PySpark Interview Handbook — Batch 8

Problems 176–200
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 176: 176 - 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 177: 177 - DataFrame Basics: Distinct challenge

Problem

DataFrame Basics

Solution (PySpark)

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

Problem 178: 178 - Window Functions: Lag 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 = products.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \
```

Problem 179: 179 - Aggregations & GroupBy: Groupby challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

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

Problem 180: 180 - Complex Types: Structs challenge

Problem

Complex Types

Solution (PySpark)

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

Problem 181: 181 - Misc Utilities: Accumulators (concept) challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 182: 182 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 183: 183 - Misc Utilities: Broadcast joins challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 184: 184 - Joins: Inner 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 = orders.join(F.broadcast(users), "user_id", "left")
```

Problem 185: 185 - Dates & Timestamps: From_unixtime challenge

Problem

Dates & Timestamps

Solution (PySpark)

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

Problem 186: 186 - MLlib Basics: Pipeline 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(transactions)
res = model.transform(transactions)
```

Problem 187: 187 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 188: 188 - Spark SQL: Functions in sql challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 189: 189 - Joins: Inner 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 = events.join(F.broadcast(users), "user_id", "left")
```

Problem 190: 190 - Performance & Tuning: Aqe challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 191: 191 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 192: 192 - Graph-ish / Hierarchical: Self-join paths challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 193: 193 - Spark SQL: Create temp view challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 194: 194 - Complex Types: Arrays challenge

Problem

Complex Types

Solution (PySpark)

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

Problem 195: 195 - UDFs & Pandas UDFs: Returntype 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 = users.withColumn("score", score("value"))
```

Problem 196: 196 - UDFs & Pandas UDFs: Returntype 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 = orders.withColumn("score", score("value"))
```

Problem 197: 197 - Streaming (Structured): Trigger challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 198: 198 - Performance & Tuning: Coalesce challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 199: 199 - Strings & Regex: Regexp_replace challenge

Problem

Strings & Regex

Solution (PySpark)

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

Problem 200: 200 - 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)
```