PySpark Interview Handbook — Batch 1

Problems 001–025
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 001: 001 - Aggregations & GroupBy: Approx_count_distinct challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

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

Problem 002: 002 - Pivot & Crosstab: Cube challenge

Problem

Pivot & Crosstab

Solution (PySpark)

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

Problem 003: 003 - Strings & Regex: Concat_ws 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 004: 004 - 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 005: 005 - 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 = products.withColumn("score", score("value"))
```

Problem 006: 006 - Aggregations & GroupBy: Groupby 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 007: 007 - Dates & Timestamps: Date_trunc 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 008: 008 - 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(transactions)
res = model.transform(transactions)
```

Problem 009: 009 - Complex Types: Structs 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 010: 010 - Pivot & Crosstab: Cube 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 011: 011 - 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 = events.join(F.broadcast(users), "user_id", "left")
```

Problem 012: 012 - Aggregations & GroupBy: Approx_count_distinct 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 013: 013 - Window Functions: Dense_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 014: 014 - Complex Types: Structs 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 015: 015 - Pivot & Crosstab: Cube 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 016: 016 - Spark SQL: Functions in sql challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 017: 017 - Pivot & Crosstab: Cube 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 018: 018 - Window Functions: Lead 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 = users.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .wi
```

Problem 019: 019 - 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 = transactions.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \
```

Problem 020: 020 - Complex Types: Explode challenge

Problem

Complex Types

Solution (PySpark)

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

Problem 021: 021 - Dates & Timestamps: Date_trunc challenge

Problem

Dates & Timestamps

Solution (PySpark)

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

Problem 022: 022 - 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 023: 023 - Dates & Timestamps: Date_trunc 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 024: 024 - 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 025: 025 - DataFrame Basics: Filter challenge

Problem

DataFrame Basics

Solution (PySpark)

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