Endpoint Documentation Audit from Datasette

This script is designed to **audit endpoint metadata** from the Open Digital Planning Datasette, specifically checking for **missing `documentation_url` values**.

Purpose:

- Identify and report endpoints that are missing associated documentation URLs.
- Provide a breakdown of affected pipelines.
- Save the full endpoint metadata to CSV for further review.

Key Features:

1. **Paginated SQL Query via Datasette API**
   Uses a paginated SQL query (with `LIMIT` and `OFFSET`) to iteratively download up to 1000 records at a time from:
   - `endpoint`
   - `source`
   - `source_pipeline`
   - `organisation`
2. **Data Analysis**
   Adds new columns:
   - `documentation_missing`: Boolean for missing or empty `documentation_url`.
   - `is_active`: Boolean for endpoints with no `end_date`.
   - Statistics: % missing, most recent missing date, top pipelines missing documentation.
3. **Output CSV**
   Saves a comprehensive dataset to:
   - `all-endpoints-and-documentation-urls.csv`

In [ ]:
```
import requests
import pandas as pd
import os
import argparse

# Constants
DATASSETTE_URL = "https://datasette.planning.data.gov.uk/digital-land.json"

# Base SQL query to retrieve endpoint metadata
BASE_SQL = """
SELECT
    o.name,
    s.organisation,
    sp.pipeline AS "pipeline/dataset",
    e.endpoint_url,
    s.documentation_url,
    s.entry_date,
```

```
    s.end_date,
    e.endpoint
FROM
    endpoint e
    INNER JOIN source s ON e.endpoint = s.endpoint
    INNER JOIN source_pipeline sp ON s.source = sp.source
    INNER JOIN organisation o ON o.organisation = s.organisation
ORDER BY s.entry_date DESC
LIMIT 1000 OFFSET {offset}
"""

def parse_args():
    """
    Parses command-line arguments for the output directory.
    """
    parser = argparse.ArgumentParser(description="Export endpoints with missing doc
    parser.add_argument(
        "--output-dir",
        type=str,
        required=True,
        help="Directory to save the output CSV"
    )
    return parser.parse_args()

def fetch_endpoint_data():
    """
    Fetches all endpoint metadata using paginated SQL from Datasette API.

    Returns:
        pd.DataFrame: Combined result of all pages as a DataFrame.
    """
    all_rows, offset = [], 0
    columns = []

    while True:
        paginated_sql = BASE_SQL.format(offset=offset)
        response = requests.get(DATASSETTE_URL, params={"sql": paginated_sql, "_siz

        if response.status_code != 200:
            print("Failed to fetch data from Datasette.")
            break

        json_data = response.json()
        rows = json_data.get("rows", [])

        if not rows:
            break

        if offset == 0:
            columns = json_data.get("columns", [])

        all_rows.extend(rows)
        offset += 1000

    return pd.DataFrame(all_rows, columns=columns) if all_rows else pd.DataFrame()

def analyze_missing_docs(df):
    """
    Analyzes the dataset to flag missing documentation URLs and report stats.

    Args:
        df (pd.DataFrame): Raw endpoint metadata.

    Returns:
```

```
        pd.DataFrame: DataFrame with added helper columns.
    """
    total = len(df)
    df["documentation_missing"] = df["documentation_url"].fillna("").str.strip() ==
    missing_count = df["documentation_missing"].sum()
    percent_missing = (missing_count / total) * 100

    print(f"Total endpoints: {total}")
    print(f"Missing documentation_url: {missing_count}")
    print(f"Percent missing: {percent_missing:.2f}%")

    top_missing = (
        df[df["documentation_missing"]]
        .groupby("pipeline/dataset")
        .size()
        .sort_values(ascending=False)
        .head(10)
    )
    print("\nTop affected pipelines:")
    print(top_missing.to_string())

    df["is_active"] = df["end_date"].fillna("").str.strip() == ""
    active_missing = df.query("documentation_missing and is_active").shape[0]
    ended_missing = df.query("documentation_missing and not is_active").shape[0]

    print(f"\nActive endpoints missing documentation: {active_missing}")
    print(f"Ended endpoints missing documentation: {ended_missing}")

    df["entry_date"] = pd.to_datetime(df["entry_date"], errors="coerce")
    recent_missing = df[df["documentation_missing"]]["entry_date"].max()
    recent_str = recent_missing.date() if pd.notnull(recent_missing) else "N/A"
    print(f"\nMost recent entry with missing documentation: {recent_str}")

    return df

def save_results(df, output_dir):
    """
    Filters endpoints missing documentation and saves to CSV.

    Args:
        df (pd.DataFrame): The analyzed DataFrame.
        output_dir (str): Output directory path.
    """
    os.makedirs(output_dir, exist_ok=True)
    #filtered = df.query("documentation_missing and is_active")
    output_path = os.path.join(output_dir, "all-endpoints-and-documentation-urls.cs
    df.to_csv(output_path, index=False)
    print(f"CSV saved: {output_path}")

def main():
    """
    Main workflow to fetch, analyze, and save data.
    """
    args = parse_args()
    df = fetch_endpoint_data()

    if df.empty:
        print("No data found to process.")
        return

    df = analyze_missing_docs(df)
    save_results(df, args.output_dir)
```

```
if __name__ == "__main__":
    main()
```