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
```

PySpark Interview Handbook — Batch 2

Problems 026–050
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 026: 026 - Spark SQL: Sql queries 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 027: 027 - Streaming (Structured): Trigger challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 028: 028 - 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 = clicks.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .w
```

Problem 029: 029 - 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 030: 030 - DataFrame Basics: Select 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 031: 031 - Misc Utilities: Accumulators (concept) challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 032: 032 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 033: 033 - Pivot & Crosstab: Pivot 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 034: 034 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 035: 035 - 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 036: 036 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 037: 037 - Dates & Timestamps: From_unixtime 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 038: 038 - Pivot & Crosstab: Rollup 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 039: 039 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 040: 040 - 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 = events.withColumn("score", score("value"))
```

Problem 041: 041 - Misc Utilities: Broadcast joins challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 042: 042 - 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(users)
res = model.transform(users)
```

Problem 043: 043 - 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 = logs.join(F.broadcast(users), "user_id", "left")
```

Problem 044: 044 - Misc Utilities: Accumulators (concept) challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 045: 045 - File IO & Formats: Orc challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 046: 046 - File IO & Formats: Json challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 047: 047 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 048: 048 - 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(transactions)
res = model.transform(transactions)
```

Problem 049: 049 - 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(users)
res = model.transform(users)
```

Problem 050: 050 - 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 = logs.withColumn("score", score("value"))
```

PySpark Interview Handbook — Batch 3

Problems 051–075
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 051: 051 - 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 = logs.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .wit
```

Problem 052: 052 - Aggregations & GroupBy: Agg challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

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

Problem 053: 053 - 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 054: 054 - Misc Utilities: Accumulators (concept) challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 055: 055 - Performance & Tuning: Repartition challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 056: 056 - 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(logs)
res = model.transform(logs)
```

Problem 057: 057 - Performance & Tuning: Checkpoint challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 058: 058 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 059: 059 - Joins: Right 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 = sessions.join(F.broadcast(users), "user_id", "left")
```

Problem 060: 060 - File IO & Formats: Delta-like challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 061: 061 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 062: 062 - Joins: Right 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 063: 063 - Performance & Tuning: Cache challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 064: 064 - Performance & Tuning: Skew 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 065: 065 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 066: 066 - Complex Types: Explode challenge

Problem

Complex Types

Solution (PySpark)

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

Problem 067: 067 - 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(logs)
res = model.transform(logs)
```

Problem 068: 068 - 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 = users.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .wi
```

Problem 069: 069 - Dates & Timestamps: From_unixtime challenge

Problem

Dates & Timestamps

Solution (PySpark)

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

Problem 070: 070 - Spark SQL: Sql queries 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 071: 071 - Strings & Regex: Regexp_replace 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 072: 072 - 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 073: 073 - Aggregations & GroupBy: Agg 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 074: 074 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 075: 075 - Joins: Anti 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 = logs.join(F.broadcast(users), "user_id", "left")
```

PySpark Interview Handbook — Batch 4

Problems 076–100
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 076: 076 - Strings & Regex: Concat_ws challenge

Problem

Strings & Regex

Solution (PySpark)

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

Problem 077: 077 - Complex Types: Structs 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 078: 078 - Performance & Tuning: Cache 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 079: 079 - DataFrame Basics: Drop challenge

Problem

DataFrame Basics

Solution (PySpark)

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

Problem 080: 080 - 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 081: 081 - File IO & Formats: Json challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 082: 082 - Performance & Tuning: Cache 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 083: 083 - 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 = clicks.withColumn("rn", F.row_number().over(w)) \    .withColumn("prev_value", F.lag("value", 1).over(w)) \    .w
```

Problem 084: 084 - Performance & Tuning: Skew challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 085: 085 - Performance & Tuning: Checkpoint 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 086: 086 - Streaming (Structured): Watermark challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 087: 087 - Complex Types: Maps challenge

Problem

Complex Types

Solution (PySpark)

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

Problem 088: 088 - Misc Utilities: Mappartitions challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 089: 089 - Joins: Right 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 = transactions.join(F.broadcast(users), "user_id", "left")
```

Problem 090: 090 - Performance & Tuning: Repartition challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 091: 091 - Misc Utilities: Broadcast joins challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 092: 092 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 093: 093 - DataFrame Basics: Select 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
```

Problem 094: 094 - UDFs & Pandas UDFs: 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 = clicks.withColumn("score", score("value"))
```

Problem 095: 095 - Misc Utilities: Broadcast joins challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 096: 096 - 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 097: 097 - Graph-ish / Hierarchical: Recursive-like with joins challenge

Problem

Graph-ish / Hierarchical

Solution (PySpark)

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

Problem 098: 098 - 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 = clicks.join(F.broadcast(users), "user_id", "left")
```

Problem 099: 099 - Performance & Tuning: Aqe challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 100: 100 - Streaming (Structured): Readstream challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

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"))
```

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
```

PySpark Interview Handbook — Batch 7

Problems 151–175
Generated: 2025-09-13 04:05:06Z (UTC)

Problem 151: 151 - Complex Types: Maps 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 152: 152 - Misc Utilities: Mappartitions challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 153: 153 - Performance & Tuning: Coalesce challenge

Problem

Performance & Tuning

Solution (PySpark)

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

Problem 154: 154 - Strings & Regex: Split challenge

Problem

Strings & Regex

Solution (PySpark)

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

Problem 155: 155 - Pivot & Crosstab: Pivot 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 156: 156 - 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 157: 157 - File IO & Formats: Orc challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 158: 158 - Joins: Right 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 159: 159 - Complex Types: Maps 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 160: 160 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 161: 161 - File IO & Formats: Orc challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 162: 162 - 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 163: 163 - Complex Types: Arrays challenge

Problem

Complex Types

Solution (PySpark)

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

Problem 164: 164 - 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(orders)
res = model.transform(orders)
```

Problem 165: 165 - DataFrame Basics: Select challenge

Problem

DataFrame Basics

Solution (PySpark)

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

Problem 166: 166 - Spark SQL: Create temp view 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 167: 167 - Aggregations & GroupBy: Agg challenge

Problem

Aggregations & GroupBy

Solution (PySpark)

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

Problem 168: 168 - Spark SQL: Sql queries challenge

Problem

Spark SQL

Solution (PySpark)

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

Problem 169: 169 - File IO & Formats: Parquet challenge

Problem

File IO & Formats

Solution (PySpark)

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

Problem 170: 170 - Complex Types: Maps 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 171: 171 - Dates & Timestamps: From_unixtime 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 172: 172 - Misc Utilities: Accumulators (concept) challenge

Problem

Misc Utilities

Solution (PySpark)

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

Problem 173: 173 - Streaming (Structured): Watermark challenge

Problem

Streaming (Structured)

Solution (PySpark)

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

Problem 174: 174 - DataFrame Basics: Drop 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 175: 175 - Strings & Regex: Concat_ws challenge

Problem

Strings & Regex

Solution (PySpark)

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

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)
```