041-sqlite

May 18, 2022

4.1. Wrangling Data with SQL

[1]:
```
import sqlite3

import pandas as pd
from IPython.display import VimeoVideo
```

[2]:
```
VimeoVideo("665414044", h="ff34728e6a", width=600)
```

[2]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13eeb3490>
```

1 Prepare Data

1.1 Connect

[3]:
```
VimeoVideo("665414180", h="573444d2f6", width=600)
```

[3]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13eeb3880>
```

Task 4.1.1: Run the cell below to connect to the `nepal.sqlite` database.

- What's ipython-sql?
- What's a Magics function?

[4]:
```
%load_ext sql
%sql sqlite:////home/jovyan/nepal.sqlite
```

[4]:
```
'Connected: @/home/jovyan/nepal.sqlite'
```

1.2 Explore

[5]:
```
VimeoVideo("665414201", h="4f30b7a95f", width=600)
```

[5]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e893d00>
```

Task 4.1.2: Select all rows and columns from the `sqlite_schema` table, and examine the output.

- What's a SQL database?
- What's a SQL table?

- Write a basic query in SQL.

How many tables are in the `nepal.sqlite` database? What information do they hold?

[6]:
```
%%sql
SELECT  *
FROM sqlite_schema
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[6]:
```
[('table', 'id_map', 'id_map', 2, 'CREATE TABLE "id_map" (\n"household_id"
INTEGER,\n  "building_id" INTEGER,\n  "vdcmun_id" INTEGER,\n  "district_id"
INTEGER\n)'),
 ('index', 'ix_id_map_household_id', 'id_map', 3, 'CREATE INDEX
"ix_id_map_household_id"ON "id_map" ("household_id")'),
 ('table', 'building_structure', 'building_structure', 2032, 'CREATE TABLE
"building_structure" (\n"building_id" INTEGER,\n  "count_floors_pre_eq"
INTEGER,\n  "count_floors_post_eq" INTEGER,\n  "age_building" IN … (198
characters truncated) … or_type" TEXT,\n  "other_floor_type" TEXT,\n
"position" TEXT,\n  "plan_configuration" TEXT,\n  "condition_post_eq" TEXT,\n
"superstructure" TEXT\n)'),
 ('index', 'ix_building_structure_building_id', 'building_structure', 2033,
'CREATE INDEX "ix_building_structure_building_id"ON "building_structure"
("building_id")'),
 ('table', 'building_damage', 'building_damage', 12302, 'CREATE TABLE
"building_damage" (\n"building_id" INTEGER,\n  "damage_overall_collapse" TEXT,\n
"damage_overall_leaning" TEXT,\n  "damage_overall_adja … (2923 characters
truncated) … ndslide" INTEGER,\n  "has_geotechnical_risk_rock_fall" INTEGER,\n
"has_geotechnical_risk_flood" INTEGER,\n  "has_geotechnical_risk_other"
INTEGER\n)'),
 ('index', 'ix_building_damage_building_id', 'building_damage', 12305, 'CREATE
INDEX "ix_building_damage_building_id"ON "building_damage" ("building_id")'),
 ('table', 'household_demographics', 'household_demographics', 31601, 'CREATE
TABLE "household_demographics" (\n"household_id" INTEGER,\n
"gender_household_head" TEXT,\n  "age_household_head" REAL,\n  "caste_household"
… (8 characters truncated) …  "education_level_household_head" TEXT,\n
"income_level_household" TEXT,\n  "size_household" REAL,\n
"is_bank_account_present_in_household" REAL\n)'),
 ('index', 'ix_household_demographics_household_id', 'household_demographics',
31602, 'CREATE INDEX "ix_household_demographics_household_id"ON
"household_demographics" ("household_id")')]
```

[7]:
```
VimeoVideo("665414239", h="d7319aa0a8", width=600)
```

[7]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e8c5640>
```

Task 4.1.3: Select the `name` column from the `sqlite_schema` table, showing only rows where the `type` is `"table"`.

- Select a column from a table in SQL.
- Subset a table using a `WHERE` clause in SQL.

[8]:
```
%%sql
SELECT name
FROM sqlite_schema
WHERE type = "table"
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[8]:
```
[('id_map',),
 ('building_structure',),
 ('building_damage',),
 ('household_demographics',)]
```

[9]:
```
VimeoVideo("665414263", h="5b7d1e875f", width=600)
```

[9]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e8c5040>
```

Task 4.1.4: Select all columns from the `id_map` table, limiting your results to the first five rows.

- Inspect a table using a `LIMIT` clause in SQL.

How is the data organized? What type of observation does each row represent? How do you think the **`household_id`**, **`building_id`**, **`vdcmun_id`**, and **`district_id`** columns are related to each other?

[10]:
```
%%sql
SELECT *
FROM id_map
LIMIT 5
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[10]:
```
[(5601, 56, 7, 1),
 (6301, 63, 7, 1),
 (9701, 97, 7, 1),
 (9901, 99, 7, 1),
 (11501, 115, 7, 1)]
```

[11]:
```
VimeoVideo("665414293", h="72fbe6b7d8", width=600)
```

[11]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e8677c0>
```

Task 4.1.5: How many observations are in the `id_map` table? Use the `count` command to find out.

- Calculate the number of rows in a table using a `count` function in SQL.

[12]:
```
%%sql
SELECT count(*)
FROM id_map
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[12]:
```
[(249932,)]
```

[13]:
```
VimeoVideo("665414303", h="6ba10ddf88", width=600)
```

[13]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e871490>
```

Task 4.1.6: What districts are represented in the `id_map` table? Use the `distinct` command to determine the unique values in the **`district_id`** column.

- Determine the unique values in a column using a `distinct` function in SQL.

[14]:
```
%%sql
SELECT distinct(district_id)
FROM id_map
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[14]:
```
[(1,), (2,), (3,), (4,)]
```

[15]:
```
VimeoVideo("665414313", h="adbab3e418", width=600)
```

[15]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e8c5d00>
```

Task 4.1.7: How many buildings are there in `id_map` table? Combine the `count` and `distinct` commands to calculate the number of unique values in **`building_id`**.

- Calculate the number of rows in a table using a `count` function in SQL.
- Determine the unique values in a column using a `distinct` function in SQL.

[16]:
```
%%sql
SELECT count(distinct(building_id))
FROM id_map
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[16]:
```
[(234835,)]
```

[17]:
```
VimeoVideo("665414336", h="5b595107c6", width=600)
```

[17]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e871790>
```

Task 4.1.8: For our model, we'll focus on Gorkha (district 4). Select all columns that from `id_map`, showing only rows where the **`district_id`** is 4 and limiting your results to the first five rows.

- Inspect a table using a `LIMIT` clause in SQL.
- Subset a table using a `WHERE` clause in SQL.

[18]:
```
%%sql
SELECT *
FROM id_map
WHERE district_id = 4
LIMIT 5
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[18]:
```
[(16400201, 164002, 38, 4),
 (16408101, 164081, 38, 4),
 (16408901, 164089, 38, 4),
 (16409801, 164098, 38, 4),
 (16410301, 164103, 38, 4)]
```

[19]:
```
VimeoVideo("665414344", h="bdcb4a50a3", width=600)
```

[19]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e871b50>
```

Task 4.1.9: How many observations in the `id_map` table come from Gorkha? Use the `count` and `WHERE` commands together to calculate the answer.

- Calculate the number of rows in a table using a `count` function in SQL.
- Subset a table using a `WHERE` clause in SQL.

[20]:
```
%%sql
SELECT count(*)
FROM id_map
WHERE district_id = 4
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[20]:
```
[(75883,)]
```

[21]:
```
VimeoVideo("665414356", h="5d2bdb3813", width=600)
```

[21]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e871a60>
```

Task 4.1.10: How many buildings in the `id_map` table are in Gorkha? Combine the `count` and `distinct` commands to calculate the number of unique values in **`building_id`**, considering only rows where the **`district_id`** is 4.

- Calculate the number of rows in a table using a `count` function in SQL.

- Determine the unique values in a column using a `distinct` function in SQL.
- Subset a table using a `WHERE` clause in SQL.

[22]:
```
%%sql
SELECT count(distinct(building_id)) AS unique_buildings_gorkha
FROM id_map
WHERE district_id = 4
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[22]:
```
[(70836,)]
```

[23]:
```
VimeoVideo("665414390", h="308ea86e4b", width=600)
```

[23]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e882640>
```

Task 4.1.11: Select all the columns from the `building_structure` table, and limit your results to the first five rows.

- Inspect a table using a `LIMIT` clause in SQL.

What information is in this table? What does each row represent? How does it relate to the information in the `id_map` table?

[24]:
```
%%sql
SELECT *
FROM building_structure
LIMIT 5
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[24]:
```
[(1, 1, 1, 9, 288, 9, 9, 'Flat', 'Other', 'Bamboo/Timber-Light roof', 'Mud',
'Not applicable', 'Not attached', 'Rectangular', 'Damaged-Used in risk', 'Stone,
mud mortar'),
 (2, 1, 1, 15, 364, 9, 9, 'Flat', 'Other', 'Bamboo/Timber-Light roof', 'Mud',
'Not applicable', 'Not attached', 'Rectangular', 'Damaged-Repaired and used',
'Stone, mud mortar'),
 (3, 1, 1, 20, 384, 9, 9, 'Flat', 'Other', 'Bamboo/Timber-Light roof', 'Mud',
'Not applicable', 'Not attached', 'Rectangular', 'Damaged-Repaired and used',
'Stone, mud mortar'),
 (4, 1, 1, 20, 312, 9, 9, 'Flat', 'Other', 'Bamboo/Timber-Light roof', 'Mud',
'Not applicable', 'Not attached', 'Rectangular', 'Damaged-Repaired and used',
'Stone, mud mortar'),
 (5, 1, 1, 30, 308, 9, 9, 'Flat', 'Other', 'Bamboo/Timber-Light roof', 'Mud',
'Not applicable', 'Not attached', 'Rectangular', 'Damaged-Repaired and used',
'Stone, mud mortar')]
```

[25]:
```
VimeoVideo("665414402", h="64875c7779", width=600)
```

[25]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e882a60>
```

Task 4.1.12: How many building are there in the `building_structure` table? Use the `count` command to find out.

- Calculate the number of rows in a table using a `count` function in SQL.

[26]:
```
%%sql
SELECT count(*)
FROM building_structure
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[26]:
```
[(234835,)]
```

[27]:
```
VimeoVideo("665414414", h="202f83f3cb", width=600)
```

[27]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e80c2b0>
```

Task 4.1.13: There are over 200,000 buildings in the `building_structure` table, but how can we retrieve only buildings that are in Gorkha? Use the `JOIN` command to join the `id_map` and `building_structure` tables, showing only buildings where **`district_id`** is `4` and limiting your results to the first five rows of the new table.

- Create an alias for a column or table using the `AS` command in SQL.
- Merge two tables using a `JOIN` clause in SQL.
- Inspect a table using a `LIMIT` clause in SQL.
- Subset a table using a `WHERE` clause in SQL.

[33]:
```
%%sql
SELECT *
FROM id_map AS i
JOIN building_structure AS s ON i.building_id = s.building_id
WHERE district_id = 4
LIMIT 5

# Above code is a example of Left Join
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[33]:
```
[(16400201, 164002, 38, 4, 164002, 3, 3, 20, 560, 18, 18, 'Flat', 'Mud mortar-
Stone/Brick', 'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not
attached', 'Rectangular', 'Damaged-Repaired and used', 'Stone, mud mortar'),
 (16408101, 164081, 38, 4, 164081, 2, 2, 21, 200, 12, 12, 'Flat', 'Mud mortar-
Stone/Brick', 'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not
```

```
attached', 'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar'),
 (16408901, 164089, 38, 4, 164089, 3, 3, 18, 315, 20, 20, 'Flat', 'Mud mortar-
Stone/Brick', 'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not
attached', 'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar'),
 (16409801, 164098, 38, 4, 164098, 2, 2, 45, 290, 13, 13, 'Flat', 'Mud mortar-
Stone/Brick', 'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not
attached', 'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar'),
 (16410301, 164103, 38, 4, 164103, 2, 2, 21, 230, 13, 13, 'Flat', 'Mud mortar-
Stone/Brick', 'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not
attached', 'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar')]
```

In the table we just made, each row represents a unique household in Gorkha. How can we create a table where each row represents a unique building?

[29]:
```
VimeoVideo("665414450", h="0fcb4dc3fa", width=600)
```

[29]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e8c5f10>
```

Task 4.1.14: Use the `distinct` command to create a column with all unique building IDs in the `id_map` table. `JOIN` this column with all the columns from the `building_structure` table, showing only buildings where **`district_id`** is 4 and limiting your results to the first five rows of the new table.

- Create an alias for a column or table using the `AS` command in SQL.
- Determine the unique values in a column using a `distinct` function in SQL.
- Merge two tables using a `JOIN` clause in SQL.
- Inspect a table using a `LIMIT` clause in SQL.
- Subset a table using a `WHERE` clause in SQL.

[35]:
```
%%sql
SELECT distinct(i.building_id),
       s.*
FROM id_map AS i
JOIN building_structure AS s ON i.building_id = s.building_id
WHERE district_id = 4
LIMIT 5
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[35]:
```
[(164002, 164002, 3, 3, 20, 560, 18, 18, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Repaired and used', 'Stone, mud mortar'),
 (164081, 164081, 2, 2, 21, 200, 12, 12, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar'),
 (164089, 164089, 3, 3, 18, 315, 20, 20, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
```

```
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar'),
 (164098, 164098, 2, 2, 45, 290, 13, 13, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar'),
 (164103, 164103, 2, 2, 21, 230, 13, 13, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar')]
```

We've combined the `id_map` and `building_structure` tables to create a table with all the buildings in Gorkha, but the final piece of data needed for our model, the damage that each building sustained in the earthquake, is in the `building_damage` table.

[36]:
```
VimeoVideo("665414466", h="37dde03c93", width=600)
```

[36]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e871670>
```

Task 4.1.15: How can combine all three tables? Using the query you created in the last task as a foundation, include the **`damage_grade`** column to your table by adding a second `JOIN` for the `building_damage` table. Be sure to limit your results to the first five rows of the new table.

- Create an alias for a column or table using the `AS` command in SQL.
- Determine the unique values in a column using a `distinct` function in SQL.
- Merge two tables using a `JOIN` clause in SQL.
- Inspect a table using a `LIMIT` clause in SQL.
- Subset a table using a `WHERE` clause in SQL.

[40]:
```
%%sql
SELECT distinct(i.building_id) AS b_id,
       s.*,
       d.damage_grade
FROM id_map AS i
JOIN building_structure AS s ON i.building_id = s.building_id
JOIN building_damage AS d ON i.building_id = d.building_id
WHERE district_id = 4
LIMIT 5
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[40]:
```
[(164002, 164002, 3, 3, 20, 560, 18, 18, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Repaired and used', 'Stone, mud mortar', 'Grade 2'),
 (164081, 164081, 2, 2, 21, 200, 12, 12, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar', 'Grade 2'),
 (164089, 164089, 3, 3, 18, 315, 20, 20, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar', 'Grade 2'),
```

```
 (164098, 164098, 2, 2, 45, 290, 13, 13, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar', 'Grade 3'),
 (164103, 164103, 2, 2, 21, 230, 13, 13, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar', 'Grade 3')]
```

1.3 Import

[41]:
```
VimeoVideo("665414492", h="9392e1a66e", width=600)
```

[41]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e8671c0>
```

Task 4.1.16: Use the `connect` method from the sqlite3 library to connect to the database. Remember that the database is located at `"/home/jovyan/nepal.sqlite"`.

- Open a connection to a SQL database using sqlite3.

[42]:
```
conn = sqlite3.connect("/home/jovyan/nepal.sqlite")
```

[43]:
```
VimeoVideo("665414501", h="812d482c73", width=600)
```

[43]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e867e80>
```

Task 4.1.17: Put your last SQL query into a string and assign it to the variable `query`.

[44]:
```
query = """
SELECT distinct(i.building_id) AS b_id,
       s.*,
       d.damage_grade
FROM id_map AS i
JOIN building_structure AS s ON i.building_id = s.building_id
JOIN building_damage AS d ON i.building_id = d.building_id
WHERE district_id = 4
LIMIT 5

"""
print(query)
```

```
SELECT distinct(i.building_id) AS b_id,
       s.*,
       d.damage_grade
FROM id_map AS i
JOIN building_structure AS s ON i.building_id = s.building_id
JOIN building_damage AS d ON i.building_id = d.building_id
WHERE district_id = 4
LIMIT 5
```

[45]:
```
VimeoVideo("665414513", h="c6a81b49ad", width=600)
```

[45]:
```
<IPython.lib.display.VimeoVideo at 0x7fe13e867fa0>
```

Task 4.1.18: Use the `read_sql` from the pandas library to create a DataFrame from your `query`. Be sure that the `building_id` is set as your index column.

- Read SQL query into a DataFrame using pandas.

Tip: Your table might have two building_id columns, and that will make it hard to set it as the index column for your DataFrame. If you face this problem, add an alias for one of the building_id columns in your query using AS.

[51]:
```
df = pd.read_sql(query, conn, index_col="b_id")

df.head()
```

[51]:

b_id	building_id	count_floors_pre_eq	count_floors_post_eq	age_building	plinth_area_sq_ft	height_ft_pre_eq	height_ft_post_eq	land_surface_condition	foundation_type	roof_type	ground_floor_type	other_floor_type
164002	164002	3	3	20	560	18	18	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	TImber/Bamboo-Mud
164081	164081	2	2	21	200	12	12	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	TImber/Bamboo-Mud
164089	164089	3	3	18	315	20	20	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	TImber/Bamboo-Mud
164098	164098	2	2	45	290	13	13	Flat	Mud mortar-Stone/Brick			
164103	164103	2	2	21	230	13	13	Flat	Mud mortar-Stone/Brick			

```
164098  Bamboo/Timber-Light roof                Mud  TImber/Bamboo-Mud
164103  Bamboo/Timber-Light roof                Mud  TImber/Bamboo-Mud

            position plan_configuration          condition_post_eq  \
b_id
164002  Not attached        Rectangular  Damaged-Repaired and used
164081  Not attached        Rectangular       Damaged-Used in risk
164089  Not attached        Rectangular       Damaged-Used in risk
164098  Not attached        Rectangular       Damaged-Used in risk
164103  Not attached        Rectangular       Damaged-Used in risk

           superstructure damage_grade
b_id
164002  Stone, mud mortar      Grade 2
164081  Stone, mud mortar      Grade 2
164089  Stone, mud mortar      Grade 2
164098  Stone, mud mortar      Grade 3
164103  Stone, mud mortar      Grade 3
```

[50]:
```python
# Check your work
assert df.shape[0] == 70836, f"`df` should have 70,836 rows, not {df.shape[0]}."
assert (
    df.shape[1] > 14
), "`df` seems to be missing columns. Does your query combine the `id_map`,␣
 ↪`building_structure`, and `building_damage` tables?"
assert (
    "damage_grade" in df.columns
), "`df` is missing the target column, `'damage_grade'`."
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [50], in <cell line: 2>()
      1 # Check your work
----> 2 assert df.shape[0] == 70836, f"`df` should have 70,836 rows, not {df.
 ↪shape[0]}."
      3 assert (
      4     df.shape[1] > 14
      5 ), "`df` seems to be missing columns. Does your query combine the␣
 ↪`id_map`, `building_structure`, and `building_damage` tables?"
      6 assert (
      7     "damage_grade" in df.columns
      8 ), "`df` is missing the target column, `'damage_grade'`."

AssertionError: `df` should have 70,836 rows, not 5.
```