Import matplotlib.pyplot as plt

Import pandas as pd

Import seaborn as sns

Import json

Import asyncio

From datetime import datetime, timedelta

Import numpy as np

From typing import Dict, List, Any

Import re

Class MockMCPClient:

“””Mock MCP client for testing without actual OpenSearch connection.”””

Def \_\_init\_\_(self):

Self.sample\_data = self.\_generate\_sample\_opensearch\_response()

Def \_generate\_sample\_opensearch\_response(self) -> Dict[str, Any]:

“””Generate realistic OpenSearch aggregation response.”””

Products = [“Laptop Pro”, “Smartphone X”, “Wireless Headphones”,

“Gaming Monitor”, “Mechanical Keyboard”, “Wireless Mouse”,

“Tablet Ultra”, “Smart Watch”, “Bluetooth Speaker”, “USB-C Hub”]

Months = [“2024-01”, “2024-02”, “2024-03”, “2024-04”, “2024-05”, “2024-06”,

“2024-07”, “2024-08”, “2024-09”, “2024-10”, “2024-11”, “2024-12”]

# Generate realistic monthly buckets

Monthly\_buckets = []

For month in months:

Month\_index = int(month.split(‘-‘)[1]) – 1

Product\_buckets = []

For product in products:

# Create realistic sales patterns with seasonality and trends

Base\_value = np.random.normal(5000, 1500) # Base monthly sales

Seasonal\_factor = 1 + 0.4 \* np.sin(2 \* np.pi \* month\_index / 12) # Seasonal variation

Trend\_factor = 1 + (month\_index \* 0.03) # 3% monthly growth

Noise = np.random.normal(1, 0.1) # Random noise

# Special patterns for different products

If “Gaming” in product and month\_index in [10, 11]: # Holiday boost for gaming

Seasonal\_factor \*= 1.5

Elif “Smart Watch” in product and month\_index in [0, 1]: # New Year fitness boost

Seasonal\_factor \*= 1.3

Sales\_value = max(1000, base\_value \* seasonal\_factor \* trend\_factor \* noise)

Product\_buckets.append({

“key”: product,

“doc\_count”: int(sales\_value / 100), # Approximate number of transactions

“total\_sales”: {

“value”: round(sales\_value, 2)

}

})

Monthly\_buckets.append({

“key\_as\_string”: month,

“key”: int(month.replace(“-“, “”)),

“doc\_count”: sum(p[“doc\_count”] for p in product\_buckets),

“products”: {

“buckets”: product\_buckets

}

})

Return {

“aggregations”: {

“monthly\_trends”: {

“buckets”: monthly\_buckets

}

},

“hits”: {

“total”: {“value”: 50000}

}

}

Async def search\_products(self, query: Dict[str, Any], index: str = “products”) -> Dict[str, Any]:

“””Mock search that returns sample data.”””

Print(f”🔍 Mock search executed on index ‘{index}’”)

Print(f”📊 Query processed: {json.dumps(query, indent=2)}”)

# Simulate some processing delay

Await asyncio.sleep(0.5)

Return self.sample\_data

Class SimpleNLQueryProcessor:

“””Simplified natural language query processor.”””

Def parse\_and\_build\_query(self, nl\_query: str) -> Dict[str, Any]:

“””Parse natural language and build OpenSearch query.”””

# Extract year (default to 2024)

Year\_match = re.search(r’(\d{4})’, nl\_query)

Year = int(year\_match.group(1)) if year\_match else 2024

# Build the OpenSearch aggregation query

Query = {

“query”: {

“bool”: {

“filter”: [

{

“range”: {

“timestamp”: {

“gte”: f”{year}-01-01”,

“lte”: f”{year}-12-31”,

“format”: “yyyy-MM-dd”

}

}

}

]

}

},

“aggs”: {

“monthly\_trends”: {

“date\_histogram”: {

“field”: “timestamp”,

“calendar\_interval”: “month”,

“format”: “yyyy-MM”

},

“aggs”: {

“products”: {

“terms”: {

“field”: “product\_name.keyword”,

“size”: 20

},

“aggs”: {

“total\_sales”: {

“sum”: {

“field”: “sales\_amount”

}

}

}

}

}

}

},

“size”: 0

}

Return query

Class ProductTrendAnalyzer:

“””Analyze and visualize product trends.”””

Def \_\_init\_\_(self):

Plt.style.use(‘default’)

Sns.set\_palette(“husl”)

Def analyze\_trends(self, opensearch\_results: Dict[str, Any]) -> pd.DataFrame:

“””Convert OpenSearch results to pandas DataFrame for analysis.”””

Data = []

Monthly\_buckets = opensearch\_results[‘aggregations’][‘monthly\_trends’][‘buckets’]

For month\_bucket in monthly\_buckets:

Month = month\_bucket[‘key\_as\_string’]

Products = month\_bucket[‘products’][‘buckets’]

For product in products:

Data.append({

‘month’: month,

‘product’: product[‘key’],

‘sales’: product[‘total\_sales’][‘value’],

‘transactions’: product[‘doc\_count’]

})

Df = pd.DataFrame(data)

Df[‘month’] = pd.to\_datetime(df[‘month’])

Return df

Def create\_trend\_visualizations(self, df: pd.DataFrame):

“””Create comprehensive trend visualizations.”””

# Set up the plot grid

Fig = plt.figure(figsize=(16, 12))

Fig.suptitle(‘📈 Product Trends Analysis 2024’, fontsize=20, fontweight=’bold’)

# Create subplots

Gs = fig.add\_gridspec(3, 2, hspace=0.3, wspace=0.3)

# 1. Line plot of top products over time

Ax1 = fig.add\_subplot(gs[0, :])

Self.\_plot\_top\_products\_timeline(df, ax1)

# 2. Total sales by product (bar chart)

Ax2 = fig.add\_subplot(gs[1, 0])

Self.\_plot\_total\_sales\_by\_product(df, ax2)

# 3. Growth analysis

Ax3 = fig.add\_subplot(gs[1, 1])

Self.\_plot\_growth\_rates(df, ax3)

# 4. Monthly heatmap

Ax4 = fig.add\_subplot(gs[2, :])

Self.\_plot\_monthly\_heatmap(df, ax4)

Plt.tight\_layout()

Plt.show()

Def \_plot\_top\_products\_timeline(self, df: pd.DataFrame, ax):

“””Plot timeline for top 6 products by total sales.”””

# Get top products by total sales

Top\_products = df.groupby(‘product’)[‘sales’].sum().nlargest(6).index

For product in top\_products:

Product\_data = df[df[‘product’] == product].sort\_values(‘month’)

Ax.plot(product\_data[‘month’], product\_data[‘sales’],

Marker=’o’, linewidth=2.5, markersize=6, label=product)

Ax.set\_title(‘🔥 Top Products Sales Timeline’, fontsize=14, fontweight=’bold’)

Ax.set\_xlabel(‘Month’, fontsize=12)

Ax.set\_ylabel(‘Sales ($)’, fontsize=12)

Ax.legend(bbox\_to\_anchor=(1.05, 1), loc=’upper left’)

Ax.grid(True, alpha=0.3)

# Format y-axis to show values in thousands

Ax.yaxis.set\_major\_formatter(plt.FuncFormatter(lambda x, p: f’${x/1000:.0f}K’))

Def \_plot\_total\_sales\_by\_product(self, df: pd.DataFrame, ax):

“””Plot total sales by product.”””

Total\_sales = df.groupby(‘product’)[‘sales’].sum().sort\_values(ascending=True)

Bars = ax.barh(total\_sales.index, total\_sales.values, color=’skyblue’, alpha=0.8)

Ax.set\_title(‘💰 Total Sales by Product’, fontsize=14, fontweight=’bold’)

Ax.set\_xlabel(‘Total Sales ($)’, fontsize=12)

# Add value labels

For bar, value in zip(bars, total\_sales.values):

Ax.text(value + value\*0.01, bar.get\_y() + bar.get\_height()/2,

F’${value/1000:.0f}K’, va=’center’, fontsize=10)

# Format x-axis

Ax.xaxis.set\_major\_formatter(plt.FuncFormatter(lambda x, p: f’${x/1000:.0f}K’))

Def \_plot\_growth\_rates(self, df: pd.DataFrame, ax):

“””Plot growth rates from first to last month.”””

Growth\_data = []

For product in df[‘product’].unique():

Product\_df = df[df[‘product’] == product].sort\_values(‘month’)

If len(product\_df) >= 2:

First\_month = product\_df.iloc[0][‘sales’]

Last\_month = product\_df.iloc[-1][‘sales’]

If first\_month > 0:

Growth\_rate = ((last\_month – first\_month) / first\_month) \* 100

Growth\_data.append({‘product’: product, ‘growth\_rate’: growth\_rate})

Growth\_df = pd.DataFrame(growth\_data).sort\_values(‘growth\_rate’)

Colors = [‘red’ if x < 0 else ‘green’ for x in growth\_df[‘growth\_rate’]]

Bars = ax.barh(growth\_df[‘product’], growth\_df[‘growth\_rate’], color=colors, alpha=0.7)

Ax.set\_title(‘📊 Growth Rate (Jan to Dec)’, fontsize=14, fontweight=’bold’)

Ax.set\_xlabel(‘Growth Rate (%)’, fontsize=12)

Ax.axvline(x=0, color=’black’, linestyle=’-‘, alpha=0.3)

# Add percentage labels

For bar, rate in zip(bars, growth\_df[‘growth\_rate’]):

Ax.text(rate + (2 if rate > 0 else -2), bar.get\_y() + bar.get\_height()/2,

F’{rate:.1f}%’, ha=’left’ if rate > 0 else ‘right’, va=’center’)

Def \_plot\_monthly\_heatmap(self, df: pd.DataFrame, ax):

“””Plot monthly sales heatmap.”””

# Pivot data for heatmap

Heatmap\_data = df.pivot\_table(values=’sales’, index=’product’, columns=’month’, fill\_value=0)

# Format month columns for better display

Heatmap\_data.columns = [col.strftime(‘%b’) for col in heatmap\_data.columns]

# Create heatmap

Sns.heatmap(heatmap\_data, annot=True, fmt=’.0f’, cmap=’YlOrRd’,

Cbar\_kws={‘label’: ‘Sales ($)’}, ax=ax)

Ax.set\_title(‘🔥 Monthly Sales Heatmap’, fontsize=14, fontweight=’bold’)

Ax.set\_xlabel(‘Month’, fontsize=12)

Ax.set\_ylabel(‘Product’, fontsize=12)

Def print\_trend\_summary(self, df: pd.DataFrame):

“””Print a text summary of trends.”””

Print(“\n” + “=”\*60)

Print(“📊 PRODUCT TRENDS SUMMARY 2024”)

Print(“=”\*60)

# Top products by total sales

Top\_products = df.groupby(‘product’)[‘sales’].sum().nlargest(5)

Print(“\n🏆 TOP 5 PRODUCTS BY TOTAL SALES:”)

For I, (product, sales) in enumerate(top\_products.items(), 1):

Print(f” {i}. {product}: ${sales:,.0f}”)

# Growth analysis

Print(“\n📈 GROWTH ANALYSIS (Jan vs Dec):”)

Growth\_data = []

For product in df[‘product’].unique():

Product\_df = df[df[‘product’] == product].sort\_values(‘month’)

If len(product\_df) >= 2:

First\_month = product\_df.iloc[0][‘sales’]

Last\_month = product\_df.iloc[-1][‘sales’]

If first\_month > 0:

Growth\_rate = ((last\_month – first\_month) / first\_month) \* 100

Growth\_data.append({‘product’: product, ‘growth\_rate’: growth\_rate})

Growth\_df = pd.DataFrame(growth\_data).sort\_values(‘growth\_rate’, ascending=False)

Print(“ Top Growers:”)

For \_, row in growth\_df.head(3).iterrows():

Print(f” 📊 {row[‘product’]}: +{row[‘growth\_rate’]:.1f}%”)

Print(“ Declining Products:”)

Declining = growth\_df[growth\_df[‘growth\_rate’] < 0]

If not declining.empty:

For \_, row in declining.head(3).iterrows():

Print(f” 📉 {row[‘product’]}: {row[‘growth\_rate’]:.1f}%”)

Else:

Print(“ 🎉 No declining products!”)

# Monthly insights

Monthly\_totals = df.groupby(‘month’)[‘sales’].sum()

Best\_month = monthly\_totals.idxmax().strftime(‘%B’)

Worst\_month = monthly\_totals.idxmin().strftime(‘%B’)

Print(f”\n📅 MONTHLY INSIGHTS:”)

Print(f” 🔥 Best Month: {best\_month} (${monthly\_totals.max():,.0f})”)

Print(f” ❄️ Slowest Month: {worst\_month} (${monthly\_totals.min():,.0f})”)

# Overall trend

Total\_sales = df[‘sales’].sum()

Print(f”\n💰 TOTAL SALES 2024: ${total\_sales:,.0f}”)

Print(“=”\*60)

Class ProductTrendClient:

“””Main client class that orchestrates the workflow.”””

Def \_\_init\_\_(self, use\_mock: bool = True):

Self.use\_mock = use\_mock

Self.query\_processor = SimpleNLQueryProcessor()

Self.analyzer = ProductTrendAnalyzer()

If use\_mock:

Self.mcp\_client = MockMCPClient()

Else:

# Here you would initialize the real MCP client

Raise NotImplementedError(“Real MCP client not implemented in this demo”)

Async def analyze\_trends(self, natural\_query: str):

“””Main method to analyze trends from natural language query.”””

Print(“🚀 Starting Product Trend Analysis”)

Print(f”📝 Query: ‘{natural\_query}’”)

Print(“-“ \* 50)

# Step 1: Process natural language query

Opensearch\_query = self.query\_processor.parse\_and\_build\_query(natural\_query)

Print(“✅ Natural language query processed”)

# Step 2: Execute search through MCP

Print(“🔍 Executing search through MCP…”)

Results = await self.mcp\_client.search\_products(opensearch\_query)

Print(“✅ OpenSearch results received”)

# Step 3: Analyze trends

Print(“📊 Analyzing trends…”)

Df = self.analyzer.analyze\_trends(results)

Print(f”✅ Processed {len(df)} data points across {df[‘product’].nunique()} products”)

# Step 4: Create visualizations

Print(“📈 Creating visualizations…”)

Self.analyzer.create\_trend\_visualizations(df)

# Step 5: Print summary

Self.analyzer.print\_trend\_summary(df)

Return df

# Demo function

Async def main():

“””Main demo function.”””

Print(“🎯 Product Trend Analysis Demo”)

Print(“Using Mock Data for OpenSearch via MCP”)

Print(“=”\*50)

# Initialize client with mock data

Client = ProductTrendClient(use\_mock=True)

# Natural language query

Query = “Show me trend of all products in 2024”

# Run analysis

Try:

Df = await client.analyze\_trends(query)

Print(“\n✅ Analysis completed successfully!”)

Print(f”📊 Data shape: {df.shape}”)

# Show sample data

Print(“\n📋 Sample Data:”)

Print(df.head(10).to\_string(index=False))

Except Exception as e:

Print(f”❌ Error during analysis: {e}”)

Import traceback

Traceback.print\_exc()

# Additional utility functions for real integration

Def create\_sample\_opensearch\_index\_data():

“””Generate sample data that you can index into OpenSearch for testing.”””

Products = [“Laptop Pro”, “Smartphone X”, “Wireless Headphones”,

“Gaming Monitor”, “Mechanical Keyboard”, “Wireless Mouse”,

“Tablet Ultra”, “Smart Watch”, “Bluetooth Speaker”, “USB-C Hub”]

Data = []

For month in range(1, 13): # Jan to Dec 2024

For day in range(1, 31, 3): # Sample every 3 days

Try:

Date = datetime(2024, month, day)

For product in products:

# Generate realistic transaction data

Num\_transactions = np.random.poisson(10) # Average 10 transactions per product per sample day

For \_ in range(num\_transactions):

Base\_price = {

“Laptop Pro”: 1200, “Smartphone X”: 800, “Wireless Headphones”: 150,

“Gaming Monitor”: 300, “Mechanical Keyboard”: 120, “Wireless Mouse”: 50,

“Tablet Ultra”: 500, “Smart Watch”: 250, “Bluetooth Speaker”: 80, “USB-C Hub”: 30

}[product]

# Add some price variation

Price = base\_price \* np.random.normal(1, 0.1)

Quantity = np.random.randint(1, 4)

Data.append({

“timestamp”: date.isoformat(),

“product\_name”: product,

“sales\_amount”: round(price \* quantity, 2),

“quantity”: quantity,

“unit\_price”: round(price, 2),

“customer\_id”: f”cust\_{np.random.randint(1000, 9999)}”,

“transaction\_id”: f”txn\_{len(data) + 1:06d}”

})

Except ValueError:

# Skip invalid dates (like Feb 30)

Continue

Return data

Def print\_integration\_instructions():

“””Print instructions for integrating with real OpenSearch.”””

Instructions = “””

🔧 INTEGRATION WITH REAL OPENSEARCH:

1. Set up your OpenSearch MCP Server:

- Use the server code from the previous artifact

- Make sure OpenSearch is running and accessible

- Configure proper authentication

2. Replace MockMCPClient with real MCP client:

```python

From mcp.client.session import ClientSession

From mcp.client.stdio import stdio\_client

Class RealMCPClient:

Async def search\_products(self, query, index=”products”):

Async with stdio\_client() as (read, write):

Async with ClientSession(read, write) as session:

Await session.initialize()

Result = await session.call\_tool(“search\_documents”, {

“index”: index,

“query”: query

})

Return json.loads(result[0].text)

```

3. Index sample data into OpenSearch:

```python

Sample\_data = create\_sample\_opensearch\_index\_data()

# Use OpenSearch bulk API to index this data

```

4. Update the client initialization:

```python

Client = ProductTrendClient(use\_mock=False)

```

5. Run your analysis:

```python

Df = await client.analyze\_trends(“trend of all products in 2024”)

```

“””

Print(instructions)

If \_\_name\_\_ == “\_\_main\_\_”:

# Run the demo

Asyncio.run(main())

# Print integration instructions

Print\_integration\_instructions()

# Optionally generate sample data for indexing

Print(“\n📝 Generating sample data for OpenSearch indexing…”)

Sample\_data = create\_sample\_opensearch\_index\_data()

Print(f”✅ Generated {len(sample\_data)} samp