Pipeline Core Code Explanation

This document provides a detailed breakdown of 'pipeline/src/pipeline_core.py'. After refactoring, this file contains specialized helper functions that power the extraction, configuration, and validation logic of the pipeline.

1. Configuration Section

These functions handle reading the YAML instructions which define the validation rules.

```
def load_config(config_path: str) -> Dict:
    with open(config_path, 'r') as f:
        return yaml.safe_load(f)
```

- What it does: Opens 'schema_config.yaml' and converts it into a Python Dictionary.
- Why: Allows the pipeline to know rules dynamically without hardcoding them.

```
def get_dataset_config(config: Dict, dataset_name: str) -> Dict:
    return config['datasets'].get(dataset_name)
```

- What it does: Retrieves specific rules for a dataset (e.g., 'inventory_snapshot').
- Why: Different files have different columns and integrity requirements.

2. Ingestion Section

This function handles the raw data loading.

```
def load_csv(data_dir: str, filename: str) -> pd.DataFrame:
    path = os.path.join(data_dir, filename)
    if not os.path.exists(path):
        raise FileNotFoundError(f'File not found: {path}')
    return pd.read_csv(path)
```

- What it does: Constructs the full path and reads the CSV into a Pandas DataFrame.
- Safety: Checks file existence first to provide clear error messages.

3. Validation Logic (The Brain)

This is the core filter of the system. It splits data into 'Good' (Valid) and 'Bad' (Quarantine) streams.

```
def validate_data(df: pd.DataFrame, rules: Dict, master_products: set) -> Tuple...
```

Key Steps inside this function:

A. Quarantine Mask:
  Creates a boolean series (True/False) for every row. Initially all are False (Valid).

Pipeline Core Code Explanation

B. Rule Loop:

Loops through columns defined in YAML. If a check like 'min_0' is found, it marks rows with values < 0 as True (Invalid) in the mask.

C. Master Data Check:

Checks if 'product_id' exists in the master list. If not, marks row as Invalid.

D. Splitting:

```
quarantine_df = df[quarantine_mask]
valid_df = df[~quarantine_mask]
```

The '~' operator inverts the mask to select only the valid rows.