I first viewed the nashville\_housing dataset in Google Sheets; I formatted the sale\_date column to fit the PostgreSQL date data type; and I renamed the columns using small letters and underscores to separate words before downloading the csv file to be imported into PostgreSQL.

The nashville\_housing dataset has a total of 19 columns and 56,477 rows

I began by creating a table using the following SQL query;

**---importing dataset**

**CREATE TABLE** public.nashville (

unique\_id integer **PRIMARY KEY**,

parcel\_id varchar(28),

land\_use text,

property\_address varchar(50),

sale\_date date,

sale\_price varchar(50),

legal\_reference varchar(50),

sold\_as\_vacant BOOLEAN,

owner\_name varchar(60),

owner\_address varchar (60),

acreage double precision,

tax\_district text,

land\_value integer,

building\_value integer,

total\_value integer,

year\_built integer,

bedrooms integer,

full\_bath integer,

half\_bath integer

);

Then, I refreshed the public schema in my database for the public.nashville table to appear, and then I imported the dataset into the table. I used the code below to view the imported data for inspection;

**SELECT** \*

**FROM** public.nashville

**--populate property\_address data**

Upon inspection, I noticed that the property\_address data was missing in 29 rows and the data could be populated from the same table, using a SELF JOIN on the basis that the parcel\_id is the same, i.e. if 2 rows share the same parcel\_id, then the property address is the same. So using the available parcel\_id data, I populated the missing property\_address values with the known ones, using the code below;

**--to view the null rows**

**SELECT** \*

**FROM** public.nashville

**WHERE** property\_address IS NULL

**--query to populate the missing values**

**UPDATE** public.nashville

**SET** property\_address = **COALESCE**(a.property\_address,b.property\_address)

**FROM** public.nashville a

**JOIN** public.nashville b

**ON** a.parcel\_id = b.parcel\_id

**AND** a.unique\_id != b.unique\_id

**WHERE** a.property\_address is null

**--Breaking out property\_address into individual columns (street, state)**

I realized that this column was too cluttered and the information could be better analyzed separately than combined.

**SELECT** property\_address

**FROM** public.nashville

**--query to separate data**

**ALTER TABLE** public.nashville

**ADD COLUMN** property\_address\_street varchar(50)

**ALTER TABLE** public.nashville

**ADD COLUMN** property\_address\_state varchar(50)

**UPDATE** public.nashville

**SET** property\_address\_street = substring(property\_address, 1, strpos(property\_address, ',') - 1);

**UPDATE** public.nashville

**SET** property\_address\_state = substring(property\_address, strpos(property\_address, ',') + 1);

**--Breaking out owner\_address into individual columns (street, city, state)**

I decided to do the same thing with the owner\_address column for the same reason, but I opted to use a different method from the first time, using **split\_part()**, as opposed to **substring()**

**SELECT** owner\_address

**FROM** public.nashville

**--query to separate data**

**ALTER TABLE** public.nashville

**ADD COLUMN** owner\_address\_street varchar(50)

**ALTER TABLE** public.nashville

**ADD COLUMN** owner\_address\_city varchar(50)

**ALTER TABLE** public.nashville

**ADD COLUMN** owner\_address\_state varchar (50)

**UPDATE** public.nashville

**SET** (owner\_address\_street, owner\_address\_city, owner\_address\_state) =

(split\_part(owner\_address, ',', 1),

split\_part(owner\_address, ',', 2),

split\_part(owner\_address, ',', 3))

**--removing irrelevant columns**

I removed the old address columns, as I no longer needed them for analysis and could easily recombine the data if needed into a single column.

**SELECT** \*

**FROM** public.nashville

**ALTER TABLE** public.nashville

**DROP COLUMN** owner\_address

**ALTER TABLE** public.nashville

**DROP COLUMN** property\_address

This was the last step of my data cleaning process. I decided not to remove any null or duplicate values, as the columns containing duplicate values make sense when duplicated, e.g., one owner can have more than one property, so the owner name can appear multiple times. The columns with null values also do not affect the analysis significantly at this point, as they are not the focus of my analysis.

The final table now has 22 columns and remains with a total of 56,477 rows.

--Github link to see full SQL query and processes