SECTION 2: LAB ACTIVITIES

Activity 1: Spark SQL Fundamentals & Large Dataset Processing

Objectives: Load large datasets, execute SQL queries, understand lazy evaluation and caching

Time: 30 minutes

Part 1: Create the Script

```
cd ~/spark-kafka-lab
source venv/bin/activate

cat > activity1_spark_sql.py << 'PYEOF'
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, desc, avg, sum as spark_sum, count as spark_count,
row_number, round as spark_round
from pyspark.sql.window import Window
import time

print("=" * 70)
print("Activity 1: Spark SQL Fundamentals")
print("=" * 70)

# Create Spark Session
spark = SparkSession.builder \
    .appName("Activity1-SparkSQL") \
    .master("local[*]") \
    .config("spark.driver.memory", "4g") \
    .config("spark.sql.shuffle.partitions", "8") \
    .getOrCreate()

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

```
Spark Version: {spark.version}")
print(f"Running Mode: {spark.sparkContext.master}
")
PYEOF
```

Part 2: Add Data Loading

```
cat >> activity1_spark_sql.py << 'PYEOF'

# Load dataset
print("=" * 70)
print("STEP 1: Loading Dataset")
print("=" * 70)
start_time = time.time()
# NOTE: Use the taxi parquet files downloaded in setup (no script logic changed here)
df = spark.read.parquet("data/*.parquet")
load_time = time.time() - start_time
print(f"✓ Schema loaded in {load_time:.2f} seconds")

start_time = time.time()
count = df.count()
count_time = time.time() - start_time
print(f"✓ Total Records: {count:,}")
print(f"✓ Count time: {count_time:.2f} seconds")

print("
" + "=" * 70)
print("STEP 2: Data Schema and Sample")
print("=" * 70)
df.printSchema()
df.show(10, truncate=False)
PYEOF
```

Part 3: Add SQL Queries

```
cat >> activity1_spark_sql.py << 'PYEOF'

# SQL Queries
print("
" + "=" * 70)
print("STEP 3: SQL Query - Average Fare by Passenger Count")
print("=" * 70)
```

```
start_time = time.time()
df.createOrReplaceTempView("trips")

result1 = spark.sql("""
    SELECT
        passenger_count,
        COUNT(*) as trip_count,
        ROUND(AVG(fare_amount), 2) as avg_fare,
        ROUND(AVG(tip_amount), 2) as avg_tip,
        ROUND(AVG(trip_distance), 2) as avg_distance
    FROM trips
    GROUP BY passenger_count
    ORDER BY passenger_count
""")
result1.show()
query1_time = time.time() - start_time
print(f"✓ Query time: {query1_time:.2f} seconds")

print("
" + "=" * 70)
print("STEP 4: SQL Query - Payment Type Analysis")
print("=" * 70)
start_time = time.time()
result2 = spark.sql("""
    SELECT
        payment_type,
        COUNT(*) as transaction_count,
        ROUND(SUM(total_amount), 2) as total_revenue,
        ROUND(AVG(total_amount), 2) as avg_transaction
    FROM trips
    GROUP BY payment_type
    ORDER BY total_revenue DESC
""")
result2.show()
query2_time = time.time() - start_time
print(f"✓ Query time: {query2_time:.2f} seconds")
PYEOF
```

Part 4: Add Caching Demo

```
cat >> activity1_spark_sql.py << 'PYEOF'

# Caching
print("
" + "=" * 70)
print("STEP 5: Caching Demonstration")
print("=" * 70)

print("Query WITHOUT cache:")
start_time = time.time()
df.groupBy("payment_type").count().show()
time_no_cache = time.time() - start_time
print(f"✓ Time without cache: {time_no_cache:.2f} seconds")

print("
Caching dataframe...")
df.cache()
df.count()

print("
Query WITH cache:")
start_time = time.time()
df.groupBy("payment_type").count().show()
time_with_cache = time.time() - start_time
print(f"✓ Time with cache: {time_with_cache:.2f} seconds")
print(f"✓ Speedup: {time_no_cache/time_with_cache:.2f}x")

spark.stop()
print("
✓ Activity 1 Complete!")
PYEOF
```

Part 5: Run the Script

```
mkdir -p output
python3 activity1_spark_sql.py
```

REQUIRED SCREENSHOTS

Screenshot 1: Complete terminal output showing all 5 steps executed successfully

CRITICAL QUESTIONS

Question 1: Explain the difference between lazy evaluation and action in Spark. Which operations in your code were transformations and which were actions?

Question 2: What was your caching speedup? Explain why caching improves performance and when it would NOT be beneficial to cache a DataFrame.

Activity 2: Performance Benchmarking (Single-Node vs Cluster)

Objectives: Compare performance with different core counts, analyze parallelism benefits
Time: 40 minutes

Part 1: Create Single-Node Benchmark

```
cat > activity2_single_node.py << 'PYEOF'
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, desc, avg, sum as spark_sum, count as spark_count
import time

def run_benchmark(cores, app_name):
    print(f"
{'='*70}")
    print(f"BENCHMARK: {cores} Core(s)")
    print(f"{'='*70}
")

    spark = SparkSession.builder \
        .appName(app_name) \
        .master(f"local[{cores}]") \
        .config("spark.driver.memory", "4g") \
        .config("spark.sql.shuffle.partitions", str(cores * 2)) \
        .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")
    results = {}

    # Test 1: Load Data
    print("Test 1: Loading Data...")
    start = time.time()
    df = spark.read.parquet('data/*.parquet')
    record_count = df.count()
    results['load_time'] = time.time() - start
    print(f"  ✓ Loaded {record_count:,} records in {results['load_time']:.2f}s")

    df.cache()
    df.count()
```

```
    # Test 2: Aggregation
    print("Test 2: Aggregation...")
    start = time.time()
    df.groupBy("payment_type", "passenger_count") \
        .agg(spark_count("*").alias("cnt"), avg("fare_amount"), spark_sum("total_amount")) \
        .collect()
    results['aggregation_time'] = time.time() - start
    print(f"  ✓ Completed in {results['aggregation_time']:.2f}s")

    # Test 3: Filter
    print("Test 3: Filter & Sort...")
    start = time.time()
    df.filter(col("fare_amount") > 50).orderBy(desc("fare_amount")).limit(1000).collect()
    results['filter_time'] = time.time() - start
    print(f"  ✓ Completed in {results['filter_time']:.2f}s")

    # Test 4: Join
    print("Test 4: Join...")
    start = time.time()
    df_sample = df.sample(0.01)
    df_sample.alias("a").join(df_sample.alias("b"), col("a.passenger_count") ==
col("b.passenger_count")).count()
    results['join_time'] = time.time() - start
    print(f"  ✓ Completed in {results['join_time']:.2f}s")

    results['total_time'] = sum(results.values())
    spark.stop()
    return results
PYEOF
```

Part 2: Add Benchmark Loop and Results

```
cat >> activity2_single_node.py << 'PYEOF'

print("=" * 70)
print("SINGLE-NODE PERFORMANCE BENCHMARK")
print("=" * 70)

configs = [1, 2, 4]
all_results = {}

for cores in configs:
    all_results[cores] = run_benchmark(cores, f"Benchmark-{cores}Cores")
```

```
    time.sleep(3)

# Print comparison
print("
" + "=" * 85)
print("PERFORMANCE COMPARISON")
print("=" * 85)
print(f"{'Test':<20} {'1 Core':>12} {'2 Cores':>12} {'4 Cores':>12} {'Speedup':>12}")
print("-" * 85)

for test in ['load_time', 'aggregation_time', 'filter_time', 'join_time', 'total_time']:
    times = [all_results[cores][test] for cores in configs]
    speedup = times[0] / times[2]
    print(f"{test:<20} {times[0]:>10.2f}s {times[1]:>10.2f}s {times[2]:>10.2f}s {speedup:>11.2f}x")

print("=" * 85)

# Save results
with open('output/single_node_results.txt', 'w') as f:
    f.write("Single Node Benchmark Results
")
    f.write("="*50 + "

")
    for cores in configs:
        f.write(f"{cores} Core(s):
")
        for test, val in all_results[cores].items():
            f.write(f"  {test}: {val:.2f}s
")
        f.write("
")

print("
✓ Results saved to output/single_node_results.txt")
PYEOF
```

Part 3: Run Single-Node Benchmark

python3 activity2_single_node.py

Part 4: Create Cluster Benchmark

```
cat > activity2_cluster.py << 'PYEOF'
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, desc, avg, sum as spark_sum, count as spark_count
import time

def run_cluster_benchmark(cores, app_name):
    print(f"
{'='*70}")
    print(f"CLUSTER BENCHMARK: {cores} Cores")
    print(f"{'='*70}
")

    spark = SparkSession.builder \
        .appName(app_name) \
        .master("spark://localhost:7077") \
        .config("spark.executor.memory", "2g") \
        .config("spark.driver.memory", "2g") \
        .config("spark.cores.max", str(cores)) \
        .config("spark.sql.shuffle.partitions", str(cores * 4)) \
        .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")
    results = {}

    print("Test 1: Loading...")
    start = time.time()
    df = spark.read.parquet('data/*.parquet')
    df.count()
    results['load_time'] = time.time() - start
    print(f"  ✓ {results['load_time']:.2f}s")

    df = df.repartition(cores * 4).cache()
    df.count()

    print("Test 2: Aggregation...")
    start = time.time()
    df.groupBy("payment_type").agg(spark_count("*"), avg("fare_amount")).collect()
```

```
    results['aggregation_time'] = time.time() - start
    print(f"  ✓ {results['aggregation_time']:.2f}s")

    print("Test 3: Filter...")
    start = time.time()
    df.filter(col("fare_amount") > 50).limit(10000).collect()
    results['filter_time'] = time.time() - start
    print(f"  ✓ {results['filter_time']:.2f}s")

    print("Test 4: Join...")
    start = time.time()
    df_left = df.sample(0.02).repartition(cores, "passenger_count")
    df_right = df.sample(0.02).repartition(cores, "passenger_count")
    df_left.join(df_right, "passenger_count").count()
    results['join_time'] = time.time() - start
    print(f"  ✓ {results['join_time']:.2f}s")

    results['total_time'] = sum(results.values())
    spark.stop()
    time.sleep(5)
    return results

print("=" * 70)
print("CLUSTER PERFORMANCE BENCHMARK")
print("=" * 70)

configurations = [2, 4, 6]
all_results = {}

for cores in configurations:
    all_results[cores] = run_cluster_benchmark(cores, f"Cluster-{cores}Cores")

# Comparison
print("
" + "=" * 85)
print("CLUSTER COMPARISON")
print("=" * 85)
print(f"{'Test':<20} {'2 Cores':>12} {'4 Cores':>12} {'6 Cores':>12} {'Speedup':>12}")
print("-" * 85)

for test in ['load_time', 'aggregation_time', 'filter_time', 'join_time', 'total_time']:
    times = [all_results[cores][test] for cores in configurations]
    speedup = times[0] / times[2]
    print(f"{test:<20} {times[0]:>10.2f}s {times[1]:>10.2f}s {times[2]:>10.2f}s {speedup:>11.2f}x")
```

```
print("=" * 85)

with open('output/cluster_results.txt', 'w') as f:
    f.write("Cluster Benchmark Results
")
    f.write("="*50 + "

")
    for cores in configurations:
        f.write(f"{cores} Cores:
")
        for test, val in all_results[cores].items():
            f.write(f"  {test}: {val:.2f}s
")
        f.write("
")

print("
✓ Results saved to output/cluster_results.txt")
PYEOF
```

Part 5: Run Cluster Benchmark

```
jps | grep -E "Master|Worker"  # Verify cluster running
python3 activity2_cluster.py
```

Part 6: View Results File

```
cat output/single_node_results.txt
cat output/cluster_results.txt
```

REQUIRED SCREENSHOTS

Screenshot 2.1: Terminal showing single-node benchmark comparison table (1, 2, 4 cores)

Screenshot 2.2: Terminal showing cluster benchmark comparison table (2, 4, 6 cores)

Screenshot 2.3: Content of output files showing detailed timing for all core configurations

CRITICAL QUESTIONS

Question 1: Which operation (load, aggregation, filter, join) showed the best speedup and why? Which operation showed the least improvement with more cores?

Question 2: Compare your single-node 4-core results with cluster 2-core results. Is cluster mode always faster? Explain the trade-offs between single-node and cluster execution.

Activity 3: Kafka Producer-Consumer & Real-Time Anomaly Detection

Objectives: Implement Kafka producer and consumer, detect anomalies in streaming sensor data

Part 1: Verify Kafka Topics

```
kafka-topics.sh --list --bootstrap-server localhost:9092
# Should show: sensor-data, transactions, alerts
```

Part 2: Create Kafka Producer

```
cat > activity3_producer.py << 'PYEOF'
from kafka import KafkaProducer
import json, time, random
from datetime import datetime
from faker import Faker

fake = Faker()
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

def gen_sensor():
    return {
        'sensor_id': f"SENSOR_{random.randint(1, 20):03d}",
        'timestamp': datetime.now().isoformat(),
        'temperature': round(random.uniform(15, 35), 2),
        'humidity': round(random.uniform(30, 80), 2),
        'pressure': round(random.uniform(980, 1050), 2),
        'status': random.choice(['normal', 'normal', 'normal', 'warning', 'critical'])
    }

print("=" * 70)
print("Kafka Producer - Sending Sensor Data")
print("=" * 70)

for i in range(500):
```

```
    msg = gen_sensor()
    producer.send('sensor-data', msg)

    if (i + 1) % 100 == 0:
        print(f"Sent {i+1} messages...")

    time.sleep(0.05)

producer.flush()
producer.close()
print("
✓ Complete! Sent 500 sensor readings")
PYEOF
```

Part 3: Create Kafka Consumer with Anomaly Detection

```
cat > activity3_consumer.py << 'PYEOF'
from kafka import KafkaConsumer
import json, signal, sys

consumer = KafkaConsumer(
    'sensor-data',
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest',
    group_id='anomaly-detector',
    value_deserializer=lambda x: json.loads(x.decode('utf-8'))
)

def signal_handler(sig, frame):
    consumer.close()
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)

print("=" * 70)
print("Kafka Consumer - Real-Time Anomaly Detection")
print("=" * 70)
print("Listening for sensor data... (Press Ctrl+C to stop)
")

count = 0
anomaly_count = 0
```

```
for msg in consumer:
    data = msg.value
    count += 1

    # Anomaly Detection Logic
    is_anomaly = False
    reasons = []

    if data['temperature'] > 32:
        is_anomaly = True
        reasons.append(f"High temp: {data['temperature']}°C")

    if data['status'] == 'critical':
        is_anomaly = True
        reasons.append(f"Critical status")

    if is_anomaly:
        anomaly_count += 1
        print(f"
🚨 ANOMALY DETECTED (#{anomaly_count})")
        print(f"   Sensor: {data['sensor_id']}")
        print(f"   Time: {data['timestamp']}")
        print(f"   Reasons: {', '.join(reasons)}")
        print(f"   Full Data: {data}")

    if count % 50 == 0:
        print(f"
--- Processed: {count} | Anomalies: {anomaly_count} ---")
PYEOF
```

Part 4: Run Producer and Consumer

Terminal 1 - Consumer (Start First):

```
python3 activity3_consumer.py
```

Terminal 2 - Producer:

```
python3 activity3_producer.py
```

REQUIRED SCREENSHOTS

Screenshot 3.1: Terminal 1 (Consumer) showing anomaly detection in action with multiple anomalies detected

Screenshot 3.2: Terminal 2 (Producer) showing messages being sent successfully

CRITICAL QUESTIONS

Question 1: Explain how Kafka's producer-consumer model works. What happens if the consumer is slower than the producer?

Question 2: What anomalies did your system detect? Modify the anomaly detection logic to add one more condition (e.g., humidity threshold) and explain why that would be useful.

Activity 4: Spark Structured Streaming with Kafka

Objectives: Use Spark to consume Kafka streams, perform windowed aggregations, write to output topic
Time: 30 minutes

Part 1: Create Spark Streaming Script

```
cat > activity4_streaming.py << 'PYEOF'
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

schema = StructType([
    StructField("sensor_id", StringType()),
    StructField("timestamp", StringType()),
    StructField("temperature", DoubleType()),
    StructField("humidity", DoubleType()),
    StructField("pressure", DoubleType()),
    StructField("status", StringType())
])

spark = SparkSession.builder \
    .appName("Activity4-Streaming") \
    .master("spark://localhost:7077") \
    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0") \
    .getOrCreate()

spark.sparkContext.setLogLevel("WARN")
print("=" * 70)
print("Spark Structured Streaming - Kafka Integration")
print("=" * 70)
print("Reading from topic: sensor-data
")

# Read stream
raw = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "sensor-data") \
```

```
    .option("startingOffsets", "latest") \
    .load()

# Parse JSON
parsed = raw.selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
    .select("data.*") \
    .withColumn("event_time", to_timestamp(col("timestamp")))

# Filter for alerts
alerts = parsed.filter((col("temperature") > 30) | (col("status") == "critical")) \
    .select(col("sensor_id"), col("temperature"), col("status")) \
    .withColumn("alert_msg", concat(lit("ALERT: "), col("sensor_id"),
                                    lit(" - temp="), col("temperature"),
                                    lit(" status="), col("status")))

# Windowed aggregation (5-second windows)
windowed = parsed \
    .withWatermark("event_time", "5 seconds") \
    .filter(col("status") == "critical") \
    .groupBy(window(col("event_time"), "5 seconds"), col("sensor_id")) \
    .count() \
    .withColumnRenamed("count", "critical_count")

# Write alerts to Kafka
kafka_output = alerts.select(
    col("sensor_id").cast("string").alias("key"),
    to_json(struct("sensor_id", "temperature", "status", "alert_msg")).alias("value")
)

query1 = kafka_output.writeStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("topic", "alerts") \
    .option("checkpointLocation", "checkpoint/alerts") \
    .start()

# Write windowed counts to console
query2 = windowed.writeStream \
    .format("console") \
    .outputMode("update") \
    .option("truncate", "false") \
    .start()
```

```
print("Streaming queries active... Press Ctrl+C to stop")
spark.streams.awaitAnyTermination()
PYEOF
```

Part 2: Run Streaming Pipeline (3 Terminals)

Terminal 1 - Start Spark Streaming:

```
rm -rf checkpoint/
python3 activity4_streaming.py
```

Terminal 2 - Start Producer (continuous):

```
cat > continuous_producer.py << 'PYEOF'
from kafka import KafkaProducer
import json, time, random
from datetime import datetime

producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

def gen_sensor():
    return {
        'sensor_id': f"SENSOR_{random.randint(1, 20):03d}",
        'timestamp': datetime.now().isoformat(),
        'temperature': round(random.uniform(15, 35), 2),
        'humidity': round(random.uniform(30, 80), 2),
        'pressure': round(random.uniform(980, 1050), 2),
        'status': random.choice(['normal', 'normal', 'warning', 'critical'])
    }

print("Sending continuous sensor data... (Ctrl+C to stop)")
count = 0
try:
    while True:
        producer.send('sensor-data', gen_sensor())
        count += 1
        if count % 100 == 0:
            print(f"Sent {count} messages...")
        time.sleep(0.1)
```

```
except KeyboardInterrupt:
    producer.close()
    print(f"
Stopped. Total sent: {count}")
PYEOF

python3 continuous_producer.py
```

Terminal 3 - Monitor Alerts Topic:

```
kafka-console-consumer.sh \
  --bootstrap-server localhost:9092 \
  --topic alerts \
  --from-beginning
```

REQUIRED SCREENSHOTS

Screenshot 4.1: Terminal 1 showing Spark Streaming console output with windowed aggregation results (5-second windows showing critical counts)

Screenshot 4.2: Terminal 3 showing alerts being written to Kafka alerts topic in real-time

CRITICAL QUESTIONS

Question 1: Explain what watermarking does in the windowed aggregation. Why is it necessary for streaming applications?

Question 2: What is the difference between batch processing and stream processing? Give an example from this activity where streaming provides an advantage.

Activity 5: Real-Time Analytics Dashboard

Objectives: Build web dashboard with Flask-SocketIO to display live Kafka metrics
Time: 30 minutes

Part 1: Create Dashboard Server

```
cat > activity5_dashboard.py << 'PYEOF'
from flask import Flask, render_template
from flask_socketio import SocketIO, emit
from kafka import KafkaConsumer
import json, threading, time

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret'
socketio = SocketIO(app, cors_allowed_origins="*")

metrics = {
    'sensor_count': 0,
    'txn_count': 0,
    'current_temp': 0.0,
    'total_revenue': 0.0,
    'status': 'Running'
}

class KafkaThread(threading.Thread):
    def __init__(self, topic):
        threading.Thread.__init__(self, daemon=True)
        self.topic = topic

    def run(self):
        consumer = KafkaConsumer(
            self.topic,
            bootstrap_servers=['localhost:9092'],
            auto_offset_reset='latest',
            group_id=f'{self.topic}_dash',
            value_deserializer=lambda x: json.loads(x.decode('utf-8'))
        )

        for msg in consumer:
            data = msg.value
            if self.topic == 'sensor-data':
```

```
            metrics['sensor_count'] += 1
            metrics['current_temp'] = data.get('temperature', 0)
        elif self.topic == 'transactions':
            metrics['txn_count'] += 1
            metrics['total_revenue'] += data.get('amount', 0)

        socketio.emit('update', metrics, namespace='/live')

@app.route('/')
def index():
    return render_template('dashboard.html')

@socketio.on('connect', namespace='/live')
def handle_connect():
    emit('update', metrics)

if __name__ == '__main__':
    KafkaThread('sensor-data').start()
    KafkaThread('transactions').start()
    print("=" * 70)
    print("Real-Time Dashboard Server")
    print("=" * 70)
    print("Dashboard URL: http://localhost:5000")
    print("Press Ctrl+C to stop")
    print("=" * 70)
    socketio.run(app, host='0.0.0.0', port=5000, allow_unsafe_werkzeug=True)
PYEOF
```

Part 2: Create Dashboard HTML

```
mkdir -p templates
cat > templates/dashboard.html << 'HTMLEOF'
<!DOCTYPE html>
<html>
<head>
    <title>Real-Time Analytics Dashboard</title>
    <script src="https://cdn.socket.io/4.0.0/socket.io.min.js"></script>
    <style>
        body {
            font-family: Arial, sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            padding: 20px;
            margin: 0;
```

```
}
h1 {
    text-align: center;
    color: white;
    margin-bottom: 40px;
    text-shadow: 2px 2px 4px rgba(0,0,0,0.3);
}
.grid {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    gap: 20px;
    max-width: 1400px;
    margin: 0 auto;
}
.card {
    background: white;
    padding: 30px;
    border-radius: 15px;
    box-shadow: 0 10px 30px rgba(0,0,0,0.2);
    text-align: center;
    transition: transform 0.3s;
}
.card:hover {
    transform: translateY(-5px);
}
.value {
    font-size: 3em;
    font-weight: bold;
    color: #667eea;
    margin: 15px 0;
}
.label {
    font-size: 0.9em;
    color: #666;
    text-transform: uppercase;
    letter-spacing: 1px;
}
.status-indicator {
    position: fixed;
    top: 20px;
    right: 20px;
    padding: 10px 20px;
    background: #4caf50;
    color: white;
```

```
            border-radius: 20px;
            font-weight: bold;
            box-shadow: 0 4px 10px rgba(0,0,0,0.2);
        }
    </style>
</head>
<body>
    <div class="status-indicator">● Live</div>
    <h1>📊 Real-Time Analytics Dashboard</h1>
    <div class="grid">
        <div class="card">
            <div class="label">Sensor Readings</div>
            <div id="sensor" class="value">0</div>
        </div>
        <div class="card">
            <div class="label">Temperature (°C)</div>
            <div id="temp" class="value">0.0</div>
        </div>
        <div class="card">
            <div class="label">Transactions</div>
            <div id="txn" class="value">0</div>
        </div>
        <div class="card">
            <div class="label">Revenue ($)</div>
            <div id="revenue" class="value">0.00</div>
        </div>
    </div>
    <script>
        var socket = io('/live');
        socket.on('update', function(data) {
            document.getElementById('sensor').innerText = data.sensor_count.toLocaleString();
            document.getElementById('temp').innerText = data.current_temp.toFixed(1);
            document.getElementById('txn').innerText = data.txn_count.toLocaleString();
            document.getElementById('revenue').innerText = data.total_revenue.toLocaleString('en',
{minimumFractionDigits: 2});
        });
    </script>
</body>
</html>
HTMLEOF
```

Part 3: Create Transaction Producer

```
cat > transaction_producer.py << 'PYEOF'
from kafka import KafkaProducer
import json, time, random
from datetime import datetime
from faker import Faker

fake = Faker()
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

def gen_transaction():
    return {
        'transaction_id': fake.uuid4(),
        'timestamp': datetime.now().isoformat(),
        'user_id': f"USER_{random.randint(1, 500):05d}",
        'amount': round(random.uniform(5, 500), 2),
        'merchant': fake.company(),
        'category': random.choice(['Shopping', 'Food', 'Transport']),
    }

print("Sending transactions... (Ctrl+C to stop)")
count = 0
try:
    while True:
        producer.send('transactions', gen_transaction())
        count += 1
        if count % 50 == 0:
            print(f"Sent {count} transactions...")
        time.sleep(0.2)
except KeyboardInterrupt:
    producer.close()
    print(f"
Stopped. Total: {count}")
PYEOF
```

Part 4: Run Dashboard (3 Terminals)

Terminal 1 - Dashboard Server:

```
python3 activity5_dashboard.py
# Open browser: http://localhost:5000
```

Terminal 2 - Sensor Data Producer:

```
python3 continuous_producer.py
```

Terminal 3 - Transaction Producer:

```
python3 transaction_producer.py
```

REQUIRED SCREENSHOTS

Screenshot 5.1: Web browser showing the live dashboard with all 4 metrics updating in real-time (initial state)

Screenshot 5.2: Same dashboard after 1–2 minutes showing updated metrics (sensor count, temperature, transactions, revenue)

Screenshot 5.3: Terminal showing dashboard server logs with connection messages

CRITICAL QUESTIONS

Question 1: How does Flask-SocketIO enable real-time communication between the server and browser? What happens when multiple users access the dashboard simultaneously?

Question 2: Why do we use threading for Kafka consumers in the dashboard? What would happen if we didn't use threads?

Activity 6: Machine Learning with Spark MLlib

Objectives: Train a classification model using Spark MLlib, evaluate model performance

Part 1: Create ML Script

bash

```
cat > activity6_ml.py << 'PYEOF'
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline

print("=" * 70)
print("Activity 6: Machine Learning with Spark MLlib")
print("=" * 70)

spark = SparkSession.builder \
    .appName("Activity6-ML") \
    .master("local[*]") \
    .config("spark.driver.memory", "2g") \
    .getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

# Load Iris dataset
print("\nSTEP 1: Loading Iris Dataset")
print("=" * 70)
data = spark.read.csv("data/iris.csv", header=True, inferSchema=True)
print(f"Total samples: {data.count()}")
data.printSchema()
data.show(5)
PYEOF
```

Part 2: Add Feature Engineering

bash
cat >> activity6_ml.py << 'PYEOF'

```
# Feature Engineering
print("\nSTEP 2: Feature Engineering")
print("=" * 70)

# Convert species labels to numeric
indexer = StringIndexer(inputCol="species", outputCol="label")

# Combine features into vector
feature_cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

print("Feature columns:", feature_cols)
print("Label column: species (converted to numeric)")
```

PYEOF

Part 3: Add Training and Evaluation

bash
cat >> activity6_ml.py << 'PYEOF'

```
# Train/Test Split
print("\nSTEP 3: Train/Test Split")
print("=" * 70)
train_data, test_data = data.randomSplit([0.7, 0.3], seed=42)
print(f"Training samples: {train_data.count()}")
print(f"Test samples: {test_data.count()}")

# Build Pipeline
print("\nSTEP 4: Training Decision Tree Model")
print("=" * 70)
dt = DecisionTreeClassifier(labelCol="label", featuresCol="features", maxDepth=5)
pipeline = Pipeline(stages=[indexer, assembler, dt])

print("Training model...")
model = pipeline.fit(train_data)
print("✓ Training complete!")

# Make Predictions
```

```
print("\nSTEP 5: Making Predictions")
print("=" * 70)
predictions = model.transform(test_data)
predictions.select("features", "label", "prediction", "species").show(10)

# Evaluate Model
print("\nSTEP 6: Model Evaluation")
print("=" * 70)
evaluator = MulticlassClassificationEvaluator(
    labelCol="label",
    predictionCol="prediction",
    metricName="accuracy"
)

accuracy = evaluator.evaluate(predictions)
print(f"Test Accuracy: {accuracy:.4f} ({accuracy*100:.2f}%)")

# Confusion analysis
print("\nPrediction Summary:")
predictions.groupBy("label", "prediction").count().show()

spark.stop()
print("\n✓ Activity 6 Complete!")
```

PYEOF

Part 4: Run ML Pipeline

bash

```
python3 activity6_ml.py
```

REQUIRED SCREENSHOTS

Screenshot 6.1: Terminal output showing dataset loading, schema, and sample data (Steps 1-2)

Screenshot 6.2: Terminal showing predictions with features, actual label, and predicted label (Step 5)

Screenshot 6.3: Terminal showing final model accuracy and prediction summary (Step 6)

(Similar to the image showing ML model training results and accuracy)

CRITICAL QUESTIONS

Question 1: What was your model's accuracy? Analyze the prediction summary - which species (if any) was harder to classify correctly and why might that be?

Lab Completion & Submission

Final Checklist

- Activity 1: Spark SQL completed with 1 screenshot
- Activity 2: Performance benchmarks with 3 screenshots
- Activity 3: Kafka anomaly detection with 2 screenshots
- Activity 4: Spark streaming with 2 screenshots
- Activity 5: Dashboard with 3 screenshots
- Activity 6: Machine learning with 3 screenshots

Stop All Services

bash
 # Stop producers (Ctrl+C in terminals)

 # Stop Spark Streaming
 # Ctrl+C in streaming terminal

 # Stop Dashboard
 # Ctrl+C in dashboard terminal

 # Stop Spark Cluster
stop-worker.sh
stop-master.sh

 # Stop Kafka
kafka-server-stop.sh
zookeeper-server-stop.sh

 # Verify all stopped

jps *# Should only show Jp*

Submission Requirements

1. Lab Report Document

Create a document containing:

- **Cover Page:** Name, ERP, Date, Lab Title
- **Activity 1-6 Sections:**

- All required screenshots (labeled clearly)
- Answers to all critical questions
- Brief observations/notes for each activity

2. Code Files

Submit a ZIP file containing:

```
spark-kafka-lab/
├── activity1_spark_sql.py
├── activity2_single_node.py
├── activity2_cluster.py
├── activity3_producer.py
├── activity3_consumer.py
├── activity4_streaming.py
├── continuous_producer.py
├── activity5_dashboard.py
├── transaction_producer.py
├── activity6_ml.py
├── templates/
│   └── dashboard.html
└── output/
    ├── single_node_results.txt
    └── cluster_results.txt
```

Outcome & Achievements (Concise)

- **Dataset:** Implemented use of NYC Yellow Taxi Parquet (Jan–Mar 2023) as the primary large dataset; fallback sample created if download fails. Parquet improves IO and preserves schema.

- **Activity 1:** Loaded taxi data, ran SQL aggregations (avg fare by passenger_count, payment-type revenue), demonstrated caching speedup.

- **Activity 2:** Benchmarked single-node vs cluster runs (load, aggregation, filter, join) to measure parallelism benefits.

- **Activity 3:** Implemented Kafka producer and consumer; detected anomalies and validated message flow.

- **Activity 4:** Built Spark Structured Streaming pipeline consuming Kafka, produced alerts back to Kafka and performed windowed aggregations with watermarking.

- **Activity 5:** Built Flask-SocketIO dashboard streaming live metrics from Kafka topics; multi-client capable and thread-safe via consumer threads.

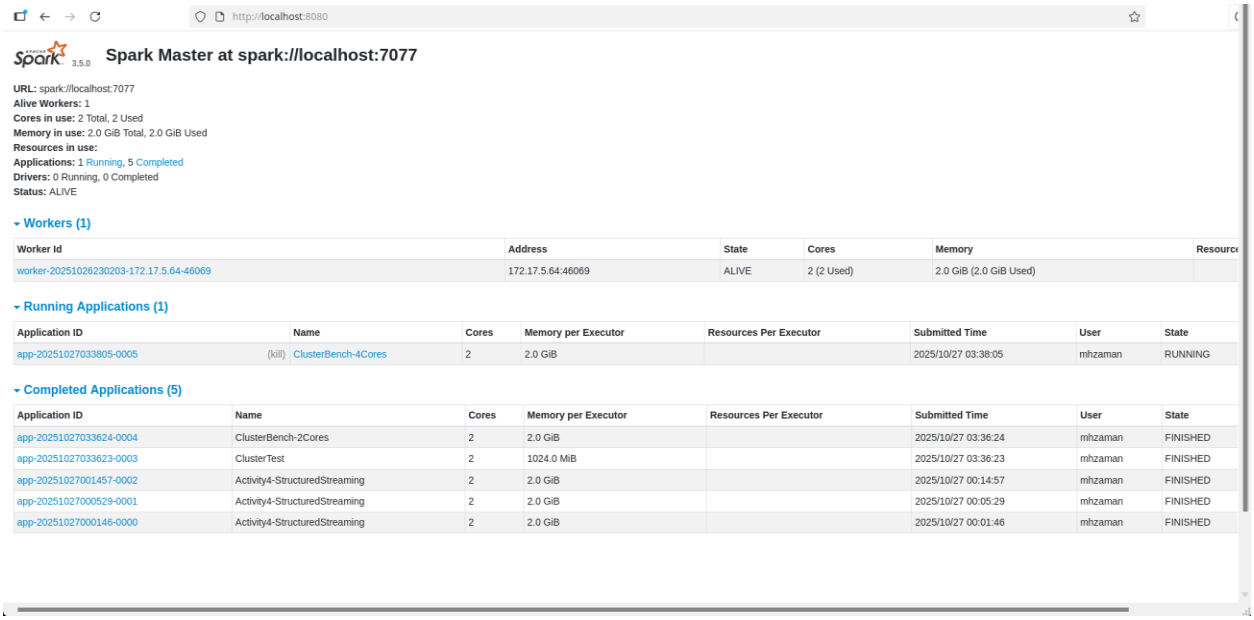
- **Activity 6:** Trained Decision Tree on Iris dataset using Spark MLlib; evaluated and printed accuracy and confusion counts.

- **Overall:** End-to-end pipeline demonstrating batch (Parquet) processing, streaming integration (Kafka + Spark Streaming), real-time visualization, and ML — suitable for coursework and demos.

Activity 1:

```
|payment_type|  count|
+------------+-------+
|        Cash|3332750|
|      Credit|3333563|
|      Mobile|3333687|
+------------+-------+

Time with cache: 1.07 seconds
Speedup: 5.47x

=== Query 6: Complex Aggregation - Trip Categories ===
+------------+-----------------+----------+--------+---------+
|payment_type|distance_category|trip_count|avg_fare|avg_total|
+------------+-----------------+----------+--------+---------+
|        Cash|             Long|    114727|   52.56|    62.41|
|        Cash|           Medium|    510648|    52.5|    62.48|
|        Cash|            Short|   1080771|   52.51|    62.46|
|        Cash|        Very Long|      4295|    52.3|    62.61|
|        Cash|       Very Short|   1622309|   52.44|    62.47|
|      Credit|             Long|    114596|   52.49|    62.62|
|      Credit|           Medium|    509211|   52.49|     62.6|
|      Credit|            Short|   1082262|   52.45|    62.51|
|      Credit|        Very Long|      4344|   51.74|    63.12|
|      Credit|       Very Short|   1623150|   52.49|     62.5|
|      Mobile|             Long|    114628|   52.35|    62.66|
|      Mobile|           Medium|    511355|   52.51|     62.5|
|      Mobile|            Short|   1080778|   52.54|    62.49|
|      Mobile|        Very Long|      4099|   52.69|    63.41|
|      Mobile|       Very Short|   1622827|   52.52|    62.51|
+------------+-----------------+----------+--------+---------+

Query execution time: 1.54 seconds

=== Saving Results ===
Results saved to output/activity1_results
Parquet results saved to output/activity1_parquet

============================================================
Activity 1 Complete!
============================================================
(venv) mhzaman@bdacourse:~/spark-kafka-lab$
```

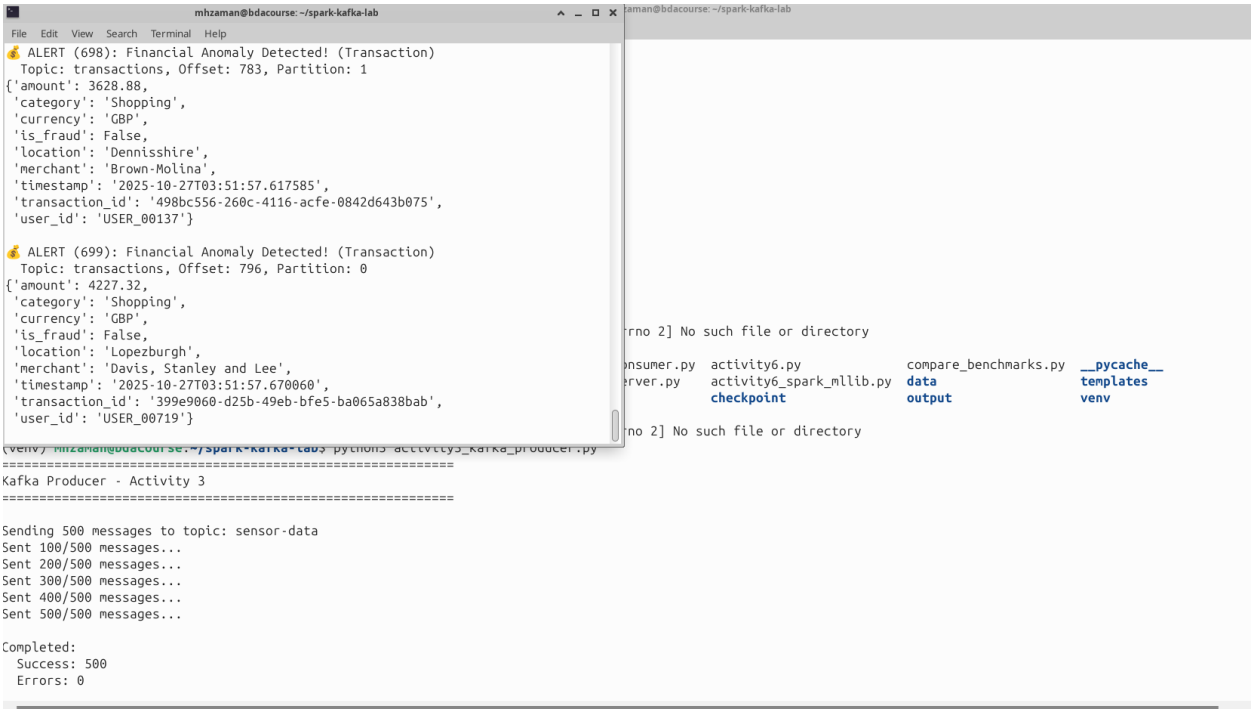
Activity 2:


http://localhost:8080
Spark 3.5.0
Spark Master at spark://localhost:7077
URL: spark://localhost:7077
Alive Workers: 1
Cores in use: 2 Total, 2 Used
Memory in use: 2.0 GiB Total, 2.0 GiB Used
Resources in use:
Applications: 1 Running, 5 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE
Workers (1)
Worker Id | Address | State | Cores | Memory
worker-20251026230203-172.17.5.64-46069 | 172.17.5.64:46069 | ALIVE | 2 (2 Used) | 2.0 GiB (2.0 GiB Used)
Running Applications (1)
Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State
app-20251027033805-0005 (kill) | ClusterBench-4Cores | 2 | 2.0 GiB | | 2025/10/27 03:38:05 | mhzaman | RUNNING
Completed Applications (5)
Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State
app-20251027033624-0004 | ClusterBench-2Cores | 2 | 2.0 GiB | | 2025/10/27 03:36:24 | mhzaman | FINISHED
app-20251027033623-0003 | ClusterTest | 2 | 1024.0 MiB | | 2025/10/27 03:36:23 | mhzaman | FINISHED
app-20251027001457-0002 | Activity4-StructuredStreaming | 2 | 2.0 GiB | | 2025/10/27 00:14:57 | mhzaman | FINISHED
app-20251027000529-0001 | Activity4-StructuredStreaming | 2 | 2.0 GiB | | 2025/10/27 00:05:29 | mhzaman | FINISHED
app-20251027000146-0000 | Activity4-StructuredStreaming | 2 | 2.0 GiB | | 2025/10/27 00:01:46 | mhzaman | FINISHED

Generate an output file with all relevant results, show how number of cores impacted your execution/processing time for different functions/queries on a dataset:

```
(venv) mhzaman@bdacourse:~/spark-kafka-lab$ cat output/cluster_benchmark.txt
Cluster Benchmark Results
================================================


2 Cores:
------------------------------
  Data Loading        :    27.83s
  Aggregation         :     2.39s
  Filter & Sort       :     2.26s
  Join Operation      :    20.11s
  Window Functions    :    11.16s
  TOTAL TIME          :    63.76s

4 Cores:
------------------------------
  Data Loading        :    26.17s
  Aggregation         :     2.03s
  Filter & Sort       :     2.36s
  Join Operation      :    19.76s
  Window Functions    :     8.62s
  TOTAL TIME          :    58.94s

6 Cores:
------------------------------
  Data Loading        :    25.48s
  Aggregation         :     2.04s
  Filter & Sort       :     2.61s
  Join Operation      :    20.35s
  Window Functions    :     8.62s
  TOTAL TIME          :    59.10s


Speedups (2 cores → 6 cores):
------------------------------
  Data Loading        :     1.09x
  Aggregation         :     1.18x
  Filter & Sort       :     0.87x
  Join Operation      :     0.99x
  Window Functions    :     1.29x
(venv) mhzaman@bdacourse:~/spark-kafka-lab$ S
```

Activity 3:
Detecting Anamolies in real time data stream

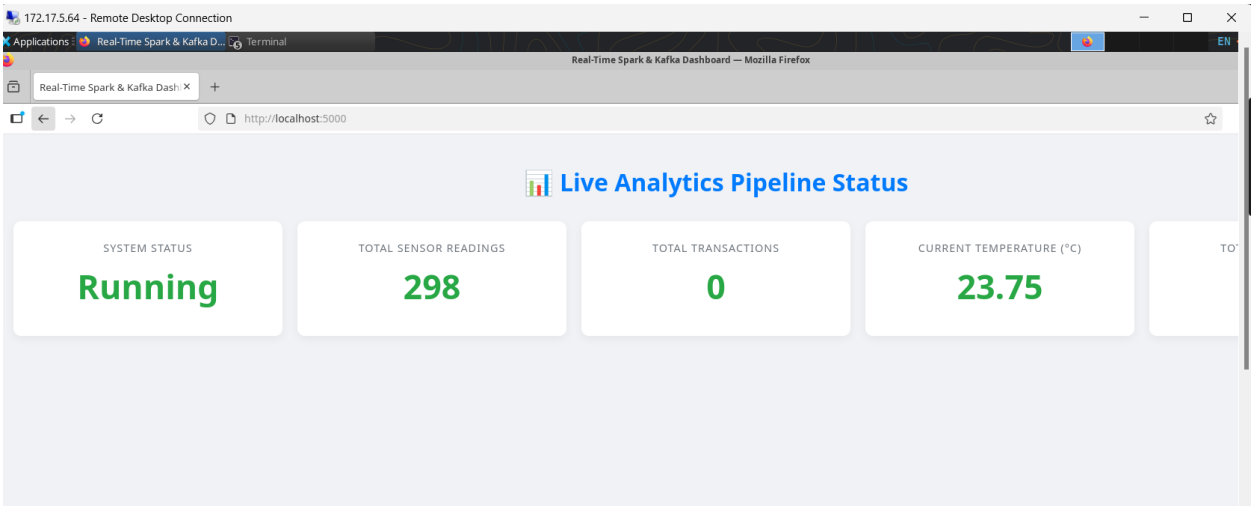


Activity 4:

```
|{2025-10-27 00:22:40, 2025-10-27 00:22:45}|warning |5    |
|{2025-10-27 00:22:40, 2025-10-27 00:22:45}|critical|2    |
+------------------------------------------+--------+-----+

-------------------------------------------
Batch: 29
-------------------------------------------
+------------------------------------------+--------+-----+
|window                                    |status  |count|
+------------------------------------------+--------+-----+
|{2025-10-27 00:22:40, 2025-10-27 00:22:45}|normal  |9    |
|{2025-10-27 00:22:40, 2025-10-27 00:22:45}|warning |12   |
|{2025-10-27 00:22:40, 2025-10-27 00:22:45}|critical|6    |
+------------------------------------------+--------+-----+

-------------------------------------------
Batch: 30
-------------------------------------------
+------------------------------------------+--------+-----+
|window                                    |status  |count|
+------------------------------------------+--------+-----+
|{2025-10-27 00:22:40, 2025-10-27 00:22:45}|warning |20   |
|{2025-10-27 00:22:40, 2025-10-27 00:22:45}|critical|9    |
+------------------------------------------+--------+-----+

-------------------------------------------
Batch: 31
-------------------------------------------
```

Activity 5:

```
(venv) mhzaman@bdacourse:~/spark-kafka-lab$ python3 activity5.py
Starting Flask server on http://localhost:5000
 * Serving Flask app 'activity5'
 * Debug mode: off
Consumer started for topic: sensor-data
Consumer started for topic: transactions
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.5.64:5000
Press CTRL+C to quit
127.0.0.1 - - [27/Oct/2025 02:51:09] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Oct/2025 02:51:10] "GET /socket.io/?EIO=4&transport=polling&t=PeYWFyO HTTP/1.1" 200 -
127.0.0.1 - - [27/Oct/2025 02:51:10] "GET /favicon.ico HTTP/1.1" 404 -
Client connected to /data_stream
127.0.0.1 - - [27/Oct/2025 02:51:10] "POST /socket.io/?EIO=4&transport=polling&t=PeYWFzQ&sid=e-0EKaV-J-7Xrq6eAAAA HTTP/1.1" 200 -
127.0.0.1 - - [27/Oct/2025 02:51:10] "GET /socket.io/?EIO=4&transport=polling&t=PeYWFzT&sid=e-0EKaV-J-7Xrq6eAAAA HTTP/1.1" 200 -
127.0.0.1 - - [27/Oct/2025 02:51:10] "GET /socket.io/?EIO=4&transport=polling&t=PeYWFzx&sid=e-0EKaV-J-7Xrq6eAAAA HTTP/1.1" 200 -
```



Show of Real time update on a html based analytical dashboard.

```
Files removed: 398 (2411.2 MB)
Installing six...
Collecting six
  Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six
Successfully installed six-1.17.0
Installing kafka-python...
Collecting kafka-python
  Downloading kafka_python-2.2.15-py2.py3-none-any.whl.metadata (10.0 kB)
Downloading kafka_python-2.2.15-py2.py3-none-any.whl (309 kB)
Installing collected packages: kafka-python
Successfully installed kafka-python-2.2.15
(venv) mhzaman@bdacourse:~/spark-kafka-lab$ python3 activity3_kafka_producer.py
============================================================
Kafka Producer - Activity 3
============================================================

Sending 500 messages to topic: sensor-data
Sent 100/500 messages...
Sent 200/500 messages...
Sent 300/500 messages...
Sent 400/500 messages...
Sent 500/500 messages...

Completed:
  Success: 500
  Errors: 0
```

Activity 6:

```
+------------+-----------+------------+-----------+-----------+
|sepal_length|sepal_width|petal_length|petal_width|    species|
+------------+-----------+------------+-----------+-----------+
|         5.1|        3.5|         1.4|        0.2|Iris-setosa|
|         4.9|        3.0|         1.4|        0.2|Iris-setosa|
|         4.7|        3.2|         1.3|        0.2|Iris-setosa|
|         4.6|        3.1|         1.5|        0.2|Iris-setosa|
|         5.0|        3.6|         1.4|        0.2|Iris-setosa|
+------------+-----------+------------+-----------+-----------+
only showing top 5 rows

Training Data Count: 104
Test Data Count: 46

=== Training Decision Tree Model... ===
Training Complete.
+-----------------+-----+----------+---------------+
|         features|label|prediction|        species|
+-----------------+-----+----------+---------------+
|[4.4,3.0,1.3,0.2]|  2.0|       2.0|    Iris-setosa|
|[4.6,3.2,1.4,0.2]|  2.0|       2.0|    Iris-setosa|
|[4.6,3.6,1.0,0.2]|  2.0|       2.0|    Iris-setosa|
|[4.7,3.2,1.3,0.2]|  2.0|       2.0|    Iris-setosa|
|[4.8,3.1,1.6,0.2]|  2.0|       2.0|    Iris-setosa|
|[4.8,3.4,1.6,0.2]|  2.0|       2.0|    Iris-setosa|
|[4.8,3.4,1.9,0.2]|  2.0|       2.0|    Iris-setosa|
|[4.9,3.1,1.5,0.1]|  2.0|       2.0|    Iris-setosa|
|[4.9,3.1,1.5,0.1]|  2.0|       2.0|    Iris-setosa|
|[5.0,2.3,3.3,1.0]|  1.0|       1.0|Iris-versicolor|
+-----------------+-----+----------+---------------+
only showing top 10 rows


============================================================
Model Training Time: DecisionTreeClassifier_fc78ceb5e25f
Test Set Accuracy = 0.9783
============================================================

=== Activity 6 Complete ===
(venv) mhzaman@bdacourse:~/spark-kafka-lab$
```