PySpark Interview Handbook — Batch 6

Problems 126–150
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 126: 126 - Pivot & Crosstab: Rollup 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 127: 127 - Pivot & Crosstab: Rollup challenge

Problem

Pivot & Crosstab

Solution (PySpark)

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

Problem 128: 128 - UDFs & Pandas UDFs: Pandas_udf 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 = transactions.withColumn("score", score("value"))
```

Problem 129: 129 - Dates & Timestamps: Window challenge

Problem

Dates & Timestamps

Solution (PySpark)

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

Problem 130: 130 - 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 131: 131 - DataFrame Basics: Filter challenge

Problem

DataFrame Basics

Solution (PySpark)

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

Problem 132: 132 - File IO & Formats: Delta-like challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 133: 133 - Aggregations & GroupBy: Agg challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

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

Problem 134: 134 - Pivot & Crosstab: Cube challenge

Problem

Pivot & Crosstab

Solution (PySpark)

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

Problem 135: 135 - Spark SQL: Functions in sql challenge

Problem

Spark SQL

Solution (PySpark)

```
clicks.createOrReplaceTempView("clicks_view")
res = spark.sql("SELECT user_id, COUNT(*) AS cnt FROM clicks_view GROUP BY user_id")
```

Problem 136: 136 - Dates & Timestamps: Date_trunc challenge

Problem

Dates & Timestamps

Solution (PySpark)

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

Problem 137: 137 - Dates & Timestamps: Date_trunc challenge

Problem

Dates & Timestamps

Solution (PySpark)

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

Problem 138: 138 - DataFrame Basics: Withcolumn challenge

Problem

DataFrame Basics

Solution (PySpark)

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

Problem 139: 139 - Graph-ish / Hierarchical: Self-join paths challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 140: 140 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 141: 141 - DataFrame Basics: Drop challenge

Problem

DataFrame Basics

Solution (PySpark)

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

Problem 142: 142 - Complex Types: Arrays challenge

Problem

Complex Types

Solution (PySpark)

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

Problem 143: 143 - Misc Utilities: Broadcast joins challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 144: 144 - Strings & Regex: Concat_ws challenge

Problem

Strings & Regex

Solution (PySpark)

```
from pyspark.sql import functions as F
res = transactions
if "email" in res.columns:
    res = res.withColumn("domain", F.regexp_extract("email", "@(.*)$", 1))
```

Problem 145: 145 - 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 146: 146 - Window Functions: Row_number challenge

Problem

Window Functions

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window
w = Window.partitionBy("user_id").orderBy("ts")
res = orders.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .w
```

Problem 147: 147 - Window Functions: Rank challenge

Problem

Window Functions

Solution (PySpark)

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window
w = Window.partitionBy("user_id").orderBy("ts")
res = orders.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .w
```

Problem 148: 148 - MLlib Basics: Vectorassembler 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(sessions)
res = model.transform(sessions)
```

Problem 149: 149 - 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 150: 150 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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