SMRT Analytics using Semantic Kernel + RAG + KQL Database

Overview

This notebook implements an advanced analytics system for SMRT (Singapore Mass Rapid Transit) operations using a modern AI stack combining **PySpark**, **Semantic Kernel**, **RAG (Retrieval-Augmented Generation)**, and **KQL (Kusto Query Language)** database.

Architecture Components

1. PySpark

What it is: Distributed data processing framework for large-scale data analytics
Why used:

- Handle large volumes of telemetry data (2000+ records)
- Perform distributed computations for feature engineering
- Integrate with Fabric Lakehouse and Delta tables
- Enable scalable data transformations and aggregations

2. Semantic Kernel

What it is: Microsoft's AI orchestration framework for integrating LLMs into applications
Why used:

- Standardized interface to OpenAI models
- Manage AI services (chat completion, embeddings)
- Enable plugin architecture for extensibility
- Provide async/await pattern for efficient AI calls

3. RAG (Retrieval-Augmented Generation)

What it is: AI technique that retrieves relevant documents/data and uses them to augment LLM prompts
Why used:

- Ground AI responses in actual operational data
- Prevent hallucination by providing factual context
- Enable domain-specific responses for SMRT operations
- Combine vector search with semantic understanding

4. KQL Database

What it is: Kusto Query Language database for time-series and telemetry data
Why used:

- Optimized for time-series operational data
- Fast ingestion and querying of telemetry streams
- Integration with Microsoft Fabric ecosystem
- Real-time analytics capabilities

Code Breakdown

1. Initialization & Dependencies

```
# Key imports for the AI stack
from pyspark.sql import SparkSession, DataFrame
from openai import OpenAI
import semantic_kernel as sk
from semantic_kernel.connectors.ai.open_ai import OpenAIChatCompletion, OpenAITextEmbedding
```

Output: Environment setup with OpenAI API configuration and model definitions.

2. Synthetic Data Generation

```
def create_smrt_synthetic(num_records: int = 500)-> DataFrame:
    # Generates realistic SMRT operational data
    train_ids = [f"TRAIN-{i:03d}" for i in range(1, 31)]
    lines = ["North-South", "East-West", "Circle", "Thomson-East Coast"]
```

```
    # ... data generation logic
```

Sample Output:

text

```
+-------------------------+-----------------+-------+---------+----------+-------------+---------------+-------------+
|timestamp                |line             |station|train_id |temperature_c|vibration_level|delay_minutes|maintenance_status|
+-------------------------+-----------------+-------+---------+-------------+---------------+-------------+-----------------+
|2025-10-04 21:16:50.354495|Circle          |STN-033|TRAIN-007|39.39        |0.951          |0.34         |Anomaly          |
|2025-10-08 00:16:50.354495|Thomson-East Coast|STN-044|TRAIN-023|30.92        |2.698          |0.11         |Anomaly          |
+-------------------------+-----------------+-------+---------+-------------+---------------+-------------+-----------------+
```

Why: Creates realistic training data with:

- Multiple train lines and stations
- Equipment telemetry (temperature, vibration)
- Operational metrics (delays, passenger counts)
- Maintenance status flags

3. Data Enrichment & Feature Engineering

```
smrt_enriched_df = smrt_df \
    .withColumn("efficiency_index",
                round(col("passenger_count") / (col("delay_minutes") + 1), 2)) \
    .withColumn("overheat_flag", when(col("temperature_c") > 38, True).otherwise(False))
```

Features Added:

- efficiency_index: Operational efficiency metric
- overheat_flag: Temperature anomaly detection
- high_vibration_flag: Mechanical stress indicator
- high_load_flag: Passenger capacity alerts
- operational_context: Human-readable context for LLMs
- metrics_vector: Embedding vectors for semantic search

4. KQL Database Integration

```
KQL_CLUSTER = "https://trd-jqrttn76947phe10ts.z4.kusto.fabric.microsoft.com"
KQL_DB = "SMRT_Operations_DB"
KQL_TABLE = "TrainTelemetry"

smrt_enriched_df.write \
    .format("com.microsoft.kusto.spark.synapse.datasource") \
    .options(**kustoOptions) \
    .mode("append") \
    .save()
```

Output:

✓ Data prepared for KQL ingestion
Records to ingest: 2,000
✓ Data saved to Delta table: SMRT_operations_staging

Why KQL:

- Time-series optimization for telemetry data
- Fast ingestion of streaming data
- Integration with Power BI for visualization
- Real-time query capabilities

5. SMRT RAG System Implementation

Core Components:

A. Memory Initialization

```
def _initialize_memory(self):
    recent_data = self.df.filter(
        (col("maintenance_status") == "Anomaly") |
```

```
        (col("temperature_c") > 38.0) |
        (col("vibration_level") > 2.0) |
        (col("delay_minutes") > 10)
    ).orderBy(col("timestamp").desc()).limit(200).collect()
```

Output: ✓ Loaded 200 operational records into memory

B. Vector Search (Semantic Search)

```
def vector_search(self, query: str, top_k: int = 5)-> List[Dict]:
    # Heuristic-based search for operational context
    if "maintenance" in q or "anomaly" in q:
        condition = col("maintenance_status") == "Anomaly"
    elif "temperature" in q or "overheat" in q:
        condition = col("temperature_c") > 35.0
    # ... more conditions
```

Sample Search Output:

→ Performing semantic search for: 'maintenance anomalies'

✓ Found 5 relevant records

1. 2025-10-10 13:24:50.354495 | Thomson-East Coast | STN-036 | TRAIN-013 | T=32.55C Vib=2.257 Delay=0.17s Status=Anomaly
2. 2025-10-10 13:18:50.354495 | Thomson-East Coast | STN-043 | TRAIN-023 | T=28.24C Vib=2.634 Delay=0.44s Status=Anomaly

C. Prompt Augmentation

```
def augment_prompt(self, query: str, context: List[Dict])-> str:
    enhanced = f"""You are an expert transport operations AI assistant for SMRT.

RELEVANT OPERATIONAL RECORDS:
{context_str}

USER QUERY: {query}
"""
```

D. AI Response Generation

```
async def generate_response_async(self, enhanced_prompt: str)-> str:
    result = await self.chat_service.get_chat_message_content(
        chat_history=chat_history,
        settings=self.chat_service.get_prompt_execution_settings_class()(
            temperature=0.5, max_tokens=800
        )
    )
```

6. Example RAG Workflow Output

Query: "temperature and vibration anomalies"

AI Response:

1) Situation Analysis:

The operational records indicate several instances of temperature and vibration anomalies across different lines and stations. The temperature anomalies range from 26.77°C to 41.5°C, and the vibration anomalies range from 0.195 to 2.634.

2) Key Findings and Observed Patterns:

The temperature anomalies are not confined to a specific line or station, indicating a widespread issue. The highest temperature recorded was 41.5°C on the Thomson-East Coast line.

3) Actionable Recommendations:
Short-term: Initiate immediate inspection and maintenance of the affected track and train assets.
Mid-term: Conduct a thorough review of the maintenance protocols and schedules.

4) Suggested Data/Feature Improvements:
Consider collecting more granular data such as the age and maintenance history of the assets.

Key Benefits of This Architecture

1. Operational Intelligence

- Real-time anomaly detection
- Predictive maintenance insights
- Route optimization recommendations
- Demand forecasting capabilities

2. Scalability

- PySpark handles large data volumes
- KQL optimized for time-series data
- Semantic Kernel enables AI orchestration
- RAG provides contextual accuracy

3. Domain Specialization

- SMRT-specific operational context
- Transportation industry metrics
- Maintenance and reliability focus
- Passenger experience optimization

4. Production Readiness

- Error handling and fallbacks
- Async/await patterns for performance
- Multiple storage backends (KQL + Delta)
- Modular, extensible architecture

Use Cases Enabled

1. **Predictive Maintenance**: Identify equipment failures before they occur
2. **Operational Efficiency**: Optimize train schedules and resource allocation
3. **Passenger Experience**: Reduce delays and improve service quality
4. **Asset Management**: Monitor train and track health in real-time
5. **Strategic Planning**: Data-driven decisions for network expansion

This implementation demonstrates a modern AI-powered analytics platform specifically tailored for mass transit operations, combining the scalability of big data technologies with the intelligence of large language models.

Outputs

```
================================================================================
LOAD DATA TO KQL DATABASE - SMRT OPERATIONS
================================================================================
✓ Data prepared for KQL ingestion
  Records to ingest: 2,000
✓ Data saved to Delta table: SMRT_operations_staging
Total records prepared for ingestion: 2000
```