

▼ proj1a.ipynb

 Download

```
In [1]: # Initialize Otter
import otter
grader = otter.Notebook("proj1a.ipynb")
```

Project 1A: Exploring Cook County Housing

Due Date: Thursday, March 10th, 11:59 PM PDT

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*

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.

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

Question	Part	Points
3	2	1
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

In [2]:

```
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
```

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.

```
In [3]: with zipfile.ZipFile('cook_county_data.zip') as item:
    item.extractall()
```

Let's load the training data.

```
In [4]: 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.

```
In [5]: # 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.

```
In [6]: training_data.columns.values
```

```
Out [6]: 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',
'Neighborhood Code (mapping)', 'Town and Neighborhood',
'Description', 'Lot Size'], dtype=object)
```

In [7]: `training_data['Description'][0]`

Out [7]: 'This property, sold on 09/14/2015, is a one-story household located at 2950 S LY

Part 1: Contextualizing the Data

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

Question 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 sales record of each property in Cook County, Illinois.

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.

I guess this data may be collected to analyze the feature and data of the property transaction, in order to find the popular housing type or in order to predict a rational price based on the current data. It may be collected by the housing agency or construction company.

Part 3

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

The Census Tract column could provide demographic-related information when linked to a data set. This is an identifier which maps to the full census bureau, which will reveal more Census tract information.

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.

1. What is the trend of Sale Price with Land Square Feet increasing?
 - I will create a scatter plot of different land square feet vs the Sale Price. Then use proper model to fit the plot.
2. What is the trend of Sale Price with age increasing?
 - I would create a scatter plot of different age vs the Sale Price. Then use proper model to fit the plot.

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**.

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.

```
In [8]: 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,
    )

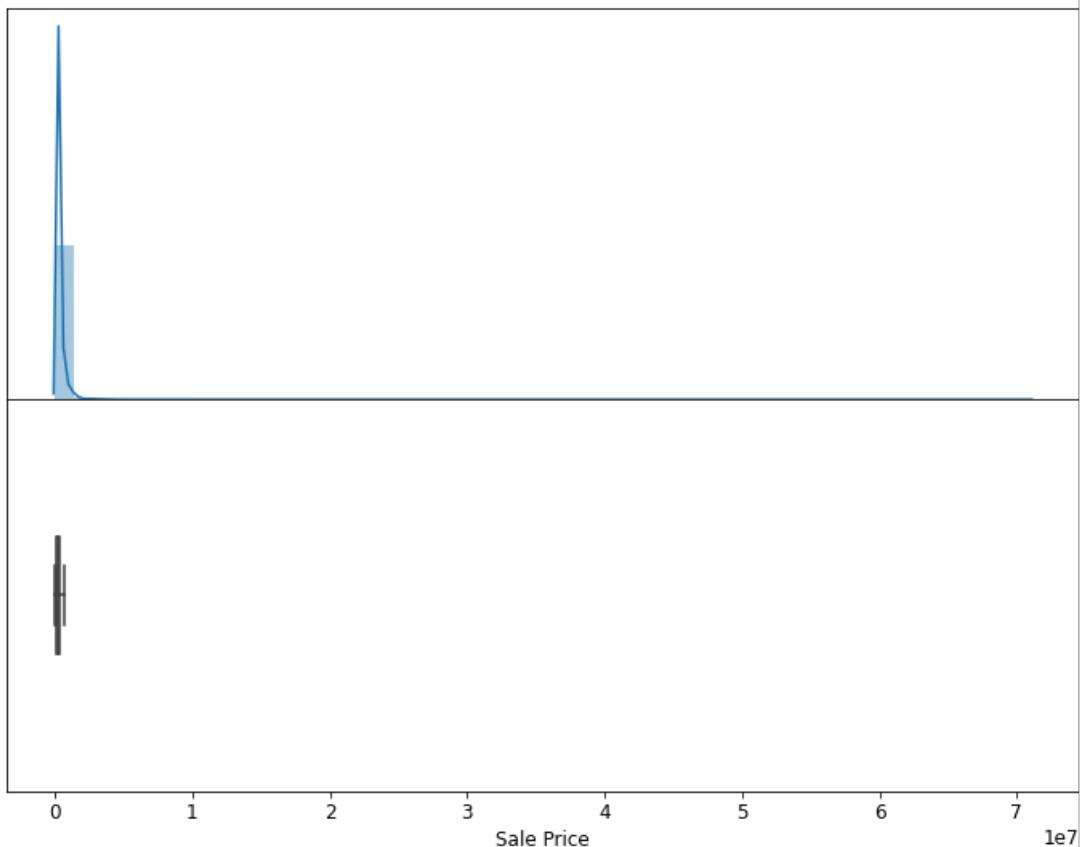
    # Align axes
    spacer = np.max(data[label]) * 0.05
    xmin = np.min(data[label]) - spacer
    xmax = np.max(data[label]) + spacer
    axs[0].set_xlim((xmin, xmax))
    axs[1].set_xlim((xmin, xmax))
```

```
# 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)

# Adjust boxplot fill to be white
axs[1].artists[0].set_facecolor('white')
```

In [9]: `plot_distribution(training_data, label='Sale Price')`



Question 2

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.

The x-axis's unit is way too big which cause the visualization be crowded in a narrow left part which make us can't see the sale price clearly. We can eliminate the outlier data such as property which sale price is 7.1000e+07 to make the x-axis fits the data well.

In [10]: `training_data['Sale Price'].describe()`

```
Out [10]:    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
```

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.

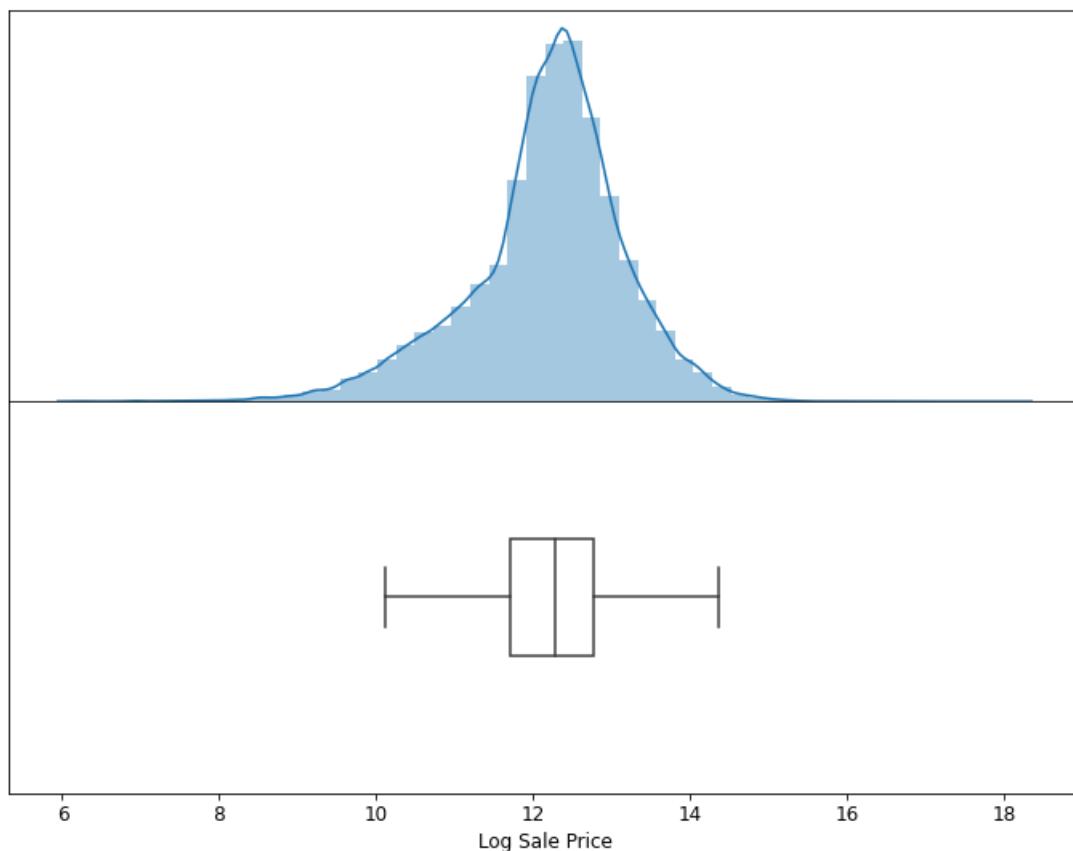
```
In [11]: training_data = training_data[training_data['Sale Price'] >= 500]
          training_data['Log Sale Price'] = np.log(training_data['Sale Price'])
```

```
In [12]: grader.check("q2b")
```

```
Out [12]: q2b results: All test cases passed!
```

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

```
In [13]: plot_distribution(training_data, label='Log Sale Price');
```



Question 3

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`.

```
In [14]: # These should be True or False
q3statement1 = True
q3statement2 = False
q3statement3 = True
```

```
In [15]: grader.check("q3a")
```

```
Out [15]: q3a results: All test cases passed!
```

Part 2

Next, we want to explore if 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.

```
In [16]: training_data['Log Building Square Feet'] = np.log(training_data['Building Square Feet'])
```

```
In [17]: grader.check("q3b")
```

```
Out [17]: q3b results: All test cases passed!
```

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?



Yes, there exist a correlation between log sale price and log building square feet since the (x, y) points are distributed around the simple regression line. Hence Log Building Square Feet would be a good candidate as one of the features.

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.

```
In [18]: 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.
    """
    rt_data = data[data[variable] <= upper]
    rt_data = rt_data[rt_data[variable] >= lower]
    return rt_data
```

```
In [19]: grader.check("q4")
```

Out [19]: q4 results: All test cases passed!

Part 3: Feature Engineering

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

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.

Question 5

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

```
In [20]: q5a = 6
```

```
In [21]: grader.check("q5a")
```

Out [21]: q5a results: All test cases passed!

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 bathrooms (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!

```
In [22]: import re

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

training_data = add_total_bedrooms(training_data)
```

```
In [23]: grader.check("q5b")
```

Out [23]: q5b results: All test cases passed!

Part 3

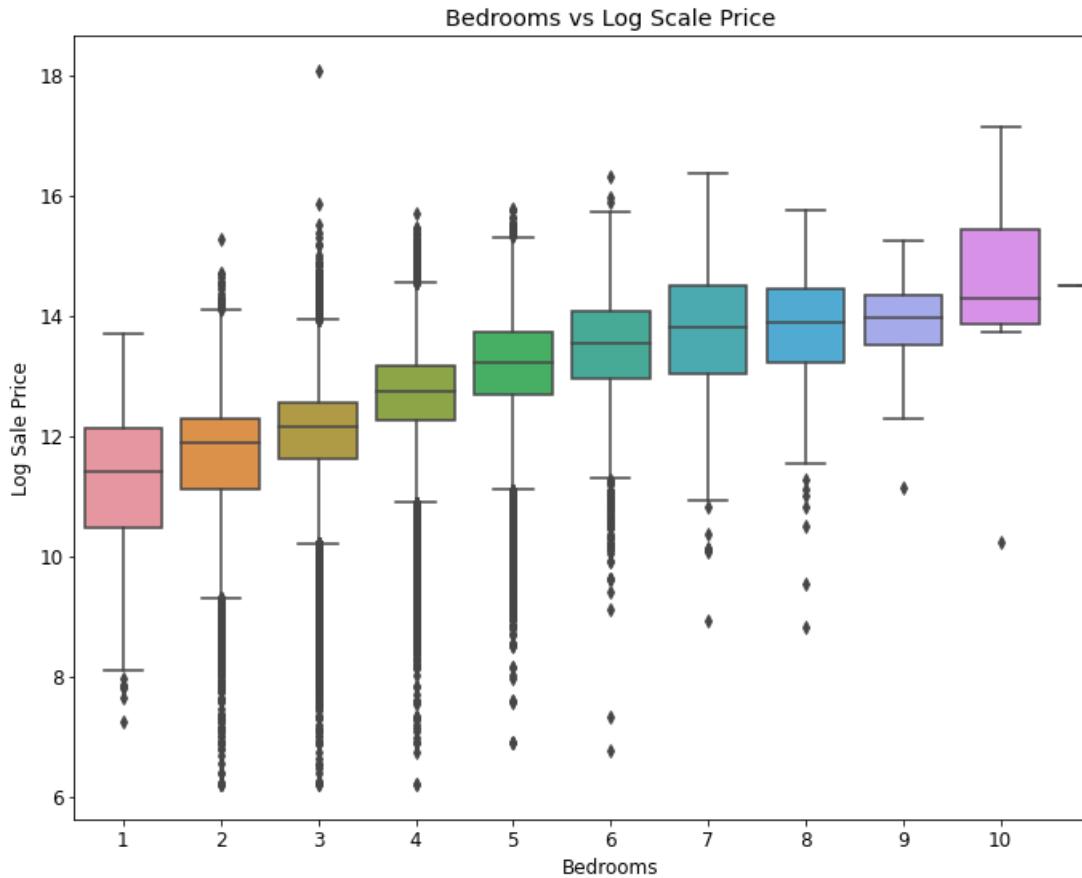
Create a visualization that clearly and succinctly shows if there exists an association between `Bedrooms` and `Log Sale Price`. A good visualization should satisfy the following requirements:

- 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.

```
In [24]: sns.boxplot(
    x='Bedrooms',
    y='Log Sale Price',
    data=training_data,
)
plt.title("Bedrooms vs Log Scale Price")
```

Out [24]: Text(0.5, 1.0, 'Bedrooms vs Log Scale Price')



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).

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`.

```
In [25]: num_neighborhoods = training_data['Neighborhood Code'].value_counts().size
num_neighborhoods
```

Out [25]: 193

```
In [26]: grader.check("q6a")
```

Out [26]: q6a results: All test cases passed!

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 visualization. 

In [27]:

```
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]
        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 neighborhoods with the top 20 number of houses.

In [28]:

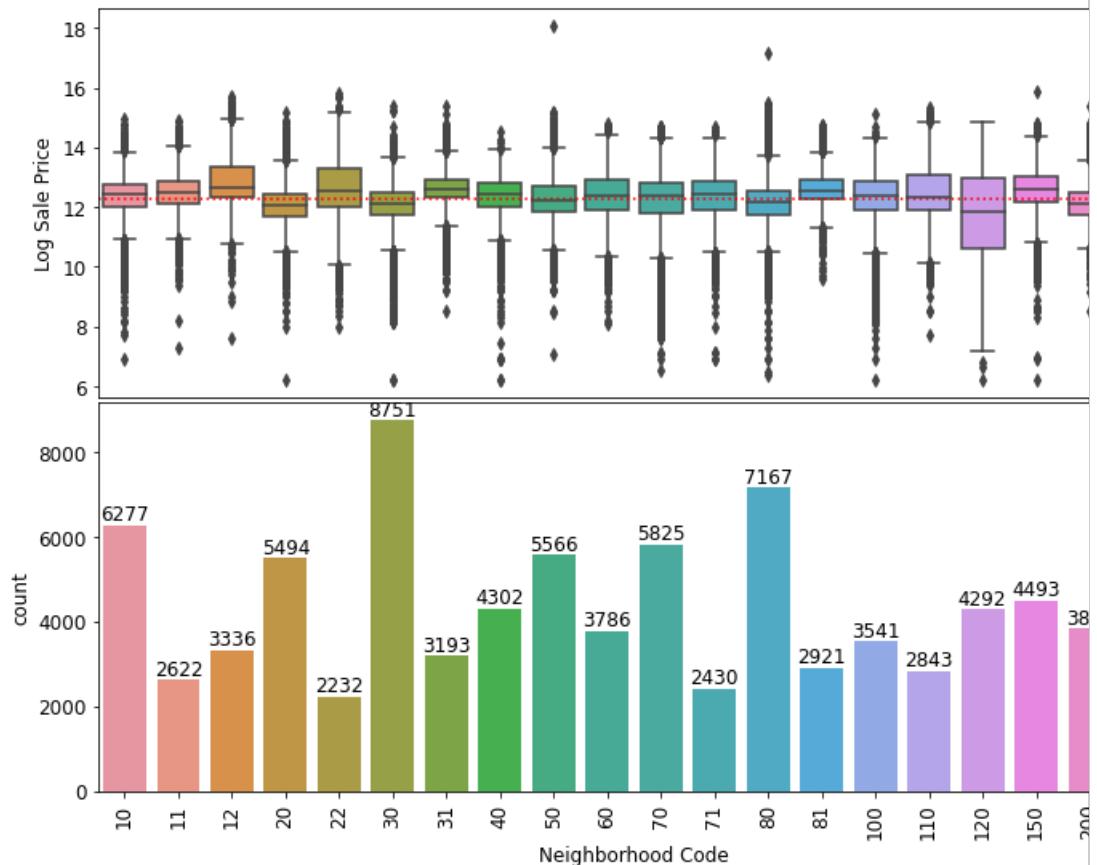
```
top_20 = training_data['Neighborhood Code'].value_counts()[:20].index
in_top_20_neighborhoods = training_data[training_data['Neighborhood
Code'].isin(top_20)]
```

In [29]: `grader.check("q6b")`

Out [29]: 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!

In [30]: `plot_categorical(neighborhoods=in_top_20_neighborhoods)`



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?

Based on the plot above, there is not a strong relationship between Log Sale Price and their neighbors, the mean value among different neighborhoods doesn't have a significant difference. Although some of the neighborhood code have outliers that Log Sale Price tend to be low, some of them don't.

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`.

```
In [31]: 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').agg(metric).sort_values('Log Sale Price', ascending = False)
[:n].index

    # 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
```

Out [31]: [44, 94, 93]

```
In [32]: grader.check("q6d")
```

Out [32]: q6d results: All test cases passed!

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_in_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.

```
In [33]: 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(int)
return data

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

```

In [34]: grader.check("q6e")

Out [34]: q6e results: All test cases passed!

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.

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.

In [35]:

```

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
"""
rt_data = data
rt_data['Roof Material'].replace({1: "Shingle/Asphalt",
                                    2 : "Tar&Gravel",
                                    3 : "Slate",
                                    4 : "Shake",
                                    5 : "Tile",
                                    6 : "Other"})

```

```

    5 : "Tile",
    6 : "Other"}, inplace = True)
return rt_data

training_data = substitute_roof_material(training_data)
training_data.head()

```

Out [35]:

	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	

	Neighborhood	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]

In [36]:

grader.check("q7a")

Out [36]:

q7a results: All test cases passed!

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 `x0_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**.

In [37]:

```
from sklearn.preprocessing import OneHotEncoder

def ohe_roof_material(data):
    """
    One-hot-encodes roof material. New columns are of the form x0_MATERIAL.
    """
    new_data = data.copy()
    encoder = OneHotEncoder()
    new_df = encoder.fit_transform(data[['Roof Material']]).toarray()
    new_data[encoder.get_feature_names()] = new_df
    return new_data

training_data = ohe_roof_material(training_data)
training_data.filter(regex='^x0').head(10)
```

Out [37]:

	x0_Other	x0_Shake	x0_Shingle/Asphalt	x0_Slate	x0_Tar&Gravel	x0_Tile
1	0.0	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0
6	0.0	0.0	1.0	0.0	0.0	0.0
7	0.0	0.0	1.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	1.0	0.0
9	0.0	0.0	1.0	0.0	0.0	0.0
10	0.0	0.0	1.0	0.0	0.0	0.0
11	0.0	0.0	1.0	0.0	0.0	0.0

In [38]:

```
grader.check("q7b")
```

Out [38]:

```
q7b results: All test cases passed!
```

Congratulations! You have finished Project 1A!

In Project 1B, you will focus on building a linear model to predict home prices. You will be well-prepared to build such a model: you have considered what is in this data set, what it can be used for, and engineered some features that should be useful for prediction. Creating a house-pricing model for Cook County has some challenging social implications

▼ proj1b.ipynb

 [Download](#)

```
In [2]: # Initialize Otter
import otter
grader = otter.Notebook("proj1b.ipynb")
```

Project 1B: Predicting Housing Prices in Cook County

Due Date: Thursday, Mar 17th, 11:59 PM

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the project, 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*

Introduction

In part A of this project, you performed some basic exploratory data analysis (EDA), laying out the thought process that leads to certain modeling decisions. Then, you added a few new features to the dataset, cleaning the data as well in the process.

In this project, you will specify and fit a linear model to a few features of the housing data to predict housing prices. Next, we will analyze the error of the model and brainstorm ways to improve the model's performance. Finally, we'll delve deeper into the implications of predictive modeling within the Cook County Assessor's Office (CCAO) case study, especially because statistical modeling is how the CCAO evaluates properties. Given the history of racial discrimination in housing policy and property taxation in Cook County, consider the impacts of your modeling results as you work through this assignment - and think about what fairness might mean to property owners in Cook County.

After this part of the project, you should be comfortable with:

- Implementing a data processing pipeline using `pandas`
- Using `scikit-learn` to build and fit linear models

Score Breakdown

Question	Points
0	5
1	2
2	2
3	3
4	2
5	2

Question	Points
6	1
7	4
8	6
9	2
10	1
11	2
Total	32

```
In [3]: 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
```

Let's load the training and test data.

```
In [4]: with zipfile.ZipFile('cook_county_data.zip') as item:
    item.extractall()
```

```
In [5]: training_data = pd.read_csv("cook_county_train.csv", index_col='Unnamed: 0')
test_data = pd.read_csv("cook_county_test.csv", index_col='Unnamed: 0')
```

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

```
In [6]: # 204792 observations and 62 features in training data
assert training_data.shape == (204792, 62)
# 68264 observations and 61 features in test data
assert test_data.shape == (68264, 61)
# Sale Price is provided in the training data
assert 'Sale Price' in training_data.columns.values
# Sale Price is hidden in the test data
assert 'Sale Price' not in test_data.columns.values
```

Let's remind ourselves of the data available to us in the Cook County dataset. Remember, a more detailed description of each variable is included in `codebook.txt`, which is in the same directory as this notebook). **If you did not attempt Project 1A**, you should take some time to familiarize yourself with the codebook before moving forward.

```
In [7]: training_data.columns.values
```

```
Out [7]: 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',
       'Neighborhood Code (mapping)', 'Town and Neighborhood',
       'Description', 'Lot Size'], dtype=object)
```

Question 0

Question 0a

"How much is a house worth?" Who might be interested in an answer to this question? Please list at least three different parties (people or organizations) and state whether each one has an interest in seeing the value be high or low.

1. Government, they hope to see the value be high since that would get paid for tax.
2. Insurance agent, they hope to see the value be high since they want more house insurance benefit.
3. Housing agent, they hope to see the value be high since high selling price get them more commission.

Question 0b

Which of the following scenarios strike you as unfair and why? You can choose more than one. There is no single right answer but you must explain your reasoning.

- A. A homeowner whose home is assessed at a higher price than it would sell for.
- B. A homeowner whose home is assessed at a lower price than it would sell for.

- C. An assessment process that systematically overvalues inexpensive properties and undervalues expensive properties.
- D. An assessment process that systematically undervalues inexpensive properties and overvalues expensive properties.

C. The definition of Fairness refers to if our model can accurately assess all residential property values, accounting for disparities in geography and other information. In this case, C and D could both cause the caused greater error which makes the prediction inaccurate. But if we overvalue inexpensive properties and undervalues expensive properties, it makes high-income family less tax and low-income family more, in this aspect, the consequence of regressive tax prediction make it more unfair.

Question 0c

Consider a model that is fit to $n = 30$ training observations. Call the response y (Log Sale Price), the predictions \hat{y} , and the residuals $y - \hat{y}$. Which of the following residual plots of y versus $y - \hat{y}$ correspond to a model that might make property assessments that result in to regressive taxation?



In [8]: `q0c = 'A'`

In [9]: `grader.check("q0c")`

Out [9]: `q0c results: All test cases passed!`

The CCAO Dataset

The dataset you'll be working with comes from the Cook County Assessor's Office (CCAO) in Illinois, a government institution that determines property taxes across most of Chicago's metropolitan area and its nearby suburbs. In the United States, all property owners are required to pay property taxes, which are then used to fund public services including education, road maintenance, and sanitation. These property tax assessments are based on property values estimated using statistical models that consider multiple factors, such as real estate value and construction cost.

This system, however, is not without flaws. In late 2017, a lawsuit was filed against the office of Cook County Assessor Joseph Berrios for producing "[racially discriminatory assessments and taxes](#)." The lawsuit included claims that the assessor's office undervalued high-priced homes and overvalued low-priced homes, creating a visible divide along racial lines: Wealthy homeowners, who were typically white, [paid less in property taxes](#), whereas [working-class, non-white homeowners paid more](#).

The Chicago Tribune's four-part series, "[The Tax Divide](#)", delves into how this was uncovered: After "compiling and analyzing more than 100 million property tax records from the years 2003 through 2015, along with thousands of pages of documents, then vetting the findings with top experts in the field," they discovered that "residential assessments had been so far off the mark for so many years." You can read more about their investigation [here](#).

And make sure to watch [Lecture 14](#) before answering the following questions!

Question 0d

What were the central problems with the earlier property tax system in Cook County as reported by the Chicago Tribune? And what were the primary causes of these problems? (Note: in addition to reading the paragraph above you will need to watch the lecture to answer this question)

- Problem: The value of a home didn't be assessed accurately, uniformly, and impartially, it is assessed unfairly toward people across different races and incomes. That make the tax burden was heavy to the low-income black family and light to rich white families.
- Causes: Historically, the racial inequality which was rooted in the US is the major factor to blame. The segregation caused by policy makes the assessors get informed which area is more black proportional lived and the discrimination makes them would prefer to assess higher for those areas and assess lower for other areas. Also, I guess the low-value property tends to be not uniform which makes it more difficult to assess them which contributes to inflated assessments. In addition, the appeal process is more friendly and easy for a rich family, which makes the assessment for them undervalued.

Question 0e

In addition to being regressive, why did the property tax system in Cook County place a disproportionate tax burden on non-white property owners?

The segregation caused by the past history makes the black family have the specific area to gather and neighborhood. After the big depression, the government provide federally-backed mortgages and coded the black family gathered area as high risky area. That makes the assessors get informed which area is more black proportional lived and the discrimination makes them would prefer to assess higher for those areas and assess lower for other areas. Also, I guess the low-value property tends to be not uniform which makes it more difficult to assess them which contributes to inflated assessments. In addition, the appeal process is more friendly and easy for a rich family, which makes the assessment for them undervalued.

Question 1

Now, let's split the data set into a training set and test set. We will use the training set to fit our model's parameters, and we will use the test set to estimate how well our model will perform on unseen data drawn from the same distribution. If we used all the data to fit our model, we would not have a way to estimate model performance on **unseen data**.

"Don't we already have a test set in `cook_county_test.csv`?" you might wonder. The sale prices for `cook_county_test.csv` aren't provided, so we're constructing our own test set for which we know the outputs.

In the cell below, complete the function `train_test_split` that splits `data` into two smaller DataFrames named `train` and `test`. Let `train` contain 80% of the data, and let `test` contain the remaining 20% of the data.

To do this, first create two NumPy arrays named `train_indices` and `test_indices`. `train_indices` should contain a *random* 80% of the indices in `full_data`, and `test_indices` should contain the remaining 20% of the indices. Then, use these arrays to index into `full_data` to create your final `train` and `test` DataFrames.

The provided tests check that you not only answered correctly, but ended up with the exact same train/test split as our reference implementation. Later testing is easier this way.

Note: You should not be importing any additional libraries for this question.

```
In [10]: # This makes the train-test split in this section reproducible across
# different runs
# of the notebook. You do not need this line to run train_test_split in
# general

# DO NOT CHANGE THIS LINE
np.random.seed(1337)
# DO NOT CHANGE THIS LINE

def train_test_split(data):
    data_len = data.shape[0]
    shuffled_indices = np.random.permutation(data_len)

    train_indices = shuffled_indices[: int(data_len*0.8)]
    test_indices = shuffled_indices[int(data_len*0.8):]

    return data.iloc[train_indices], data.iloc[test_indices]

train, test = train_test_split(training_data)
```

```
In [11]: grader.check("q1")
```

```
Out [11]: q1 results: All test cases passed!
```

Now, let's fit our updated linear regression model using the ordinary least squares estimator! We will start you off with something simple by using only 2 features: the **number of bedrooms** in the household and the **log-transformed total area covered by the building** (in square feet).

Consider the following expression for our 1st linear model that contains one of the features:

$$\text{Log Sale Price} = \theta_0 + \theta_1 \cdot (\text{Bedrooms})$$

In parallel, we will also consider a 2nd model that contains both features:

$$\text{Log Sale Price} = \theta_0 + \theta_1 \cdot (\text{Bedrooms}) + \theta_2 \cdot (\text{Log Building Square Feet})$$

Question 2

Without running any calculation or code, complete the following statement by filling in the blank with one of the comparators below:

\geq

\leq

$=$

Suppose we quantify the loss on our linear models using MSE (Mean Squared Error). Consider the training loss of the 1st model and the training loss of the 2nd model. We are

guaranteed that:

Training Loss of the 1st Model _____ Training Loss of the 2nd Model

\geq

- We are adding one more factor which is Log Building Square Feet, it increases the prediction dimension by 1 to predict the log sale price in the second model, hence the mean squared error should be less in the second case.

Question 3

In part A, you wrote a few functions that added features to the dataset. Instead of manually calling each function to add these features to the dataset, it is best practice to encapsulate all of this feature engineering into one "pipeline" function. Defining and using a pipeline reduces all the feature engineering to just one function call and ensures that the same transformations are applied to all data. In this question, we will build a pipeline with the function `process_data_gm`.

Take a look at the following function `process_data_gm`, which takes in a dataframe `data`, a list `pipeline_functions` containing 3-element tuples `(function, arguments, keyword_arguments)` that will be called on `data` in the pipeline, and the label `prediction_col` that represents the column of our target variable (`Sale Price` in this case). It returns two NumPy arrays: `x`, which is our design matrix, and `y` which is the vector containing the observed data. Take a look at our use of `pd.DataFrame.pipe`; you can use this function with each of the tuples passed in through `pipeline_functions`.

```
In [12]: from feature_func import *      # imports functions from Project 1A
# run this cell to define process_data_gm and select_columns

def process_data_gm(data, pipeline_functions, prediction_col):
    """Process the data for a guided model."""
    for function, arguments, keyword_arguments in pipeline_functions:
        if keyword_arguments and (not arguments):
            data = data.pipe(function, **keyword_arguments)
        elif (not keyword_arguments) and (arguments):
            data = data.pipe(function, *arguments)
        else:
            data = data.pipe(function)
    X = data.drop(columns=[prediction_col]).to_numpy()
    y = data.loc[:, prediction_col].to_numpy()
    return X, y
```

```
In [13]: def select_columns(data, *columns):
    """Select only columns passed as arguments."""
    return data.loc[:, columns]

def log_transform(data, col):
    """Add the log transformation of a column to the data frame"""
    data['Log ' + col] = np.log(data[col])
    return data
```

It is time to prepare the training and validation data for the two models we proposed above. Use the following 2 cells to reload a fresh dataset from scratch and run them through the following preprocessing steps for each model:

- Perform a `train_test_split` on the original dataset. Let 80% of the set be training data and 20% of the set be validation data. **Even though we are splitting our dataset into training and validation sets, this question will refer to the validation set as the test set.**
- For both the training and validation set,
 1. Remove outliers in `Sale Price` by so that we are considering households with a price that is strictly greater than 499 dollars (i.e., greater than or equal to 500 dollars).
 2. Apply log transformations to `Sale Price` and the `Building Square Feet` columns to create 2 new columns `Log Sale Price` and `Log Building Square Feet`.
 3. Extract the total number of bedrooms into a new column `Bedrooms` from the `Description` column.
 4. Select the columns `Log Sale Price` and `Bedrooms` (and `Log Building Square Feet` as well if this is the 2nd model).
 5. Return the design matrix X and the observed vector y . **Your design matrix and observed vector should either be numpy arrays or pandas dataframes.**

Assign the final training data and validation data for both models to the following set of variables:

- 1st Model: `X_train_m1`, `y_train_m1`, `X_test_m1`, `y_test_m1`
- 2nd Model: `X_train_m2`, `y_train_m2`, `X_test_m2`, `y_test_m2`

We have automatically imported staff implementations of the functions you wrote in Project 1A. These functions are `remove_outliers`, `add_total_bedrooms`, `find_expensive_neighborhoods`, `add_in_expensive_neighborhood`, and `ohe_roof_material`. You are welcome to copy over your own implementations if you like.

Hint: We have processed the data for the first model for you below to use as an example.

Note: Do not change the line `np.random.seed(1337)` as it ensures we are partitioning the dataset exactly the same way for both models (otherwise their performance isn't directly comparable).

In [14]:

```
# Reload the data
full_data = pd.read_csv("cook_county_train.csv")

# Process the data using the pipeline for the first model
np.random.seed(1337)
train_m1, test_m1 = train_test_split(full_data)

m1_pipelines = [
    (remove_outliers, None, {
        'variable': 'Sale Price',
        'lower': 499,
    }),
    (log_transform, None, {'col': 'Sale Price'}),
    (add_total_bedrooms, None, None),
    (select_columns, ['Log Sale Price', 'Bedrooms'], None)
]

X_train_m1, y_train_m1 = process_data_gm(train_m1, m1_pipelines, 'Log Sale Price')
X_test_m1, y_test_m1 = process_data_gm(test_m1, m1_pipelines, 'Log Sale Price')
```

In [15]:

```
# DO NOT CHANGE THIS LINE
np.random.seed(1337)
```

```
# DO NOT CHANGE THIS LINE

# Process the data using the pipeline for the second model
train_m2, test_m2 = train_test_split(full_data)

m2_pipelines = [
    (remove_outliers, None, {
        'variable': 'Sale Price',
        'lower': 499,
    }),
    (log_transform, None, {'col': 'Sale Price'}),
    (log_transform, None, {'col': 'Building Square Feet'}),
    (add_total_bedrooms, None, None),
    (select_columns, ['Log Sale Price', 'Bedrooms', 'Log Building Square
Feet'], None)
]

X_train_m2, y_train_m2 = process_data_gm(train_m2, m2_pipelines, 'Log Sale
Price')
X_test_m2, y_test_m2 = process_data_gm(test_m2, m2_pipelines, 'Log Sale
Price')
```

In [16]: grader.check("q3")

Out [16]: q3 results: All test cases passed!

Question 4

Finally, let's do some regression!

We first initialize a `sklearn.linear_model.LinearRegression` object for both of our models. We set the `fit_intercept = True` to ensure that the linear model has a non-zero intercept (i.e., a bias term).

In [17]:

```
from sklearn import linear_model as lm

linear_model_m1 = lm.LinearRegression(fit_intercept=True)
linear_model_m2 = lm.LinearRegression(fit_intercept=True)
```

Now it's time to fit our linear regression model. Use the cell below to fit both models, and then use it to compute the fitted values of `Log Sale Price` over the training data, and the predicted values of `Log Sale Price` for the testing data.

Assign the predicted values from both of your models on the training and testing set to the following variables:

- 1st Model: prediction on training set: `y_fitted_m1`, prediction on testing set: `y_predicted_m1`
- 2nd Model: prediction on training set: `y_fitted_m2`, prediction on testing set: `y_predicted_m2`

Note: To make sure you understand how to find the predicted value for both the training and testing data set, there won't be any hidden tests for this part.

In [18]:

```
# Fit the 1st model
# Compute the fitted and predicted values of Log Sale Price for 1st model
```

```
linear_model_m1.fit(X_train_m1, y_train_m1)
y_fitted_m1 = linear_model_m1.predict(X_train_m1)
y_predicted_m1 = linear_model_m1.predict(X_test_m1)

# Fit the 2nd model
# Compute the fitted and predicted values of Log Sale Price for 2nd model
linear_model_m2.fit(X_train_m2, y_train_m2)
y_fitted_m2 = linear_model_m2.predict(X_train_m2)
y_predicted_m2 = linear_model_m2.predict(X_test_m2)
```

In [19]: `grader.check("q4")`

Out [19]: q4 results: All test cases passed!

Question 5

We are moving into analysis of our two models! Let's compare the performance of our two regression models using the Root Mean Squared Error function.

$$RMSE = \sqrt{\frac{\sum_{\text{houses in test set}} (\text{actual price for house} - \text{predicted price for house})^2}{\text{number of houses}}}$$

The function is provided below.

In [20]:

```
def rmse(predicted, actual):
    """
    Calculates RMSE from actual and predicted values
    Input:
        predicted (1D array): vector of predicted/fitted values
        actual (1D array): vector of actual values
    Output:
        a float, the root-mean square error
    """
    return np.sqrt(np.mean((actual - predicted)**2))
```

Now use your `rmse` function to calculate the training error and test error for both models in the cell below.

Assign the error from both of your models to the following variables:

- 1st model: `training_error_m1`, `test_error_m1`
- 2nd model: `training_error_m2`, `test_error_m2`

Since the target variable we are working with is log-transformed, it can also be beneficial to transform it back to its original form so we will have more context on how our model is performing when compared to actual housing prices.

Assign the error on the "de-log-transformed" sale price from both of your models to the following variables:

- 1st model: `training_error_m1_delog`, `test_error_m1_delog`
- 2nd model: `training_error_m2_delog`, `test_error_m2_delog`

In [21]:

```
# Training and test errors for the 1st model
training_error_m1 = rmse(y_fitted_m1, y_train_m1)
test_error_m1 = rmse(y_predicted_m1, y_test_m1)
```

```
# Training and test errors for the 1st model (in its original values before
# the log transform)
training_error_m1_delog = rmse(np.exp(y_fitted_m1), np.exp(y_train_m1))
test_error_m1_delog = rmse(np.exp(y_fitted_m1), np.exp(y_train_m1))

# Training and test errors for the 2nd model
training_error_m2 = rmse(y_fitted_m2, y_train_m2)
test_error_m2 = rmse(y_predicted_m2, y_test_m2)

# Training and test errors for the 2nd model (in its original values before
# the log transform)
training_error_m2_delog = rmse(np.exp(y_fitted_m2), np.exp(y_train_m2))
test_error_m2_delog = rmse(np.exp(y_fitted_m2), np.exp(y_train_m2))

print("1st Model\nTraining RMSE: {}\\nTest RMSE:
{}\\n".format(training_error_m1, test_error_m1))
print("1st Model (no log transform)\nTraining RMSE: {}\\nTest RMSE:
{}\\n".format(training_error_m1_delog, test_error_m1_delog))
print("2nd Model\nTraining RMSE: {}\\nTest RMSE:
{}\\n".format(training_error_m2, test_error_m2))
print("2nd Model (no log transform)\nTraining RMSE: {}\\nTest RMSE:
{}\\n".format(training_error_m2_delog, test_error_m2_delog))
```

1st Model
 Training RMSE: 0.9025651719699077
 Test RMSE: 0.9068644732045896

1st Model (no log transform)
 Training RMSE: 382697.78149699024
 Test RMSE: 382697.78149699024

2nd Model
 Training RMSE: 0.8042009333446841
 Test RMSE: 0.8113963052434995

2nd Model (no log transform)
 Training RMSE: 325716.40819160367
 Test RMSE: 325716.40819160367

In [22]: grader.check("q5")

Out [22]: q5 results: All test cases passed!

Question 6

Let's compare the actual parameters (θ_0 and θ_1) from both of our models. As a quick reminder,

for the 1st model,

$$\text{Log Sale Price} = \theta_0 + \theta_1 \cdot (\text{Bedrooms})$$

for the 2nd model,

$$\text{Log Sale Price} = \theta_0 + \theta_1 \cdot (\text{Bedrooms}) + \theta_2 \cdot (\text{Log Building Square Feet})$$

Run the following cell and compare the values of θ_1 from both models. Why does θ_1 change from positive to negative when we introduce an additional feature in our 2nd model?

In [23]:

```
# Parameters from 1st model
theta0_m1 = linear_model_m1.intercept_
theta1_m1 = linear_model_m1.coef_[0]

# Parameters from 2nd model
theta0_m2 = linear_model_m2.intercept_
theta1_m2, theta2_m2 = linear_model_m2.coef_

print("1st Model\nθ₀: {}\nθ₁: {}".format(theta0_m1, theta1_m1))
print("2nd Model\nθ₀: {}\nθ₁: {}\nθ₂: {}".format(theta0_m2, theta1_m2,
theta2_m2))
```

```
1st Model
θ₀: 10.571725401040084
θ₁: 0.4969197463141442
2nd Model
θ₀: 1.9339633173823696
θ₁: -0.030647249803554506
θ₂: 1.4170991378689644
```

Multicollinearity is the primary possible reason. It is due to the correlation between the two predictors. Basically, if our variables are positively correlated, then the coefficients will be negatively correlated, which can lead to a wrong sign on one of the coefficients. We add one more factor Log Building Square Feet in the prediction model, the building square feet just restate the information of Bedrooms numbers in a sense, which caused the weird flip sign of Bedrooms' coefficient.

Question 7

Question 7a

Another way of understanding the performance (and appropriateness) of a model is through a plot of the model the residuals versus the observations.

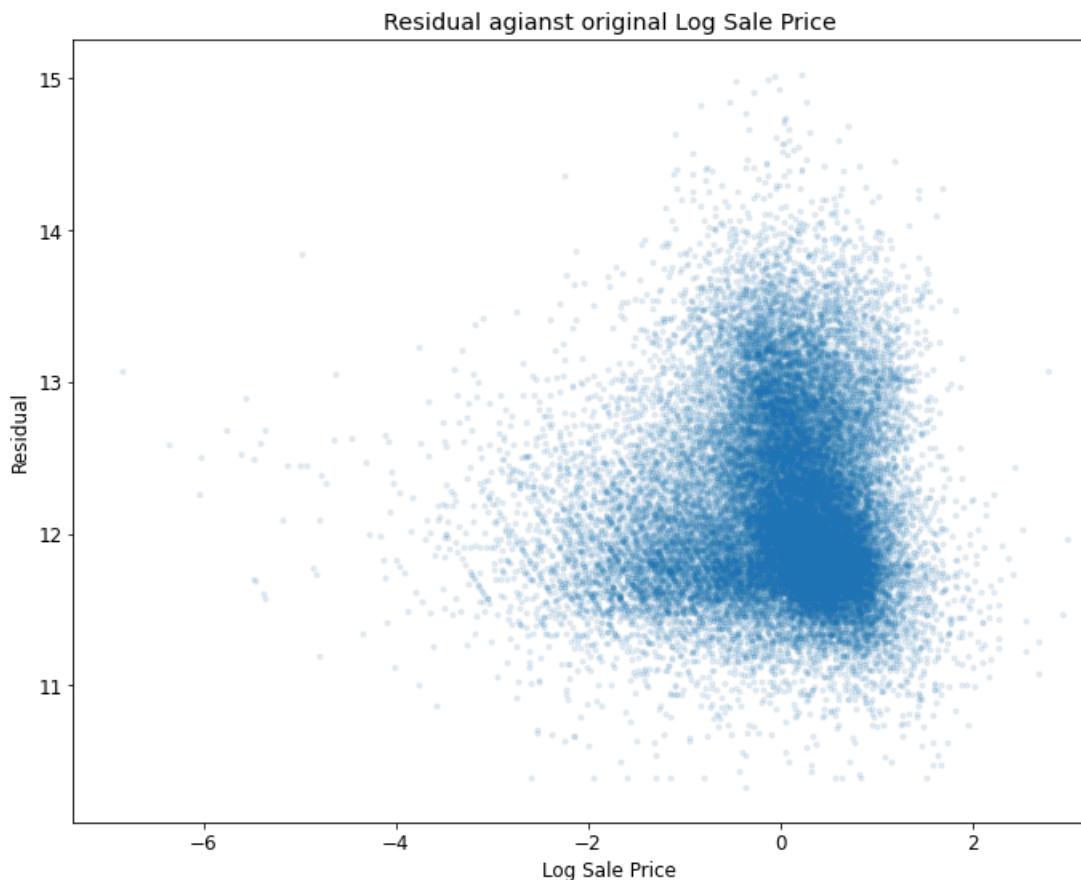
In the cell below, use `plt.scatter` to plot the residuals from predicting `Log Sale Price` using **only the 2nd model** against the original `Log Sale Price` for the **test data**. You should also ensure that the dot size and opacity in the scatter plot are set appropriately to reduce the impact of overplotting.

In [24]:

```
plt.scatter(y_test_m2 - y_predicted_m2, y_predicted_m2, alpha=0.1, s = 8)
plt.xlabel('Log Sale Price')
plt.ylabel('Residual')
plt.title('Residual agianst original Log Sale Price')
```

Out [24]:

```
Text(0.5, 1.0, 'Residual agianst original Log Sale Price')
```



Question 7b

Based on the structure you see in your plot, does this model seem like it will correspond to *regressive*, *fair*, or *progressive* taxation?

```
In [25]: q7b = 'regressive'
```

```
In [26]: grader.check("q7b")
```

```
Out [26]: q7b results: All test cases passed!
```

While our simple model explains some of the variability in price, there is certainly still a lot of room for improvement to be made -- one reason is we have been only utilizing 1 or 2 features (out of a total of 70+) so far! Can you engineer and incorporate more features to improve the model's fairness and accuracy? We won't be asking you to provide your answers here, but this would be important going into the next part (also last part, wohoo!) of this assignment.

Question 8

It is time to build your own model!

Just as in the guided model from the previous question, you should encapsulate as much of your workflow into functions as possible. Your job is to select better features and define your own feature engineering pipeline inside the function `process_data_fm` in the following cell. **You must not change the parameters inside** `process_data_fm`.

To evaluate your model, we will start by defining a linear regression model called `final_model`. Then, we will process training data using your `process_data_fm`, fit `final_model` with this training data, and compute the training RMSE. Then, we will process some test data with your `process_data_fm`, use `final_model` to predict `Log Sale Price` for the test data, transform the predicted and original log values back into their original forms, and compute the test RMSE. See below for an example of the code we will run to grade your model:

Note: `delog` is a function we will run to undo the log transformation on your predictions/original sale prices.

Note: We will not use the test data as provided in `cook_county_test.csv`, but we will assess your model using `cook_county_contest_test.csv`.

```
final_model = lm.LinearRegression(fit_intercept=True)

training_data = pd.read_csv('cook_county_train.csv')
test_data = pd.read_csv('cook_county_contest_test.csv')

X_train, y_train = process_data_fm(training_data)
X_test, y_test = process_data_fm(test_data)

final_model.fit(X_train, y_train)
y_predicted_train = final_model.predict(X_train)
y_predicted_test = final_model.predict(X_test)

training_rmse = rmse(delog(y_predicted_train), delog(y_train))
test_rmse = rmse(delog(y_predicted_test), delog(y_test))
```

Note: It is your duty to make sure that all of your feature engineering and selection happens in `process_data_fm`, and that the function performs as expected without errors. We will **NOT** accept regrade requests that require us to go back and run code that require typo/bug fixes.

Hint: Some features may have missing values in the test set but not in the training set. Make sure `process_data_fm` handles missing values appropriately for each feature!

Note: You **MUST remove any additional new cells you add below the current one before submitting to Gradescope** to avoid any autograder errors.

Grading Scheme

Your grade for Question 8 will be based on your training RMSE and contest **test** RMSE (note that this is another test set, separate from our existing test set!). The thresholds are as follows:

Points	3	2	1	0
Training RMSE	Less than 200k	[200k, 240k)	[240k, 280k)	More than 280k
Test RMSE	Less than 240k	[240k, 280k)	[280k, 300k)	More than 300k

In [27]:

```
# Define any additional helper functions you need here
from sklearn.preprocessing import OneHotEncoder
def substitute_wallm(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
    """
    replacements = {
        'Wall Material': {
            1: 'Wood',
            2: 'Masonry',
            3: 'WM',
            4: 'Stucco'
        }
    }
    data = data.replace(replacements)
    return data

def ohe_wallm(data):
    """
    One-hot-encodes roof material. New columns are of the form x0_MATERIAL.
    """
    oh_enc = OneHotEncoder()
    oh_enc.fit(data[['Wall Material']])
    dummies = pd.DataFrame(oh_enc.transform(data[['Wall
Material']]).todense(),
                           columns=oh_enc.get_feature_names(),
                           index = data.index)
    return data.join(dummies)

def add_total_bathrooms(data):
    """
    Input:
        data (data frame): a data frame containing at least the Description
        column.
    """
    with_rooms = data.copy()
    rooms_regex = r'and (\d+\.\d+) of which are bathrooms'
    rooms = with_rooms['Description'].str.extract(rooms_regex).astype(float)
    with_rooms['Bathrooms'] = rooms
    return with_rooms

m3_pipelines = [
    (remove_outliers, None, {'variable': 'Sale Price', 'lower': 30000,
    'upper': 450000}),
    (log_transform, None, {'col': 'Sale Price'}),
    (log_transform, None, {'col': 'Building Square Feet'}),
    (add_total_bedrooms, None, None),
    (add_total_bathrooms, None, None),
    (substitute_wallm, None, None),
    (ohe_wallm, None, None)
]

m3_test_pipelines = [
    (log_transform, None, {'col': 'Building Square Feet'}),
    (add_total_bedrooms, None, None),
    (add_total_bathrooms, None, None),
    (substitute_wallm, None, None),
```

```

        (ohe_wallm, None, None)
    ]

def apply_pipe(data, pipeline_functions):
    for function, arguments, keyword_arguments in pipeline_functions:
        if keyword_arguments and (not arguments):
            data = data.pipe(function, **keyword_arguments)
        elif (not keyword_arguments) and (arguments):
            data = data.pipe(function, *arguments)
        else:
            data = data.pipe(function)
    return data

# Please include all of your feature engineering process inside this
function.
# Do not modify the parameters of this function.
def process_data_fm(data, is_test_set=False):
    # Return predictors and response variables separately
    if is_test_set:
        data = apply_pipe(data, m3_test_pipelines)
        data['AgeD Square'] = data['Age Decade']**2
        data['LLE Land'] = np.log(data['Estimate (Land)'] + 1)
        return data[['Bedrooms', 'Bathrooms', 'Log Building Square
Feet', 'Garage 1 Size',
                    'Age Decade', 'AgeD Square', 'LLE Land',
                    'x0_Wood', 'x0_Masonry', 'x0_WM']]
    else:
        data = apply_pipe(data, m3_pipelines)
        data['AgeD Square'] = data['Age Decade']**2
        data['LLE Land'] = np.log(data['Estimate (Land)'] + 1)
        data = data[['Log Sale Price', 'Bedrooms', 'Bathrooms', 'Log Building
Square Feet', 'Garage 1 Size',
                    'Age Decade', 'AgeD Square', 'LLE Land',
                    'x0_Wood', 'x0_Masonry', 'x0_WM']]
    X = data.drop(['Log Sale Price'], axis = 1)
    y = data.loc[:, 'Log Sale Price']
    return X, y

training_data = pd.read_csv('cook_county_train.csv')
process_data_fm(training_data)

```

Out [27]:

	Bedrooms	Bathrooms	Log Building Square Feet	Garage 1 Size	\
1	3	1.0	6.904751	2.0	
3	3	1.5	7.068172	3.0	
6	4	1.5	7.458186	3.0	
9	3	1.5	7.343426	1.0	
10	2	1.0	6.841615	3.0	
...
204787	2	1.0	6.813445	1.0	
204788	4	1.5	7.603399	1.0	
204789	3	2.0	6.815640	3.0	
204790	3	1.0	7.092574	3.0	
204791	2	1.0	6.946976	0.0	

	Age Decade	AgeD Square	LLE Land	x0_Wood	x0_Masonry	x0_WM
1	9.6	92.16	10.802449	0.0	1.0	0.0
3	6.3	39.69	10.293196	0.0	0.0	1.0
6	10.9	118.81	10.474213	1.0	0.0	0.0
9	4.8	23.04	10.918736	1.0	0.0	0.0
10	7.4	54.76	10.426765	1.0	0.0	0.0
...
204787	5.8	33.64	9.887409	0.0	1.0	0.0
204788	9.3	86.49	12.602029	1.0	0.0	0.0
204789	5.9	34.81	10.191332	0.0	1.0	0.0
204790	6.0	36.00	10.711770	0.0	1.0	0.0

204791	4.7	22.09	9.964159	1.0	0.0	0.0
1	12.560244					
3	12.323856					
6	11.512925					
9	12.506177					
10	11.695247					
	...					
204787	10.521372					
204788	12.323856					
204789	11.813030					
204790	12.879017					
204791	11.736069					
	Name: Log Sale Price, Length: 131989, dtype: float64)					

In [28]: `grader.check("q8")`

Out [28]: q8 results: All test cases passed!

To determine the error on the test set, please submit your predictions on the contest test set to the Gradescope assignment: **Project 1B Test Set Predictions**. The CSV file to submit is generated below and you should not modify the cell below. Simply download the CSV file and submit it to the appropriate Gradescope assignment.

Note that **you will not receive credit for the test set predictions (i.e. up to 3 points) unless you submit to this assignment!**

In [29]:

```
from datetime import datetime

Y_test_pred =
    run_linear_regression_test(lm.LinearRegression(fit_intercept=True),
    process_data_fm, None, 'cook_county_train.csv',
    'cook_county_contest_test.csv',
                                is_test = True, is_ranking = False,
    return_predictions = True
    )

# Construct and save the submission:
submission_df = pd.DataFrame({
    "Id": pd.read_csv('cook_county_contest_test.csv')['Unnamed: 0'],
    "Value": Y_test_pred,
}, columns=['Id', 'Value'])
timestamp = datetime.isoformat(datetime.now()).split(".") [0]
submission_df.to_csv("submission_{}.csv".format(timestamp), index=False)

print('Created a CSV file:
{}.'.format("submission_{}.csv".format(timestamp)))
print('You may now upload this CSV file to Gradescope for scoring.'))
```

Created a CSV file: submission_2022-03-19T17:08:53.csv.
You may now upload this CSV file to Gradescope for scoring.

Congratulations on finishing your prediction model for home sale prices in Cook County! In the following section, we'll delve deeper into the implications of predictive modeling within the CCAO case study - especially because statistical modeling is how the CCAO evaluates properties.

Refer to [Lecture 14](#) if you're having trouble getting started!

Question 9

When evaluating your model, we used root mean squared error. In the context of estimating the value of houses, what does error mean for an individual homeowner? How does it affect them in terms of property taxes?

- The error means shows the difference between the actual sale price and the predicted sale price. It shows how accurately their property gets assessed in the model and how fair all the homeowners are being treated in the assessment. In terms of property taxes, if we could make the RMSE less enough, we are efficiently preventing the regressive tax effect and could lower the low-income family's tax burden while making the high-income family pay the reasonable tax amounts which they should pay. With the accurate error mean in hand, the government could assess the rationality for mass appraisal.

In the case of the Cook County Assessor's Office, Chief Data Officer Rob Ross states that fair property tax rates are contingent on whether property values are assessed accurately - that they're valued at what they're worth, relative to properties with similar characteristics. This implies that having a more accurate model results in fairer assessments. The goal of the property assessment process for the CCAO, then, is to be as accurate as possible.

When the use of algorithms and statistical modeling has real-world consequences, we often refer to the idea of fairness as a measurement of how socially responsible our work is. But fairness is incredibly multifaceted: Is a fair model one that minimizes loss - one that generates accurate results? Is it one that utilizes "unbiased" data? Or is fairness a broader goal that takes historical contexts into account?

These approaches to fairness are not mutually exclusive. If we look beyond error functions and technical measures of accuracy, we'd not only consider *individual* cases of fairness, but also what fairness - and justice - means to marginalized communities on a broader scale. We'd ask: What does it mean when homes in predominantly Black and Hispanic communities in Cook County are consistently overvalued, resulting in proportionally higher property taxes? When the white neighborhoods in Cook County are consistently undervalued, resulting in proportionally lower property taxes?

Having "accurate" predictions doesn't necessarily address larger historical trends and inequities, and fairness in property assessments in taxes works beyond the CCAO's valuation model. Disassociating accurate predictions from a fair system is vital to approaching justice at multiple levels. Take Evanston, IL - a suburb in Cook County - as an example of housing equity beyond just improving a property valuation model: Their City Council members [recently approved reparations for African American residents](#).

Question 10

In your own words, describe how you would define fairness in property assessments and taxes.

- In the statistic aspect, fairness means being more accurately assessed and close to real house values as close as possible in most cases. But considering the historical fact, the low-income black family has been suffered for heavy tax burden for certain years. In this aspect, I believe property assessment should consider the previous history as an important factor toward fairness.

For example, they could calculate how long the unfair tax burden last and the mean error costs for each family. Then give them some lenient reduction in future years to compensate for their unfair treatment. Once the compensation amount has been settled, then use the better prediction we get to give all people a fair tax burden.

The CCAO and Transparency

Additionally, in their approach to fair property valuations, the CCAO has also pushed for transparency initiatives in the property tax assessment system. After a lawsuit was filed against the CCAO for producing "[racially discriminatory assessments and taxes,](#)" the Office decided that these inequities would be best addressed by making the assessment process more transparent to Cook County constituents.

These transparency initiatives include publishing all of the CCAO's work on [GitLab](#). By allowing the public to access any updates to the system in real-time, the Office argues that they increase accessibility to a process that had previously been blackboxed - obscured and hidden - from the public. Ultimately, the hope is that, by exposing the inner workings of the CCAO's property valuation process, the CCAO's assessment results could be publicly verified as accurate and therefore trusted to be fair.

Question 11

Take a look at the Residential Automated Valuation Model files under the Models subgroup in the CCAO's [GitLab](#). Without directly looking at any code, do you feel that the documentation sufficiently explains how the residential valuation model works? Which part(s) of the documentation might be difficult for nontechnical audiences to understand?

- I feel the documentation sufficiently explains how the residential valuation model works. It explains the steps of modeling and prediction, he first trained the model and iteratively updated the mathematical function from the pattern recognition, then use the model he got to predict all residential properties in Cook County. Then train a secondary simple model to correct for systemic bias for the first model. Then combine the two models to produce initial assessed values. This explanation is pretty straightforward. He also gave the reason for model and hyperparameter selection and demonstrated all the features used and gave much detail of their types and possible values. That makes the results seem transparent and convincing.
- I guess the Hyperparameter Selection part may be hard for nontechnical audiences to understand, the terminology is not that straightforward. It's hard for them to understand how it is used to control the learning process. In addition, the model selection part is not that easy to understand either. It contains many terminologies such as feature transformation, one-hot encoding, etc.

You might feel that the model's inner workings are beyond your pay grade - it's far more complex than the model you built in this assignment, after all! Though we won't delve further into the role of transparency in the broader CCAO case study, consider its effectiveness and/or ineffectiveness: Is the system truly transparent if it's inaccessible to Cook County constituents? Do transparency measures actually bolster the accuracy of a model - or do they only affect the *perceived* accuracy of a model?

And if you're interested in thinking more about transparency measures, take Data 104! But for now...