Week 5 Apache Beam Exercise - Complete Solution Guide

Overview

This solution implements all tasks from the Week 5 exercise using Apache Beam pipelines to process user data.

Installation

First, install the required packages:

```bash
pip install apache-beam
```

Running the Pipelines

Task 1: Format User Data to Marketing Format

Transforms the user data to the required marketing format with proper date and address formatting.

```bash
python3 week5_beam.py --input users.csv --output outputs/marketing_format.txt --task task1
```

Expected Output Format:

```
User;Gender;Age;Address;Date joined
Amy Sullivan;female;20;Westlake,OH,44145;2020-08-31
Paige Dixon;female;43;Hicksville,NY,11801;2020-03-22
```

Task 2.1: Gender Percentage Split

Calculates the percentage distribution of male vs female customers.

```bash
python3 week5_beam.py --input users.csv --output outputs/gender_totals.txt --task task2_gender
```

Expected Output:

```
('gender', {'Female': 0.52, 'Male': 0.48})
```

Task 2.2: Customer Count by Join Date

Counts how many customers joined on each date.

```bash
```

```
python3 week5_beam.py --input users.csv --output outputs/customer_totals.txt --task task2_dates
```

Expected Output:

```
('2020-08-31', 1)
('2020-03-22', 1)
('2020-05-28', 2)
('2020-04-04', 1)
```

Task 2.3: Customer Count by State

Counts customers in each state.

```
bash

python3 week5_beam.py --input users.csv --output outputs/state_totals.txt --task task2_states
```

Expected Output:

```
('OH', 6)
('NY', 10)
('CT', 2)
('NJ', 12)
```

How the Solution Works

Task 1: FormatUserData Transform

- **Reads** CSV data line by line
- **Splits** each line into fields
- **Converts** gender to lowercase
- **Reformats** address from `City-State-Zip` to `City,State,Zip`
- **Converts** date from `YYYY/MM/DD` to `YYYY-MM-DD`
- **Outputs** with semicolon delimiter

Task 2.1: Gender Aggregation

1. `ExtractGender`: Extracts gender field from each row
2. `Count.PerElement()`: Counts occurrences of each gender
3. `ToDict()`: Combines counts into a dictionary
4. `CalculateGenderPercentages`: Computes percentages

Task 2.2: Date Aggregation

1. `ExtractJoinDate`: Extracts and formats the join date
2. `Count.PerElement()`: Counts customers per date
3. Output is written as (date, count) tuples

Task 2.3: State Aggregation

1. `ExtractState`: Parses state from address field
2. `Count.PerElement()`: Counts customers per state
3. Output is written as (state, count) tuples

Key Beam Concepts Used

PTransforms

- **ReadFromText**: Reads input file into PCollection
- **ParDo**: Applies DoFn to each element
- **Map**: Applies simple function to each element
- **Count.PerElement()**: Counts unique elements
- **ToDict()**: Combines key-value pairs into dictionary
- **WriteToText**: Writes PCollection to output file

DoFn Classes

Custom DoFn classes process elements:

- `FormatUserData`: Multi-field transformation
- `ExtractGender`: Simple field extraction
- `ExtractJoinDate`: Field extraction with formatting
- `ExtractState`: Parsing nested data
- `CalculateGenderPercentages`: Aggregation calculation

Answering Concluding Questions

1. Is the PCollection bounded or unbounded?

Answer: The PCollection is **bounded**.

We're reading from static CSV files that have a fixed, finite number of records. The pipeline processes all records and then completes.

2. Real-time Data Considerations

Would the data set be bounded or unbounded? The data set would be **unbounded** (streaming data).

Additional Considerations for Streaming Pipeline:

1. **Windowing Strategy**
   - Need to define time windows (e.g., fixed, sliding, session windows)
   - Example: Count customers joining per hour or day
2. **Triggers**
   - Define when to emit results (e.g., after watermark, processing time)
   - Handle late-arriving data
3. **State Management**
   - Manage stateful operations across windows
   - Handle updates to aggregations
4. **Watermarks**
   - Track event time progress
   - Determine when windows are complete
5. **Infrastructure**
   - Use appropriate runner (e.g., DataflowRunner, FlinkRunner)
   - Consider scalability and fault tolerance
6. **Example Streaming Code:**

```python
# Add windowing to pipeline
(p
 | 'Read stream' >> beam.io.ReadFromPubSub(topic='users')
 | 'Window' >> beam.WindowInto(beam.window.FixedWindows(60))  # 1-minute windows
 | 'Extract gender' >> beam.ParDo(ExtractGender())
 | 'Count' >> beam.combiners.Count.PerElement()
 | 'Write' >> beam.io.WriteToPubSub(topic='gender-counts'))
```

Troubleshooting

Output Files Have Suffixes

Beam appends suffixes like `-00000-of-00001` to output files. This is normal behavior.

Multiple Output Files

For larger datasets, Beam may create multiple sharded output files. Use `num_shards=1` parameter in `WriteToText()` to force single file output.

Module Import Errors

Ensure apache-beam is properly installed:

```bash
pip install --upgrade apache-beam
```

Testing Your Solution

Create a small test file to verify transformations:

```bash
# Create test file
echo -e 'User,Gender,Age,Address,Date joined\nTest User,Female,25,TestCity-NY-12345,2020/01/01' > test.csv

# Run pipeline
python3 week5_beam.py --input test.csv --output test_output.txt --task task1

# Check output
cat test_output.txt-00000-of-00001
```

Additional Resources

- Apache Beam Documentation
- Beam Programming Guide
- PTransform Catalog