PySpark Lab 1 – Complete Solutions (Q1 – Q4)

1. Operations on CSV File

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder.appName("Lab1_Exp1_CSV").getOrCreate()

# (a) Read data from CSV File
df = spark.read.csv("input.csv", header=True, inferSchema=True)

# (b) Get basic statistics
df.describe().show()

# (c) Count total number of rows
row_count = df.count()
print("Total rows:", row_count)

# (d) Number of unique values in a column (example: 'Name')
unique_count = df.select("Name").distinct().count()
print("Unique values in Name:", unique_count)

# (e) Update specific value (example: change Age 25 to 26)
df_updated = df.withColumn(
"Age",
F.when(F.col("Age") == 25, 26).otherwise(F.col("Age"))
)

# (f) Write to CSV File
df_updated.write.csv("output.csv", header=True)
```

2. Operations on JSON File

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder.appName("Lab1_Exp2_JSON").getOrCreate()

# (a) Read data from JSON file
df = spark.read.json("input.json")

# (b) Count total number of records
record_count = df.count()
print("Total records:", record_count)

# (c) Insert new record (example: Name, Age)
new_data = [("David", 40)]
new_df = spark.createDataFrame(new_data, ["Name", "Age"])
df_combined = df.union(new_df)

# (d) Update a specific record (example: change Alice's Age to 28)
df_updated = df_combined.withColumn(
"Age",
F.when(F.col("Name") == "Alice", 28).otherwise(F.col("Age"))
```

```
)

# (e) Write to JSON file
df_updated.write.json("output.json")
```

3. Text Processing and Word Count

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder.appName("Lab1_Exp3_TextProcessing").getOrCreate()

# (a) Ingest selected text file into a DataFrame
df = spark.read.text("input_text.txt") # one line per row, column 'value'

# (b) 'value' column is already StringType, one record per line

# (c) Remove punctuation, quotation marks, and symbols using regex
df_clean = df.select(
F.regexp_replace("value", r"[^a-zA-Z0-9\s]", " ").alias("clean_text")
)

# (d) Convert to lowercase
df_clean = df_clean.select(F.lower("clean_text").alias("clean_text"))

# Stopwords list
stopwords = ["is", "the", "and", "or", "a", "an", "to", "of", "in"]

# (e) Split each string into a list of words (tokenization)
df_words = df_clean.select(
F.split("clean_text", r"\s+").alias("words")
)

# Explode words so that each row has a single word
df_tokens = df_words.select(F.explode("words").alias("word"))

# Remove empty strings and stopwords
df_tokens_filtered = df_tokens.where(
(F.col("word") != "") & (~F.col("word").isin(stopwords))
)

# (f) Count frequency of each word present in the text
word_counts = df_tokens_filtered.groupBy("word").count().orderBy(F.desc("count"))

word_counts.show(truncate=False)
```

4. Select Operations and Array Column Access

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder.appName("Lab1_Exp4_SelectOperations").getOrCreate()

# (a) Ingest selected text file into a DataFrame
df = spark.read.text("input_text.txt") # one line per row with 'value'
```

```
# (b) 'value' is StringType, one record per line

# Add an ID/line number column for demonstration
df_with_id = df.withColumn("line_number", F.monotonically_increasing_id())

# (c.i) Selecting specific columns by name
selected_by_name = df_with_id.select("line_number", "value")
selected_by_name.show(truncate=False)

# (c.ii) Selecting columns with expressions (e.g., transforming data)
selected_with_expr = df_with_id.select(
"line_number",
F.length("value").alias("line_length"),
F.upper("value").alias("value_upper")
)
selected_with_expr.show(truncate=False)

# (c.iii) Selecting specific elements from an array column (first word)
df_words = df_with_id.select(
"line_number",
F.split("value", r"\s+").alias("words")
)

first_word_df = df_words.select(
"line_number",
F.col("words")[0].alias("first_word")
)
first_word_df.show(truncate=False)
```

PySpark Lab 1 – Solutions (Q5 – Q8)

5. Selecting Columns and Ordering Results

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("Lab1_Exp5_SelectAndOrder").getOrCreate()

# (a) Ingest selected text file into a DataFrame
# One line per row, default column name is 'value'
df = spark.read.text("input_text.txt")

# (b) 'value' column is already StringType, one record per line

# (c.i) Selecting columns using col() function
df_selected_col = df.select(col("value"))
df_selected_col.show(truncate=False)

# (c.ii) Selecting with column transformations (e.g., changing case)
df_transformed = df.select(
col("value"),
F.upper(col("value")).alias("value_upper"),
F.lower(col("value")).alias("value_lower")
)
df_transformed.show(truncate=False)

# (c.iii) Selecting exploded words to create a new row for each word
df_words = df.select(
F.split(col("value"), r"\s+").alias("words")
)
df_exploded = df_words.select(
F.explode(col("words")).alias("word")
)
df_exploded.show(truncate=False)

# (d) Ordering the results using orderBy
# Example: count word frequency and show top 10
word_counts = df_exploded.groupBy("word").count()

# Remove empty strings
word_counts = word_counts.where(col("word") != "")

top_10 = word_counts.orderBy(col("count").desc()).limit(10)
top_10.show(truncate=False)

# Similarly, to get top 20, 50, 100:
# top_20 = word_counts.orderBy(col("count").desc()).limit(20)
# top_50 = word_counts.orderBy(col("count").desc()).limit(50)
# top_100 = word_counts.orderBy(col("count").desc()).limit(100)
```

6. Filtering Text and Word Counts

```
from pyspark.sql import SparkSession
```

```
from pyspark.sql import functions as F
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("Lab1_Exp6_Filtering").getOrCreate()

# (a) Ingest selected text file into a DataFrame
df = spark.read.text("input_text.txt") # column 'value'

# (b) 'value' is StringType, one record per line

# Prepare a word-count DataFrame for numeric filters
df_clean = df.select(
F.regexp_replace("value", r"[^a-zA-Z0-9\s]", " ").alias("clean_text")
)
df_clean = df_clean.select(F.lower(col("clean_text")).alias("clean_text"))

df_words = df_clean.select(
F.split(col("clean_text"), r"\s+").alias("words")
)
df_exploded = df_words.select(F.explode(col("words")).alias("word"))
df_exploded = df_exploded.where(col("word") != "")

word_counts = df_exploded.groupBy("word").count()

# (c.i) Filtering rows with specific word (lines that contain 'Big')
lines_with_Big = df.filter(col("value").contains("Big"))
lines_with_Big.show(truncate=False)

# (c.ii) Filtering numeric values
# Words with count greater than 5
words_count_gt_5 = word_counts.where(col("count") > 5)
words_count_gt_5.show(truncate=False)

# Words with count between 3 and 10
words_count_between = word_counts.where(
(col("count") >= 3) & (col("count") <= 10)
)
words_count_between.show(truncate=False)

# (c.iii) Filtering rows with multiple conditions
# Lines where text contains both 'Big' and 'Data'
lines_with_Big_and_Data = df.where(
col("value").contains("Big") & col("value").contains("Data")
)
lines_with_Big_and_Data.show(truncate=False)
```

7. Read, Filter, Select, Rename, and Write MySQL Table

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("Lab1_Exp7_MySQL_ReadFilter").getOrCreate()

# MySQL connection properties
jdbc_url = "jdbc:mysql://localhost:3306/mydatabase" # change to your DB
table_name = "people" # change to your table
properties = {
```

```
"user": "root", # your MySQL username
"password": "password", # your MySQL password
"driver": "com.mysql.cj.jdbc.Driver"
}

# (a) Read data from MySQL Table
df = spark.read.jdbc(url=jdbc_url, table=table_name, properties=properties)

# (b) Filtering Data, example: Get rows where Age > 30
df_age_gt_30 = df.where(col("Age") > 30)

# (c) Filter with multiple conditions, example: Age > 30 and City = 'London'
df_filtered_multi = df.where(
(col("Age") > 30) & (col("City") == "London")
)

# (d) Select specific columns, example: Name and Age
df_selected_cols = df_filtered_multi.select("Name", "Age")

# (e) Rename a column, example: rename 'Age' to 'PersonAge'
df_renamed = df_selected_cols.withColumnRenamed("Age", "PersonAge")

df_renamed.show(truncate=False)

# (f) Write to MySQL Table (e.g., to a new table 'people_filtered')
output_table = "people_filtered"
df_renamed.write.jdbc(url=jdbc_url, table=output_table, mode="overwrite",
properties=properties)
```

8. Add/Drop Columns, Sort, Group & Aggregate in MySQL Table

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("Lab1_Exp8_MySQL_Transform").getOrCreate()

# MySQL connection properties
jdbc_url = "jdbc:mysql://localhost:3306/mydatabase" # change to your DB
table_name = "people" # change to your table
properties = {
"user": "root", # your MySQL username
"password": "password", # your MySQL password
"driver": "com.mysql.cj.jdbc.Driver"
}

# (a) Read data from MySQL Table
df = spark.read.jdbc(url=jdbc_url, table=table_name, properties=properties)

# (b) Add a new column, example: add a constant column 'Status' = 'Active'
df_added = df.withColumn("Status", F.lit("Active"))

# (c) Drop a column, example: drop 'Status' column
df_dropped = df_added.drop("Status")

# (d) Sorting Data, example: Sort by Age (ascending)
df_sorted = df_dropped.orderBy(col("Age").asc())
```

```
# (e) Grouping and Aggregations
# Example: Find average age and count of people by City
df_grouped = df_sorted.groupBy("City").agg(
F.count("*").alias("count_people"),
F.avg("Age").alias("avg_age")
)

df_grouped.show(truncate=False)

# (f) Write to MySQL Table (e.g., to a new summary table 'people_summary')
output_table = "people_summary"
df_grouped.write.jdbc(url=jdbc_url, table=output_table, mode="overwrite",
properties=properties)
```

HDFS & PySpark Lab – Solutions (Q9 – Q11)

9. Basic HDFS Operations (hdfs dfs)

```
# a. Create a Directory in HDFS
hdfs dfs -mkdir /user/hduser/example_dir

# b. List files and directories in HDFS
hdfs dfs -ls /user/hduser
hdfs dfs -ls /user/hduser/example_dir

# c. Upload a File to HDFS
# Assume local file: /home/hduser/sample.txt
hdfs dfs -put /home/hduser/sample.txt /user/hduser/example_dir/

# d. Read a File from HDFS
hdfs dfs -cat /user/hduser/example_dir/sample.txt

# e. Remove a File/Directory from HDFS
# Remove a file
hdfs dfs -rm /user/hduser/example_dir/sample.txt

# Remove a directory (recursively)
hdfs dfs -rm -r /user/hduser/example_dir

# f. Overwrite an Existing File Using -put -f
# Recreate directory and upload file with force overwrite
hdfs dfs -mkdir /user/hduser/example_dir
hdfs dfs -put /home/hduser/sample.txt /user/hduser/example_dir/sample.txt

# Overwrite with a new local file
hdfs dfs -put -f /home/hduser/sample_new.txt /user/hduser/example_dir/sample.txt

# g. Read the File After Overwriting
hdfs dfs -cat /user/hduser/example_dir/sample.txt

# h. Download a File from HDFS Using -get
hdfs dfs -get /user/hduser/example_dir/sample.txt
/home/hduser/downloaded_sample.txt
```

10. Basic HDFS Operations with copyFromLocal/copyToLocal

```
# a. Create a Directory in HDFS
hdfs dfs -mkdir /user/hduser/copy_example

# b. Upload a File to HDFS
# Assume local file: /home/hduser/data.txt
hdfs dfs -put /home/hduser/data.txt /user/hduser/copy_example/

# c. Read a File from HDFS
hdfs dfs -cat /user/hduser/copy_example/data.txt

# d. Overwrite an Existing File Using -copyFromLocal
# Modify local file /home/hduser/data_updated.txt and overwrite
hdfs dfs -copyFromLocal -f /home/hduser/data_updated.txt
```

```
/user/hduser/copy_example/data.txt

# e. Download a File Using -copyToLocal
hdfs dfs -copyToLocal /user/hduser/copy_example/data.txt
/home/hduser/data_from_hdfs.txt

# f. Download and Rename the File
# Here we give a different target name while copying
hdfs dfs -copyToLocal /user/hduser/copy_example/data.txt
/home/hduser/data_renamed.txt

# g. Check HDFS Storage Usage

# i. Show Total Storage Used by a directory
hdfs dfs -du -h /user/hduser

# ii. Show Available Storage in HDFS (filesystem report)
hdfs dfsadmin -report
```

11. Store and Retrieve Data from HDFS using PySpark

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("HDFS_PySpark_ReadWriteCSV").getOrCreate()

# a. Create a Sample CSV File on local system
# Example (run in Linux shell, not PySpark):
# echo "id,name,age" > sample.csv
# echo "1,Alice,25" >> sample.csv
# echo "2,Bob,30" >> sample.csv

# b. Upload to HDFS
# In terminal (not in PySpark):
# hdfs dfs -mkdir /user/hduser/hdfs_csv
# hdfs dfs -put sample.csv /user/hduser/hdfs_csv/

# c. Read CSV with PySpark from HDFS
df = spark.read.csv(
"hdfs://localhost:9000/user/hduser/hdfs_csv/sample.csv",
header=True,
inferSchema=True
)

# Verify DataFrame Output
df.show()
df.printSchema()

# d. Write the DataFrame Back to HDFS (overwrite)
df.write.mode("overwrite").csv(
"hdfs://localhost:9000/user/hduser/hdfs_csv/output_sample"
)

# e. Verify Output in HDFS
# Run in terminal:
# hdfs dfs -ls /user/hduser/hdfs_csv/output_sample
# hdfs dfs -cat /user/hduser/hdfs_csv/output_sample/part-*.csv
```

Big Data Lab – PySpark, HDFS & Kafka Solutions (Q12 – Q19)

12. Processing Large Log Files on HDFS with PySpark

```
# 12. Processing Large Log Files on HDFS

# a. Upload a Log File to HDFS (run in terminal)
# hdfs dfs -mkdir -p /user/hduser/logs
# hdfs dfs -put access.log /user/hduser/logs/

from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder.appName("Process_Logs_HDFS").getOrCreate()

# b. Read Logs Using PySpark
log_df = spark.read.text("hdfs://localhost:9000/user/hduser/logs/access.log")

# c. Extract Important Information
# Example: Simple Apache log parsing (IP, timestamp, request, status, size)
# Pattern will depend on your log format.
log_parsed_df = log_df.select(
F.regexp_extract("value", r"^(\S+)", 1).alias("ip"),
F.regexp_extract("value", r"\[(.*?)\]", 1).alias("timestamp"),
F.regexp_extract("value", r"\"(GET|POST|PUT|DELETE|HEAD)\s+(\S+)",
2).alias("endpoint"),
F.regexp_extract("value", r"\" \d{3}", 0).alias("status_raw"),
)

# Extract numeric status code
log_parsed_df = log_parsed_df.withColumn(
"status",
F.regexp_extract("status_raw", r"\d{3}", 0).cast("int")
).drop("status_raw")

# d. Save Processed Logs to HDFS
log_parsed_df.write.mode("overwrite").parquet(
"hdfs://localhost:9000/user/hduser/logs/processed_logs"
)
```

13. Analyzing Student Performance Data on HDFS with PySpark

```
# 13. Analyzing Student Performance Data

# a. Upload a Dataset (terminal)
# hdfs dfs -mkdir -p /user/hduser/student_data
# hdfs dfs -put students.csv /user/hduser/student_data/

from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder.appName("Student_Performance_HDFS").getOrCreate()
```

```
# b. Read and Process Data
# Example CSV columns: id,name,grade,marks
students_df = spark.read.csv(
"hdfs://localhost:9000/user/hduser/student_data/students.csv",
header=True,
inferSchema=True
)

# Example: Count students per grade
students_per_grade = students_df.groupBy("grade").count()

# verify the output
students_per_grade.show()

# c. Write Aggregated Data to HDFS
students_per_grade.write.mode("overwrite").csv(
"hdfs://localhost:9000/user/hduser/student_data/students_per_grade"
)
```

14. MapReduce Word Frequency Using PySpark with HDFS

```
# 14. Word Frequency (MapReduce-style) with PySpark

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("WordCount_HDFS").getOrCreate()
sc = spark.sparkContext

# Read file from HDFS as RDD
text_rdd =
sc.textFile("hdfs://localhost:9000/user/hduser/input/wordcount_input.txt")

# MapReduce steps
# Map: split lines into words, emit (word, 1)
words_rdd = text_rdd.flatMap(lambda line: line.split())

pairs_rdd = words_rdd.map(lambda w: (w, 1))

# Reduce: sum counts per word
word_counts_rdd = pairs_rdd.reduceByKey(lambda a, b: a + b)

# Save result back to HDFS
word_counts_rdd.saveAsTextFile("hdfs://localhost:9000/user/hduser/output/wordcoun
t_output")
```

15. MapReduce Matrix Multiplication Using PySpark with HDFS

```
# 15. Matrix Multiplication with PySpark (RDD, MapReduce-style)

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("MatrixMultiplication_HDFS").getOrCreate()
sc = spark.sparkContext

# Assume matrices A and B stored in HDFS as text files:
```

```
# A: rows of "i,k,value"
# B: rows of "k,j,value"

A_rdd = sc.textFile("hdfs://localhost:9000/user/hduser/matrix/A.csv") \
.map(lambda line: line.split(",")) \
.map(lambda x: (int(x[0]), int(x[1]), float(x[2]))) # (i, k, val)

B_rdd = sc.textFile("hdfs://localhost:9000/user/hduser/matrix/B.csv") \
.map(lambda line: line.split(",")) \
.map(lambda x: (int(x[0]), int(x[1]), float(x[2]))) # (k, j, val)

# Step 1: Map A to ((k), (i, valA)), Map B to ((k), (j, valB))
A_mapped = A_rdd.map(lambda x: (x[1], ("A", x[0], x[2]))) # (k, ("A", i, valA))
B_mapped = B_rdd.map(lambda x: (x[0], ("B", x[1], x[2]))) # (k, ("B", j, valB))

# Step 2: Join on k
joined = A_mapped.join(B_mapped)
# joined: (k, (("A", i, valA), ("B", j, valB)))

# Step 3: For each (i, j), multiply and sum
products = joined.map(
lambda x: ((x[1][0][1], x[1][1][1]), x[1][0][2] * x[1][1][2])
) # ((i,j), valA*valB)

result = products.reduceByKey(lambda a, b: a + b) # sum over k

# Save result to HDFS: lines of "i,j,value"
result.map(lambda x: f"{x[0][0]},{x[0][1]},{x[1]}") \
.saveAsTextFile("hdfs://localhost:9000/user/hduser/matrix/C_result")
```

16. Real-time Streaming from Kafka into Spark

```
# 16. Real-time Data Streaming from Kafka into Spark

from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder \
.appName("Kafka_Streaming_Basic") \
.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

# Read from Kafka topic 'input-topic'
kafka_df = spark.readStream \
.format("kafka") \
.option("kafka.bootstrap.servers", "localhost:9092") \
.option("subscribe", "input-topic") \
.option("startingOffsets", "latest") \
.load()

# Convert value from binary to string
value_df = kafka_df.selectExpr("CAST(value AS STRING) as message")

# Simple processing: add timestamp
processed_df = value_df.withColumn("processed_time", F.current_timestamp())

# Print output to console in real-time
```

```
query = processed_df.writeStream \
.outputMode("append") \
.format("console") \
.option("truncate", "false") \
.start()

query.awaitTermination()
```

17. Real-time Word Count from Kafka Messages

```
# 17. Real-time Word Count from Kafka

from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder \
.appName("Kafka_Streaming_WordCount") \
.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

lines = spark.readStream \
.format("kafka") \
.option("kafka.bootstrap.servers", "localhost:9092") \
.option("subscribe", "wordcount-topic") \
.option("startingOffsets", "latest") \
.load() \
.selectExpr("CAST(value AS STRING) as line")

# Split into words
words = lines.select(F.explode(F.split(F.col("line"), r"\s+")).alias("word"))

# Count words in a sliding window or global (here, update counts over time)
word_counts = words.groupBy("word").count()

# Write to console
query = word_counts.writeStream \
.outputMode("complete") \
.format("console") \
.option("truncate", "false") \
.start()

query.awaitTermination()
```

18. Real-time Temperature Sensor Data – Average Temperature

```
# 18. Real-time Temperature Sensor Data Processing

from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType, DoubleType

spark = SparkSession.builder \
.appName("Kafka_Temperature_Avg") \
.getOrCreate()
```

```
spark.sparkContext.setLogLevel("WARN")

# Kafka topic 'temperature-topic' sends JSON like:
# {"sensorId": "s1", "temperature": 28.5, "timestamp": "2025-12-07T12:34:56"}

schema = StructType([
StructField("sensorId", StringType(), True),
StructField("temperature", DoubleType(), True),
StructField("timestamp", StringType(), True)
])

raw_df = spark.readStream \
.format("kafka") \
.option("kafka.bootstrap.servers", "localhost:9092") \
.option("subscribe", "temperature-topic") \
.option("startingOffsets", "latest") \
.load()

json_df = raw_df.select(
F.from_json(F.col("value").cast("string"), schema).alias("data")
).select("data.*")

# Convert timestamp to proper type
temp_df = json_df.withColumn("event_time", F.to_timestamp("timestamp"))

# Compute average temperature over a 1-minute window, updated every 30 seconds
avg_temp_df = temp_df \
.withWatermark("event_time", "2 minutes") \
.groupBy(
F.window("event_time", "1 minute", "30 seconds"),
F.col("sensorId")
).agg(F.avg("temperature").alias("avg_temperature"))

query = avg_temp_df.writeStream \
.outputMode("update") \
.format("console") \
.option("truncate", "false") \
.start()

query.awaitTermination()
```

19. Real-time Temperature Alert System (Anomaly Detection)

```
# 19. Real-time Alert System for High Temperature

from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType, DoubleType

spark = SparkSession.builder \
.appName("Kafka_Temperature_Alert") \
.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

# Same JSON format as Q18:
# {"sensorId": "s1", "temperature": 28.5, "timestamp": "2025-12-07T12:34:56"}
```

```
schema = StructType([
StructField("sensorId", StringType(), True),
StructField("temperature", DoubleType(), True),
StructField("timestamp", StringType(), True)
])

raw_df = spark.readStream \
.format("kafka") \
.option("kafka.bootstrap.servers", "localhost:9092") \
.option("subscribe", "temperature-topic") \
.option("startingOffsets", "latest") \
.load()

json_df = raw_df.select(
F.from_json(F.col("value").cast("string"), schema).alias("data")
).select("data.*")

temp_df = json_df.withColumn("event_time", F.to_timestamp("timestamp"))

# Define temperature threshold
threshold = 50.0

# Filter readings above threshold and create alert message
alerts_df = temp_df.where(F.col("temperature") > threshold) \
.withColumn(
"alert_message",
F.concat(
F.lit("ALERT! High temperature detected. Sensor: "),
F.col("sensorId"),
F.lit(", Temperature: "),
F.col("temperature")
)
)

# For lab/demo, print alerts to console
# (In real system, this could be written to another Kafka topic, email, etc.)
query = alerts_df.select("sensorId", "temperature", "event_time",
"alert_message") \
.writeStream \
.outputMode("append") \
.format("console") \
.option("truncate", "false") \
.start()

query.awaitTermination()
```