PySpark Interview Handbook — Batch 5

Problems 101–125
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 101: 101 - Pivot & Crosstab: Cube challenge

Problem

Pivot & Crosstab

Solution (PySpark)

```
from pyspark.sql import functions as F
res = events.groupBy("user_id").pivot("event_type").agg(F.count("*")).fillna(0)
```

Problem 102: 102 - File IO & Formats: Delta-like challenge

Problem

File IO & Formats

Solution (PySpark)

```
# Example write (commented):
# sessions.write.mode("overwrite").partitionBy("country").parquet("/path/out")
res = sessions
```

Problem 103: 103 - Complex Types: Explode challenge

Problem

Complex Types

Solution (PySpark)

```
from pyspark.sql import functions as F
res = users
if "tags" in res.columns:
    res = res.withColumn("tag", F.explode_outer("tags"))
```

Problem 104: 104 - DataFrame Basics: Drop challenge

Problem

DataFrame Basics

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window
res = clicks.select("id", "user_id", "event_type", "value") \    .withColumn("value_norm", (F.col("value") - F.mean("va
assert "value_norm" in res.columns
```

Problem 105: 105 - Dates & Timestamps: From_unixtime challenge

Problem

Dates & Timestamps

Solution (PySpark)

```
from pyspark.sql import functions as F
res = transactions.withColumn("day", F.to_date("ts")).groupBy("day").agg(F.count("*").alias("events"), F.avg("value").a
```

Problem 106: 106 - Dates & Timestamps: From_unixtime challenge

Problem

Dates & Timestamps

Solution (PySpark)

```
from pyspark.sql import functions as F
res = transactions.withColumn("day", F.to_date("ts")).groupBy("day").agg(F.count("*").alias("events"), F.avg("value").a
```

Problem 107: 107 - DataFrame Basics: Select challenge

Problem

DataFrame Basics

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window
res = clicks.select("id", "user_id", "event_type", "value") \    .withColumn("value_norm", (F.col("value") - F.mean("va
assert "value_norm" in res.columns
```

Problem 108: 108 - UDFs & Pandas UDFs: Vectorized challenge

Problem

UDFs & Pandas UDFs

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
@F.udf(DoubleType())
def score(x): return float(x)*1.1 if x is not None else None
res = events.withColumn("score", score("value"))
```

Problem 109: 109 - MLlib Basics: Logisticregression challenge

Problem

MLlib Basics

Solution (PySpark)

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression
label_indexer = StringIndexer(inputCol="event_type", outputCol="label", handleInvalid="skip")
assembler = VectorAssembler(inputCols=["value"], outputCol="features")
lr = LogisticRegression(maxIter=10)
model = Pipeline(stages=[label_indexer, assembler, lr]).fit(orders)
res = model.transform(orders)
```

Problem 110: 110 - Joins: Semi challenge

Problem

Joins

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
users = spark.createDataFrame([("u1","US"),("u2","IN")], ["user_id","country"])
res = clicks.join(F.broadcast(users), "user_id", "left")
```

Problem 111: 111 - Performance & Tuning: Aqe challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = products.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```

Problem 112: 112 - Aggregations & GroupBy: Approx_count_distinct challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

```
from pyspark.sql import functions as F
user_stats = clicks.groupBy("user_id").agg(F.count("*").alias("cnt"), F.sum("value").alias("sum_value"), F.avg("value")
global_distinct = clicks.select(F.countDistinct("user_id").alias("distinct_users"))
res = user_stats
```

Problem 113: 113 - Pivot & Crosstab: Rollup challenge

Problem

Pivot & Crosstab

Solution (PySpark)

```
from pyspark.sql import functions as F
res = orders.groupBy("user_id").pivot("event_type").agg(F.count("*")).fillna(0)
```

Problem 114: 114 - Dates & Timestamps: To_date challenge

Problem

Dates & Timestamps

Solution (PySpark)

```
from pyspark.sql import functions as F
res = events.withColumn("day", F.to_date("ts")).groupBy("day").agg(F.count("*").alias("events"), F.avg("value").alias("
```

Problem 115: 115 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

```
res = users.hint("broadcast")
```

Problem 116: 116 - Joins: Semi challenge

Problem

Joins

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
users = spark.createDataFrame([("u1","US"),("u2","IN")], ["user_id","country"])
res = products.join(F.broadcast(users), "user_id", "left")
```

Problem 117: 117 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

```
# streaming example would use readStream; here batch placeholder
res = users
```

Problem 118: 118 - File IO & Formats: Delta-like challenge

Problem

File IO & Formats

Solution (PySpark)

```
# Example write (commented):
# products.write.mode("overwrite").partitionBy("country").parquet("/path/out")
res = products
```

Problem 119: 119 - Joins: Left challenge

Problem

Joins

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
users = spark.createDataFrame([("u1","US"),("u2","IN")], ["user_id","country"])
res = users.join(F.broadcast(users), "user_id", "left")
```

Problem 120: 120 - Performance & Tuning: Checkpoint challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = orders.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```

Problem 121: 121 - Pivot & Crosstab: Pivot challenge

Problem

Pivot & Crosstab

Solution (PySpark)

```
from pyspark.sql import functions as F
res = users.groupBy("user_id").pivot("event_type").agg(F.count("*")).fillna(0)
```

Problem 122: 122 - Performance & Tuning: Checkpoint challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = events.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```

Problem 123: 123 - File IO & Formats: Csv challenge

Problem

File IO & Formats

Solution (PySpark)

```
# Example write (commented):
# logs.write.mode("overwrite").partitionBy("country").parquet("/path/out")
res = logs
```

Problem 124: 124 - File IO & Formats: Csv challenge

Problem

File IO & Formats

Solution (PySpark)

```
# Example write (commented):
# products.write.mode("overwrite").partitionBy("country").parquet("/path/out")
res = products
```

Problem 125: 125 - Performance & Tuning: Repartition challenge

Problem

Performance & Tuning

Solution (PySpark)

```
from pyspark.sql import functions as F
res = events.repartition(200, "user_id").groupBy("user_id").agg(F.count("*").alias("cnt"))
```