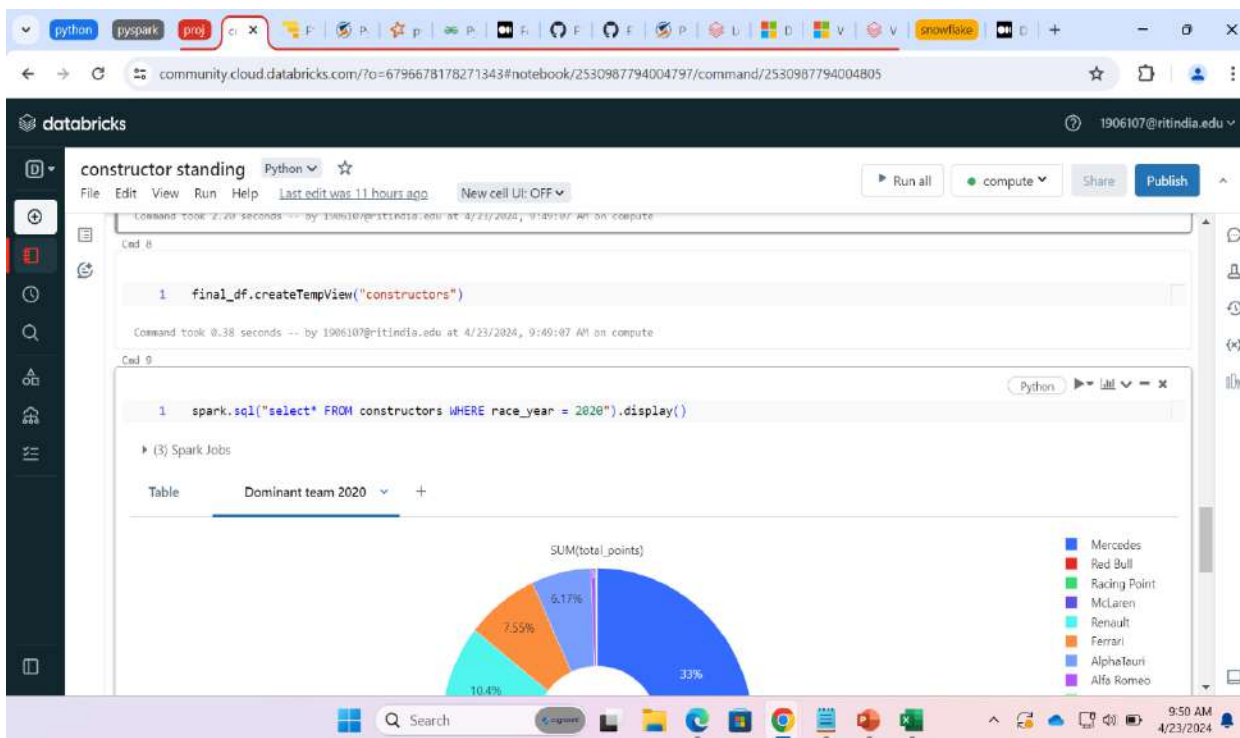
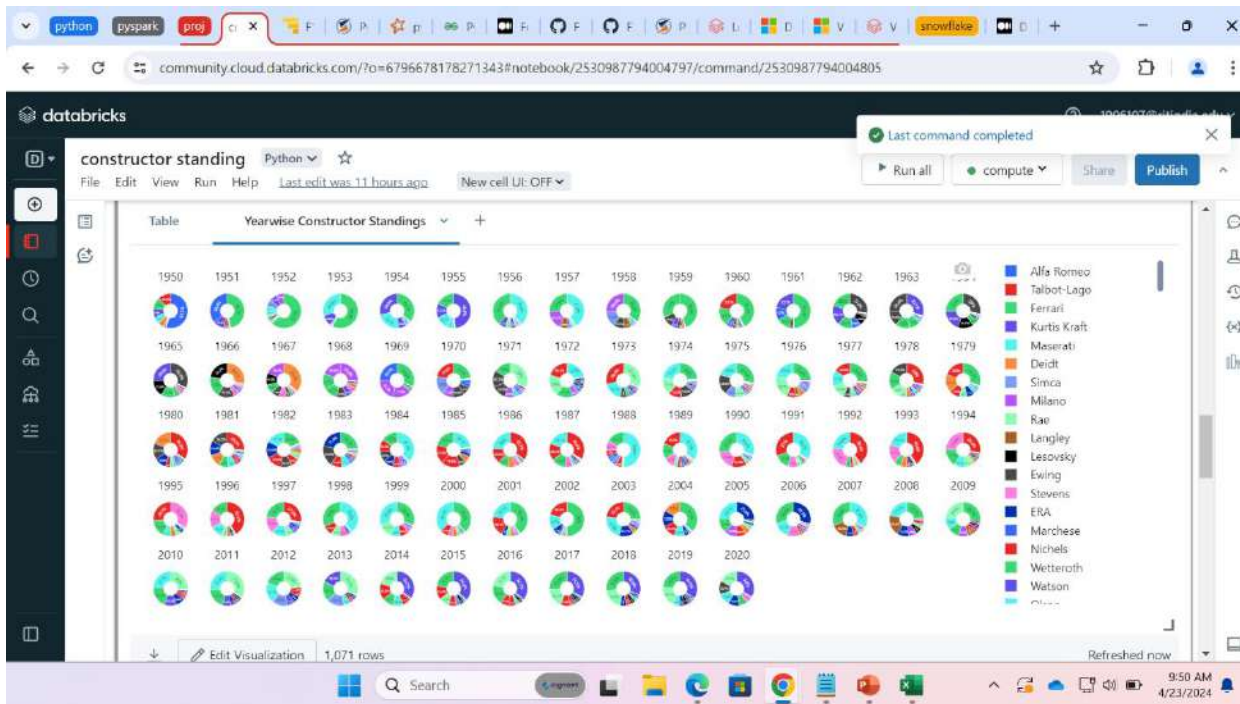
Table Yearwise Constructor Standings

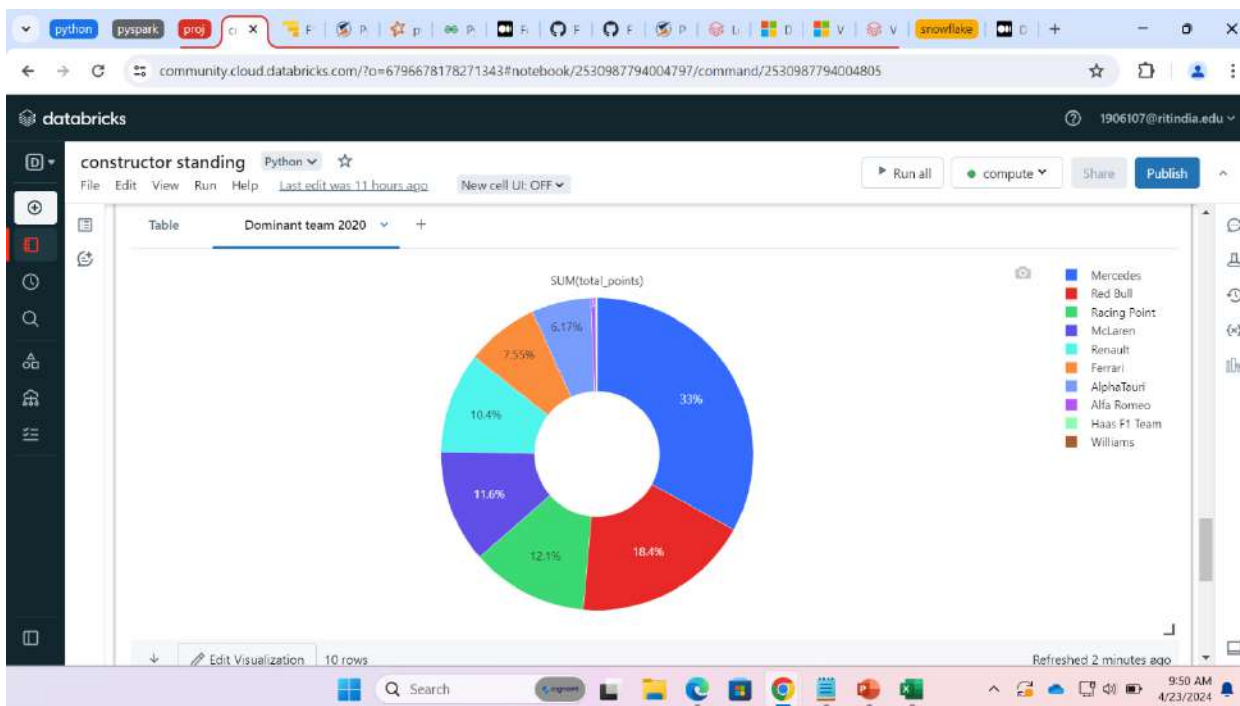
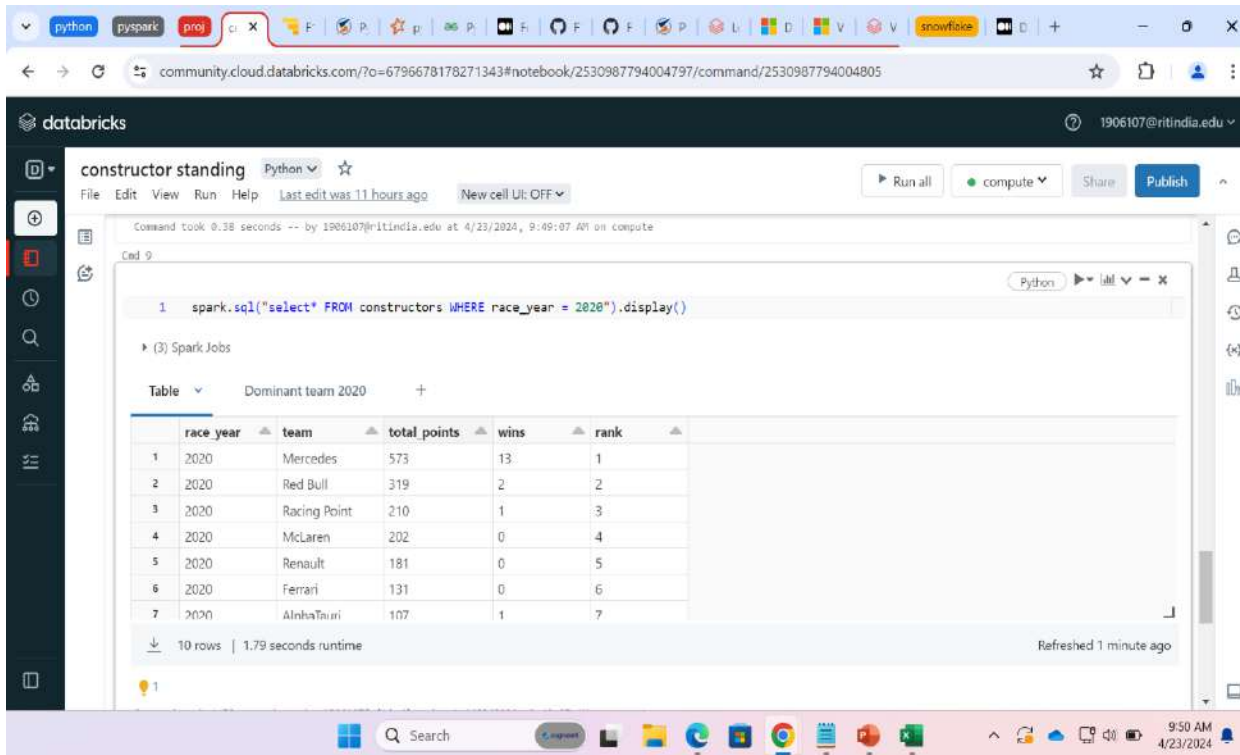
```

This screenshot shows the same Databricks notebook, but now displaying the output of the query in Cell 7. The output is a table titled "Yearwise Constructor Standings" with 7 rows of data for the year 1950. The table includes columns for race_year, team, total_points, wins, and rank.

	race_year	team	total_points	wins	rank
1	1950	Alfa Romeo	89	6	1
2	1950	Talbot-Lago	20	0	2
3	1950	Ferrari	18	0	3
4	1950	Kurtis Kraft	13	1	4
5	1950	Maserati	11	0	5
6	1950	Deidt	10	0	6
7	1950	Simca	3	0	7

1,071 rows | 2.20 seconds runtime
Command took 2.20 seconds -- by 1906107@nitindia.edu at 4/23/2024, 9:49:07 AM on compute





2. Driver standing and dominant driver.

The screenshot shows a Databricks notebook titled "driver standing". The first cell contains the code to read a Parquet file from the cloud storage:

```
1 race_results_df = spark.read.parquet("dbfs:/FileStore/Formula1/trans_before_result/file_for_standing")
```

The output shows the Spark job completed successfully, and the DataFrame schema is displayed: (race_id: integer, race_year: integer ... 13 more fields). The second cell contains the code to display the first 10 rows of the DataFrame:

```
1 race_results_df.show(truncate=False)
```

The output shows a table with 10 rows of race results, including columns for race_id, race_year, race_name, driver_name, position, and points.

race_id	race_year	race_name	driver_name	position	points
51	2009	Australian Grand Prix	Jenson Button	22	1
17	2009	Australian Grand Prix	Mark Webber	1	8
30	2009	Australian Grand Prix	Jarno Trulli	1	20
50	2009	Australian Grand Prix	Felipe Massa	19	6
30	2009	Australian Grand Prix	Nelson Piquet Jr.	1	14
17	2009	Australian Grand Prix	Timo Glock	1	19

The screenshot shows the same Databricks notebook with additional code to calculate driver standings. The third cell contains the code to calculate the total points for each driver and the number of wins:

```
1 from pyspark.sql.functions import sum, when, count, col
2
3 driver_standings_df = race_results_df \
4     .groupBy("race_year", "driver_name", "driver_nationality") \
5     .agg(sum("points").alias("total_points"),
6         count(when(col("position") == 1, True)).alias("wins"))
```

The output shows the Spark job completed successfully, and the DataFrame schema is displayed: (race_year: integer, driver_name: string ... 3 more fields). The fourth cell contains the code to display the top 10 drivers by total points:

```
1 from pyspark.sql.functions import desc, asc
2 driver_standings_df.orderBy(driver_standings_df.wins.desc()).take(10)
```

The output shows the top 10 drivers by total points, including columns for race_year, driver_name, driver_nationality, total_points, and wins.

race_year	driver_name	driver_nationality	total_points	wins
2009	Jenson Button	British	1	1
2009	Mark Webber	Australian	8	1
2009	Jarno Trulli	Italian	20	1
2009	Felipe Massa	Brazilian	6	1
2009	Nelson Piquet Jr.	Brazilian	14	1
2009	Timo Glock	German	19	1

community.cloud.databricks.com/?o=6796678178271343#notebook/489064752877537/command/489064752877539

Run all cells in this notebook. 1906107@ritindia.edu

driver standing Python

```

1 from pyspark.sql.functions import desc, asc
2 driver_standings_df.orderBy(driver_standings_df.wins.desc()).take(10)

```

Out[5]: [Row(race_year=2004, driver_name='Michael Schumacher', driver_nationality='German', total_points=148.0, wins=13), Row(race_year=2013, driver_name='Sebastian Vettel', driver_nationality='German', total_points=397.0, wins=13), Row(race_year=2011, driver_name='Sebastian Vettel', driver_nationality='German', total_points=392.0, wins=11), Row(race_year=2019, driver_name='Lewis Hamilton', driver_nationality='British', total_points=413.0, wins=11), Row(race_year=2014, driver_name='Lewis Hamilton', driver_nationality='British', total_points=384.0, wins=11), Row(race_year=2002, driver_name='Michael Schumacher', driver_nationality='German', total_points=144.0, wins=11), Row(race_year=2018, driver_name='Lewis Hamilton', driver_nationality='British', total_points=408.0, wins=11), Row(race_year=2020, driver_name='Lewis Hamilton', driver_nationality='British', total_points=347.0, wins=11), Row(race_year=2015, driver_name='Lewis Hamilton', driver_nationality='British', total_points=381.0, wins=10), Row(race_year=2016, driver_name='Lewis Hamilton', driver_nationality='British', total_points=380.0, wins=10)]

Command took 2.48 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:51:58 AM on compute

```

1 top_driver = driver_standings_df.orderBy(col("wins").desc())
2 overll_top_10_drivers = top_driver.limit(10)

```

top_driver: pyspark.sql.dataframe.DataFrame = [race_year: integer, driver_name: string ... 3 more fields]
overll_top_10_drivers: pyspark.sql.dataframe.DataFrame = [race_year: integer, driver_name: string ... 3 more fields]

community.cloud.databricks.com/?o=6796678178271343#notebook/489064752877537/command/489064752877539

Run all cells in this notebook. 1906107@ritindia.edu

driver standing Python

```

1 top_driver = driver_standings_df.orderBy(col("wins").desc())
2 overll_top_10_drivers = top_driver.limit(10)

```

top_driver: pyspark.sql.dataframe.DataFrame = [race_year: integer, driver_name: string ... 3 more fields]
overll_top_10_drivers: pyspark.sql.dataframe.DataFrame = [race_year: integer, driver_name: string ... 3 more fields]

Command took 0.29 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:51:58 AM on compute

```

1 overll_top_10_drivers.display()

```

Table

	race_year	driver_name	driver_nationality	total_points	wins
1	2004	Michael Schumacher	German	148	13
2	2013	Sebastian Vettel	German	397	13
3	2011	Sebastian Vettel	German	392	11
4	2019	Lewis Hamilton	British	413	11
5	2014	Lewis Hamilton	British	384	11

The screenshot shows a Databricks notebook titled "driver standing" in Python. The code in Cell 9 defines a window function to calculate driver ranks based on total points and wins for each race year. Cell 10 displays the resulting DataFrame.

```

1 #que 2 driver standing

Command took 0.08 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:51:58 AM on compute

Cell 9

1 from pyspark.sql.window import Window
2 from pyspark.sql.functions import desc, rank, asc
3
4 driver_rank_spec = Window.partitionBy("race_year").orderBy(desc("total_points"), desc("wins"))
5 final_df = driver_standings_df.withColumn("rank", rank().over(driver_rank_spec))

final_df: pyspark.sql.dataframe.DataFrame = (race_year: integer, driver_name: string ... 4 more fields)
Command took 0.29 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:51:58 AM on compute

Cell 10

1 final_df.display()

(3) Spark Jobs

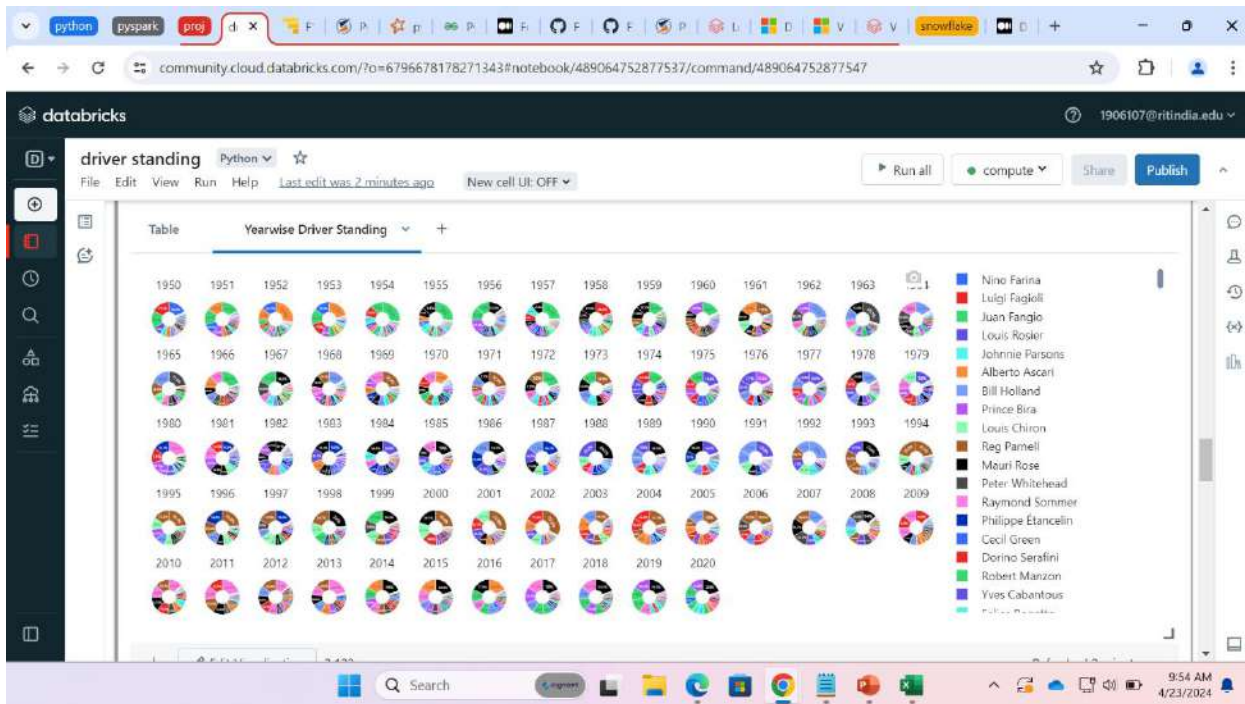
Table Yearwise Driver Standing +

```

The screenshot shows the same Databricks notebook, but now displaying the output of the code. The table "Yearwise Driver Standing" is shown with 7 rows of data for the year 1950. The table has columns for race_year, driver_name, driver_nationality, total_points, wins, and rank.

	race_year	driver_name	driver_nationality	total_points	wins	rank
1	1950	Nino Farina	Italian	30	3	1
2	1950	Luigi Fagioli	Italian	28	0	2
3	1950	Juan Fangio	Argentine	27	3	3
4	1950	Louis Rosier	French	13	0	4
5	1950	Johnnie Parsons	American	9	1	5
6	1950	Alberto Ascari	Italian	8	0	6
7	1950	Bill Holland	American	6	0	7

3,122 rows | 2.70 seconds runtime Refreshed 1 minute ago



```
1 final_df.createTempView("trial")
```

Command took 0.17 seconds -- by 1906107@ritindia.edu at 4/23/2024, 9:53:58 AM on compute

```
1 spark.sql("select* FROM trial WHERE race_year = 2020").show()
```

(4) Spark Jobs

race_year	driver_name	driver_nationality	total_points	wins	rank
2020	Lewis Hamilton	British	347.0	11	1
2020	Valtteri Bottas	Finnish	223.0	2	2
2020	Max Verstappen	Dutch	214.0	2	3
2020	Sergio Pérez	Mexican	125.0	1	4
2020	Daniel Ricciardo	Australian	119.0	0	5
2020	Carlos Sainz	Spanish	105.0	0	6
2020	Alexander Albon	Thai	105.0	0	6
2020	Charles Leclerc	Monegasque	98.0	0	8
2020	Lando Norris	British	97.0	0	9
2020	Pierre Gasly	French	75.0	1	10
2020	Lance Stroll	Canadian	75.0	0	11

