



# **PySpark Scenario-Based Interview Questions (Complete Notes Series)**

**DAY 16 – Error Handling**

**&**

**Debugging in PySpark :**



**Karthik Kondpak**  
**9989454737**

# Error Handling & Debugging in PySpark

## ● 1. Why Error Handling is Important in PySpark?

- Spark jobs run **distributed across multiple executors**.
- Errors can be **silent** (performance issues) or **hard failures** (job crash).
- Proper error handling helps in:

Debugging failures quickly

Maintaining data quality

Avoiding job re-runs on huge datasets

## ● 2. Common Sources of Errors in PySpark

### 1. Schema mismatch

a. Example: Expected IntegerType but got StringType.

### 2. Null or Missing values

a. Unexpected nulls in joins or aggregations.

### 3. Data Skew

a. One partition having too much data.

### 4. Shuffle & Memory errors

- a. OutOfMemoryError, ExecutorLostFailure.

## 5. Invalid operations

- a. Calling unsupported functions or wrong column references.

## 3. Debugging with Logs

- Spark writes logs at **driver** and **executor** level.
- Use:

```
spark.ark.Con{text}.setLogLevel("IDNEFOB,UGWA"), ERROR
```

- **Spark UI** (<http://localhost:4040>) gives:

DAG Visualization

Stages & Tasks breakdown

Shuffle read/write stats

Skew detection

## 4. Using `explain()` to Debug Plans

- `explain()` shows the **logical & physical plan** (Catalyst Optimizer).
- Helps detect **unnecessary shuffles, scans, or broadcasts**.

```
df = spark.read.csv("data.csv", header=True, inferSchema=True)  
df.groupBy("category").count().explain(True)
```

Output shows:

- Parsed Logical Plan
- Analyzed Logical Plan
- Optimized Logical Plan
- Physical Plan

## 5. Handling Null & Missing Data

### Drop Nulls

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

### Fill Nulls

```
df.na.fill({"age": 0, "city": "Unknown"})
```

### Replace Specific Values

```
df.na.replace("?", None)
```

Avoids **NullPointerException** and ensures consistency.

## 6. Using try...except in PySpark

Python-level exceptions can be handled with **try-except**.

```
try:  
    df = spark.read.csv("invalid_path.csv", header=True)
```

```
except Exception as e:  
    print(f"Error reading file: {e}")
```

## ● 7. Data Type Errors & Casting

- Mismatchedt ypes causeruntime errors.
- **Usesafecasting** with whenand otherwise.

```
from pyspark.sql.functions import col, when  
  
df = df.withColumn(  
    "age_int",  
    when(col("age").rlike("[^0-9]+$"),  
        col("age").cast("int")).otherwise(None)  
)
```

This avoids job failures when non-numeric data exists.

## ● 8. Handling Job Failures

- Use **checkpointing** for long pipelines.
- Re-run only failed stages instead of full job.

```
spark.sparkContext.setCheckpointDir("/tmp/checkpoints")  
df.checkpoint()
```

Useful in iterative jobs (ML, graph processing).

## 9. Debugging Joins

Common issue: **duplicate columns, nulls, or skew.**

### Duplicate Columns

```
df1.join(df2, "id", "inner").drop(df2.id)
```

### Broadcast Join to Fix Skew

```
from pyspark.sql.functions import broadcast  
df1.join(broadcast(df2), "id")
```

Prevents large shuffles and memory errors.

## 10. Debugging Performance Issues

- Check partitions:

```
df.rdd.getNumPartitions()
```

- Repartition or coalesce:

```
df = df.repartition(10)      #Increase parallelism  
df = df.coalesce(2)         #Reduce shuffle
```

- Tune shuffle partitions:

```
spark.conf.set("spark.sql.shuffle.partitions", 100)
```

## ● 11. Using Accumulators & Logging for Debugging

- **Accumulators** help debug data counts during job execution.

```
acc = spark.sparkContext.accumulator(0)

def count_errors(row):
    global acc
    if row["status"] == "error":
        acc += 1
    return row

df.rdd.map(count_errors).collect()
print(f"Total Errors: {acc.value}")
```

## ● 12. Best Practices for Error Handling in PySpark

Validate schema before preprocessing.

Use **try-except** for external reads/writes.

Handle **nulls** explicitly.

Use `explain()` to analyze plans.

Monitor jobs in **Spark UI**.

Optimize joins with **broadcast** and **skew handling**.

Enable **checkpointing** for long pipelines.



**Let's build your Data  
Engineering journey  
together!**

 Call us directly at: 9989454737

 <https://seekhobigdata.com/>





**Let's build your Data  
Engineering journey  
together!**



Call us directly at: 9989454737



<https://seekhobigdata.com/>

