**🚀 Relearning PySpark with Databricks: A Quick Dive into Data Reading & Transformations!**

**📥 DATA READING**

🔹 **Reading JSON Files**  
Using .read.format('json') to load structured JSON data efficiently.

df\_json = spark.read.format('json')\

.option('inferSchema', True)\

.option('header', True)\

.option('multiLine', False)\

.load('/FileStore/tables/drivers.json')

df\_json.display()

🔹 **Exploring Files with dbutils**  
Handy to list files available in Databricks.

dbutils.fs.ls('/FileStore/tables/')

🔹 **Reading CSV Files with Inferred Schema**  
Automatically detects column types and reads data.

df = spark.read.format('csv')\

.option('inferSchema', True)\

.option('header', True)\

.load('/FileStore/tables/BigMart\_Sales.csv')

df.display()

🔹 **Printing Schema**  
Always useful to verify structure post-ingestion.

df.printSchema()

**🧱 SCHEMA DEFINITION METHODS**

🔹 **DDL Format**  
Schema defined as a DDL string (like SQL table structure).

my\_ddl\_schema = '''

Item\_Identifier STRING,

Item\_Weight STRING,

Item\_Fat\_Content STRING,

Item\_Visibility DOUBLE,

Item\_Type STRING,

Item\_MRP DOUBLE,

Outlet\_Identifier STRING,

Outlet\_Establishment\_Year INT,

Outlet\_Size STRING,

Outlet\_Location\_Type STRING,

Outlet\_Type STRING,

Item\_Outlet\_Sales DOUBLE

'''

df = spark.read.format('csv')\

.schema(my\_ddl\_schema)\

.option('header', True)\

.load('/FileStore/tables/BigMart\_Sales.csv')

df.display()

🔹 **StructType Schema**  
More explicit schema using PySpark's native data types.

my\_strct\_schema = StructType([

StructField('Item\_Identifier', StringType(), True),

StructField('Item\_Weight', StringType(), True),

StructField('Item\_Fat\_Content', StringType(), True),

StructField('Item\_Visibility', StringType(), True),

StructField('Item\_MRP', StringType(), True),

StructField('Outlet\_Identifier', StringType(), True),

StructField('Outlet\_Establishment\_Year', StringType(), True),

StructField('Outlet\_Size', StringType(), True),

StructField('Outlet\_Location\_Type', StringType(), True),

StructField('Outlet\_Type', StringType(), True),

StructField('Item\_Outlet\_Sales', StringType(), True)

])

df = spark.read.format('csv')\

.schema(my\_strct\_schema)\

.option('header', True)\

.load('/FileStore/tables/BigMart\_Sales.csv')

df.printSchema()

**🔄 TRANSFORMATIONS**

🔹 **SELECT Columns**  
Pick only specific columns from the dataset.

df.select(col('Item\_Identifier'), col('Item\_Weight'), col('Item\_Fat\_Content')).display()

🔹 **ALIAS Columns**  
Renaming a column on the fly.

df.select(col('Item\_Identifier').alias('Item\_ID')).display()

🔹 **FILTER Rows**  
Apply conditions to limit rows.

Scenario 1: Basic equality

df.filter(col('Item\_Fat\_Content') == 'Regular').display()

Scenario 2: AND condition

df.filter((col('Item\_Type') == 'Soft Drinks') & (col('Item\_Weight') < 10)).display()

Scenario 3: NULL check & value filter

df.filter((col('Outlet\_Size').isNull()) & (col('Outlet\_Location\_Type').isin('Tier 1','Tier 2'))).display()

🔹 **Renaming Columns**  
Use withColumnRenamed for permanent rename.

df.withColumnRenamed('Item\_Weight', 'Item\_Wt').display()

🔹 **Adding Columns**  
You can use withColumn to add or transform values.

Scenario 1: Add static and **calculated** columns

df = df.withColumn('flag', lit("new"))

df.withColumn('multiply', col('Item\_Weight') \* col('Item\_MRP')).display()

Scenario 2: String **replacement** using regexp\_replace

df = df.withColumn('Item\_Fat\_Content', regexp\_replace(col('Item\_Fat\_Content'), "Regular", "Reg"))\

.withColumn('Item\_Fat\_Content', regexp\_replace(col('Item\_Fat\_Content'), "Low Fat", "Lf"))

df.display()

🔹 **Type Casting**  
Change data types as needed for transformations or storage.

df = df.withColumn('Item\_Weight', col('Item\_Weight').cast(StringType()))

df.printSchema()

**🔄 SORTING**

🔹 **Scenario 1: Sort by Column in Descending Order**  
Sort the dataset by Item\_Weight from highest to lowest.

df.sort(col('Item\_Weight').desc()).display()

🔹 **Scenario 2: Sort by Column in Ascending Order**  
Sort the dataset by Item\_Visibility from lowest to highest.

df.sort(col('Item\_Visibility').asc()).display()

🔹 **Scenario 3: Sort by Multiple Columns (Descending)**  
Sort by both Item\_Weight and Item\_Visibility in descending order.

df.sort(['Item\_Weight', 'Item\_Visibility'], ascending = [0, 0]).display()

🔹 **Scenario 4: Sort by Multiple Columns (Mixed Order)**  
Sort Item\_Weight in descending and Item\_Visibility in ascending order.

df.sort(['Item\_Weight', 'Item\_Visibility'], ascending = [0, 1]).display()

**⚖️ LIMIT**

🔹 **Limiting Results**  
Use limit() to restrict the output to the top 10 rows.  
This is super useful when working with large datasets and just need a quick glance.

df.limit(10).display()

**🧹 DROP Columns**

🔹 **Scenario 1: Drop a Single Column**  
Remove Item\_Visibility from the dataset.

df.drop('Item\_Visibility').display()

🔹 **Scenario 2: Drop Multiple Columns**  
Remove both Item\_Visibility and Item\_Type from the dataset.

df.drop('Item\_Visibility', 'Item\_Type').display()

📌 **Next Steps:**  
I’ll be exploring more complex operations soon, like **aggregations** and **joins**. Stay tuned! 😊