PySpark Tutorial

January 20, 2025

0.1 PySpark Complete Tutorial

0.1.1 Data Path

- File uploaded to /FileStore/tables/drivers.json
- File uploaded to /FileStore/tables/BigMart_Sales.csv

0.1.2 Data Reading

[0]:
```
df_json = spark.read.format('json').option('inferSchema',True).
  ↪option('header',True).option('multiline',False).load('/FileStore/tables/
  ↪drivers.json')
df_json.limit(10).display()
```

[0]:
```
df_csv = spark.read.format('csv').option('inferSchema',True).
  ↪option('header',True).load('/FileStore/tables/BigMart_Sales.csv')
df_csv.limit(10).display()
```

0.1.3 Schema Definition

[0]:
```
df_csv.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)
```

0.1.4 DDL Schema

[0]:
```
# Item_Weight datatype has been modified
custom_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 integer,
                    Outlet_Size string,
                    Outlet_Location_Type string,
                    Outlet_Type string,
                    Item_Outlet_Sales double
                   """
df_csv_custom_schema = spark.read.format('csv').schema(custom_ddl_schema).
 ↪option('header',True).load('/FileStore/tables/BigMart_Sales.csv')
df_csv_custom_schema.printSchema()
```

```
root
 |-- Item_Identifier: string (nullable = true)
 |-- Item_Weight: string (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)
```

[0]:
```
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

```
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()
```

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

0.1.5 TRANSFORMATIONS

[0]:
```
df_csv.limit(10).display()
```

[0]:
```
df_csv.select('Item_Identifier','Item_Weight').limit(10).display()
```

[0]:
```
df_csv.select(col('Item_Identifier').alias('ITEM_IDENTIFIER')).limit(10).
 ↪display()
```

0.1.6 Filter

[0]:
```
df_csv.filter(col('Item_Fat_Content')=='Regular').limit(10).display()
```

[0]:
```
df_csv.filter((col('Item_Type') == 'Soft Drinks') & (col('Item_Weight')<10)).
 ↪display()
```

[0]:
```
df_csv.filter((col('Outlet_Size').isNull()) & (col('Outlet_Location_Type').
 ↪isin('Tier 1','Tier 2'))).limit(10).display()
```

0.1.7 Rename the column

[0]:
```
df_csv.withColumnRenamed('Item_Weight', 'Item_Wt').limit(10).display()
```

0.1.8 Create new column

[0]:
```
df_csv.withColumn('flag',lit('New')).limit(10).display()
```

[0]:
```
df_csv.withColumn('multiply',col('Item_Weight')*col('Item_MRP')).limit(10).
 ↪display()
```

[0]:
```
df_csv.
 ↪withColumn('Item_Fat_Content_Updated',regexp_replace(col('Item_Fat_Content'),"Regular","Reg
.
 ↪withColumn('Item_Fat_Content_Updated',regexp_replace(col('Item_Fat_Content'),"Low␣
 ↪Fat","LF")).limit(10).display()
```

0.1.9 Type Casting

[0]:
```
df_csv.withColumn('Item_Weight',col('Item_Weight').cast(StringType())).
 ↪printSchema()
```

```
root
 |-- Item_Identifier: string (nullable = true)
 |-- Item_Weight: string (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)
```

0.1.10 Sort

[0]:
```
df_csv.sort(col('Item_Weight').desc()).limit(10).display()
```

[0]:
```
df_csv.sort(col('Item_Weight').asc()).limit(10).display()
```

[0]:
```
df_csv.sort(['Item_Weight', 'Item_Visibility'], ascending=[0,0]).limit(10).
↪display()
```

[0]:
```
df_csv.sort(['Item_Weight', 'Item_Visibility'], ascending = [0,1]).limit(10).
↪display()
```

0.1.11 Limit

[0]:
```
df_csv.limit(10).display()
```

0.1.12 Drop Columns

[0]:
```
df_csv.drop('Item_Weight').limit(10).display()
```

[0]:
```
df_csv.drop(col('Item_Weight'), col('Item_Visibility')).limit(10).display() #␣
↪It will work without col as well
```

0.1.13 Drop Duplicates

[0]:
```
df_csv.dropDuplicates().limit(10).display()
```

[0]:
```
df_csv.dropDuplicates(subset=['Item_Type']).limit(10).display()
```

0.1.14 Union and Union By Name

[0]:
```
data1 = [('1','Kid'),('2','Sid')]
schema1 ='id STRING, name STRING'
df1 = spark.createDataFrame(data1, schema1)
df1.display()

data2 = [('3','Rahul'), ('4','Jas')]
schema2 = 'id STRING, name STRING'
df2 = spark.createDataFrame(data2, schema2)
df2.display()

data3 = [('Subhash','5'), ('Anushka', '6')]
schema3 = 'name STRING, id STRING'
df3 = spark.createDataFrame(data3, schema3)
df3.display()
```

[0]:
```
df1.union(df2).display()
```

[0]:
```
df1.union(df3).display()
```

[0]:
```
df1.unionByName(df2).display()
```

[0]:
```
df1.unionByName(df3).display()
```

0.1.15 String Functions

[0]:
```
df_csv.select(upper('Item_Type').alias('Upper_Item_Type')).limit(10).display()
```

0.1.16 Date Functions

[0]:
```
df_csv = df_csv.withColumn('curr_date',current_date())
df_csv.limit(10).display()
```

[0]:
```
df_csv.withColumn('week_after',date_add('curr_date',7)).limit(10).display()
```

[0]:
```
df_csv.withColumn('week_before',date_sub('curr_date', 7)).limit(10).display()
```

[0]:
```
df_csv = df_csv.withColumn('week_before',date_add('curr_date',-7))
df_csv.limit(10).display()
```

[0]:
```
df_csv.withColumn('date_diff',datediff('curr_date','week_before')).limit(10).
↪display()
```

[0]:
```
df_csv.withColumn('week_before',date_format('week_before', 'dd-MM-yyy')).
↪limit(10).display()
```

0.1.17 Handling Nulls

[0]:
```
df_csv.dropna('all').limit(10).display()
```

[0]:
```
df_csv.dropna('any').limit(10).display()
```

[0]:
```
df_csv.dropna(subset=['Outlet_Size']).limit(10).display()
```

0.1.18 Filling Nulls

[0]:
```
df_csv.fillna('Not_Available').limit(10).display()
```

[0]:
```
df_csv.fillna('Not_Available',subset=['Outlet_Size']).limit(10).display()
```

0.1.19 Split and Indexing

[0]:
```
df_csv.withColumn('Outlet_Type', split('Outlet_Type', ' ')).limit(10).display()
```

[0]:
```
df_csv.withColumn('Outlet_Type',split('Outlet_Type',' ')[1]).limit(10).display()
```

0.1.20 Explode and Array Contains

[0]:
```
df_csv.withColumn('Outlet_Type',split('Outlet_Type', ' '))\
  .withColumn('Outlet_Type', explode('Outlet_Type')).limit(10).display()
```

[0]:
```
df_csv.withColumn('Outlet_Type',split('Outlet_Type', ' '))\
    .withColumn('Type1_Flag',array_contains('Outlet_Type', 'Type1')).limit(10).
 ↪display()
```

0.1.21 GroupBy

[0]:
```
df_csv.groupBy('Item_Type').agg(sum('Item_MRP')).limit(10).display()
```

[0]:
```
df_csv.groupBy('Item_Type').agg(avg('Item_MRP')).display()
```

[0]:
```
df_csv.groupBy('Item_Type', 'Outlet_Size').agg(sum('Item_MRP').
 ↪alias('Total_Item_MRP')).display()
```

[0]:
```
df_csv.groupBy('Item_Type', 'Outlet_Size').agg(sum('Item_MRP'),␣
 ↪avg('Item_MRP')).display()
```

0.1.22 Collect List and Select

[0]:
```
data4 = [
    ('user1', 'book1'),
    ('user1', 'book2'),
    ('user2', 'book2'),
    ('user2', 'book4'),
    ('user1', 'book1')
    ]
schema4 = 'user STRING, book STRING'
df_book = spark.createDataFrame(data4, schema4)
df_book.display()

df_book.groupBy('user').agg(collect_list('book')).display()
```

[0]:
```
df_csv.select('Item_Type', 'Outlet_Size', 'Item_MRP').limit(10).display()
```

0.1.23 Pivot

[0]:
```
df_csv.groupBy('Item_Type').pivot('Outlet_Size').agg(sum('Item_MRP')).display()
```

0.1.24 When-Otherwise like case-when in SQL

[0]:
```
df_csv.withColumn('veg_flag',when(col('Item_Type')=='Meat','Non-Veg').
 ↪otherwise('Veg')).limit(10).display()
```

[0]:
```
df_csv.withColumn('Item_Type_Flag', when(col('Item_Type') == 'Dairy', 'Dairy').
 ↪when(col('Item_Type') == 'Meat', 'Meat').otherwise('Others')).limit(10).
 ↪display()
```

[0]:
```
df_csv.withColumn('Item_Type_Flag', when((col('Item_Type') == 'Dairy') &␣
 ↪(col('Outlet_Size') == 'Medium'), 'Dairy_Medium').otherwise('Others')).
 ↪limit(10).display()
```

0.1.25 Joins

[0]:
```
dataj1 = [('1','gaur','d01'),
          ('2','kit','d02'),
          ('3','sam','d03'),
          ('4','tim','d03'),
          ('5','aman','d05'),
          ('6','nad','d06')]

schemaj1 = 'emp_id STRING, emp_name STRING, dept_id STRING'
df1 = spark.createDataFrame(dataj1,schemaj1)
df1.display()

dataj2 = [('d01','HR'),
          ('d02','Marketing'),
          ('d03','Accounts'),
          ('d04','IT'),
          ('d05','Finance')]

schemaj2 = 'dept_id STRING, department STRING'
df2 = spark.createDataFrame(dataj2,schemaj2)
df2.display()
```

[0]:
```
df1.join(df2, df1['dept_id'] == df2['dept_id'], 'inner').display()
```

[0]:
```
df1.join(df2, df1['dept_id'] == df2['dept_id'], 'left').display()
```

[0]:
```
df1.join(df2, df1['dept_id'] == df2['dept_id'], 'right').display()
```

[0]:
```
df1.join(df2, df1['dept_id'] == df2['dept_id'], 'anti').display()
```

0.1.26 Windows Functions

[0]:
```
from pyspark.sql.window import Window
df_csv.withColumn('rowCol', row_number().over(Window.
 ↪orderBy('Item_Identifier'))).limit(10).display()
```

[0]:
```
df_csv.withColumn('rank', rank().over(Window.orderBy(col('Item_Identifier')))).
 ↪withColumn('dense_rank', dense_rank().over(Window.
 ↪orderBy(col('Item_Identifier')))).limit(10).display()
```

0.1.27 Sum Aggregation

[0]:
```
df_csv.withColumn('rolling_sum', sum('Item_MRP').over(Window.
 ↪orderBy(col('Item_Identifier')))).limit(10).display()
```

0.1.28 Cummulative Sum

[0]:
```
# df_csv.withColumn('cumsum', sum('Item_MRP').over(Window.
 ↪orderBy('Item_Type'))).display() # It will not show us cummulative sum
df_csv.withColumn('cumsum', sum('Item_MRP').over(Window.orderBy('Item_Type').
 ↪rowsBetween(Window.unboundedPreceding, Window.currentRow))).limit(10).
 ↪display() # Cummulative Sum
df_csv.withColumn('total_sum', sum('Item_MRP').over(Window.orderBy('Item_Type').
 ↪rowsBetween(Window.unboundedPreceding, Window.unboundedFollowing))).
 ↪display() # Total Sum
df_csv.select(sum('Item_MRP')).limit(10).display() # Total Sum
```

0.1.29 User Defined Functions (UDF)

[0]:
```
# Step 1: Create function like Python
def my_func(x):
  return x*x
# Step 2: Converting function into UDF
my_udf = udf(my_func)

df_csv.withColumn('mynewcol',my_udf('Item_MRP')).limit(10).display()
```

0.1.30 Data Writing

[0]:
```
# Saving with bydefault mode
# df_csv.write.format('csv').save('/FileStore/tables/CSV/data.csv')

# Append
# df_csv.write.format('csv').mode('append').save('/FileStore/tables/CSV/data.
 ↪csv')
```

```
# df_csv.write.format('csv').mode('append').option('path', '/FileStore/tables/
 ↪CSV/data.csv').save() # Another way of saving

# Overwrite
df_csv.write.format('csv').mode('overwrite').save('/FileStore/tables/CSV/data.
 ↪csv')

# Ignore
df_csv.write.format('csv').mode('ignore').save('/FileStore/tables/CSV/data.csv')

# Error
# df_csv.write.format('csv').mode('error').option('path', '/FileStore/tables/
 ↪CSV/data.csv').save() # Another of saving
```

[0]:
```
df_csv.write.format('parquet').mode('overwrite').save('/FileStore/tables/CSV/
 ↪data.parquet')
```

[0]:
```
df_csv.write.format('parquet').mode('overwrite').saveAsTable('my_table')
```

0.1.31 Spark SQL

[0]:
```
# df_csv.createTempView('my_view')
```

[0]:
```
%sql
select * from my_view
```

[0]:
```
df_sql = spark.sql("select * from my_view")
df_sql.limit(10).display()
```

0.1.32 THE END