Duplicate Geometry Checker

This notebook/script processes and reports on spatial data quality issues—specifically, cases of duplicate geometries found during automated validation checks on the Open Digital Planning platform. It performs the following steps:

Step-by-step breakdown:

1. **Input Parsing**:
   - Uses `argparse` to take an output directory as input for saving results.
2. **Expectation Data Fetching**:
   - Downloads `expectation.csv` from Datasette, filtering for `duplicate_geometry_check` entries.
3. **Details Parsing & Record Extraction**:
   - Parses each expectation's `details` field into Python dictionaries.
   - Extracts and flattens both `complete_matches` (two entities with identical geometries) and `single_matches` (likely partial overlaps or ambiguous matches).
4. **Entity Metadata Enrichment**:
   - Downloads and joins entity-level metadata (e.g., geometry, name, organisation) from the relevant datasets:
     - `conservation-area`, `article-4-direction-area`, `listed-building-outline`, `tree-preservation-zone`, `tree`.
5. **Organisation Metadata Enrichment**:
   - Merges readable organisation names for both `entity_a` and `entity_b` using the organisation registry.
6. **Output Generation**:
   - **Detailed Output**: `duplicate_entity_expectation.csv` listing each pair of duplicate entities with enriched metadata.
   - **Summary Output**: `duplicate_entity_expectation_summary.csv` aggregating stats (counts, actual vs expected) for each dataset.

This tool is designed to support QA of spatial datasets by identifying structural overlaps that may indicate mis-published features. It uses streamed `.csv` endpoints for efficient access to national-scale datasets and supports integration into broader workflow automation pipelines.

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

def parse_args():
    """
    Parses command-line arguments for specifying the output directory.
    Returns:
        argparse.Namespace: Contains the output directory path.
    """
    parser = argparse.ArgumentParser(description="Duplicate geometry checker")
    parser.add_argument(
        "--output-dir",
        type=str,
        required=True,
        help="Directory to save exported CSVs"
    )
    return parser.parse_args()

def parse_details(val):
    """
    Safely parses a stringified dictionary from the 'details' column using `ast.lit

    Parameters:
        val (str): A string containing a dictionary-like structure.

    Returns:
        dict: Parsed dictionary, or empty dict if parsing fails.
    """
    try:
        return ast.literal_eval(val)
    except Exception:
        return {}

def extract_stats(details_dict):
    """
    Extracts summary statistics from a parsed 'details' dictionary.

    Parameters:
        details_dict (dict): Parsed dictionary from the 'details' column.

    Returns:
        pd.Series: Contains actual/expected values and match counts.
    """
    return pd.Series({
        "actual": details_dict.get("actual"),
        "expected": details_dict.get("expected"),
        "complete_match_count": len(details_dict.get("complete_matches", [])) if is
        "single_match_count": len(details_dict.get("single_matches", [])) if isinst
        "complete_matches": details_dict.get("complete_matches", []),
        "single_matches": details_dict.get("single_matches", [])
    })

def main(output_dir):
    """
    Main function for processing duplicate geometry checks.

    - Downloads expectations with 'duplicate_geometry_check' operation.
    - Parses match results and enriches them with entity and organisation metadata.
    - Outputs both detailed and summary CSVs to the specified output directory.
    """

    # Load expectation records where operation is 'duplicate_geometry_check'
    url = "https://datasette.planning.data.gov.uk/digital-land/expectation.csv?_str
    df = pd.read_csv(url)
    df = df[df["operation"] == "duplicate_geometry_check"].copy()

    # Parse the 'details' field into dictionaries
    df["details_parsed"] = df["details"].apply(parse_details)
```

```
# Extract match records (complete and single) from details
records = []
for _, row in df.iterrows():
    dataset = row["dataset"]
    operation = row["operation"]
    details = row["details_parsed"]

    for match in details.get("complete_matches", []):
        records.append({
            "dataset": dataset,
            "operation": operation,
            "message": "complete_match",
            "entity_a": match.get("entity_a"),
            "organisation_entity_a": match.get("organisation_entity_a"),
            "entity_b": match.get("entity_b"),
            "organisation_entity_b": match.get("organisation_entity_b"),
        })

    for match in details.get("single_matches", []):
        records.append({
            "dataset": dataset,
            "operation": operation,
            "message": "single_match",
            "entity_a": match.get("entity_a"),
            "organisation_entity_a": match.get("organisation_entity_a"),
            "entity_b": match.get("entity_b"),
            "organisation_entity_b": match.get("organisation_entity_b"),
        })

df_matches = pd.DataFrame(records)

# URLs for entity tables by dataset
url_map = {
    "conservation-area": "https://datasette.planning.data.gov.uk/conservation-a
    "article-4-direction-area": "https://datasette.planning.data.gov.uk/article
    "listed-building-outline": "https://datasette.planning.data.gov.uk/listed-b
    "tree-preservation-zone": "https://datasette.planning.data.gov.uk/tree-pres
    "tree": "https://datasette.planning.data.gov.uk/tree/entity.csv?_stream=on"
}

# Columns to retain from entity tables
columns_to_keep = ["entity", "dataset", "end_date", "entry_date", "geometry", '
entity_tables = {}

# Download and store each dataset's entity table
for dataset_name, entity_url in url_map.items():
    df_entity = pd.read_csv(entity_url)
    df_entity["dataset"] = dataset_name
    entity_tables[dataset_name] = df_entity[columns_to_keep].copy()

# Combine all entity tables into one DataFrame
df_entities = pd.concat(entity_tables.values(), ignore_index=True)

# Merge metadata for entity_a
df_matches = df_matches.merge(
    df_entities,
    how="left",
    left_on=["dataset", "entity_a"],
    right_on=["dataset", "entity"]
).rename(columns={
    "end_date": "entity_a_end_date",
    "entry_date": "entity_a_entry_date",
    "geometry": "entity_a_geometry",
    "name": "entity_a_name",
    "organisation_entity": "entity_a_organisation"
}).drop(columns=["entity"])

# Merge metadata for entity_b
df_matches = df_matches.merge(
    df_entities,
    how="left",
    left_on=["dataset", "entity_b"],
    right_on=["dataset", "entity"]
).rename(columns={
    "end_date": "entity_b_end_date",
    "entry_date": "entity_b_entry_date",
    "geometry": "entity_b_geometry",
    "name": "entity_b_name",
    "organisation_entity": "entity_b_organisation"
}).drop(columns=["entity"])

# Reorder final column layout
ordered_cols = [
    "dataset", "operation", "message",
    "entity_a", "entity_a_name", "entity_a_organisation", "entity_a_entry_date'
    "entity_b", "entity_b_name", "entity_b_organisation", "entity_b_entry_date'
]
df_matches = df_matches[ordered_cols]

# Load organisation lookup table
org_url = "https://datasette.planning.data.gov.uk/digital-land/organisation.csv
df_org = pd.read_csv(org_url)[["entity", "name"]].rename(columns={
    "entity": "organisation_entity",
    "name": "organisation_name"
})

# Add readable name for entity_a_organisation
df_matches = df_matches.merge(
    df_org,
    how="left",
    left_on="entity_a_organisation",
    right_on="organisation_entity"
).rename(columns={"organisation_name": "entity_a_organisation_name"}).drop(colu

# Insert entity_a_organisation_name after entity_a_organisation
a_cols = df_matches.columns.tolist()
a_index = a_cols.index("entity_a_organisation") + 1
a_cols.insert(a_index, a_cols.pop(a_cols.index("entity_a_organisation_name")))
df_matches = df_matches[a_cols]

# Add readable name for entity_b_organisation
df_matches = df_matches.merge(
    df_org,
    how="left",
    left_on="entity_b_organisation",
    right_on="organisation_entity"
).rename(columns={"organisation_name": "entity_b_organisation_name"}).drop(colu

# Insert entity_b_organisation_name after entity_b_organisation
b_cols = df_matches.columns.tolist()
b_index = b_cols.index("entity_b_organisation") + 1
b_cols.insert(b_index, b_cols.pop(b_cols.index("entity_b_organisation_name")))
df_matches = df_matches[b_cols]

# Save detailed match output
os.makedirs(output_dir, exist_ok=True)
matches_csv = os.path.join(output_dir, "duplicate_entity_expectation.csv")
df_matches.to_csv(matches_csv, index=False)
# Re-parse stats and generate summary view
df["details_parsed"] = df["details"].apply(parse_details)
stats_df = pd.concat([df[["dataset", "severity"]], df["details_parsed"].apply(e
stats_df = stats_df.sort_values(by="complete_match_count", ascending=False).res

# Save summary CSV
summary_csv = os.path.join(output_dir, "duplicate_entity_expectation_summary.cs
stats_df.drop(columns=["complete_matches", "single_matches"]).to_csv(summary_cs

# Entry point
if __name__ == "__main__":
args = parse_args()
main(args.output_dir)
```