PySpark Top 50 — Focus Pack

1. Window Functions & Analytics: Advanced Task on `transactions`

Question

Scenario. You have a large `transactions` dataset with columns like `user_id`, `created_at`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a window functions & analytics problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Window Functions & Analytics (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

2. Complex Joins & Skew Handling: Advanced Task on `orders`

Question

Scenario. You have a large `orders` dataset with columns like `customer_id`, `event_time`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a complex joins & skew handling problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Complex Joins & Skew Handling (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

3. Nested JSON & Semi-structured Data: Advanced Task on `metrics`

Question

Scenario. You have a large `metrics` dataset with columns like `user_id`, `updated_at`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a nested json & semi-structured data problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Nested JSON & Semi-structured Data (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

4. UDFs vs Pandas UDFs & Vectorization: Advanced Task on `events`

Question

Scenario. You have a large `events` dataset with columns like `user_id`, `ts`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a udfs vs pandas udfs & vectorization problem: - Ingest data with proper schema handling. - Apply necessary transformations

(null-safety, casting, deduplication). - Implement the core logic related to UDFs vs Pandas UDFs & Vectorization (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

5. Stateful Structured Streaming: Advanced Task on `metrics`

Question

Scenario. You have a large `metrics` dataset with columns like `session_id`, `event_time`, and `latency_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a stateful structured streaming problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Stateful Structured Streaming (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

6. Watermarking & Late Data: Advanced Task on `orders`

Question

Scenario. You have a large `orders` dataset with columns like `order_id`, `updated_at`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a watermarking & late data problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Watermarking & Late Data (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

7. Checkpointing & Exactly-once Semantics: Advanced Task on `impressions`

Question

Scenario. You have a large `impressions` dataset with columns like `session_id`, `created_at`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a checkpointing & exactly-once semantics problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Checkpointing & Exactly-once Semantics (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

8. File-based Incremental Ingestion: Advanced Task on `clicks`

Question

Scenario. You have a large `clicks` dataset with columns like `order_id`, `updated_at`, and `score`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a file-based incremental ingestion problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to File-based Incremental Ingestion (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

9. Delta Lake Optimize/Z-Order (conceptual with PySpark): Advanced Task on `sessions`

Question

Scenario. You have a large `sessions` dataset with columns like `customer_id`, `ts`, and `score`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a delta lake optimize/z-order (conceptual with pyspark) problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Delta Lake Optimize/Z-Order (conceptual with PySpark) (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

10. CDC/Merge into Delta (conceptual with PySpark): Advanced Task on `transactions`

Question

Scenario. You have a large `transactions` dataset with columns like `user_id`, `updated_at`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a cdc/merge into delta (conceptual with pyspark) problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to CDC/Merge into Delta (conceptual with PySpark) (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

11. Bucketing, Partitioning & Writer Jobs: Advanced Task on `logs`

Question

Scenario. You have a large `logs` dataset with columns like `account_id`, `created_at`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a bucketing, partitioning & writer jobs problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Bucketing, Partitioning & Writer Jobs (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

12. Adaptive Query Execution (AQE) and Shuffle Partitions: Advanced Task on `impressions`

Question

Scenario. You have a large `impressions` dataset with columns like `session_id`, `event_time`, and `duration_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a adaptive query execution (aqe) and shuffle partitions problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Adaptive Query Execution (AQE) and Shuffle Partitions (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

13. Broadcast Joins and Hints: Advanced Task on `transactions`

Question

Scenario. You have a large `transactions` dataset with columns like `session_id`, `created_at`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a broadcast joins and hints problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Broadcast Joins and Hints (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

14. Skew Join Salting Techniques: Advanced Task on `logs`

Question

Scenario. You have a large `logs` dataset with columns like `session_id`, `ts`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a skew join salting techniques problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Skew Join Salting Techniques (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

15. Aggregations with Complex Grouping Sets: Advanced Task on `transactions`

Question

Scenario. You have a large `transactions` dataset with columns like `user_id`, `ts`, and `score`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a aggregations with complex grouping sets problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Aggregations with

Complex Grouping Sets (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

16. Explode + Window Hybrids: Advanced Task on `transactions`

Question

Scenario. You have a large `transactions` dataset with columns like `customer_id`, `event_time`, and `duration_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a explode + window hybrids problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Explode + Window Hybrids (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

17. Sessionization (clickstreams): Advanced Task on `sessions`

Question

Scenario. You have a large `sessions` dataset with columns like `device_id`, `created_at`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a sessionization (clickstreams) problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Sessionization (clickstreams) (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

18. Time-series Gaps & Islands: Advanced Task on `logs`

Question

Scenario. You have a large `logs` dataset with columns like `account_id`, `ts`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a time-series gaps & islands problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Time-series Gaps & Islands (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

19. Surrogate Keys & Deduplication: Advanced Task on `sessions`

Question

Scenario. You have a large `sessions` dataset with columns like `order_id`, `event_time`, and `latency_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a surrogate keys & deduplication problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Surrogate Keys & Deduplication (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

20. SCD Type 2 with MERGE logic (Delta/Parquet): Advanced Task on `clicks`

Question

Scenario. You have a large `clicks` dataset with columns like `session_id`, `ts`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a scd type 2 with merge logic (delta/parquet) problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to SCD Type 2 with MERGE logic (Delta/Parquet) (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

21. Advanced Window: Last non-null forward-fill: Advanced Task on `impressions`

Question

Scenario. You have a large `impressions` dataset with columns like `device_id`, `created_at`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a advanced window: last non-null forward-fill problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Advanced Window: Last non-null forward-fill (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

22. Top-K per Group at Scale: Advanced Task on `metrics`

Question

Scenario. You have a large `metrics` dataset with columns like `customer_id`, `created_at`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a top-k per group at scale problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Top-K per Group at Scale (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

23. Rolling Distinct Counts (HLL sketch concept): Advanced Task on `orders`

Question

Scenario. You have a large `orders` dataset with columns like `user_id`, `created_at`, and `duration_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a rolling distinct counts (hll sketch concept) problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Rolling Distinct Counts (HLL sketch concept) (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

24. Cross-file Schema Evolution: Advanced Task on `sessions`

Question

Scenario. You have a large `sessions` dataset with columns like `user_id`, `ts`, and `latency_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a cross-file schema evolution problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Cross-file Schema Evolution (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

25. Dynamic File Pruning: Advanced Task on `logs`

Question

Scenario. You have a large `logs` dataset with columns like `customer_id`, `updated_at`, and `duration_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a dynamic file pruning problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Dynamic File Pruning (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

26. Data Quality Checks & Expectations: Advanced Task on `impressions`

Question

Scenario. You have a large `impressions` dataset with columns like `customer_id`, `created_at`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a data quality checks & expectations problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Data Quality Checks &

Expectations (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

27. Unit Testing with pytest & chispa: Advanced Task on `orders`

Question

Scenario. You have a large `orders` dataset with columns like `order_id`, `created_at`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a unit testing with pytest & chispa problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Unit Testing with pytest & chispa (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

28. Performance Debugging with UI & Query Plans: Advanced Task on `payments`

Question

Scenario. You have a large `payments` dataset with columns like `session_id`, `updated_at`, and `score`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a performance debugging with ui & query plans problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Performance Debugging with UI & Query Plans (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

29. Caching vs Checkpointing vs Persist: Advanced Task on `orders`

Question

Scenario. You have a large `orders` dataset with columns like `customer_id`, `updated_at`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a caching vs checkpointing vs persist problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Caching vs Checkpointing vs Persist (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

30. Reusable Jobs & Parameterized Notebooks: Advanced Task on `events`

Question

Scenario. You have a large `events` dataset with columns like `user_id`, `ts`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a reusable jobs & parameterized notebooks problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Reusable Jobs & Parameterized Notebooks (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

31. DataFrame <-> Spark SQL Interop: Advanced Task on `clicks`

Question

Scenario. You have a large `clicks` dataset with columns like `order_id`, `updated_at`, and `latency_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a dataframe <-> spark sql interop problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to DataFrame <-> Spark SQL Interop (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

32. Pivot/Unpivot Large Datasets: Advanced Task on `transactions`

Question

Scenario. You have a large `transactions` dataset with columns like `device_id`, `updated_at`, and `latency_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a pivot/unpivot large datasets problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Pivot/Unpivot Large Datasets (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

33. Joins over Ranges (temporal joins): Advanced Task on `impressions`

Question

Scenario. You have a large `impressions` dataset with columns like `session_id`, `created_at`, and `latency_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a joins over ranges (temporal joins) problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Joins over Ranges (temporal joins) (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

34. Windowed UDAFs (via Pandas UDFs): Advanced Task on `events`

Question

Scenario. You have a large `events` dataset with columns like `order_id`, `event_time`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a windowed udafs (via pandas udfs) problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Windowed UDAFs (via Pandas UDFs) (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

35. Binary Files & Image Ingestion: Advanced Task on `metrics`

Question

Scenario. You have a large `metrics` dataset with columns like `account_id`, `created_at`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a binary files & image ingestion problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Binary Files & Image Ingestion (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

36. Graph-Style Problems without GraphFrames: Advanced Task on `sessions`

Question

Scenario. You have a large `sessions` dataset with columns like `device_id`, `ts`, and `duration_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a graph-style problems without graphframes problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Graph-Style Problems without GraphFrames (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

37. MLlib Pipelines with Custom Transformers: Advanced Task on `events`

Question

Scenario. You have a large `events` dataset with columns like `order_id`, `created_at`, and `latency_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a mllib pipelines with custom transformers problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to MLlib

Pipelines with Custom Transformers (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

38. Streaming Joins & State Timeout: Advanced Task on `clicks`

Question

Scenario. You have a large `clicks` dataset with columns like `session_id`, `event_time`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a streaming joins & state timeout problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Streaming Joins & State Timeout (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

39. Idempotent Sinks Design: Advanced Task on `sessions`

Question

Scenario. You have a large `sessions` dataset with columns like `order_id`, `ts`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a idempotent sinks design problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Idempotent Sinks Design (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

40. Out-of-order Event Handling: Advanced Task on `logs`

Question

Scenario. You have a large `logs` dataset with columns like `customer_id`, `event_time`, and `latency_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a out-of-order event handling problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Out-of-order Event Handling (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

41. Checkpoint Recovery Simulation: Advanced Task on `logs`

Question

Scenario. You have a large `logs` dataset with columns like `device_id`, `event_time`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a checkpoint recovery simulation problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Checkpoint Recovery Simulation (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

42. File Compaction Job: Advanced Task on `logs`

Question

Scenario. You have a large `logs` dataset with columns like `account_id`, `ts`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a file compaction job problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to File Compaction Job (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

43. Small-file Problem Mitigation: Advanced Task on `orders`

Question

Scenario. You have a large `orders` dataset with columns like `session_id`, `event_time`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a small-file problem mitigation problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Small-file Problem Mitigation (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

44. Reading from Hive Metastore & External Tables: Advanced Task on `impressions`

Question

Scenario. You have a large `impressions` dataset with columns like `user_id`, `ts`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a reading from hive metastore & external tables problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Reading from Hive Metastore & External Tables (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

45. Security & PII Masking Patterns: Advanced Task on `impressions`

Question

Scenario. You have a large `impressions` dataset with columns like `session_id`, `ts`, and `score`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a security & pii masking patterns problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Security & PII Masking Patterns (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

46. Column-level Encryption (conceptual + UDF demo): Advanced Task on `metrics`

Question

Scenario. You have a large `metrics` dataset with columns like `session_id`, `updated_at`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a column-level encryption (conceptual + udf demo) problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Column-level Encryption (conceptual + UDF demo) (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

47. Debugging Serialization / Pickling issues: Advanced Task on `metrics`

Question

Scenario. You have a large `metrics` dataset with columns like `order_id`, `ts`, and `quantity`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a debugging serialization / pickling issues problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Debugging Serialization / Pickling issues (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

48. Handling Very Wide Schemas: Advanced Task on `sessions`

Question

Scenario. You have a large `sessions` dataset with columns like `device_id`, `created_at`, and `duration_ms`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a handling very wide schemas problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Handling Very Wide Schemas (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

49. Reading Multi-line JSON & Corrupt Records: Advanced Task on `metrics`

Question

Scenario. You have a large `metrics` dataset with columns like `device_id`, `event_time`, and `value`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a reading multi-line json & corrupt records problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Reading Multi-line JSON & Corrupt Records (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).

50. Optimizing fromRDD / mapPartitions: Advanced Task on `orders`

Question

Scenario. You have a large `orders` dataset with columns like `user_id`, `created_at`, and `amount`. The data arrives from multiple sources as Parquet/JSON with evolving schemas.

Task. Using PySpark, implement a robust solution to solve a optimizing fromrdd / mappartitions problem: - Ingest data with proper schema handling. - Apply necessary transformations (null-safety, casting, deduplication). - Implement the core logic related to Optimizing fromRDD / mapPartitions (detailed below). - Produce an optimized output suitable for downstream consumption (partitioning/bucketing where applicable).