databricks DataBricks_tutorial

(https://databricks.com)
PySpark Scenarios

1. Data Ingestion

Load the CSV file into a PySpark DataFrame. Infer schema automatically and check the data types of each column.

2. Data Cleaning

Handle missing values: Fill missing values in Item_Weight with the average weight. Fill missing values in Outlet_Size with the mode. Remove duplicate rows if any.

3. Data Transformation

Convert Outlet_Establishment_Year to Outlet_Age by subtracting it from the current year. Standardize Item_Fat_Content (since variations like 'low fat' and 'LF' exist).

4. Aggregation & Analysis

Compute total sales (Item_Outlet_Sales) per Outlet_Type. Find the most sold item category (Item_Type) based on total sales. Determine the top 5 items with the highest visibility.

5. Filtering and Queries

Retrieve records where Item_Visibility is greater than the average visibility. Fetch details of all items with Item_MRP greater than 200. Get all sales data for a specific Outlet_Identifier.

6. Joins & Grouping

If another dataset (e.g., Outlet_Details) exists, join it based on Outlet_Identifier. Group sales data by Outlet_Location_Type and calculate the average sales.

7. Machine Learning Preparation

Encode categorical variables (Outlet_Type, Item_Type, Outlet_Location_Type). Prepare a dataset for predicting Item_Outlet_Sales using features like Item_Weight, Item_MRP, and Outlet_Age. Would you like PySpark code for any of these scenarios?

Here are some complex PySpark scenarios based on the dataset:

1. Predictive Sales Analysis Using Window Functions

Scenario: Identify the top-selling item in each Outlet_Identifier using window functions.

Use Window Functions to rank Item_Outlet_Sales per outlet. Get the top 3 selling items per outlet. Compare sales across different Outlet_Type.

2. Advanced Feature Engineering for Sales Prediction

Scenario: Build new features that could improve sales prediction. Compute rolling average sales for each outlet using window functions. Create a sales-to-visibility ratio: Sales per unit of Visibility = Item_Outlet_Sales / Item_Visibility Generate Outlet Performance Score: (Total Sales per Outlet) / (Years in Business)

3. Market Basket Analysis - Association Rules

Scenario: Identify frequently co-purchased item categories.

Group transactions by Outlet_Identifier and find co-occurrence of Item_Type. Use FP-Growth algorithm in PySpark ML to identify association rules.

4. Anomaly Detection in Sales Data

Scenario: Detect unusual spikes or dips in sales for an outlet.

Use Z-score or IQR (Interquartile Range) to detect sales anomalies. Identify outlets with unusual sales drop (e.g., 30% drop in the last month). Flag items with an abnormally high Item_Visibility.

5. Time Series Analysis: Predicting Future Sales

Scenario: Use Lag features to predict future sales.

Create lagged sales features (Sales on Day -1, Day -7, etc.). Use exponential smoothing or ARIMA to forecast sales. Compare sales trends before and after an outlet was established.

6. Customer Segmentation Based on Outlet Performance

Scenario: Cluster outlets into high, medium, and low performers.

Use K-Means clustering to group outlets based on: Total Sales Outlet_Age Avg Item Price Assign labels: High-Performing, Medium, and Low-Performing. ###7. Recommendation System for Product Discounts Scenario: Suggest discounts for underperforming products.

Identify items with: Low Sales High MRP but low visibility Recommend discount strategies for items that haven't been selling well.

JSON READING

3

```
json_df = spark.read.format('json')\
  .option('inferschema', True)\
  .option('header', True)\
  .option('multiline', False)\
    .load('/FileStore/tables/drivers.json')
```

json_df: pyspark.sql.dataframe.DataFrame = [code: string, dob: string ... 6 more fields]

4

```
json_df.show()
```

Data Reading

6

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

7

```
df = spark.read.format('csv')\
    .option('InferSchema', True)\
        .option('header', True)\
            .load('/FileStore/tables/BigMart_Sales.csv')
```

df: pyspark.sql.dataframe.DataFrame = [Item_Identifier: string, Item_Weight: double ... 10 more fields]

8

```
df.printSchema()
```

```
root
 |-- Item_Identifier: string (nullable = true)
 |-- Item_Weight: double (nullable = true)
 |-- Item_Fat_Content: string (nullable = true)
 |-- Item_Visibility: double (nullable = true)
 |-- Item_Type: string (nullable = true)
 |-- Item_MRP: double (nullable = true)
 |-- Outlet_Identifier: string (nullable = true)
 |-- Outlet_Establishment_Year: integer (nullable = true)
 |-- Outlet_Size: string (nullable = true)
 |-- Outlet_Location_Type: string (nullable = true)
 |-- Outlet_Type: string (nullable = true)
 |-- Item_Outlet_Sales: double (nullable = true)
```

Schema Defenition

10

```
my_ddl_schema='''
                Item_Identifier STRING,
                ITEM_wieght STRING,
                Item_Fat_Content STRING,
                Item_Visibility STRING,
                Item_Type STRING,
                Item_MRP DOUBLE,
                Outlet_Identifier STRING,
                Outlet_Establishment_Year STRING,
                Outlet_Size STRING,
                Outlet_Location_Type STRING,
                Outlet_Type STRING,
                Item_Outlet_Sales STRING
                '''
```

11

```
df = spark.read.format('csv')\     .schema(my_ddl_schema)\
        .option('head ...
```

12

```
df_withdefined_schema.display()
```

13

```
df_withdefined_schema.printSchema()
```

StructType Schema

15

```
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

Command skipped

16

```
strcttype_schema = StructType([
    StructField('Item_Identifier', StringType(), True),
    StructField('Item_wieght', DoubleType(), True),
    StructField('Item_Fat_Content', StringType(), True),
    StructField('Item_Visibility', StringType(), True),
    StructField('Item_Type', 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)
])
```

Command skipped

17

```
df = spark.read.format('csv')\
            .schema(strcttype_schema)\
                .option('header', True)\
                    .load('/FileStore/tables/BigMart_Sales.csv')
```

Command skipped

18

```
strct_type_schema_df.display()
```

Command skipped

19

```
df.printSchema()
```

Command skipped

SELECT

21

```
#Selecting with Comma Sperated Columns
df.select('Item_Identifier', 'Item_Weight',
'Item_Fat_Content').display()
```

Command skipped

22

```
#Selecting with LIst OF Columns
df.select(['Item_Identifier', 'Item_Weight',
'Item_Fat_Content']).display(5)
```

Command skipped

23

```
#Select with Col()
# Alias is Used to Rename teh Column, This is possibel with Col
Object

df.select(col('Item_Identifier').alias('Item_ID'),
col('Item_Weight'), col('Item_fat_content')).display()
```

Command skipped

FILTER/WHERE

Scenario-1 : Filter Data With fat Content = Regular

26

```
df.filter(col('Item_Fat_Content')=='Regular')\

.select(col('Item_Identifier'),col('Item_Fat_Content'),col('Item_Type
'),col('Item_MRP')\
                    .alias('MRP'))\
                        .display()
```

```
Command skipped
```

Scenario-2 : Display records with Item wight <10 & Type is Soft Drinks

28

```
df.filter(((col('Item_Weight') < 10) & (col('Item_Type')=='Soft
Drinks'))).display()
```

```
Command skipped
```

Filter the Data With Outlet_Size = null & Location in Tier 1 or Tier2

30

```
df.filter((col('Outlet_Location_Type').isin('Tier 1','Tier 2'))&(c
ol('Outlet_Siz ...
```

WithColumnRenamed

32

```
df.withColumnRenamed('Outlet_Establishment_year','Establishment_year'
).display()
```

```
Command skipped
```

WithColumn

Scenario 1 : Add a new Column with Value based on calculation from two Columns

35

```
df.withColumn('flag', lit('True')).display()
```

36

```
df.withColumn('Total Price', col('Item_Weight') * col('Item_MR
P')).display()
```

replacing teh Value on teh value of Existing column

38

```
df.withColumn('Item_Fat_Content', regexp_replace('Item_Fat_Conten
t','Regular', ' ...
```

TypeCasting

40

```
df = df.withColumn('Item_Weight',
col('Item_Weight').cast(StringType()))
```

```
Command skipped
```

41

```
df.printSchema()
```

```
Command skipped
```

42

```
df.filter(col('Item_Weight').isNotNull())\
.sort(col('Item_Weight').asc())\
    .sort(col('Item_Visibility').asc())\
    .display()
```

```
Command skipped
```

Sceanrio 2: Sorting based on multiple Column

Item Weight Ascending & Item Visibility Descending

45

```
df.filter(col('Item_Weight').isNotNull())\     .filter((col('Item_
Visibility').i ...
```

Limit- Just to Limit teh number of Records as we do in SQL

47

```
df.limit(10).display()
# Just to get First 10 Records
```

```
Command skipped
```

Drop: Drop the Column

Sceanrio 1: Drop 1 Column Item_Visibility

Scenario 2: Drop Multiple Columns

50

```
#Scenario 1: Drop 1 Column Item_Visibility df.drop('Item_Visibilit
y').display()
```

51

```
#Scenario 2: Drop two Columns Item_Visibility & Item_Type
df.drop('Item_Visibility', 'Item_Type', 'Outlet_Size').display()
```

```
Command skipped
```

Drop Duplicates

53

```
df.dropDuplicates().display()
```

```
Command skipped
```

Scenario 2: Drop Duplicates in One Column

55

```
df.dropDuplicates(subset=['Item_Type', 'Item_Fat_Content']).display()
```

```
Command skipped
```

Union- It Combines teh Data

UnionBYName- It Combines the Data by name

57

```
data_set1 = [('Saurabh', 34),('Smriti', 31)]

schema = 'name STRING','age INT'

data_set2 = [('Lavit', 3),('Shambhu', 72)]

df1 = spark.createDataFrame(data_set1, schema)
df2 = spark.createDataFrame(data_set2, schema)

df1.display()
df2.display()
```

```
Command skipped
```

58

```
(df1.union(df2)).sort(col('age INT').asc()).display()
```

```
Command skipped
```

UnionByName()- It will combine teh Dataframes based on Column Name

60

```
schema = 'age INT', 'name STRING'
data_set1 = [(34, 'Saurabh'),(31, 'Smriti')]


df1 = spark.createDataFrame(data_set1, schema=schema)

df1.display()

df1.union(df2).display()

df2.unionByName(df1).display()
```

```
Command skipped
```

String Functions

INITCAP()

UPPER()

LOWER()

62

```
df.select(initcap(col('Item_Type')).alias('Init_Cap_Item_Type')).display()
```

63

```
df.select(upper(col('Item_Type')).alias('Upper_Item_Type')).limit(5).display()
```

```
Command skipped
```

64

```
df.select(lower(col('Item_type')).alias('Lower_Item_type')).limit
(10).display()
```

Date Functions

Current_Date()

Date_Add()

Date_Sub()

66

```
df = df.withColumn('curr_date', current_date())
df.display()
```

```
Command skipped
```

67

```
df = df.withColumn('Week_After', date_add('curr_date',7)) df.displ
ay()
```

68

```
df = df.withColumn('Week_Before', date_sub('curr_date', 7)) df.lim
it(10).display ...
```

Date Subtraction using date_add()

70

```
df = df.withColumn('Two_Weeks_Before', date_add('Week_Before',-7))
df.display()
```

Date Diff

72

```
df = df.withColumn('Date_Difference_1', datediff(col('Week_After'),
col('curr_date')))
df.display()
```

```
Command skipped
```

Date_Format

74

```
df =df.withColumn('Week_Before', date_format('Week_Before','dd-MM-
yyyy'))
df.display()
```

```
Command skipped
```

Handling Nulls

Dropping Null

- **Dropping NA with All as an option deletes all teh rows having all columns as Null `This` happens to be rare event where all teh columns will have null values**

- **Dropping null with 'any' it drops all the rows with any collumn having null --> It imposes data loss**
- **Dropping Null values for a specific Column by supplying the Subset of teh Columns.**
- **Filling null values.**

78

```
df.dropna('all').display()
```

Command skipped

79

```
df.dropna('any').display()
```

Command skipped

80

```
df.dropna(subset=['Outlet_Size']).display()
```

Command skipped

81

```
df.fillna('Not Available', subset=['Outlet_Size']).display()
```

Command skipped

----------- Split & Indexing --------------

Split splits the String based on some Delimiter like Space, Comma Et or any sptring you want your string to be splitted on

84

```
df = df.withColumn('Splitted_Column',split('Outlet_Type', ' '))

df.display()
```

Command skipped

------------------------EXPLODE ------------------------

86

```
df_exploded = df.withColumn('Splitted_Column',
explode('Splitted_Column'))
df_exploded.display()
```

Command skipped

------------------------ Array_Contains------------------

88

```
df = df.withColumns({'Outlet_type_supermarket': array_contains('Sp
litted_Column' ...
```

-------------------GROUP_BY -------------------------

90

```
df.groupBy('Item_type').agg(sum('Item_MRP')).display()
```

INstead of Finding Sum What is the Average MRP grouped by Type

92

```
df.groupBy('Item_Type').agg(avg('Item_MRP').alias('Item_Wise_Average_MRP')).display()
```

```
Command skipped
```

- ## Gruop By multiple Columns

94

```
df.groupBy('Item_Type',
'Outlet_Size').agg(sum('Item_MRP').alias('Grouped_MRP'))\
    .sort('Item_Type','Outlet_Size').display()
```

```
Command skipped
```

- ## Group by Item_Type & Outlet Size and Calculate the Total & the Average MRP

96

```
df.groupBy('Item_Type',
'Outlet_Size').agg(sum('Item_MRP').alias('Total_MRP'),
avg('Item_MRP').alias('Average_MRP'))\
    .sort('Item_Type', 'Outlet_Size').display()
```

```
Command skipped
```

-----------------CoLLect_List() & Collect_Set() ----------------

98

```
df.groupBy('Item_type').agg(collect_list('Outlet_Size').alias('Outlet_Size_list')).display()
df.groupBy('Item_type').agg(collect_set('Outlet_Size').alias('Outlet_Size_list')).display()
```

```
Command skipped
```

---------------- Pivot -------------------

100

```
df.groupBy('Item_Type').pivot('Outlet_Size').agg(avg('Item_MRP').alias('Average_MRP')).display()
```

```
Command skipped
```

---------------- WHEN -OtherWise ----------

Create A Column `veg_flag` if `item_type` is not Meat

103

```
df = df.withColumn('veg_flag', when(col('Item_Type') == 'Meat', 'Non-Veg').otherwise('Veg'))
df.display()
```

df: pyspark.sql.dataframe.DataFrame = [Item_Identifier: string, ITEM_wieght: string ... 11 more fields]

Table

Create a `Veg_Expensive_flag` column if it is `veg_flag` is true & `Item_MRP` is > 100, it should be `veg_expensive` if `Item_MRP` < 100, It should be veg_Inexpensive, Otherwise it can be Simple Nonveg

105

```
# df = df.withColumn('veg_exp_flag',when((col('veg_flag') == 'Veg') &
(col('Item_MRP') <= 100), 'Veg Inexpensive'))\
#                                   .when((col('veg_flag') == 'Veg') &
(col('Item_MRP') > 100), 'Veg Expensive')\
#                                   .otherwise('Non-Veg')

df = df.withColumn(
    'veg_exp_flag',\
    when((col('veg_flag') == 'Veg') & (col('Item_MRP') <= 100), 'Veg
Inexpensive')\
    .when((col('veg_flag') == 'Veg') & (col('Item_MRP') > 100), 'Veg
Expensive')\
    .otherwise('Non-Veg')
)
df.display()
```

df: pyspark.sql.dataframe.DataFrame = [Item_Identifier: string, ITEM_wieght: string ... 12 more fields]

Table

JOINS

- **INNER JOINS**
- **OUTER JOINS**
- **LEFT JOIN**
- **RIGHT JOIN**
- **FULL JOIN**
- `ANTI JOIN`

108

Table