ODP Issue-Level Detail Report Generator

This script generates a detailed CSV (`odp-issue.csv`) containing issue-level diagnostics for datasets submitted to the **Open Digital Planning (ODP)** service. It merges **expected provisions** with **issue summaries** from the ODP Datasette platform, enabling fine-grained monitoring of dataset conformance at the resource and field level.

Purpose:

- Join `provision` data (expected datasets per cohort/organisation) with actual `endpoint_dataset_issue_type_summary` data.
- Provide detailed information per issue including:
  - Dataset pipeline
  - Issue type, severity, responsibility
  - Affected endpoint/resource
  - Metadata like entry/end dates and latest statuses

Data Sources:

- `digital-land` Datasette database:
  - `provision` , `cohort` , `organisation`
- `performance` Datasette database:
  - `endpoint_dataset_issue_type_summary`
  - `endpoint_dataset_summary`

In [ ]:

```
"""
Script to generate a detailed CSV of issue-level data from the Open Digital Plannir
Datasette service. It joins issue summaries with provision data to create a merged
of expected dataset performance per organisation and cohort.
"""

import os
import pandas as pd
import requests
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry
import argparse

# Dataset Definitions
SPATIAL_DATASETS = [
    "article-4-direction-area",
    "conservation-area",
    "listed-building-outline",
    "tree-preservation-zone",
    "tree",
]
DOCUMENT_DATASETS = [
    "article-4-direction",
    "conservation-area-document",
    "tree-preservation-order",
]
```

```
ALL_DATASETS = SPATIAL_DATASETS + DOCUMENT_DATASETS

# HTTP Helpers
def get_datasette_http():
    """
    Returns a requests.Session object with retry logic for querying Datasette endpc

    Returns:
        requests.Session: A session object with retry strategy for robustness.
    """
    retry_strategy = Retry(total=3, status_forcelist=[400], backoff_factor=0.2)
    adapter = HTTPAdapter(max_retries=retry_strategy)
    http = requests.Session()
    http.mount("https://", adapter)
    return http

# Datasette Query Helper
def get_datasette_query(db: str, sql: str, url="https://datasette.planning.data.gov
    """
    Executes an SQL query against the specified Datasette database.

    Args:
        db (str): Datasette database name.
        sql (str): SQL query string.
        url (str): Base Datasette URL.

    Returns:
        pd.DataFrame: Resulting data as a DataFrame, or empty on failure.
    """
    full_url = f"{url}/{db}.json"
    params = {"sql": sql, "_shape": "array", "_size": "max"}
    try:
        http = get_datasette_http()
        response = http.get(full_url, params=params)
        response.raise_for_status()
        return pd.DataFrame(response.json())
    except Exception as e:
        print(f"[ERROR] Datasette query failed: {e}")
        return pd.DataFrame()

# Provision Query
def get_provisions():
    """
    Retrieves all expected dataset provisions from the 'provision' table.

    Returns:
        pd.DataFrame: Provision records joined with cohort and organisation names.
    """
    sql = """
        SELECT
            p.cohort,
            p.organisation,
            c.start_date AS cohort_start_date,
            o.name AS organisation_name
        FROM provision p
        INNER JOIN cohort c ON c.cohort = p.cohort
        INNER JOIN organisation o ON o.organisation = p.organisation
        WHERE p.provision_reason = 'expected'
          AND p.project = 'open-digital-planning'
        GROUP BY p.organisation, p.cohort
    """
    return get_datasette_query("digital-land", sql)

# Issue Query (Paged)
```

```
def get_issue_type_chunk(dataset_clause, offset):
    """
    Retrieves a paged chunk of issue type summaries joined with endpoint metadata.

    Args:
        dataset_clause (str): SQL clause to filter datasets.
        offset (int): Pagination offset for the query.

    Returns:
        pd.DataFrame: Chunk of issue summary data.
    """
    sql = f"""
        SELECT
            edits.*,
            eds.endpoint_end_date,
            eds.endpoint_entry_date,
            eds.latest_status,
            eds.latest_exception
        FROM endpoint_dataset_issue_type_summary edits
        LEFT JOIN (
            SELECT endpoint, end_date as endpoint_end_date,
                   entry_date as endpoint_entry_date,
                   latest_status, latest_exception
            FROM endpoint_dataset_summary
        ) eds ON edits.endpoint = eds.endpoint
        {dataset_clause}
        LIMIT 1000 OFFSET {offset}
    """
    return get_datasette_query("performance", sql)

def get_full_issue_type_summary(datasets):
    """
    Retrieves the full issue summary table across all datasets using pagination.

    Args:
        datasets (list): List of dataset names to include.

    Returns:
        pd.DataFrame: Combined issue summary for all specified datasets.
    """
    dataset_clause = "WHERE " + " OR ".join(f"edits.dataset = '{ds}'" for ds in dat
    df_list = []
    offset = 0
    while True:
        chunk = get_issue_type_chunk(dataset_clause, offset)
        if chunk.empty:
            break
        df_list.append(chunk)
        if len(chunk) < 1000:
            break
        offset += 1000
    return pd.concat(df_list, ignore_index=True)

# Main CSV Generator
def generate_detailed_issue_csv(output_dir: str, dataset_type="all") -> str:
    """
    Generates a CSV containing detailed issue-level data for ODP datasets.

    Args:
        output_dir (str): Path to the output directory.
        dataset_type (str): One of 'spatial', 'document', or 'all' (default).

    Returns:
        str: Path to the saved CSV file.
```

```
    """
    # Select datasets based on type
    datasets = {
        "spatial": SPATIAL_DATASETS,
        "document": DOCUMENT_DATASETS,
        "all": ALL_DATASETS
    }.get(dataset_type, ALL_DATASETS)

    print("[INFO] Fetching provisions...")
    provisions = get_provisions()

    print("[INFO] Fetching detailed issue-level data...")
    issues = get_full_issue_type_summary(datasets)

    print("[INFO] Merging data...")
    merged = provisions.merge(
        issues.drop(columns=["organisation_name"], errors="ignore"),
        on=["organisation", "cohort"],
        how="inner"
    )

    print("[INFO] Saving CSV...")
    os.makedirs(output_dir, exist_ok=True)
    output_path = os.path.join(output_dir, "odp-issue.csv")
    merged[
        [
            "organisation",
            "cohort",
            "organisation_name",
            "pipeline",
            "issue_type",
            "severity",
            "responsibility",
            "count_issues",
            "collection",
            "endpoint",
            "endpoint_url",
            "latest_status",
            "latest_exception",
            "resource",
            "latest_log_entry_date",
            "endpoint_entry_date",
            "endpoint_end_date",
            "resource_start_date",
            "resource_end_date",
        ]
    ].to_csv(output_path, index=False)

    print(f"[SUCCESS] CSV saved: {output_path} ({len(merged)} rows)")
    return output_path

# CLI Argument Parser
def parse_args():
    """
    Parses command-line arguments for the script.

    Returns:
        argparse.Namespace: Contains the '--output-dir' argument.
    """
    parser = argparse.ArgumentParser(description="Generate detailed ODP issue-level
    parser.add_argument(
        "--output-dir",
        type=str,
        required=True,
```

```
        help="Directory to save the output CSV"
    )
    return parser.parse_args()

# Script Entry Point
if __name__ == "__main__":
    args = parse_args()
    generate_detailed_issue_csv(args.output_dir, dataset_type="all")
```