✅ Step-by-Step PySpark Practice Sheet (Real-World Style)

🔹 1. Ingest the Dataset

- Upload the CSV to **Databricks FileStore** or **DBFS**.

Read using PySpark:

```
df = spark.read.option("header", True).option("inferSchema",
True).csv("/FileStore/your_dataset.csv")
```

-

🔹 2. Basic Exploration

- Print schema: `df.printSchema()`
- View sample: `df.show(5, truncate=False)`
- Check row count: `df.count()`

🔹 3. Data Quality Checks

- Missing/null counts per column
- Unique counts for `OrderID`, `CustomerID`, `ProductID`
- Invalid values:
  - Negative prices (`UnitPrice < 0`)
  - Invalid dates (e.g., `'INVALID_DATE'`)
  - Ratings not in 1–5 range

- ◦ NULLs in critical fields like `OrderDate`, `ProductID`, etc.

```
from pyspark.sql.functions import col, isnan, count, when

df.select([count(when(col(c).isNull() | isnan(c), c)).alias(c) for c
in df.columns]).show()
```

🔹 4. Data Cleaning

- Remove or fix:
  - Rows with `INVALID_DATE`
  - Rows with `NULL` `ProductID` or `UnitPrice`
  - Convert `DeliveryDate`, `OrderDate` to `DateType`
  - Fill null `CustomerRating` with average rating per `CustomerID`
  - Cap `CustomerRating` to [1, 5] if required
  - Convert `"TRUE"`/`"FALSE"` string fields to actual booleans

🔹 5. De-duplication

- Detect duplicate orders or products in orders

Check for exact duplicates using all fields:

```
df.groupBy(df.columns).count().filter("count > 1").show()
```

-

🔹 6. Feature Engineering

- Calculate `TotalPrice = Quantity * UnitPrice`

- Create:
  - `DeliveryTimeDays = DeliveryDate - OrderDate`
  - Flag for delayed delivery (e.g., over 5 days)
  - Flag for returned orders
  - Year-Month columns for time series

🔹 7. Aggregations & Insights

Perform groupings and summarizations:

- 🛒 Revenue per `Country`, `ProductName`
- 🛍️ Top 5 products by revenue
- 📅 Order volume by day/week/month
- 🧾 Count of orders per `CustomerSegment`
- 📦 Return rate by `ProductName` or `Country`

🎯 Discount/promotion effectiveness (`DiscountCode` & `PromotionApplied`)

```
df.groupBy("ProductName").agg(
    sum("Quantity").alias("TotalSold"),
    sum("TotalPrice").alias("Revenue")
).orderBy("Revenue", ascending=False).show()
```

🔹 8. Joins (Optional Practice)

Create a static DataFrame for:

- Customer demographics (age, city, gender)
- Product catalog (category, brand)

Then join with the main DF for richer insights.

🔹 9. Data Validations

- Verify referential integrity (e.g., every ProductID has a name)
- Check for duplicate `OrderID + ProductID` combos
- Assert schema types, ranges

🔹 10. Save Cleaned Data

Save to Parquet:

```
df_cleaned.write.mode("overwrite").parquet("/mnt/cleaned_data/retail
_orders")
```

-

Optionally save as Delta Table or Register as a SQL table:

```
df_cleaned.write.saveAsTable("cleaned_orders")
```

-

🔹 11. Optional: Load to BigQuery (if chosen)

- Export to GCS → Load to BigQuery
- Or use Databricks connector to BigQuery

Sample config:

```
df.write.format("bigquery").option("table",
"project.dataset.table").save()
```

🔹 12. Optional: Visualize in Power BI

- Export to CSV or Parquet
- Upload to Power BI
- Suggested visuals:
    - Time series: Orders per day/month
    - Pie chart: Customer segment distribution
    - Bar chart: Top products by revenue
    - Matrix: Return rate by product and country

🔹 13. Bonus: Optimize & Cache

- Partition by `Country` or `OrderDate`
- Cache popular tables
- Use `.repartition()` and `.coalesce()` if needed for performance

🔹 14. Optional Advanced Practice

- Use **Window functions**: Ranks, cumulative revenue
- Write **Unit tests** for transformations using `assert`
- Add **logging** and **error handling** blocks

🧠 Final Output

By the end, you should have:

- Cleaned, transformed DataFrame

- Loaded to BigQuery or Power BI-ready dataset
- Reusable code for ingestion → cleaning → analysis
- Insights similar to a production ETL pipeline