# Databricks Community Edition - Full Practice Set (30 Exercises)

**Instructions:**

Run each code block in Databricks Community Edition.

Observe the results, compare with expected outputs.

**Exercise 1: Load a CSV file into a DataFrame**

Code:

```
df = spark.read.option("header", True).option("inferSchema", True).csv("/databricks-datasets/airlines/part-00000")
df.show(5)
```

*Expected Result:*

*Shows first 5 rows of the Airlines dataset.*

**Exercise 2: Cache a DataFrame for faster access**

Code:

```
df.cache()
df.count()
```

*Expected Result:*

*Caches the DataFrame into memory.*

**Exercise 3: Select specific columns from a DataFrame**

Code:

```
df.select("Year", "Month", "DepDelay").show(5)
```

*Expected Result:*

*Shows selected columns.*

**Exercise 4: Filter rows where DepDelay > 30 minutes**

Code:

```
df.filter(df.DepDelay > 30).show(5)
```

*Expected Result:*

*Shows flights delayed more than 30 minutes.*

**Exercise 5: Add a new column 'DelayCategory'**

Code:

```
from pyspark.sql.functions import when


df2 = df.withColumn("DelayCategory", when(df.DepDelay > 30, "Late").otherwise("OnTime"))

df2.select("DepDelay", "DelayCategory").show(5)
```

*Expected Result:*

*Adds column categorizing flights as Late or OnTime.*

**Exercise 6: Group by Origin and count flights**

Code:

```
df.groupBy("Origin").count().orderBy("count", ascending=False).show(5)
```

*Expected Result:*

*Shows airports with most flights.*

**Exercise 7: Sort flights by Departure Delay descending**

Code:

```
df.orderBy(df.DepDelay.desc()).show(5)
```

*Expected Result:*

*Flights with highest departure delay first.*

**Exercise 8: Rename column DepDelay to DepartureDelayMinutes**

Code:

```
df2 = df.withColumnRenamed("DepDelay", "DepartureDelayMinutes")
```

```
df2.show(5)
```

*Expected Result:*

*Column renamed.*

**Exercise 9: Drop rows with null DepDelay**

Code:

```
df_clean = df.na.drop(subset=["DepDelay"])

df_clean.show(5)
```

*Expected Result:*

*Null DepDelay rows removed.*

**Exercise 10: Get distinct Origin airports**

Code:

```
df.select("Origin").distinct().show(5)
```

*Expected Result:*

*Lists distinct origin airports.*

**Exercise 11: Save DataFrame as Delta Table**

Code:

```
df.write.format("delta").mode("overwrite").save("/tmp/airlines_delta")
```

*Expected Result:*

*Saves as Delta format.*

**Exercise 12: Load Delta Table into DataFrame**

Code:

```
df_delta = spark.read.format("delta").load("/tmp/airlines_delta")

df_delta.show(5)
```

*Expected Result:*

*Loads Delta format back.*

**Exercise 13: Create a SQL table from Delta location**

Code:

```
spark.sql("CREATE TABLE IF NOT EXISTS airlines_delta USING DELTA LOCATION '/tmp/airlines_delta'")
```

*Expected Result:*

*Creates SQL table linked to Delta files.*

**Exercise 14: Update records in Delta Table**

Code:

```
spark.sql("UPDATE airlines_delta SET DepDelay = 0 WHERE DepDelay IS NULL")
```

*Expected Result:*

*Updates NULL DepDelay values.*

**Exercise 15: Time Travel to old Delta Table version**

Code:

```
df_old = spark.read.format("delta").option("versionAsOf", 0).load("/tmp/airlines_delta")
df_old.show(5)
```

*Expected Result:*

*Reads earlier version.*

**Exercise 16: Merge new data into Delta Table**

Code:

```
from delta.tables import DeltaTable


deltaTable = DeltaTable.forPath(spark, "/tmp/airlines_delta")
deltaTable.alias("old").merge(
    df.alias("new"), "old.FlightNum = new.FlightNum"
).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
```

*Expected Result:*

*Upserts data into Delta Table.*

**Exercise 17: Optimize Delta Table**

Code:

```
spark.sql("OPTIMIZE airlines_delta")
```

*Expected Result:*

*Improves Delta query speed.*

**Exercise 18: Vacuum old files from Delta Table**

Code:

```
spark.sql("VACUUM airlines_delta RETAIN 168 HOURS")
```

*Expected Result:*

*Cleans up old files safely.*

**Exercise 19: Add a new column to Delta Table**

Code:

```
spark.sql("ALTER TABLE airlines_delta ADD COLUMNS (NewColumn STRING)")
```

*Expected Result:*

*New column added.*

**Exercise 20: Drop a column from Delta Table**

Code:

```
spark.sql("ALTER TABLE airlines_delta DROP COLUMN NewColumn")
```

*Expected Result:*

*Column removed from Delta Table.*

**Exercise 21: Run SQL query on Delta Table**

Code:

```
spark.sql("SELECT Carrier, AVG(DepDelay) AS AvgDepDelay FROM airlines_delta GROUP BY Carrier ORDER BY AvgDepDelay DESC").show()
```

*Expected Result:*

*Average delays per Carrier.*

**Exercise 22: Create managed Delta Table from DataFrame**

Code:

```
df.write.saveAsTable("managed_airlines_table")
```

*Expected Result:*

*Creates managed SQL table.*

**Exercise 23: Drop SQL table**

Code:

```
spark.sql("DROP TABLE IF EXISTS managed_airlines_table")
```

*Expected Result:*

*Deletes SQL table.*

**Exercise 24: Create a temporary SQL View**

Code:

```
df.createOrReplaceTempView("temp_view_flights")
spark.sql("SELECT * FROM temp_view_flights LIMIT 5").show()
```

*Expected Result:*

*Creates view for easy querying.*

**Exercise 25: Check Delta Table history**

Code:

```
spark.sql("DESCRIBE HISTORY airlines_delta").show(truncate=False)
```

*Expected Result:*

*Shows table modification history.*

**Exercise 26: Top 5 flights with highest Arrival Delay**

Code:

```
df.orderBy(df.ArrDelay.desc()).select("FlightNum", "ArrDelay").show(5)
```

*Expected Result:*

*Top delayed flights displayed.*

**Exercise 27: Find flights with negative DepDelay (early departures)**

Code:

```
df.filter(df.DepDelay < 0).select("FlightNum", "DepDelay").show(5)
```

*Expected Result:*

*Early departing flights listed.*

**Exercise 28: Group by Carrier and calculate avg DepDelay**

Code:

```
df.groupBy("Carrier").avg("DepDelay").orderBy("avg(DepDelay)").show(5)
```

*Expected Result:*

*Best and worst airlines by delay.*

**Exercise 29: Register DataFrame as SQL Table then query**

Code:

```
df.write.format("delta").mode("overwrite").saveAsTable("airport_summary")
spark.sql("SELECT Origin, COUNT(*) FROM airport_summary GROUP BY Origin ORDER BY
COUNT(*) DESC").show(5)
```

*Expected Result:*

*Top airports displayed.*

**Exercise 30: Perform Delta Time Travel using SQL**

Code:

```
spark.sql("SELECT * FROM airlines_delta VERSION AS OF 0 LIMIT 5").show()
```

*Expected Result:*

*Reads old snapshot of table.*