proj1a

November 25, 2024

[41]:
```python
# Initialize Otter
import otter
grader = otter.Notebook("proj1a.ipynb")
```

1 Project 1A: Exploring Cook County Housing

1.1 Due Date: Thursday, October 13th, 11:59 PM PDT

1.1.1 Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** in the collaborators cell below.

Collaborators: *list names here*

1.2 Introduction

This project explores what can be learned from an extensive housing data set that is embedded in a dense social context in Cook County, Illinois.

Here in part A, we will guide you through some basic exploratory data analysis (EDA) to understand the structure of the data. Next, you will be adding a few new features to the dataset, while cleaning the data as well in the process.

In part B, you will specify and fit a linear model for the purpose of prediction. Finally, we will analyze the error of the model and brainstorm ways to improve the model's performance.

1.3 Score Breakdown

Question	Part	Points
1	1	1
1	2	1
1	3	1
1	4	1
2	1	1
2	2	1
3	1	3
3	2	1

Question	Part	Points
3	3	1
4	-	2
5	1	1
5	2	2
5	3	2
6	1	1
6	2	2
6	3	1
6	4	2
6	5	1
7	1	1
7	2	2
Total	-	28

[42]:
```
import numpy as np

import pandas as pd
from pandas.api.types import CategoricalDtype

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

import zipfile
import os

from ds100_utils import run_linear_regression_test

# Plot settings
plt.rcParams['figure.figsize'] = (12, 9)
plt.rcParams['font.size'] = 12
```

2 The Data

The data set consists of over 500 thousand records from Cook County, Illinois, the county where Chicago is located. The data set we will be working with has 61 features in total; the 62nd is sales price, which you will predict with linear regression in the next part of this project. An explanation of each variable can be found in the included `codebook.txt` file. Some of the columns have been filtered out to ensure this assignment doesn't become overly long when dealing with data cleaning and formatting.

The data are split into training and test sets with 204,792 and 68,264 observations, respectively, but we will only be working on the training set for this part of the project.

Let's first extract the data from the `cook_county_data.zip`. Notice we didn't leave the `csv` files directly in the directory because they take up too much space without some prior compression.

[43]:
```
with zipfile.ZipFile('cook_county_data.zip') as item:
    item.extractall()
```

Let's load the training data.

[44]:
```
training_data = pd.read_csv("cook_county_train.csv", index_col='Unnamed: 0')
```

As a good sanity check, we should at least verify that the data shape matches the description.

[45]:
```
# 204792 observations and 62 features in training data
assert training_data.shape == (204792, 62)
# Sale Price is provided in the training data
assert 'Sale Price' in training_data.columns.values
```

The next order of business is getting a feel for the variables in our data. A more detailed description of each variable is included in `codebook.txt` (in the same directory as this notebook). **You should take some time to familiarize yourself with the codebook before moving forward.**

Let's take a quick look at all the current columns in our training data.

[46]:
```
training_data.head(2)
```

[46]:
```
              PIN  Property Class  Neighborhood Code  Land Square Feet  \
0  17294100610000             203                 50            2500.0
1  13272240180000             202                120            3780.0

   Town Code  Apartments  Wall Material  Roof Material  Basement  \
0         76         0.0            2.0            1.0       1.0
1         71         0.0            2.0            1.0       1.0

   Basement Finish  …  Sale Month of Year  Sale Half of Year  \
0              3.0  …                   9                  2
1              1.0  …                   5                  1

   Most Recent Sale  Age Decade  Pure Market Filter  Garage Indicator  \
0               1.0        13.2                   0               0.0
1               1.0         9.6                   1               1.0

   Neigborhood Code (mapping)  Town and Neighborhood  \
0                          50                   7650
1                         120                  71120

                                         Description  Lot Size
0  This property, sold on 09/14/2015, is a one-st…    2500.0
```

```
1  This property, sold on 05/23/2018, is a one-st…    3780.0

[2 rows x 62 columns]
```

[47]:
```python
training_data.columns
```

[47]:
```
Index(['PIN', 'Property Class', 'Neighborhood Code', 'Land Square Feet',
       'Town Code', 'Apartments', 'Wall Material', 'Roof Material', 'Basement',
       'Basement Finish', 'Central Heating', 'Other Heating', 'Central Air',
       'Fireplaces', 'Attic Type', 'Attic Finish', 'Design Plan',
       'Cathedral Ceiling', 'Construction Quality', 'Site Desirability',
       'Garage 1 Size', 'Garage 1 Material', 'Garage 1 Attachment',
       'Garage 1 Area', 'Garage 2 Size', 'Garage 2 Material',
       'Garage 2 Attachment', 'Garage 2 Area', 'Porch', 'Other Improvements',
       'Building Square Feet', 'Repair Condition', 'Multi Code',
       'Number of Commercial Units', 'Estimate (Land)', 'Estimate (Building)',
       'Deed No.', 'Sale Price', 'Longitude', 'Latitude', 'Census Tract',
       'Multi Property Indicator', 'Modeling Group', 'Age', 'Use',
       'O'Hare Noise', 'Floodplain', 'Road Proximity', 'Sale Year',
       'Sale Quarter', 'Sale Half-Year', 'Sale Quarter of Year',
       'Sale Month of Year', 'Sale Half of Year', 'Most Recent Sale',
       'Age Decade', 'Pure Market Filter', 'Garage Indicator',
       'Neigborhood Code (mapping)', 'Town and Neighborhood', 'Description',
       'Lot Size'],
      dtype='object')
```

[48]:
```python
training_data['Property Class'].unique()
```

[48]:
```
array([203, 202, 208, 205, 207, 206, 204, 278, 209])
```

[49]:
```python
training_data['Sale Quarter'].unique()
```

[49]:
```
array([75, 86, 77, 67, 78, 87, 84, 79, 90, 82, 70, 65, 73, 91, 89, 72, 74,
       88, 92, 80, 83, 71, 81, 76, 69, 66, 68, 85])
```

[50]:
```python
training_data['Sale Quarter of Year'].unique()
```

[50]:
```
array([3, 2, 1, 4])
```

[51]:
```python
training_data['Age'].unique()
```

[51]:
```
array([132,  96, 112,  63,  58, 109,  17, 100,  48,  74,  34,  13, 122,
        16,  59,  94,  87,  41,  65,  69,   1,  64,  95,  27,  92,  62,
        73,  67, 107,  93,  54,  33,  40,   7,  91,  42,  10,  38,  28,
        25, 102,  49,  57,  61,  30,  39,  66, 101,  71,  47,   9,  51,
        60,  88, 105,  75,  89,  77,  44,  46, 115,  90,  68, 117,  43,
       106,  55, 110,  85,  37,  76, 134,  80, 129,  45,  50,  19,  84,
```

```
        53,  35,   8,   4, 123, 140,  56,  78,  36, 121, 118,  86,  31,
       127, 128,  15,  29,  52,  18,  83,  82, 104,  98,  97, 114, 113,
        99,  24,  70, 120, 116, 111, 125,  32, 124, 136,  72,  21, 103,
        22,  20, 108, 133,   6, 150, 130,   5,  12,  79, 137,  14,  26,
       143,  11, 119,  81, 126, 147, 144, 151, 135,  23, 142, 131, 139,
       138, 155, 148, 141, 145, 159,   3, 172, 146, 154, 156, 149, 152,
       153,   2, 160, 165, 161, 163, 169, 164, 158, 157])
```

[52]:
```python
training_data.columns.values
```

[52]:
```
array(['PIN', 'Property Class', 'Neighborhood Code', 'Land Square Feet',
       'Town Code', 'Apartments', 'Wall Material', 'Roof Material',
       'Basement', 'Basement Finish', 'Central Heating', 'Other Heating',
       'Central Air', 'Fireplaces', 'Attic Type', 'Attic Finish',
       'Design Plan', 'Cathedral Ceiling', 'Construction Quality',
       'Site Desirability', 'Garage 1 Size', 'Garage 1 Material',
       'Garage 1 Attachment', 'Garage 1 Area', 'Garage 2 Size',
       'Garage 2 Material', 'Garage 2 Attachment', 'Garage 2 Area',
       'Porch', 'Other Improvements', 'Building Square Feet',
       'Repair Condition', 'Multi Code', 'Number of Commercial Units',
       'Estimate (Land)', 'Estimate (Building)', 'Deed No.', 'Sale Price',
       'Longitude', 'Latitude', 'Census Tract',
       'Multi Property Indicator', 'Modeling Group', 'Age', 'Use',
       "O'Hare Noise", 'Floodplain', 'Road Proximity', 'Sale Year',
       'Sale Quarter', 'Sale Half-Year', 'Sale Quarter of Year',
       'Sale Month of Year', 'Sale Half of Year', 'Most Recent Sale',
       'Age Decade', 'Pure Market Filter', 'Garage Indicator',
       'Neigborhood Code (mapping)', 'Town and Neighborhood',
       'Description', 'Lot Size'], dtype=object)
```

[53]:
```python
training_data.values
```

[53]:
```
array([[17294100610000, 203, 50, …, 7650,
        'This property, sold on 09/14/2015, is a one-story houeshold located at
2950 S LYMAN ST.It has a total of 6 rooms, 3 of which are bedrooms, and 1.0 of
which are bathrooms.',
        2500.0],
       [13272240180000, 202, 120, …, 71120,
        'This property, sold on 05/23/2018, is a one-story houeshold located at
2844 N LOWELL AVE.It has a total of 6 rooms, 3 of which are bedrooms, and 1.0 of
which are bathrooms.',
        3780.0],
       [25221150230000, 202, 210, …, 70210,
        'This property, sold on 02/18/2016, is a one-story houeshold located at
11415 S PRAIRIE AVE.It has a total of 7 rooms, 3 of which are bedrooms, and 1.0
of which are bathrooms.',
        4375.000000000001],
```

```
        …,
        [16333020150000, 202, 90, …, 1590,
         'This property, sold on 01/31/2014, is a one-story houeshold located at
3525 S 55TH AVE.It has a total of 5 rooms, 3 of which are bedrooms, and 2.0 of
which are bathrooms.',
         3810.0],
        [9242030500000, 203, 80, …, 2280,
         'This property, sold on 02/22/2018, is a one-story houeshold located at
8430 N OSCEOLA AVE.It has a total of 5 rooms, 3 of which are bedrooms, and 1.0
of which are bathrooms.',
         6649.999999999999],
        [19102030080000, 203, 30, …, 7230,
         'This property, sold on 04/22/2014, is a one-story houeshold located at
4209 W 47TH ST.It has a total of 4 rooms, 2 of which are bedrooms, and 1.0 of
which are bathrooms.',
         2500.0]], dtype=object)
```

[54]:
```python
training_data['Description'][0]
```

[54]:
```
'This property, sold on 09/14/2015, is a one-story houeshold located at 2950 S
LYMAN ST.It has a total of 6 rooms, 3 of which are bedrooms, and 1.0 of which
are bathrooms.'
```

[55]:
```python
training_data['Description']
```

[55]:
```
0         This property, sold on 09/14/2015, is a one-st…
1         This property, sold on 05/23/2018, is a one-st…
2         This property, sold on 02/18/2016, is a one-st…
3         This property, sold on 07/23/2013, is a one-st…
4         This property, sold on 06/10/2016, is a one-st…
                                …
204787    This property, sold on 07/23/2014, is a one-st…
204788    This property, sold on 03/27/2019, is a one-st…
204789    This property, sold on 01/31/2014, is a one-st…
204790    This property, sold on 02/22/2018, is a one-st…
204791    This property, sold on 04/22/2014, is a one-st…
Name: Description, Length: 204792, dtype: object
```

3 Part 1: Contextualizing the Data

Let's try to understand the background of our dataset before diving into a full-scale analysis.

3.1 Question 1

3.1.1 Part 1

Based on the columns present in this data set and the values that they take, what do you think each row represents? That is, what is the granularity of this data set?

Each row represents one property (apartment buildings count as one property; essentially a property is the physical building, not a specific living place for a family).

3.1.2 Part 2

Why do you think this data was collected? For what purposes? By whom?

This question calls for your speculation and is looking for thoughtfulness, not correctness.

There's great motive to see how properties with certain features affect its sale price, aka how other people value the property. This is helpful for home sellers and buyers to know. The data could have been collected by either one of those groups, who stand to benefit from the information.

3.1.3 Part 3

Certain variables in this data set contain information that either directly contains demographic information (data on people) or could when linked to other data sets. Identify at least one demographic-related variable and explain the nature of the demographic data it embeds.

Census Tract feature links the people who live in the property to the results of the census they filled out. This gives demographic data common in census data, including race/age/sex.

3.1.4 Part 4

Craft at least two questions about housing in Cook County that can be answered with this data set and provide the type of analytical tool you would use to answer it (e.g. "I would create a _____ plot of _____ and ***" or "I would calculate the*** [summary statistic] for _____ and ______"). Be sure to reference the columns that you would use and any additional data sets you would need to answer that question.

A question I would want to answer is how the age of a property relates to its sale price. I would create a scatter plot of Age and Sale Price to see the association.

Another question I would answer is how the majority demographic of a neighborhood relates to the sale price/value of the proprety, with the hypothesis being that certain demographics are living in clumps and in different valued properties. I would group by Neighborhood Codes, find the average sale price of a property in that neighborhood, and draw a bar chart with neighborhood codes on the x axis, sale price on the y axis, and use some sort of coloring scheme to indicate which demographics are the majority in which neighborhoods.

4 Part 2: Exploratory Data Analysis

This data set was collected by the Cook County Assessor's Office in order to build a model to predict the monetary value of a home (if you didn't put this for your answer for Question 1 Part 2, please don't go back and change it - we wanted speculation!). You can read more about data collection in the CCAO's Residential Data Integrity Preliminary Report. In part 2 of this project you will be building a linear model that predict sales prices using training data but it's important to first understand how the structure of the data informs such a model. In this section, we will make a series of exploratory visualizations and feature engineering in preparation for that prediction task.

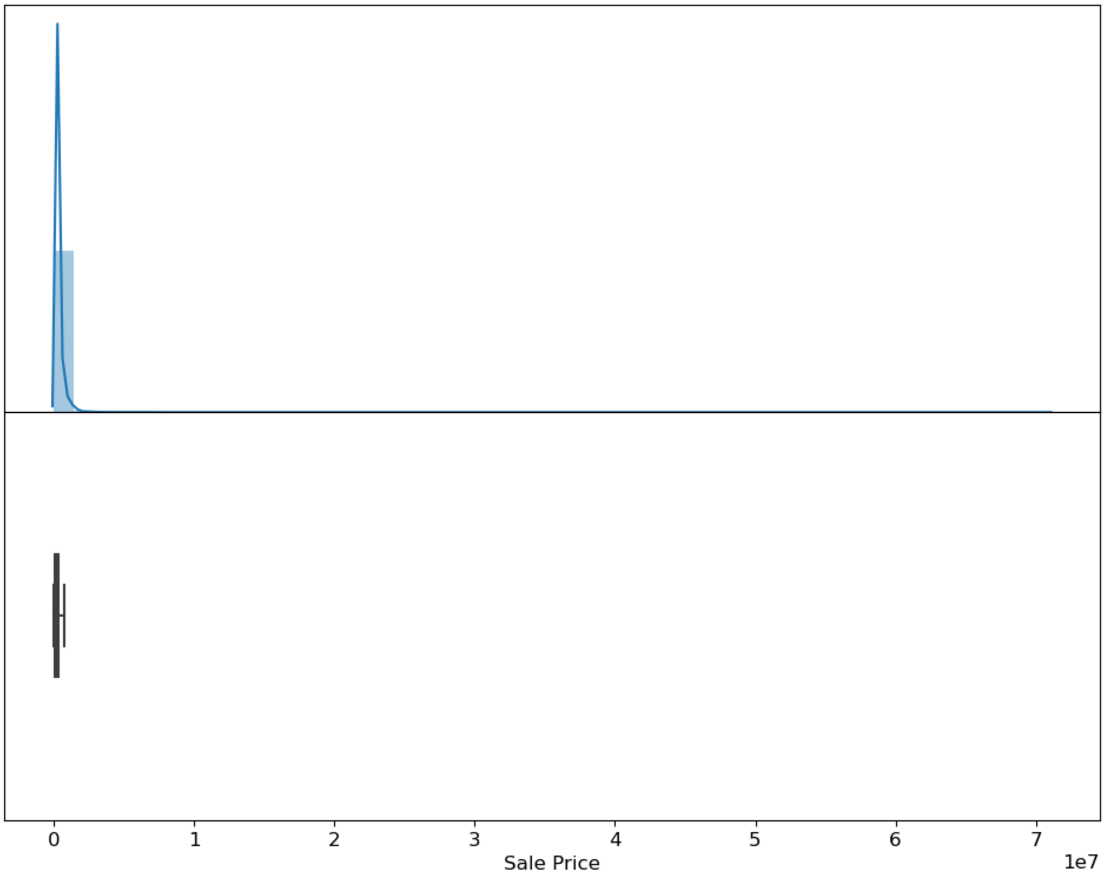
Note that we will perform EDA on the **training data**.

4.0.1 Sale Price

We begin by examining the distribution of our target variable `SalePrice`. At the same time, we also take a look at some descriptive statistics of this variable. We have provided the following helper method `plot_distribution` that you can use to visualize the distribution of the `SalePrice` using both the histogram and the box plot at the same time. Run the following 2 cells and describe what you think is wrong with the visualization.

[56]:
```python
def plot_distribution(data, label):
    fig, axs = plt.subplots(nrows=2)

    sns.distplot(
        data[label],
        ax=axs[0]
    )
    sns.boxplot(
        data[label],
        width=0.3,
        ax=axs[1],
        showfliers=False,
    )

    # Align axes
    spacer = np.max(data[label]) * 0.05
    xmin = np.min(data[label]) - spacer
    xmax = np.max(data[label]) + spacer
    axs[0].set_xlim((xmin, xmax))
    axs[1].set_xlim((xmin, xmax))

    # Remove some axis text
    axs[0].xaxis.set_visible(False)
    axs[0].yaxis.set_visible(False)
    axs[1].yaxis.set_visible(False)

    # Put the two plots together
    plt.subplots_adjust(hspace=0)
```

[57]:
```python
plot_distribution(training_data, label='Sale Price')
```



4.1 Question 2

4.1.1 Part 1

Identify one issue with the visualization above and briefly describe one way to overcome it. You may also want to try running `training_data['Sale Price'].describe()` in a different cell to see some specific summary statistics on the distribution of the target variable. Make sure to delete the cell afterwards as the autograder may not work otherwise.

There's too much white space on the horizontal axis. The scale is also terrible; the view is too zoomed out to see anything meaninful. I would shorten the x axis considerably, probably through taking the log of the x axis.

[58]:
```
training_data['Sale Price'].describe()
```

[58]:
```
count    2.047920e+05
mean     2.451646e+05
std      3.628694e+05
min      1.000000e+00
25%      4.520000e+04
```

```
50%      1.750000e+05
75%      3.120000e+05
max      7.100000e+07
Name: Sale Price, dtype: float64
```

4.1.2 Part 2

To zoom in on the visualization of most households, we will focus only on a subset of `Sale Price` for this assignment. In addition, it may be a good idea to apply log transformation to `Sale Price`. In the cell below, reassign `training_data` to a new dataframe that is the same as the original one **except with the following changes**:

- `training_data` should contain only households whose price is at least $500.
- `training_data` should contain a new `Log Sale Price` column that contains the log-transformed sale prices.

Note: This also implies from now on, our target variable in the model will be the log transformed sale prices from the column `Log Sale Price`.

Note: You should **NOT** remove the original column `Sale Price` as it will be helpful for later questions.

To ensure that any error from this part does not propagate to later questions, there will be no hidden test here.

[59]:
```
training_data[['Sale Price']]
```

[59]:
```
        Sale Price
0                1
1           285000
2            22000
3           225000
4            22600
…                …
204787       37100
204788      225000
204789      135000
204790      392000
204791      125000

[204792 rows x 1 columns]
```

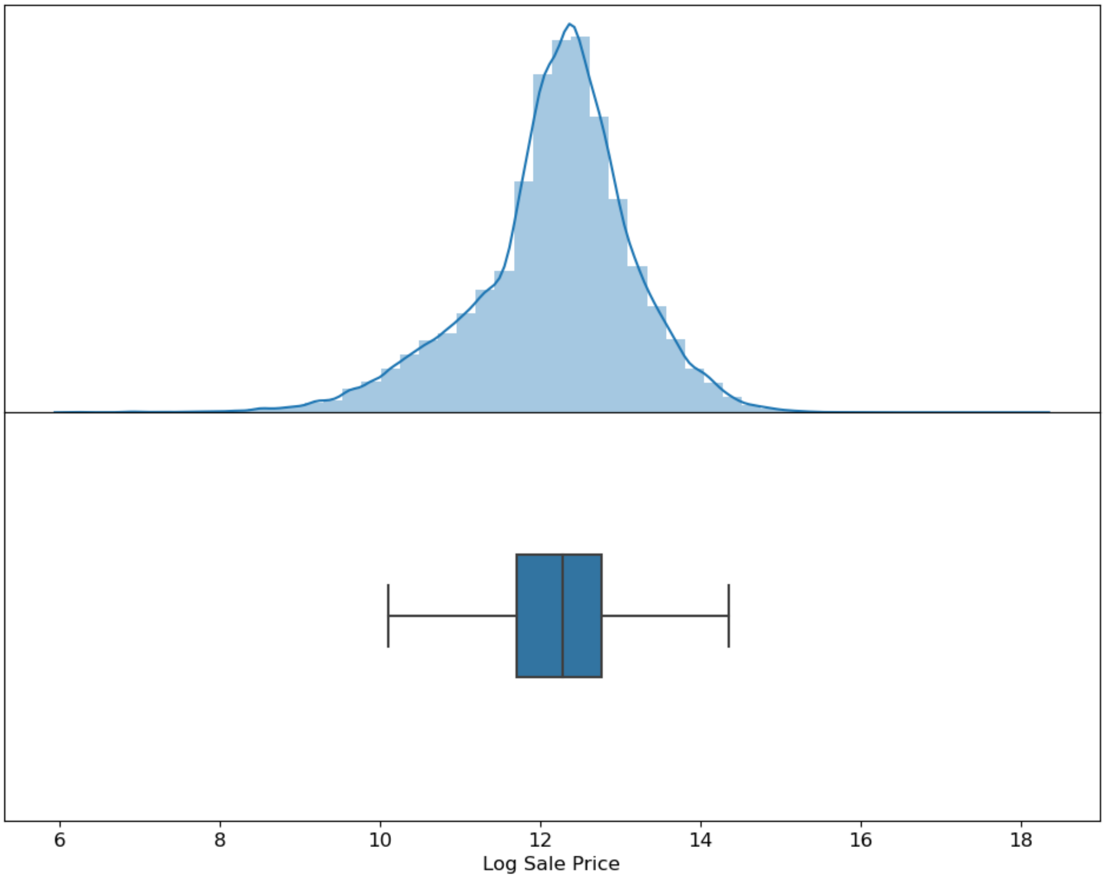
[60]:
```
training_data = training_data[training_data['Sale Price'] >= 500]
training_data['Log Sale Price'] = np.log(training_data['Sale Price'])
```

[61]:
```
grader.check("q2b")
```

[61]:
```
q2b results: All test cases passed!
```

Let's create a new distribution plot on the log-transformed sale price.

[62]:
```
plot_distribution(training_data, label='Log Sale Price');
```



4.2 Question 3

4.2.1 Part 1

To check your understanding of the graph and summary statistics above, answer the following `True` or `False` questions:

1. The distribution of `Log Sale Price` in the training set is symmetric.
2. The mean of `Log Sale Price` in the training set is greater than the median.
3. At least 25% of the houses in the training set sold for more than $200,000.00.

The provided tests for this question do not confirm that you have answered correctly; only that you have assigned each variable to `True` *or* `False`*.*

[63]:
```
training_data['Log Sale Price'].describe()
```

[63]:
```
count    168931.000000
mean         12.168227
std           0.999586
min           6.214608
25%          11.703546
50%          12.278393
75%          12.765688
max          18.078190
Name: Log Sale Price, dtype: float64
```

[64]:
```
training_data['Sale Price'].describe()
```

[64]:
```
count    1.689310e+05
mean     2.972082e+05
std      3.796823e+05
min      5.000000e+02
25%      1.210000e+05
50%      2.150000e+05
75%      3.500000e+05
max      7.100000e+07
Name: Sale Price, dtype: float64
```

[65]:
```
# These should be True or False
q3statement1 = True
q3statement2 = False
q3statement3 = True
```

[66]:
```
grader.check("q3a")
```

[66]:
```
q3a results: All test cases passed!
```

4.2.2 Part 2

Next, we want to explore if any there is any correlation between `Log Sale Price` and the total area occupied by the household. The `codebook.txt` file tells us the column `Building Square Feet` should do the trick – it measures "(from the exterior) the total area, in square feet, occupied by the building".

Before creating this jointplot however, let's also apply a log transformation to the `Building Square Feet` column.

In the following cell, create a new column `Log Building Square Feet` in our `training_data` that contains the log transformed area occupied by each household.

You should NOT remove the original `Building Square Feet` column this time as it will be used for later questions.

To ensure that any errors from this part do not propagate to later questions, there will be no hidden tests here.

[67]:
```
training_data['Log Building Square Feet'] = np.log(training_data['Building␣
↪Square Feet'])
```

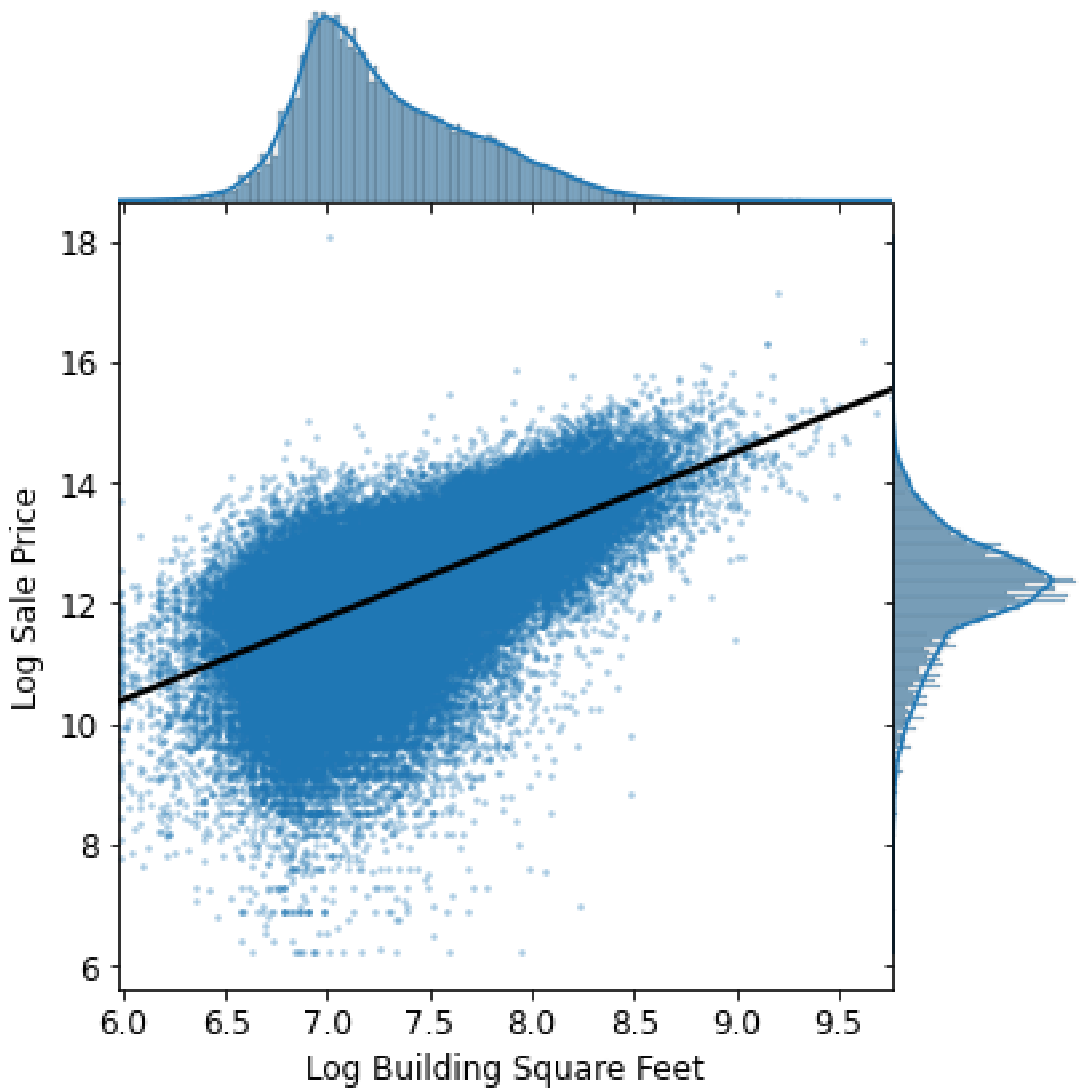
[68]:
```
grader.check("q3b")
```

[68]: q3b results: All test cases passed!

4.2.3 Part 3

As shown below, we created a joint plot with `Log Building Square Feet` on the x-axis, and `Log Sale Price` on the y-axis. In addition, we fit a simple linear regression line through the bivariate scatter plot in the middle.

Based on the following plot, does there exist a correlation between `Log Sale Price` and `Log Building Square Feet`? Would `Log Building Square Feet` make a good candidate as one of the features for our model?



There seems to be a decent correlation between Log Sale Price and Log Building Square Feet because there is a general positive association between the two variables. However, there's a distinct pattern to the residuals of the regression line, in that there's a bigger variance among smaller log building square feet values than bigger values. This suggests there's a better predictor available.

4.3 Question 4

Continuing from the previous part, as you explore the data set, you might still run into more outliers that prevent you from creating a clear visualization or capturing the trend of the majority of the houses.

For this assignment, we will work to remove these outliers from the data as we run into them. Write a function `remove_outliers` that removes outliers from a data set based off a threshold value of a variable. For example, `remove_outliers(training_data, 'Building Square Feet', upper=8000)` should return a data frame with only observations that satisfy `Building Square Feet` less than or equal to 8000.

The provided tests check that training_data was updated correctly, so that future analyses are not corrupted by a mistake. However, the provided tests do not check that you have implemented remove_outliers correctly so that it works with any data, variable, lower, and upper bound.

[69]:
```
def remove_outliers(data, variable, lower=-np.inf, upper=np.inf):
    """
    Input:
      data (data frame): the table to be filtered
      variable (string): the column with numerical outliers
      lower (numeric): observations with values lower than this will be removed
      upper (numeric): observations with values higher than this will be removed

    Output:
      a data frame with outliers removed

    Note: This function should not change mutate the contents of data.
    """
    filtered_df = data[(data[variable] >= lower) & (data[variable] <= upper)]
    return filtered_df
```

[70]:
```
grader.check("q4")
```

[70]:
```
q4 results: All test cases passed!
```

5 Part 3: Feature Engineering

In this section we will walk you through a few feature engineering techniques.

5.0.1 Bedrooms

Let's start simple by extracting the total number of bedrooms as our first feature for the model. You may notice that the `Bedrooms` column doesn't actually exist in the original dataframe! Instead, it is part of the `Description` column.

5.1 Question 5

5.1.1 Part 1

Let's take a closer look at the `Description` column first. Compare the description across a few rows together at the same time. For the following list of variables, how many of them can be extracted from the `Description` column? Assign your answer as an integer to the variable `q4a`. - The date the property was sold on - The number of stories the property contains - The previous owner of the property - The address of the property - The number of garages the property has - The total number of rooms inside the property - The total number of bedrooms inside the property - The total number of bathrooms inside the property

[71]:
```
training_data['Description']
```

[71]:
```
1         This property, sold on 05/23/2018, is a one-st…
2         This property, sold on 02/18/2016, is a one-st…
3         This property, sold on 07/23/2013, is a one-st…
4         This property, sold on 06/10/2016, is a one-st…
6         This property, sold on 10/26/2017, is a one-st…
                                    …
204787    This property, sold on 07/23/2014, is a one-st…
204788    This property, sold on 03/27/2019, is a one-st…
204789    This property, sold on 01/31/2014, is a one-st…
204790    This property, sold on 02/22/2018, is a one-st…
204791    This property, sold on 04/22/2014, is a one-st…
Name: Description, Length: 168931, dtype: object
```

[72]:
```python
print(training_data['Description'][1])
print(training_data['Description'][2])
print(training_data['Description'][3])
print(training_data['Description'][4])
print(training_data['Description'][6])
print(training_data['Description'][204787])
```

```
This property, sold on 05/23/2018, is a one-story houshold located at 2844 N
LOWELL AVE.It has a total of 6 rooms, 3 of which are bedrooms, and 1.0 of which
are bathrooms.
This property, sold on 02/18/2016, is a one-story houshold located at 11415 S
PRAIRIE AVE.It has a total of 7 rooms, 3 of which are bedrooms, and 1.0 of which
are bathrooms.
This property, sold on 07/23/2013, is a one-story with partially livable attics
houshold located at 2012 DOBSON ST.It has a total of 5 rooms, 3 of which are
bedrooms, and 1.5 of which are bathrooms.
This property, sold on 06/10/2016, is a one-story houshold located at 104 SAUK
TRL.It has a total of 5 rooms, 2 of which are bedrooms, and 1.0 of which are
bathrooms.
This property, sold on 10/26/2017, is a one-story with partially livable attics
houshold located at 2820 186TH ST.It has a total of 6 rooms, 4 of which are
bedrooms, and 1.5 of which are bathrooms.
This property, sold on 07/23/2014, is a one-story houshold located at 10732 S
UNION AVE.It has a total of 4 rooms, 2 of which are bedrooms, and 1.0 of which
are bathrooms.
```

[73]:
```python
q5a = 6
...
```

[74]:
```python
grader.check("q5a")
```

[74]:
```
q5a results: All test cases passed!
```

[75]:
```python
# optional cell for scratch work
```

5.1.2 Part 2

Write a function `add_total_bedrooms(data)` that returns a copy of `data` with an additional column called `Bedrooms` that contains the total number of bedrooms (as integers) for each house. **Treat missing values as zeros if necessary**. Remember that you can make use of vectorized code here; you shouldn't need any `for` statements.

Hint: You should consider inspecting the `Description` column to figure out if there is any general structure within the text. Once you have noticed a certain pattern, you are set with the power of Regex!

[80]:
```
training_data['Description'].shape
```

[80]: (168931,)

[81]:
```
def add_total_bathrooms(data):
    """
    Input:
      data (data frame): a data frame containing at least the Description␣
↪column.
    """
    with_rooms = data.copy()
    pattern = r'(\d+)\.(\d+) of which are bathrooms'
    with_rooms['Bathrooms'] = with_rooms['Description'].str.extract(pattern) #.
↪astype(float)
    #with_rooms['Bathrooms'] = with_rooms['Bathrooms'].fillna(0)
    return with_rooms

training_data = add_total_bathrooms(training_data)
training_data.head()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/indexes/base.p␣
↪in get_loc(self, key, method, tolerance)
   3620             try:
-> 3621                 return self._engine.get_loc(casted_key)
   3622             except KeyError as err:

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/_libs/index.pyx in␣
↪pandas._libs.index.IndexEngine.get_loc()

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/_libs/index.pyx in␣
↪pandas._libs.index.IndexEngine.get_loc()
```

```
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

KeyError: 'Bathrooms'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/frame.py in␣
↪_set_item_mgr(self, key, value)
   3798         try:
-> 3799             loc = self._info_axis.get_loc(key)
   3800         except KeyError:

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/indexes/base.py␣
↪in get_loc(self, key, method, tolerance)
   3622             except KeyError as err:
-> 3623                 raise KeyError(key) from err
   3624             except TypeError:

KeyError: 'Bathrooms'

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
/tmp/ipykernel_683/2965983650.py in <module>
     10     return with_rooms
     11
---> 12 training_data = add_total_bathrooms(training_data)
     13 training_data.head()

/tmp/ipykernel_683/2965983650.py in add_total_bathrooms(data)
      6     with_rooms = data.copy()
      7     pattern = r'(\d+)\.(\d+) of which are bathrooms'
----> 8     with_rooms['Bathrooms'] = with_rooms['Description'].str.
↪extract(pattern) #.astype(float)
      9     #with_rooms['Bathrooms'] = with_rooms['Bathrooms'].fillna(0)
     10     return with_rooms

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/frame.py in␣
↪__setitem__(self, key, value)
   3643             self._setitem_array(key, value)
   3644         elif isinstance(value, DataFrame):
-> 3645             self._set_item_frame_value(key, value)
   3646         elif (
```

```
   3647             is_list_like(value)

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/frame.py in␣
 ↪_set_item_frame_value(self, key, value)
   3786         # now align rows
   3787         arraylike = _reindex_for_setitem(value, self.index)
-> 3788         self._set_item_mgr(key, arraylike)
   3789
   3790     def _iset_item_mgr(

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/frame.py in␣
 ↪_set_item_mgr(self, key, value)
   3800         except KeyError:
   3801             # This item wasn't present, just insert at end
-> 3802             self._mgr.insert(len(self._info_axis), key, value)
   3803         else:
   3804             self._iset_item_mgr(loc, value)

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/internals/
 ↪managers.py in insert(self, loc, item, value)
   1233             value = value.T
   1234             if len(value) > 1:
-> 1235                 raise ValueError(
   1236                     f"Expected a 1D array, got an array with shape␣
 ↪{value.T.shape}"
   1237                 )

ValueError: Expected a 1D array, got an array with shape (168931, 2)
```

[ ]:
```
training_data.head(2)
```

[ ]:
```
training_data.loc[3, 'Description']
```

[38]:
```
def add_total_bedrooms(data):
    """
    Input:
      data (data frame): a data frame containing at least the Description␣
 ↪column.
    """
    with_rooms = data.copy()
    pattern = r'(\d+) of which are bedrooms'
    with_rooms['Bedrooms'] = with_rooms['Description'].str.extract(pattern).
 ↪astype(int)
    with_rooms['Bedrooms'] = with_rooms['Bedrooms'].fillna(0)
    return with_rooms
```

```
training_data = add_total_bedrooms(training_data)
training_data.head()
```

[38]:
```
              PIN  Property Class  Neighborhood Code  Land Square Feet  \
0  17294100610000             203                 50            2500.0
1  13272240180000             202                120            3780.0
2  25221150230000             202                210            4375.0
3  10251130030000             203                220            4375.0
4  31361040550000             202                120            8400.0

   Town Code  Apartments  Wall Material  Roof Material  Basement  \
0         76         0.0            2.0            1.0       1.0
1         71         0.0            2.0            1.0       1.0
2         70         0.0            2.0            1.0       2.0
3         17         0.0            3.0            1.0       1.0
4         32         0.0            3.0            1.0       2.0

   Basement Finish  …  Most Recent Sale  Age Decade  Pure Market Filter  \
0              3.0  …               1.0        13.2                   0
1              1.0  …               1.0         9.6                   1
2              3.0  …               0.0        11.2                   1
3              3.0  …               1.0         6.3                   1
4              3.0  …               0.0         6.3                   1

   Garage Indicator  Neigborhood Code (mapping)  Town and Neighborhood  \
0               0.0                          50                   7650
1               1.0                         120                  71120
2               1.0                         210                  70210
3               1.0                         220                  17220
4               1.0                         120                  32120

                                         Description  Lot Size  Bathrooms  \
0  This property, sold on 09/14/2015, is a one-st…    2500.0          0
1  This property, sold on 05/23/2018, is a one-st…    3780.0          0
2  This property, sold on 02/18/2016, is a one-st…    4375.0          0
3  This property, sold on 07/23/2013, is a one-st…    4375.0          5
4  This property, sold on 06/10/2016, is a one-st…    8400.0          0

   Bedrooms
0         3
1         3
2         3
3         3
4         2

[5 rows x 64 columns]
```

[33]:
```
grader.check("q5b")
```

[33]: q5b results: All test cases passed!

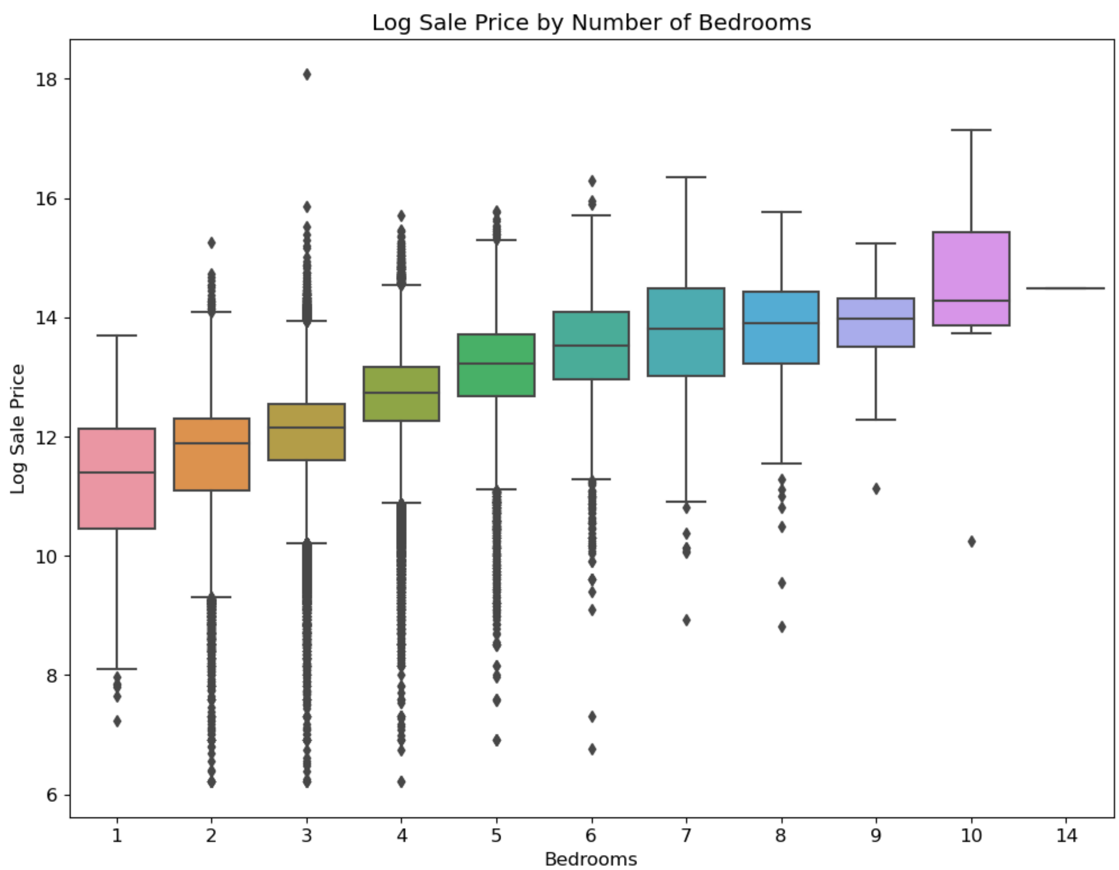
5.1.3 Part 3

Create a visualization that clearly and succintly shows if there exists an association between `Bedrooms` and `Log Sale Price`. A good visualization should satisfy the following requirements: - It should avoid overplotting. - It should have clearly labeled axes and succinct title. - It should convey the strength of the correlation between the sale price and the number of rooms.

Hint: A direct scatter plot of the sale price against the number of rooms for all of the households in our training data might risk overplotting.

[34]:
```
sns.boxplot(data = training_data, x='Bedrooms', y='Log Sale Price')
plt.ylabel("Log Sale Price")
plt.title("Log Sale Price by Number of Bedrooms")
```

[34]: Text(0.5, 1.0, 'Log Sale Price by Number of Bedrooms')



5.2 Question 6

Now, let's take a look at the relationship between neighborhood and sale prices of the houses in our data set. Notice that currently we don't have the actual names for the neighborhoods. Instead we will use a similar column `Neighborhood Code` (which is a numerical encoding of the actual neighborhoods by the Assessment office).

5.2.1 Part 1

Before creating any visualization, let's quickly inspect how many different neighborhoods we are dealing with.

Assign the variable `num_neighborhoods` with the total number of neighborhoods in `training_data`.

[35]:
```
num_neighborhoods = len(training_data['Neighborhood Code'].unique())
num_neighborhoods
```

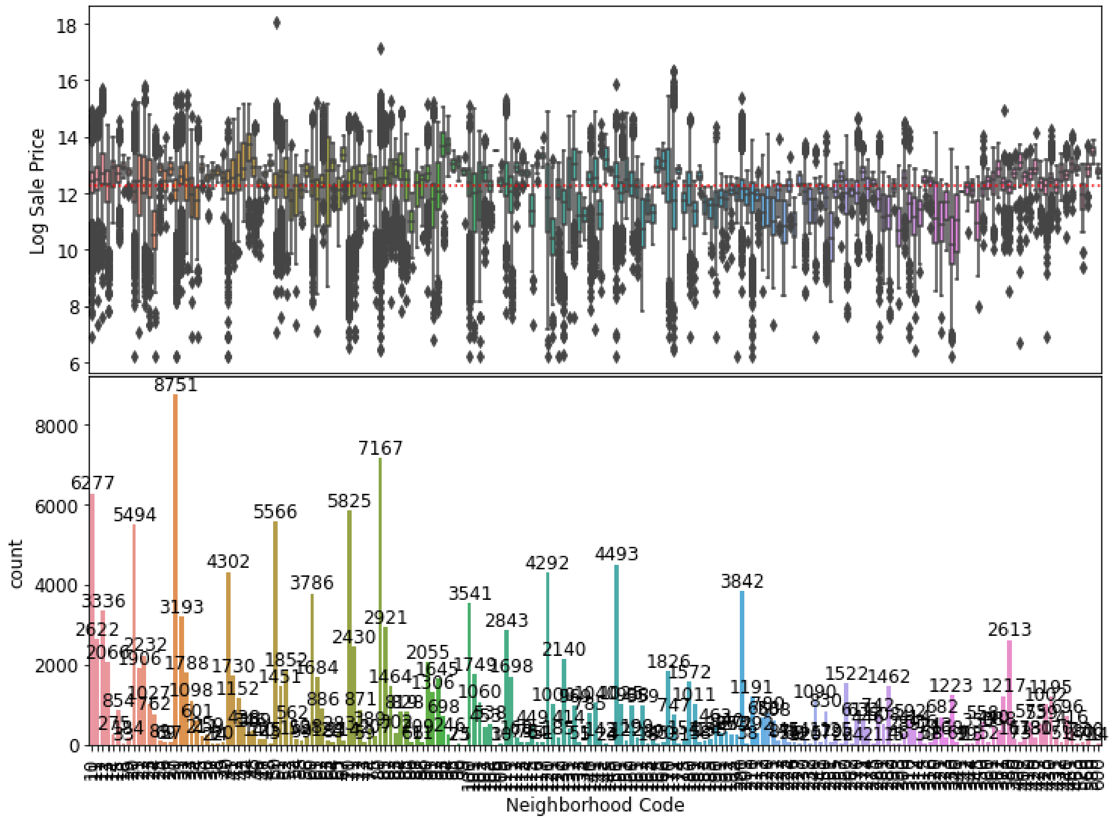
[35]:
```
193
```

[36]:
```
grader.check("q6a")
```

[36]:
```
q6a results: All test cases passed!
```

5.2.2 Part 2

If we try directly plotting the distribution of `Log Sale Price` for all of the households in each neighborhood using the `plot_categorical` function from the next cell, we would get the following visual-



ization.

[37]:
```
def plot_categorical(neighborhoods):
    fig, axs = plt.subplots(nrows=2)

    sns.boxplot(
        x='Neighborhood Code',
        y='Log Sale Price',
        data=neighborhoods,
        ax=axs[0],
    )

    sns.countplot(
        x='Neighborhood Code',
        data=neighborhoods,
        ax=axs[1],
    )

    # Draw median price
    axs[0].axhline(
        y=training_data['Log Sale Price'].median(),
        color='red',
        linestyle='dotted'
    )
```

```
    # Label the bars with counts
    for patch in axs[1].patches:
        x = patch.get_bbox().get_points()[:, 0]
        y = patch.get_bbox().get_points()[1, 1]
        axs[1].annotate(f'{int(y)}', (x.mean(), y), ha='center', va='bottom')

    # Format x-axes
    axs[1].set_xticklabels(axs[1].xaxis.get_majorticklabels(), rotation=90)
    axs[0].xaxis.set_visible(False)

    # Narrow the gap between the plots
    plt.subplots_adjust(hspace=0.01)
```

Oh no, looks like we have run into the problem of overplotting again!

You might have noticed that the graph is overplotted because **there are actually quite a few neighborhoods in our dataset**! For the clarity of our visualization, we will have to zoom in again on a few of them. The reason for this is our visualization will become quite cluttered with a super dense x-axis.

Assign the variable `in_top_20_neighborhoods` to a copy of `training_data` that contains only top 20 neighborhoods with the most number of houses.

[38]:
```
training_data_copy = training_data.copy()
codes = training_data['Neighborhood Code'].value_counts().head(20).index.
 ↪tolist()
in_top_20_neighborhoods = training_data[training_data['Neighborhood Code'].
 ↪isin(codes)]
in_top_20_neighborhoods
```

[38]:
```
                   PIN  Property Class  Neighborhood Code  Land Square Feet  \
1       13272240180000             202                120            3780.0
4       31361040550000             202                120            8400.0
8       13232040260000             205                 70            3100.0
10      19074270080000             202                380            3750.0
11      15083050330000             203                 20            5092.0
…                  …              …                  …                 …
204781  20361190390000             203                 80            4405.0
204785   9284030280000             202                 40            6650.0
204786   8141120110000             203                100           10010.0
204790   9242030500000             203                 80            6650.0
204791  19102030080000             203                 30            2500.0

        Town Code  Apartments  Wall Material  Roof Material  Basement  \
1              71         0.0            2.0            1.0       1.0
4              32         0.0            3.0            1.0       2.0
8              71         0.0            2.0            2.0       1.0
10             72         0.0            1.0            1.0       2.0
```

```
11            31         0.0            2.0            1.0       1.0
…           …         …            …            …       …
204781        70         0.0            2.0            1.0       1.0
204785        22         0.0            1.0            1.0       1.0
204786        16         0.0            2.0            1.0       1.0
204790        22         0.0            2.0            1.0       1.0
204791        72         0.0            1.0            1.0       1.0

        Basement Finish  …  Age Decade  Pure Market Filter  \
1                   1.0  …         9.6                   1
4                   3.0  …         6.3                   1
8                   3.0  …        10.0                   1
10                  3.0  …         7.4                   1
11                  1.0  …         5.8                   1
…                     …  …           …                   …
204781              3.0  …         5.7                   1
204785              3.0  …         6.1                   1
204786              1.0  …         5.6                   1
204790              3.0  …         6.0                   1
204791              3.0  …         4.7                   1

        Garage Indicator  Neigborhood Code (mapping)  Town and Neighborhood  \
1                    1.0                         120                  71120
4                    1.0                         120                  32120
8                    1.0                          70                   7170
10                   1.0                         380                  72380
11                   1.0                          20                   3120
…                      …                           …                      …
204781               1.0                          80                   7080
204785               1.0                          40                   2240
204786               1.0                         100                  16100
204790               1.0                          80                   2280
204791               0.0                          30                   7230

                                              Description  Lot Size  \
1       This property, sold on 05/23/2018, is a one-st…    3780.0
4       This property, sold on 06/10/2016, is a one-st…    8400.0
8       This property, sold on 08/25/2016, is a two-st…    3100.0
10      This property, sold on 05/01/2017, is a one-st…    3750.0
11      This property, sold on 04/29/2014, is a one-st…    5092.0
…                                                     …       …
204781  This property, sold on 07/15/2013, is a one-st…    4405.0
204785  This property, sold on 04/03/2014, is a one-st…    6650.0
204786  This property, sold on 09/08/2016, is a one-st…   10010.0
204790  This property, sold on 02/22/2018, is a one-st…    6650.0
204791  This property, sold on 04/22/2014, is a one-st…    2500.0
```

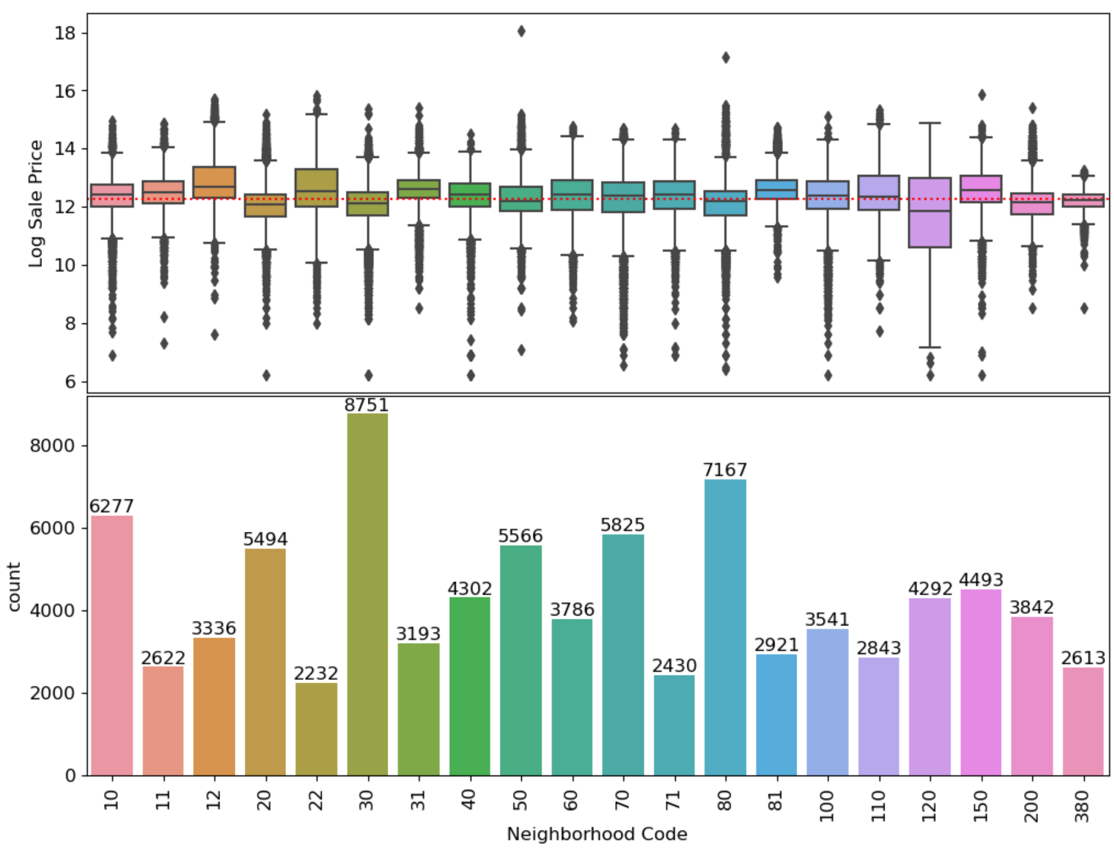
```
        Log Sale Price  Log Building Square Feet  Bedrooms
1            12.560244                  6.904751         3
4            10.025705                  6.855409         2
8            13.422468                  7.636270         4
10           11.695247                  6.841615         2
11           11.184421                  6.911747         3
…                  …                         …       …
204781       10.913269                  7.141245         3
204785       11.736069                  6.761573         3
204786       12.568978                  6.948897         3
204790       12.879017                  7.092574         3
204791       11.736069                  6.946976         2

[85526 rows x 65 columns]
```

[39]:
```
grader.check("q6b")
```

[39]:
```
q6b results: All test cases passed!
```

Let's create another of the distribution of sale price within in each neighborhood again, but this time with a narrower focus!

[40]:
```
plot_categorical(neighborhoods=in_top_20_neighborhoods)
```



5.2.3 Part 3

It looks a lot better now than before, right? Based on the plot above, what can be said about the relationship between the houses' `Log Sale Price` and their neighborhoods?

There's almost no variation between Neighborhood Code and Log Sale Price, as the median of each boxplot is near the same value. To be specific, the log sale price is slightly higher when there is a lower count of properties in the neighborhood.

5.2.4 Part 4

One way we can deal with the lack of data from some neighborhoods is to create a new feature that bins neighborhoods together. Let's categorize our neighborhoods in a crude way: we'll take the top 3 neighborhoods measured by median `Log Sale Price` and identify them as "expensive neighborhoods"; the other neighborhoods are not marked.

Write a function that returns list of the neighborhood codes of the top `n` most pricy neighborhoods as measured by our choice of aggregating function. For example, in the setup above, we would want to call `find_expensive_neighborhoods(training_data, 3, np.median)` to find the top 3 neighborhoods measured by median `Log Sale Price`.

[41]:
```
training_data.head()
```

[41]:
```
              PIN  Property Class  Neighborhood Code  Land Square Feet  \
1  13272240180000             202                120            3780.0
2  25221150230000             202                210            4375.0
3  10251130030000             203                220            4375.0
4  31361040550000             202                120            8400.0
6  30314240080000             203                181           10890.0

   Town Code  Apartments  Wall Material  Roof Material  Basement  \
1         71         0.0            2.0            1.0       1.0
2         70         0.0            2.0            1.0       2.0
3         17         0.0            3.0            1.0       1.0
4         32         0.0            3.0            1.0       2.0
6         37         0.0            1.0            1.0       1.0

   Basement Finish  …  Age Decade  Pure Market Filter  Garage Indicator  \
1              1.0  …         9.6                   1               1.0
2              3.0  …        11.2                   1               1.0
3              3.0  …         6.3                   1               1.0
4              3.0  …         6.3                   1               1.0
6              3.0  …        10.9                   1               1.0
```

```
   Neigborhood Code (mapping)  Town and Neighborhood  \
1                         120                  71120
2                         210                  70210
3                         220                  17220
4                         120                  32120
6                         181                  37181

                                        Description  Lot Size  \
1  This property, sold on 05/23/2018, is a one-st…    3780.0
2  This property, sold on 02/18/2016, is a one-st…    4375.0
3  This property, sold on 07/23/2013, is a one-st…    4375.0
4  This property, sold on 06/10/2016, is a one-st…    8400.0
6  This property, sold on 10/26/2017, is a one-st…   10890.0

   Log Sale Price  Log Building Square Feet  Bedrooms
1       12.560244                  6.904751         3
2        9.998798                  6.810142         3
3       12.323856                  7.068172         3
4       10.025705                  6.855409         2
6       11.512925                  7.458186         4

[5 rows x 65 columns]
```

[42]:
```
training_data.groupby('Neighborhood Code')['Log Sale Price'].agg(np.median).
↪sort_values(ascending=False).iloc[:3].index
```

[42]:
```
Int64Index([44, 94, 93], dtype='int64', name='Neighborhood Code')
```

[43]:
```
def find_expensive_neighborhoods(data, n=3, metric=np.median):
    """
    Input:
      data (data frame): should contain at least a string-valued 'Neighborhood␣
↪Code'
        and a numeric 'Sale Price' column
      n (int): the number of top values desired
      metric (function): function used for aggregating the data in each␣
↪neighborhood.
        for example, np.median for median prices

    Output:
      a list of the the neighborhood codes of the top n highest-priced␣
↪neighborhoods as measured by the metric function
    """
    neighborhoods = data.groupby('Neighborhood Code')['Log Sale Price'].
↪agg(metric).sort_values(ascending=False).head(n).index.values
```

```
    # This makes sure the final list contains the generic int type used in␣
↪Python3, not specific ones used in numpy.
    return [int(code) for code in neighborhoods]

expensive_neighborhoods = find_expensive_neighborhoods(training_data, 3, np.
↪median)
expensive_neighborhoods
```

[43]: [44, 94, 93]

[44]:
```
grader.check("q6d")
```

[44]: q6d results: All test cases passed!

5.2.5 Part 5

We now have a list of neighborhoods we've deemed as higher-priced than others. Let's use that information to write a function `add_expensive_neighborhood` that adds a column `in_expensive_neighborhood` which takes on the value 1 if the house is part of `expensive_neighborhoods` and the value 0 otherwise. This type of variable is known as an **indicator variable**.

Hint: `pd.Series.astype` may be useful for converting True/False values to integers.

[65]:
```
def add_in_expensive_neighborhood(data, neighborhoods):
    """
    Input:
      data (data frame): a data frame containing a 'Neighborhood Code' column␣
↪with values
        found in the codebook
      neighborhoods (list of strings): strings should be the names of␣
↪neighborhoods
        pre-identified as expensive
    Output:
      data frame identical to the input with the addition of a binary
      in_expensive_neighborhood column
    """
    data['in_expensive_neighborhood'] = data['Neighborhood Code'].
↪isin(neighborhoods).astype('int32')
    return data

expensive_neighborhoods = find_expensive_neighborhoods(training_data, 3, np.
↪median)
training_data = add_in_expensive_neighborhood(training_data,␣
↪expensive_neighborhoods)
```

```
training_data
```

[65]:

```
                   PIN  Property Class  Neighborhood Code  Land Square Feet  \
1       13272240180000             202                120            3780.0
2       25221150230000             202                210            4375.0
3       10251130030000             203                220            4375.0
4       31361040550000             202                120            8400.0
6       30314240080000             203                181           10890.0
…                  …              …                  …                …
204787  25163010260000             202                321            4375.0
204788   5063010090000             204                 21           16509.0
204789  16333020150000             202                 90            3810.0
204790   9242030500000             203                 80            6650.0
204791  19102030080000             203                 30            2500.0

        Town Code  Apartments  Wall Material   Roof Material  Basement  \
1              71         0.0            2.0  Shingle/Asphalt       1.0
2              70         0.0            2.0  Shingle/Asphalt       2.0
3              17         0.0            3.0  Shingle/Asphalt       1.0
4              32         0.0            3.0  Shingle/Asphalt       2.0
6              37         0.0            1.0  Shingle/Asphalt       1.0
…             …          …             …              …       …
204787         72         0.0            2.0  Shingle/Asphalt       1.0
204788         23         0.0            1.0  Shingle/Asphalt       1.0
204789         15         0.0            2.0  Shingle/Asphalt       1.0
204790         22         0.0            2.0  Shingle/Asphalt       1.0
204791         72         0.0            1.0  Shingle/Asphalt       1.0

        Basement Finish  …  Pure Market Filter  Garage Indicator  \
1                   1.0  …                   1               1.0
2                   3.0  …                   1               1.0
3                   3.0  …                   1               1.0
4                   3.0  …                   1               1.0
6                   3.0  …                   1               1.0
…                 …  …                 …                …
204787              1.0  …                   1               1.0
204788              1.0  …                   1               1.0
204789              1.0  …                   1               1.0
204790              3.0  …                   1               1.0
204791              3.0  …                   1               0.0

        Neigborhood Code (mapping)  Town and Neighborhood  \
1                              120                  71120
2                              210                  70210
3                              220                  17220
4                              120                  32120
6                              181                  37181
```

```
…                           …                    …
204787                    321                72321
204788                     21                 2321
204789                     90                 1590
204790                     80                 2280
204791                     30                 7230

                                             Description  Lot Size  \
1       This property, sold on 05/23/2018, is a one-st…    3780.0
2       This property, sold on 02/18/2016, is a one-st…    4375.0
3       This property, sold on 07/23/2013, is a one-st…    4375.0
4       This property, sold on 06/10/2016, is a one-st…    8400.0
6       This property, sold on 10/26/2017, is a one-st…   10890.0
…                                                   …       …
204787  This property, sold on 07/23/2014, is a one-st…    4375.0
204788  This property, sold on 03/27/2019, is a one-st…   16509.0
204789  This property, sold on 01/31/2014, is a one-st…    3810.0
204790  This property, sold on 02/22/2018, is a one-st…    6650.0
204791  This property, sold on 04/22/2014, is a one-st…    2500.0

        Log Sale Price  Log Building Square Feet  Bedrooms  \
1            12.560244                  6.904751         3
2             9.998798                  6.810142         3
3            12.323856                  7.068172         3
4            10.025705                  6.855409         2
6            11.512925                  7.458186         4
…                  …                       …        …
204787       10.521372                  6.813445         2
204788       12.323856                  7.603399         4
204789       11.813030                  6.815640         3
204790       12.879017                  7.092574         3
204791       11.736069                  6.946976         2

        in_expensive_neighborhood
1                               0
2                               0
3                               0
4                               0
6                               0
…                           …
204787                          0
204788                          0
204789                          0
204790                          0
204791                          0

[168931 rows x 66 columns]
```

[66]:
```
grader.check("q6e")
```

[66]: q6e results: All test cases passed!

5.3 Question 7

In the following question, we will take a closer look at the `Roof Material` feature of the dataset and examine how we can incorporate categorical features into our linear model.

5.3.1 Part 1

If we look at `codebook.txt` carefully, we can see that the Assessor's Office uses the following mapping for the numerical values in the `Roof Material` column.

```
Central Heating (Nominal):

       1    Shingle/Asphalt
       2    Tar&Gravel
       3    Slate
       4    Shake
       5    Tile
       6    Other
```

Write a function `substitute_roof_material` that replaces each numerical value in `Roof Material` with their corresponding roof material. Your function should return a new DataFrame, not modify the existing DataFrame.

Hint: the DataFrame.replace method may be useful here.

[67]:
```
training_data['Roof Material'].unique()
```

[67]:
```
array(['Shingle/Asphalt', 'Tar&Gravel', 'Other', 'Tile', 'Shake', 'Slate'],
      dtype=object)
```

[69]:
```
def substitute_roof_material(data):
    """
    Input:
      data (data frame): a data frame containing a 'Roof Material' column.  Its␣
↪values
                          should be limited to those found in the codebook
    Output:
      data frame identical to the input except with a refactored 'Roof␣
↪Material' column
    """
    ...
    materials = {1: 'Shingle/Asphalt', 2: 'Tar&Gravel', 3: 'Slate', 4: 'Shake',␣
↪5: 'Tile', 6: 'Other'}

    datacopy = data.copy()
```

```
    datacopy['Roof Material'] = datacopy['Roof Material'].replace(materials)
    return datacopy

training_data = substitute_roof_material(training_data)
training_data.head()
```

[69]:

```
              PIN  Property Class  Neighborhood Code  Land Square Feet  \
1  13272240180000             202                120            3780.0
2  25221150230000             202                210            4375.0
3  10251130030000             203                220            4375.0
4  31361040550000             202                120            8400.0
6  30314240080000             203                181           10890.0

   Town Code  Apartments  Wall Material   Roof Material  Basement  \
1         71         0.0            2.0  Shingle/Asphalt       1.0
2         70         0.0            2.0  Shingle/Asphalt       2.0
3         17         0.0            3.0  Shingle/Asphalt       1.0
4         32         0.0            3.0  Shingle/Asphalt       2.0
6         37         0.0            1.0  Shingle/Asphalt       1.0

   Basement Finish  …  Pure Market Filter  Garage Indicator  \
1              1.0  …                   1               1.0
2              3.0  …                   1               1.0
3              3.0  …                   1               1.0
4              3.0  …                   1               1.0
6              3.0  …                   1               1.0

   Neigborhood Code (mapping)  Town and Neighborhood  \
1                         120                  71120
2                         210                  70210
3                         220                  17220
4                         120                  32120
6                         181                  37181

                                         Description  Lot Size  \
1  This property, sold on 05/23/2018, is a one-st…    3780.0
2  This property, sold on 02/18/2016, is a one-st…    4375.0
3  This property, sold on 07/23/2013, is a one-st…    4375.0
4  This property, sold on 06/10/2016, is a one-st…    8400.0
6  This property, sold on 10/26/2017, is a one-st…   10890.0

   Log Sale Price  Log Building Square Feet  Bedrooms  \
1       12.560244                  6.904751         3
2        9.998798                  6.810142         3
3       12.323856                  7.068172         3
4       10.025705                  6.855409         2
6       11.512925                  7.458186         4
```

```
   in_expensive_neighborhood
1                          0
2                          0
3                          0
4                          0
6                          0

[5 rows x 66 columns]
```

[70]:
```
grader.check("q7a")
```

[70]:
```
q7a results: All test cases passed!
```

5.3.2 Part 2

An Important Note on One Hot Encoding Unfortunately, simply fixing these missing values isn't sufficient for using `Roof Material` in our model. Since `Roof Material` is a categorical variable, we will have to one-hot-encode the data. Notice in the example code below that we have to pre-specify the categories. For more information on categorical data in pandas, refer to this link. For more information on why we want to use one-hot-encoding, refer to this link.

Complete the following function `ohe_roof_material` that returns a dataframe with the new column one-hot-encoded on the roof material of the household. These new columns should have the form `Roof Material_MATERIAL`. Your function should return a new DataFrame, not modify the existing DataFrame.

Note: You should **avoid using `pd.get_dummies`** in your solution as it will remove your original column and is therefore not as reusable as your constructed data preprocessing pipeline. Instead, you can one-hot-encode one column into multiple columns **using Scikit-learn's One Hot Encoder**. It's far more customizable!

Hint: To get you started with this subpart, here is code that initializes a `OneHotEncoding` preprocessing "model" from Scikit-learn and fits it on a simple dataset containing (some of) the first names of your instructional staff this summer! Please play with this code before jumping into the roof material data if you are unsure how to approach the question using `OneHotEncoder`.

```
>>> oh_enc = OneHotEncoder()
>>> oh_enc.fit([['Anirudhan'], ['Dominic'], ['Rahul'], ['Rahul'], ['Anirudhan'], ['Yike'], ['Va
>>> oh_enc.transform([['Anirudhan'], ['Rahul'], ['Dominic']]).toarray()
array([[1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

[71]:
```
enc = OneHotEncoder(handle_unknown='ignore')
X = [['Male', 1], ['Female', 3], ['Female', 2]]
enc.fit(X)
enc.categories_
```

```
enc.get_feature_names_out()
```

[71]: array(['x0_Female', 'x0_Male', 'x1_1', 'x1_2', 'x1_3'], dtype=object)

[72]:
```
training_data['Roof Material'].unique()
```

[72]:
```
array(['Shingle/Asphalt', 'Tar&Gravel', 'Other', 'Tile', 'Shake', 'Slate'],
      dtype=object)
```

[73]:
```
training_data['Roof Material'].head(3)
```

[73]:
```
1    Shingle/Asphalt
2    Shingle/Asphalt
3    Shingle/Asphalt
Name: Roof Material, dtype: object
```

[74]:
```
training_data[['Roof Material']]
```

[74]:

	Roof Material
1	Shingle/Asphalt
2	Shingle/Asphalt
3	Shingle/Asphalt
4	Shingle/Asphalt
6	Shingle/Asphalt
…	…
204787	Shingle/Asphalt
204788	Shingle/Asphalt
204789	Shingle/Asphalt
204790	Shingle/Asphalt
204791	Shingle/Asphalt

[168931 rows x 1 columns]

[77]:
```
from sklearn.preprocessing import OneHotEncoder

def ohe_roof_material(data):
    """
    One-hot-encodes roof material.  New columns are of the form x0_MATERIAL.
    """
    oh_enc = OneHotEncoder()
    oh_enc.fit(data[['Roof Material']])
    dummies = oh_enc.transform(data[['Roof Material']]).todense()

    df = pd.DataFrame(dummies, columns=oh_enc.get_feature_names_out(),␣
 ↪index=data.index)
    return data.join(df)
```

```python
training_data = ohe_roof_material(training_data)
training_data.filter(regex='^Roof Material_').head(10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_99/699652353.py in <module>
     12     return data.join(df)
     13
---> 14 training_data = ohe_roof_material(training_data)
     15 training_data.filter(regex='^Roof Material_').head(10)

/tmp/ipykernel_99/699652353.py in ohe_roof_material(data)
     10
     11     df = pd.DataFrame(dummies, columns=oh_enc.get_feature_names_out(),␣
 ↪index=data.index)
---> 12     return data.join(df)
     13
     14 training_data = ohe_roof_material(training_data)

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/frame.py in␣
 ↪join(self, other, on, how, lsuffix, rsuffix, sort)
   9258         5  K1  A5   B1
   9259         """
-> 9260         return self._join_compat(
   9261             other, on=on, how=how, lsuffix=lsuffix, rsuffix=rsuffix,␣
 ↪sort=sort
   9262         )

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/frame.py in␣
 ↪_join_compat(self, other, on, how, lsuffix, rsuffix, sort)
   9289                 sort=sort,
   9290             )
-> 9291             return merge(
   9292                 self,
   9293                 other,

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/reshape/merge.
 ↪py in merge(left, right, how, on, left_on, right_on, left_index, right_index,␣
 ↪sort, suffixes, copy, indicator, validate)
    120         validate=validate,
    121     )
--> 122     return op.get_result()
    123
    124

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/reshape/merge.
 ↪py in get_result(self)
```

```
    716         join_index, left_indexer, right_indexer = self._get_join_info()
    717
--> 718         llabels, rlabels = _items_overlap_with_suffix(
    719             self.left._info_axis, self.right._info_axis, self.suffixes
    720         )

/srv/conda/envs/notebook/lib/python3.9/site-packages/pandas/core/reshape/merge.
 ↪py in _items_overlap_with_suffix(left, right, suffixes)
   2315
   2316     if not lsuffix and not rsuffix:
-> 2317         raise ValueError(f"columns overlap but no suffix specified:␣
 ↪{to_rename}")
   2318
   2319     def renamer(x, suffix):

ValueError: columns overlap but no suffix specified: Index(['Roof␣
 ↪Material_Other', 'Roof Material_Shake',
       'Roof Material_Shingle/Asphalt', 'Roof Material_Slate',
       'Roof Material_Tar&Gravel', 'Roof Material_Tile'],
      dtype='object')
```

[78]:
```
training_data.filter(regex='^Roof Material_').head(10)
display(training_data)
```

```
                    PIN  Property Class  Neighborhood Code  Land Square Feet  \
1        13272240180000             202                120            3780.0
2        25221150230000             202                210            4375.0
3        10251130030000             203                220            4375.0
4        31361040550000             202                120            8400.0
6        30314240080000             203                181           10890.0
…                   …              …                  …                …
204787   25163010260000             202                321            4375.0
204788    5063010090000             204                 21           16509.0
204789   16333020150000             202                 90            3810.0
204790    9242030500000             203                 80            6650.0
204791   19102030080000             203                 30            2500.0

        Town Code  Apartments  Wall Material   Roof Material  Basement  \
1              71         0.0            2.0  Shingle/Asphalt       1.0
2              70         0.0            2.0  Shingle/Asphalt       2.0
3              17         0.0            3.0  Shingle/Asphalt       1.0
4              32         0.0            3.0  Shingle/Asphalt       2.0
6              37         0.0            1.0  Shingle/Asphalt       1.0
…             …          …             …              …       …
204787         72         0.0            2.0  Shingle/Asphalt       1.0
204788         23         0.0            1.0  Shingle/Asphalt       1.0
204789         15         0.0            2.0  Shingle/Asphalt       1.0
```

```
204790        22         0.0            2.0  Shingle/Asphalt       1.0
204791        72         0.0            1.0  Shingle/Asphalt       1.0

        Basement Finish  …  Log Sale Price  Log Building Square Feet  \
1                   1.0  …       12.560244                  6.904751
2                   3.0  …        9.998798                  6.810142
3                   3.0  …       12.323856                  7.068172
4                   3.0  …       10.025705                  6.855409
6                   3.0  …       11.512925                  7.458186
…                     …  …             …                       …
204787              1.0  …       10.521372                  6.813445
204788              1.0  …       12.323856                  7.603399
204789              1.0  …       11.813030                  6.815640
204790              3.0  …       12.879017                  7.092574
204791              3.0  …       11.736069                  6.946976

        Bedrooms  in_expensive_neighborhood  Roof Material_Other  \
1              3                          0                  0.0
2              3                          0                  0.0
3              3                          0                  0.0
4              2                          0                  0.0
6              4                          0                  0.0
…            …                          …                  …
204787         2                          0                  0.0
204788         4                          0                  0.0
204789         3                          0                  0.0
204790         3                          0                  0.0
204791         2                          0                  0.0

        Roof Material_Shake  Roof Material_Shingle/Asphalt  \
1                       0.0                            1.0
2                       0.0                            1.0
3                       0.0                            1.0
4                       0.0                            1.0
6                       0.0                            1.0
…                       …                            …
204787                  0.0                            1.0
204788                  0.0                            1.0
204789                  0.0                            1.0
204790                  0.0                            1.0
204791                  0.0                            1.0

        Roof Material_Slate  Roof Material_Tar&Gravel  Roof Material_Tile
1                       0.0                       0.0                 0.0
2                       0.0                       0.0                 0.0
3                       0.0                       0.0                 0.0
4                       0.0                       0.0                 0.0
6                       0.0                       0.0                 0.0
```