de_pyspark_ssj40 (Python)

Import notebook

```
%sql
SHOW VOLUMES;
SHOW CATALOGS;

SHOW SCHEMAS IN workspace;
```

▸ _sqldf: pyspark.sql.connect.dataframe.DataFrame = [databaseName: string]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

;

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, expr, split, size, year, try_to_date, when, round, avg, count

spark = SparkSession.builder.appName("MoviesDataPipeline").getOrCreate()

# Full path to your volume
full_volume_path = "/Volumes/workspace/default/movies_data/"

# Use DROPMALFORMED mode to ignore broken rows
df_movies = (
    spark.read.option("header", True)
    .option("inferSchema", True)
    .option("mode", "DROPMALFORMED")
    .csv(f"{full_volume_path}movies_metadata.csv")
)

df_credits = (
    spark.read.option("header", True)
    .option("inferSchema", True)
    .option("mode", "DROPMALFORMED")
    .csv(f"{full_volume_path}credits.csv")
)

df_keywords = (
    spark.read.option("header", True)
    .option("inferSchema", True)
    .option("mode", "DROPMALFORMED")
    .csv(f"{full_volume_path}keywords.csv")
)

# Preview
display(df_movies.limit(5))
```

Table

```
from pyspark.sql.functions import year, to_date, size

# Safely cast numeric columns
numeric_cols = ["budget", "revenue", "vote_average", "popularity", "runtime"]
for c in numeric_cols:
    df_movies = df_movies.withColumn(c, col(c).cast("double"))

# Extract release year safely
df_movies = df_movies.withColumn("release_year", year(to_date(col("release_date"), "yyyy-MM-dd")))

# Count genres as a simple numeric feature
df_movies = df_movies.withColumn("genre_count", size(split(col("genres"), "\\|")))

# Fill nulls in numeric columns
df_movies = df_movies.fillna({c:0 for c in numeric_cols})
```

```
#Clean movies data

from pyspark.sql.functions import expr, col, try_to_date, year, size, split

# List of numeric columns
numeric_cols = ["budget", "revenue", "vote_average", "popularity", "runtime"]

# Safely cast to double using try_cast()
for c in numeric_cols:
    df_movies = df_movies.withColumn(c, expr(f"try_cast({c} as double)"))

# Safely extract release_year using try_to_date
df_movies = df_movies.withColumn("release_year", year(try_to_date(col("release_date"), "yyyy-MM-dd")))

# Count number of genres
df_movies = df_movies.withColumn("genre_count", size(split(col("genres"), "\\|")))

# Fill nulls in numeric columns
df_movies = df_movies.fillna({c: 0 for c in numeric_cols})
```

```
#Filter movies data
df_filtered = df_movies.filter(col("release_year").isNotNull())\
                       .filter(col("release_year") >= 2010)\
                       .filter(col("vote_average") > 7)

display(df_filtered.limit(5))
```

Table

```
#Clean credits and keywords for joins

# Keep only rows with valid IDs
df_credits_clean = df_credits.filter(col("id").isNotNull())
df_keywords_clean = df_keywords.filter(col("id").isNotNull())
df_movies_clean = df_movies.filter(col("id").isNotNull())

# Safely cast IDs to int
df_credits_clean = df_credits_clean.withColumn("id", expr("try_cast(id as int)"))
df_keywords_clean = df_keywords_clean.withColumn("id", expr("try_cast(id as int)"))
df_movies_clean = df_movies_clean.withColumn("id", expr("try_cast(id as int)"))
```

```
#JOINS
df_joined = df_movies_clean.join(df_credits_clean, on="id", how="left")\
                           .join(df_keywords_clean, on="id", how="left")

display(df_joined.limit(5))
```

Table

```
# Movie data by release year – counting how many movies were released, their average rating, average budget,
and average revenue.
from pyspark.sql.functions import round

df_yearly_stats = (
    df_joined
    .filter(col("release_year").isNotNull())
    .groupBy("release_year")
    .agg(
        count("*").alias("movie_count"),
        round(avg("vote_average"), 2).alias("avg_rating"),
        round(avg("budget"), 0).alias("avg_budget"),
        round(avg("revenue"), 0).alias("avg_revenue")
    )
    .orderBy("release_year")
)

display(df_yearly_stats)
```

Table

```
# Register DataFrames as SQL views
df_movies.createOrReplaceTempView("movies")
df_credits.createOrReplaceTempView("credits")
df_keywords.createOrReplaceTempView("keywords")
df_joined.createOrReplaceTempView("joined_movies")
```

```
#Top 10 Highest-Rated Movies Since 2015

query1 = """
SELECT title, release_year, vote_average, popularity
FROM joined_movies
WHERE release_year >= 2015
ORDER BY vote_average DESC, popularity DESC
LIMIT 10
"""
df_top_movies = spark.sql(query1)
df_top_movies.show(truncate=False)
```

```
+-----------------------------------+------------+------------+----------+
|title                              |release_year|vote_average|popularity|
+-----------------------------------+------------+------------+----------+
|LEGO DC Super Hero Girls: Brain Drain|2017        |10.0        |8.413734  |
|Tokyo Ghoul                        |2017        |10.0        |0.802191  |
|Stephen Lynch: Hello Kalamazoo     |2016        |10.0        |0.724499  |
|Long Strange Trip                  |2017        |10.0        |0.617655  |
|The Human Surge                    |2016        |10.0        |0.484825  |
|First Round Down                   |2017        |10.0        |0.422836  |
|Sum of Histories                   |2015        |10.0        |0.404432  |
|Sum of Histories                   |2015        |10.0        |0.404432  |
|Bazodee                            |2016        |10.0        |0.379968  |
|My Future Love                     |2016        |10.0        |0.371238  |
+-----------------------------------+------------+------------+----------+
```

```
#Query 2: Average Rating per Release Year

query2 = """
SELECT release_year,
       ROUND(AVG(vote_average), 2) AS avg_rating,
       COUNT(*) AS num_movies
FROM joined_movies
WHERE release_year IS NOT NULL
GROUP BY release_year
ORDER BY release_year DESC
"""
df_yearly_stats = spark.sql(query2)
df_yearly_stats.show()
```

```
|        2018|       0.0|         4|
|        2017|      5.88|       511|
|        2016|      5.83|      1587|
|        2015|      5.77|      1930|
|        2014|      5.64|      1946|
|        2013|      5.75|      1822|
|        2012|      5.66|      1660|
|        2011|      5.62|      1595|
|        2010|      5.72|      1401|
|        2009|      5.59|      1499|
|        2008|      5.57|      1392|
|        2007|      5.66|      1222|
|        2006|      5.58|      1173|
|        2005|      5.58|      1055|
|        2004|      5.68|       904|
|        2003|      5.64|       804|
|        2002|      5.55|       837|
|        2001|       5.5|       808|
|        2000|      5.42|       726|
+------------+----------+----------+
only showing top 20 rows
```

```
output_path = f"{full_volume_path}processed_movies_data/"

# Save both query outputs
df_top_movies.write.mode("overwrite").parquet(output_path + "top_movies")
df_yearly_stats.write.mode("overwrite").parquet(output_path + "yearly_stats")

print("Results written successfully to:", output_path)
```

```
Results written successfully to: /Volumes/workspace/default/movies_data/processed_movies_data/
```

```
display(dbutils.fs.ls(output_path))
```

Table

```
df_loaded = spark.read.parquet(output_path + "top_movies")
df_loaded.show(5)
```

```
+--------------------+------------+------------+----------+
|               title|release_year|vote_average|popularity|
+--------------------+------------+------------+----------+
|LEGO DC Super Her...|        2017|        10.0|  8.413734|
|         Tokyo Ghoul|        2017|        10.0|  0.802191|
|Stephen Lynch: He...|        2016|        10.0|  0.724499|
|   Long Strange Trip|        2017|        10.0|  0.617655|
|     The Human Surge|        2016|        10.0|  0.484825|
+--------------------+------------+------------+----------+
only showing top 5 rows
```

```
# Performance Analysis
# Analyze execution plan for first SQL query
print(" Execution Plan: Top Rated Movies Query ")
df_top_movies.explain(extended=True)

# Analyze execution plan for second SQL query
print("\n Execution Plan: Average Rating by Year Query ")
df_yearly_stats.explain(extended=True)
```

```
                                      :        +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#15871]
                                      :           +- PhotonShuffleExchangeSink hashpartitioning(id#11514, 1024)
                                      :              +- PhotonProject [try_cast(id#11443 as int) AS id#11514]
                                      :                 +- PhotonFilter (isnotnull(id#11443) AND isnotnull(try_cast(id#11443 as int)))
                                      :                    +- PhotonRowToColumnar
                                      :                       +- FileScan csv [id#11443] Batched: false, DataFilters: [isnotnull(id#11443), isnotnull(try_cast(id#11443 as int))], Format: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/Volumes/workspace/default/movies_data/credits.csv], PartitionFilters: [], PushedFilters: [IsNotNull(id)], ReadSchema: struct<id:string>
                                      +- PhotonShuffleExchangeSource
                                         +- PhotonShuffleMapStage EXECUTOR_BROADCAST, [id=#15883]
                                            +- PhotonShuffleExchangeSink SinglePartition
                                               +- PhotonFilter isnotnull(id#11462)
                                                  +- PhotonRowToColumnar
                                                     +- FileScan csv [id#11462] Batched: false, DataFilters: [isnotnull(id#11462)], Format: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/Volumes/workspace/default/movies_data/keywords.csv], PartitionFilters: [], PushedFilters: [IsNotNull(id)], ReadSchema: struct<id:int>

== Photon Explanation ==
The query is fully supported by Photon.
```