Data Pipelines and Workflow Orchestration

Optimization & Troubleshooting Data Pipelines

Objective

By the end of this lesson, you will:

- Understand key performance bottlenecks in data pipelines.
- Learn techniques to optimize pipeline efficiency.
- Identify common errors and debug pipeline failures.

Identifying Performance Bottlenecks

Data pipelines often slow down due to inefficient processing, poor memory management, or unoptimized I/O operations.

Example 1: Inefficient Data Loading

Problem: Loading large CSV files inefficiently

Copy

```
import pandas as pd

df = pd.read_csv("large_dataset.csv")
```

Issue: Pandas loads the entire dataset into memory, which can cause crashes on large files.

Optimization: Load data in chunks

Copy

```
chunk_size = 100000  # Process 100k rows at a time
chunks = []
for chunk in pd.read_csv("large_dataset.csv", chunksize=chunk_size):
    chunks.append(chunk)
df = pd.concat(chunks, ignore_index=True)
```

Outcome: Reduces memory usage and prevents crashes.

Parallel Processing for Speed

Example 2: Slow Data Transformations

Problem: Processing rows one-by-one

Copy

```
df['fare_with_tax'] = df['fare_amount'].apply(lambda x: x * 1.08)
```

Issue: `apply()` is slow for large datasets.

Optimization: Vectorized Operations

Copy

```
df['fare_with_tax'] = df['fare_amount'] * 1.08
```

Outcome: 10-100x faster by leveraging Pandas' internal optimizations.

Troubleshooting Common Pipeline Failures

Example 3: Handling Missing Data

Problem: Pipeline crashes due to missing values

Copy

```
df['passenger_count'] = df['passenger_count'].astype(int)  # Causes error if NaNs
```

Fix: Handle missing values before conversion

Copy

```
df['passenger_count'] = df['passenger_count'].fillna(1).astype(int)
```

Outcome: Prevents runtime errors by setting a default value.

Debugging Pipeline Failures with Logging

Problem: Silent failures when processing data

Copy

```
def transform(df):
    df['total_fare'] = df['fare_amount'] + df['extra'] + df['mta_tax'] + df['tip_
    return df
```

Issue: If a column is missing, the function fails but provides no feedback.

Solution: Add Logging and Error Handling

Copy

```python
import logging

def transform(df):
    try:
        df['total_fare'] = df['fare_amount'] + df['extra'] + df['mta_tax'] + df['
    except KeyError as e:
        logging.error(f"Missing column: {e}")
        raise
    return df
```

Outcome: Captures missing column errors and logs details for debugging.

Wrap-Up & Next Steps

You have learned:

- How to optimize data loading, transformation, and processing.
- Common debugging techniques for troubleshooting pipelines.
- Logging and error handling best practices.

< Previous

Next >