Introduction

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

In Project A1 (this assignment), 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 Project A2 (the following assignment), 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.

In [5]:
```
import numpy as np

import pandas as pd

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

import warnings
warnings.filterwarnings("ignore")

import zipfile
import os

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

The Data

The dataset consists of over 500,000 records from Cook County, Illinois, the county where Chicago is located. The dataset has 61 features in total; the 62nd is `Sale 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 (you can optionally open this by first clicking the `data` folder, then clicking `codebook.txt` file in the navigation pane). 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. Just run the cells below:

In [6]:
```
with zipfile.ZipFile('data/cook_county_data.zip') as item:
    item.extractall()
```

Let's load the initial data.

In [7]:
```
initial_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 [8]:
```
# 204,792 observations and 62 features in training data
assert initial_data.shape == (204792, 62)
# Sale Price is provided in the training data
assert 'Sale Price' in initial_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 initial data.

In [9]:
```
initial_data.columns.values
```

Out[9]:
```
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)
```

In [10]:
```
initial_data['Description'][0]
```

Out[10]:
```
'This property, sold on 09/14/2015, is a one-story household 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.'
```

In [11]:
```
initial_data.head()
```

Out[11]:

	PIN	Property Class	Neighborhood Code	Land Square Feet	Town Code	Apartments	Wall Material	Roof Material	Basement
0	17294100610000	203	50	2500.0	76	0.0	2.0	1.0	1.0
1	13272240180000	202	120	3780.0	71	0.0	2.0	1.0	1.0
2	25221150230000	202	210	4375.0	70	0.0	2.0	1.0	2.0
3	10251130030000	203	220	4375.0	17	0.0	3.0	1.0	1.0
4	31361040550000	202	120	8400.0	32	0.0	3.0	1.0	2.0

5 rows × 62 columns

Part 1: Contextualizing the Data

Although we've already explored this dataset and its social context in Lecture 15, let's refresh our memory on the background of our dataset before diving into a full-scale analysis.

Question 1a

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

The unit of observation is one house / sale of one house. granularity is the individual house by their pin, because address is not listed.

Question 1b

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.

The data was collected by the city of chicago- I think the purpose (among many possible reasons besides just giving insight to the people and government at large) may have been to study the changing size and price of homes, as cities naturally change from spread out house to more condensed homes/ multi unit properties. It could also be to track what areas are more densely populated and in need of more business/ city services.

Question 1c

Craft at least two questions about housing in Cook County that can be answered with this dataset 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 datasets you would need to answer that question.

I would create a dot plot with a regreesion line comparing age and respective lot size. I want to see if newer properties are bigger using columns lot size and age decade, and i woud overlay property type, town code or apartments as colors to see if there is an indicator of what areas or type of building fits in to age and size.

I would also make a histogram by neighborhood code with mean and median "lot size" or after we devise a way to determine predicted sale price, i would take the average of those across each neighborhood in a historgram or dot plot.

Question 1d

Suppose now, in addition to the information already contained in the dataset, you also have access to several new columns containing demographic data about the owner, including race/ethnicity, gender, age, annual income, and occupation. Provide one new question about housing in Cook County that can be answered using at least one column of demographic data and at least one column of existing data and provide the type of analytical tool you would use to answer it.

we could investigate grouping by ethnicity and pulling what proportion of each property type are owned by each ethnicity uisng columns property class and the ethnicity column to be added. Using a simple analytical tool such as using .groupby and further dividing each subcategory of)ethnicty == ethnicityX) / Dataframe height to gather proportionality of an ethnicity's total ownership by property type.

Part 2: Exploratory Data Analysis

This dataset was collected by the Cook County Assessor's Office in order to build a model to predict the monetary value of a home. You can read more about data collection in the CCAO's Residential Data Integrity Preliminary Report. In Project A2, you will be building a linear regression model that predicts 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 **initial data**.

Sale Price

We begin by examining the distribution of our target variable `Sale Price`. We have provided the following helper method `plot_distribution` that you can use to visualize the distribution of the `Sale Price` using both the histogram and the box plot at the same time. Run the following 2 cells.

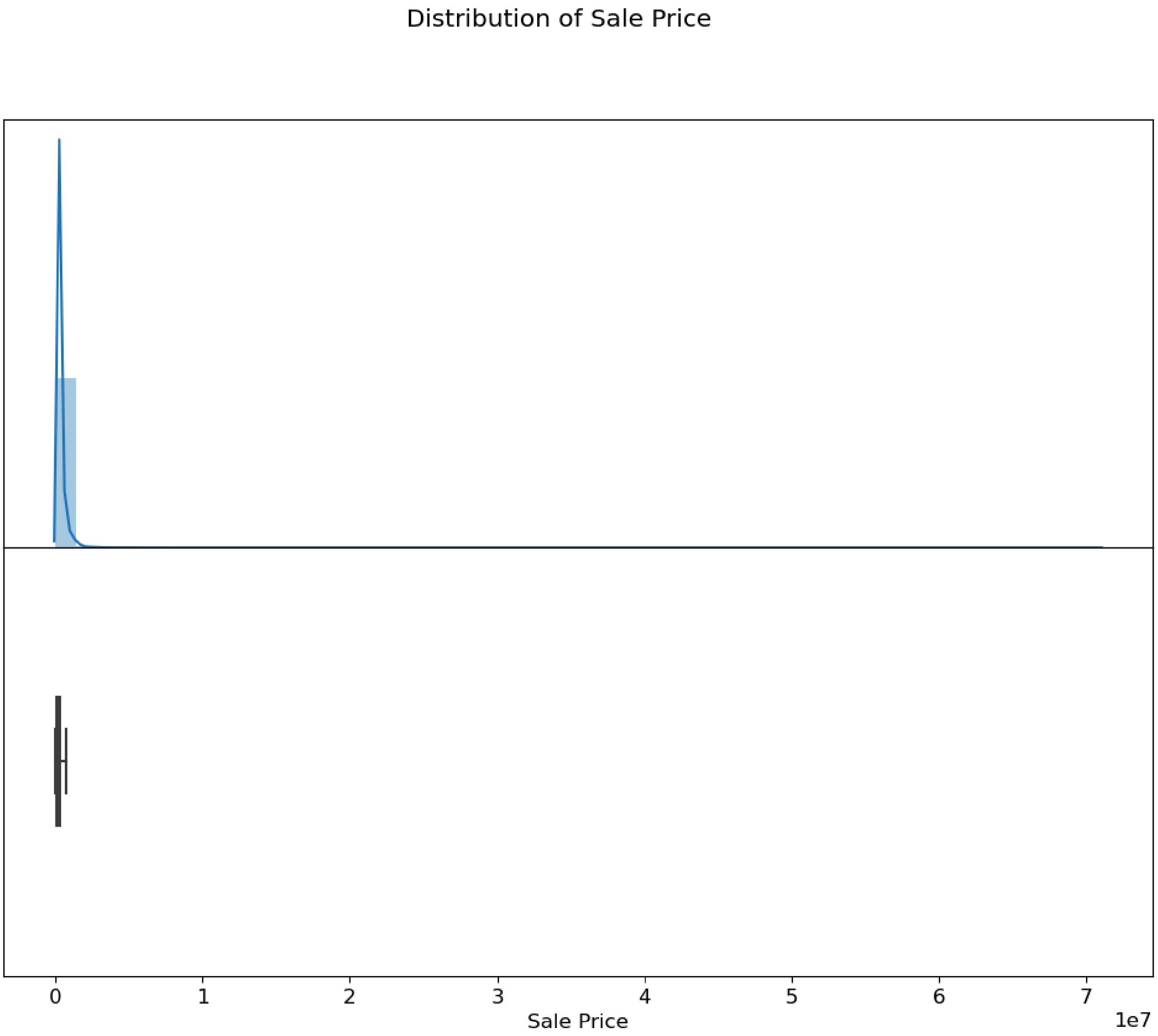
In [12]:
```
def plot_distribution(data, label):
    fig, axs = plt.subplots(nrows=2)

    sns.distplot(
        data[label],
        ax=axs[0]
    )
    sns.boxplot(
        x=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)
    fig.suptitle("Distribution of " + label)
```

In [13]:
```
plot_distribution(initial_data, label='Sale Price')
```



At the same time, we also take a look at some descriptive statistics of this variable. Run the following cell.

In [14]:
```
initial_data['Sale Price'].describe()
```

Out[14]:
```
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
```

Question 2a

Using the plots and the descriptive statistics from `initial_data['Sale Price'].describe()` above, identify one issue with the visualization above and briefly describe one way to overcome it.

It is hard to see the true range because the visualization tools we have chosen squish the graphics down really small and likely omit useful metrics we could use to actually understand whats happening in this data. Zooming in could help, or we could use either a transformation to normalize some of the data (log/ sq. root the prices) or pick a different tool such as a scatterplot or a density curve.

Question 2b

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 a log transformation to `Sale Price`. In the cell below, assign `training_data` to a new `DataFrame` that is the same as `initial_data` **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.

You should NOT remove or modify the original column `Sale Price` as it will be helpful for later questions. If you accidentally remove it, just restart your kernel and run the cells again.

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`.

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

In [15]:
```python
training_data = initial_data[initial_data["Sale Price"] >= 500]
training_data["Log Sale Price"] = np.log(initial_data["Sale Price"])
training_data.shape
```

```
Out[15]:  (168931, 63)
```

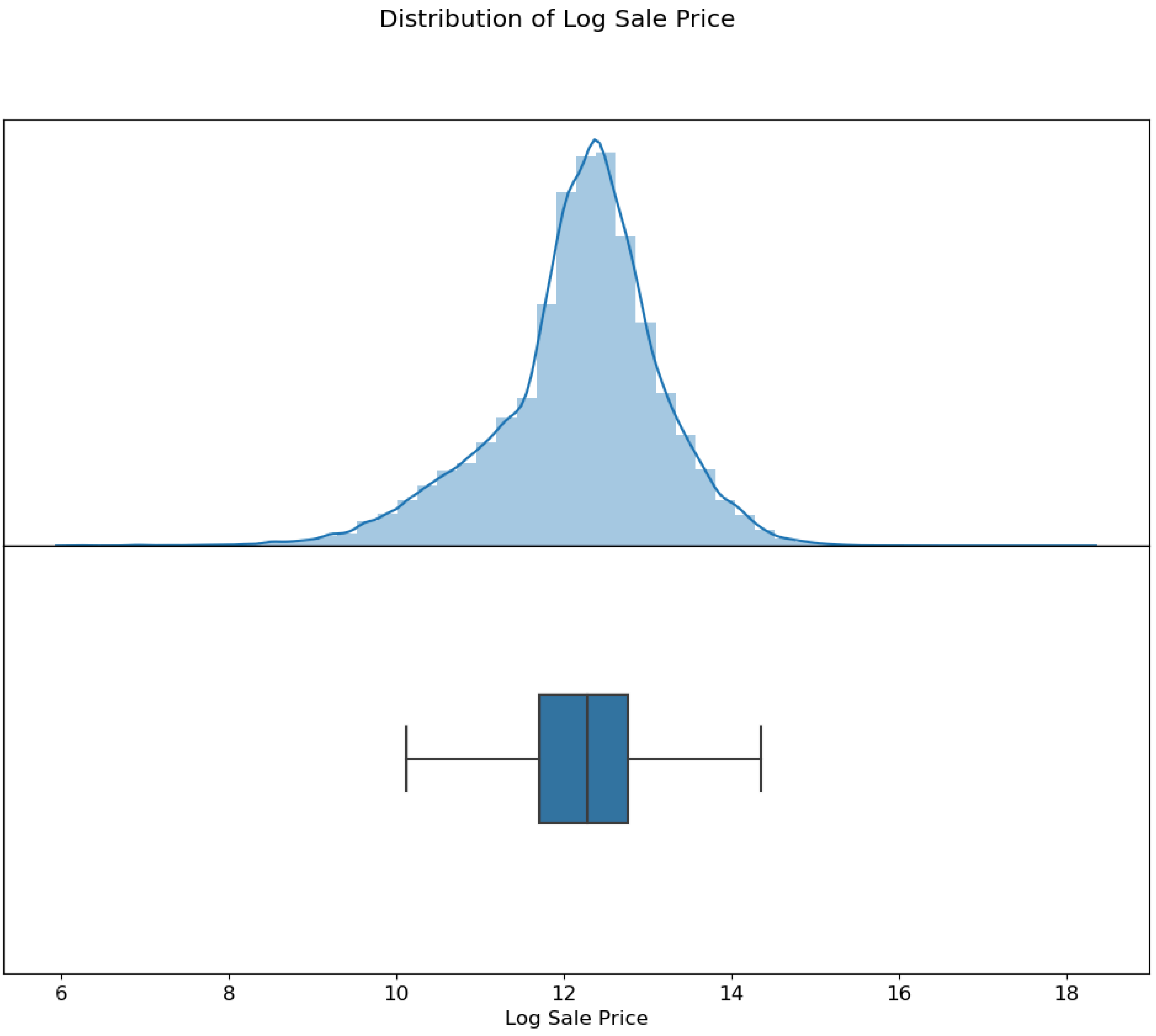
```
In [16]:  grader.check("q2b")
```

```
Out[16]:
```

q2b passed! 🎉

Let's create a new distribution plot using the log-transformed sale prices. As a sanity check, you should see that the distribution for `Log Scale Price` is much more uniform.

```
In [17]:  plot_distribution(training_data, label='Log Sale Price');
```



```
In [18]:  train200k = training_data[training_data["Sale Price"] > 200000]
          train200k.shape[0] / training_data.shape[0] * 100
```

```
Out[18]:  53.28270122120866
```

Question 3a

Is the following statement correct? Assign your answer to `q3statement`.

```
 "At least 25% of the properties in the training set sold for more than
$200,000.00."
```

Note: The provided test for this question does not confirm that you have answered correctly; only that you have assigned `q3statement` to `True` or `False`.

In [19]:
```
# This should be set to True or False
q3statement = True
```

In [20]:
```
grader.check("q3a")
```

Out[20]:

q3a passed! ✨

Question 3b

Next, we want to explore if there is any correlation between `Log Sale Price` and the total area occupied by the property. 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".

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 property.

You should NOT remove or modify the original `Building Square Feet` column as it will be used for later questions. If you accidentally remove it, just restart your kernel and run the cells again.

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

In [21]:
```
training_data['Log Building Square Feet'] = np.log(training_data["Building Square Feet"])
```

In [22]:
```
grader.check("q3b")
```

Out[22]:

q3b passed! 🙌

Question 3c

In the visualization below, we created a `jointplot` 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, would `Log Building Square Feet` make a good candidate as one of the features for our model? Why or why not?

Hint: To help answer this question, ask yourself: what kind of relationship does a "good" feature share with the target variable we aim to predict?

Joint Plot

My answer is sort of. The line of best fit does a pretty good job of fitting the data in that it cuts it pretty much down the middle, however we notice on lower x values (log sq.feet) that there is bulging down (more negative residuals than positive it looks like). This to me says that the relationship is not quite linear due to the heteroscedasticity exhbitied towards lower x values. Despite already log transforming both varaiubles and the relationship not quite being linear, we could either try a new model or simply pick a better predictor/ set of predictors than square feet alone, as there are likely other variables at play that are very important.

Question 4

Continuing from the previous part, as you explore the dataset, 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.

Write a function `remove_outliers` that removes outliers from the dataset based on a threshold value of a variable. For example, `remove_outliers(training_data, 'Building Square Feet', lower=500, upper=8000)` should return a copy of `data` with only observations that satisfy `Building Square Feet` less than or equal to 8000 (inclusive) and `Building Square Feet` greater than 500 (inclusive).

Note: The provided tests simply check that the `remove_outliers` function you defined does not mutate the input data in-place. They do not check that you have implemented `remove_outliers` correctly so that it works with any data, variable, lower, and upper bound.

In [23]:
```python
def remove_outliers(data, variable, lower=-np.inf, upper=np.inf):
    """
    Input:
      data (DataFrame): the table to be filtered
      variable (string): the column with numerical outliers
      lower (numeric): observations with values lower than or equal to this will be removed
      upper (numeric): observations with values higher than or equal to this will be removed

    Output:
      a DataFrame with outliers removed

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

In [24]:
```
grader.check("q4")
```

Out[24]:

q4 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 5a

Let's take a closer look at the `Description` column first. Compare the description for a few rows. For the following list of variables, how many of them can be extracted from the `Description` column? Assign your answer to a list of integers corresponding to the statements that you think are true (ie. `[1, 2, 3]`).

1. The date the property was sold on.
2. The number of stories the property contains.
3. The previous owner of the property.
4. The address of the property.
5. The number of garages the property has.
6. The total number of rooms inside the property.
7. The total number of bedrooms inside the property.
8. The total number of bathrooms inside the property.

In [25]:
```
# optional cell for scratch work
```

In [26]:
```
q5a = [1,2,4,6,7,8]
```

In [27]:
```
grader.check("q5a")
```

Out[27]:

q5a passed! 🍀

```
In [28]: initial_data.Description[3]

Out[28]: 'This property, sold on 07/23/2013, is a one-story with partially livable attics houshold locat
         ed at 2012 DOBSON ST.It has a total of 5 rooms, 3 of which are bedrooms, and 1.5 of which are ba
         throoms.'
```

Question 5b

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!

```
In [29]: def add_total_bedrooms(data):
             """
             Input:
               data (DataFrame): a DataFrame containing at least the Description column.

             Output:
               a Dataframe with a new column "Bedrooms" containing ints.

             """
             with_rooms = data.copy()
             with_rooms["Bedrooms"] = data['Description'].str.extract(r'(\d+)\s+of which are bedrooms').a
             ...
             return with_rooms

         training_data = add_total_bedrooms(training_data)
```

```
In [30]: grader.check("q5b")
```

Out[30]: **q5b** passed! 💯

Question 5c

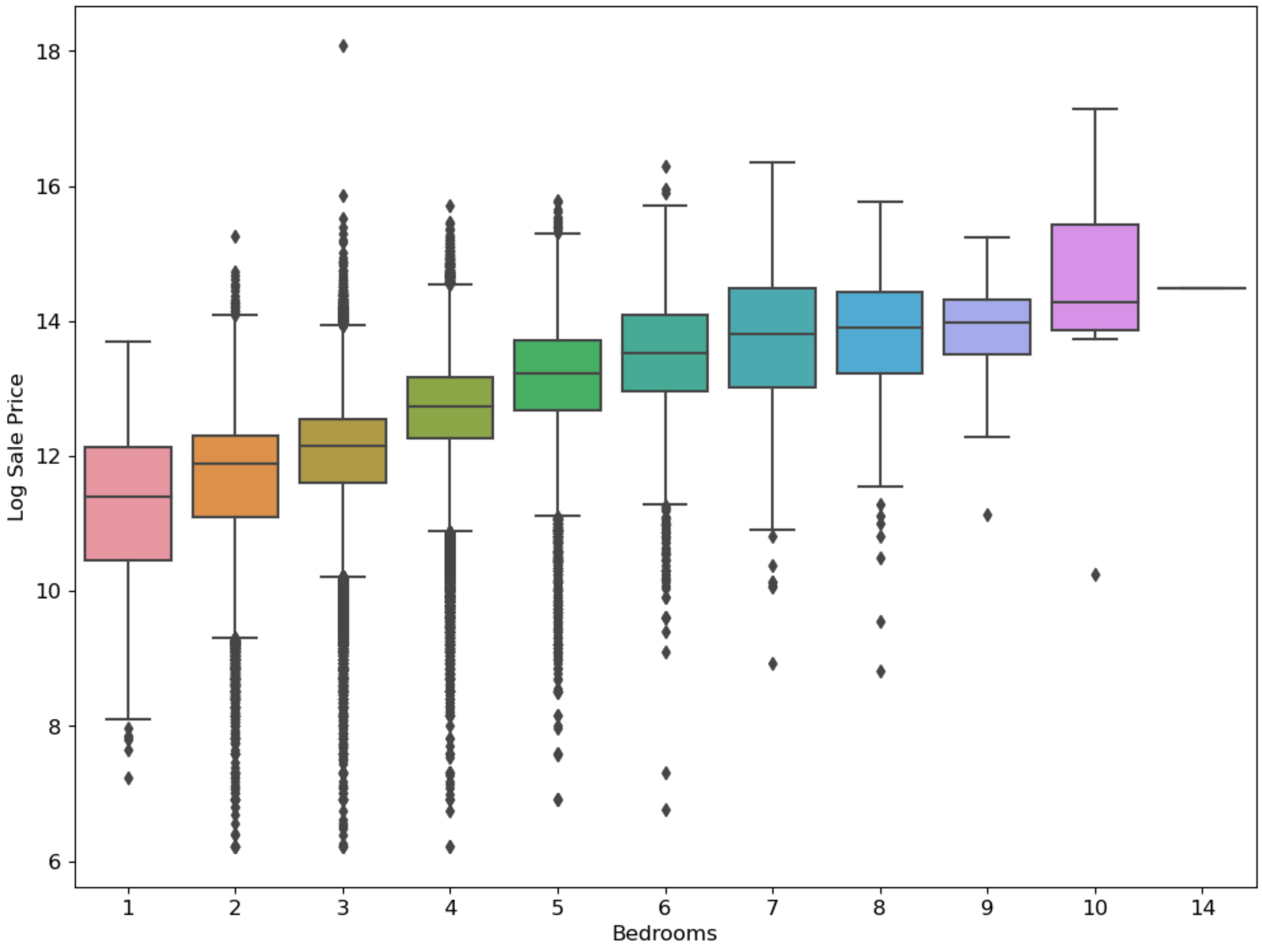
Create a visualization that clearly and succinctly shows if there exists an association between `Bedrooms` and `Log Sale Price`. A good visualization should satisfy the following requirements:

- It should avoid overplotting.
- It should have clearly labeled axes and a succinct title.
- It should convey the strength of the correlation between `Sale Price` and the number of rooms: in other words, you should be able to look at the plot and describe the general relationship between `Log`

`Sale Price` and `Bedrooms`.

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 [37]:
```
sns.boxplot(data = training_data, x = "Bedrooms", y= "Log Sale Price")
plt.xlabel = "Number of Bedrooms"
plt.title = "A box and whisker plot for each bedroom count value recorded in training_data"
```



Now, let's take a look at the relationship between neighborhood and sale prices of the houses in our dataset. 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).

Question 6a

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

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

In [44]:
```
num_neighborhoods = len(training_data["Neighborhood Code"].unique())
num_neighborhoods
```

Out[44]:
```
193
```

In [45]:
```
grader.check("q6a")
```

Out[45]:

q6a passed! 🚀

Question 6b

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 get the following visualization.

overplot

In [46]:
```
# Feel free to create a cell below this and run plot_cateogrical(training_data) if you want to se
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 has been filtered to only contain rows corresponding to properties that are in one of the top 20 most populous neighborhoods. We define the top 20 neighborhoods as being the 20 neighborhood codes that have the greatest number of properties within them.

In [55]:
```
top_20_neighborhood_codes = training_data.groupby("Neighborhood Code").size()
in_top_20 = top_20_neighborhood_codes.sort_values(ascending= False).head(20).index
in_top_20_neighborhoods = training_data[training_data["Neighborhood Code"].isin(in_top_20)]
```

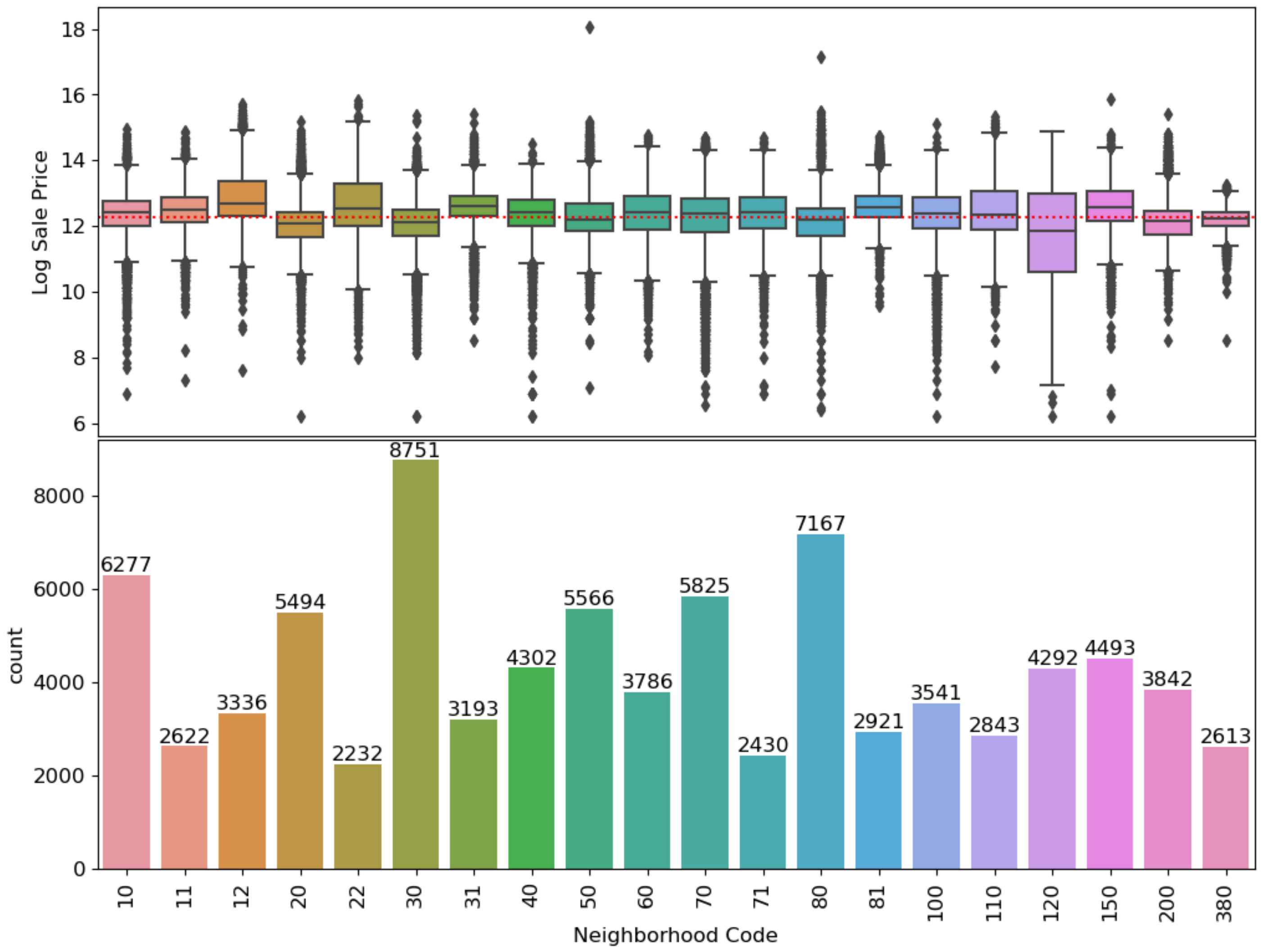
In [53]:
```
grader.check("q6b")
```

Out[53]:

q6b passed! 🌈

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

In [56]:
```
plot_categorical(neighborhoods=in_top_20_neighborhoods)
```



Question 6c

From the plot above, we can see that there is much less data available for some neighborhoods. For example, Neighborhood 71 has only around 27% of the number of datapoints as Neighborhood 30.

One way we can deal with the lack of data from some neighborhoods is to create a new feature that bins neighborhoods together. We'll categorize our neighborhoods in a crude way. In this question, we'll compute how "expensive" each neighborhood is by aggregating the `Log Sale Price`s for all properties in a particular neighborhood using a `metric`, such as the median. We'll use this `metric` to find the top `n` most expensive neighborhoods. Then, in `q6d`, we'll label these "expensive neighborhoods" and leave all other neighborhoods unmarked.

Fill in `find_expensive_neighborhoods` to return a **list** of the neighborhood codes of the **top** `n` most expensive neighborhoods as measured by our choice of aggregating function, `metric`.

For example, calling `find_expensive_neighborhoods(training_data, n=3, metric=np.median)` should return the 3 neighborhood codes with the highest median `Log Sale Price` computed across all properties in those neighborhood codes.

In [67]:
```
def find_expensive_neighborhoods(data, n=3, metric=np.median):
    """
    Input:
      data (DataFrame): should contain at least an int-valued 'Neighborhood Code'
        and a numeric 'Log 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(

    # This makes sure the final list contains the generic int type used in Python3, not specific

    return [int(code) for code in neighborhoods]

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

Out[67]:
```
[44, 94, 93]
```

In [68]:
```
grader.check("q6c")
```

Out[68]:

q6c passed! 🙌

Question 6d

We now have a list of neighborhoods we've deemed as higher-priced than others. Let's use that information to write an additional function `add_expensive_neighborhood` that takes in a `DataFrame` of housing data (`data`) and a list of neighborhood codes considered to be expensive (`expensive_neighborhoods`). You can think of `expensive_neighborhoods` as being the output of the function `find_expensive_neighborhoods` from `q6c` .

Using these inputs, `add_expensive_neighborhood` should add a column to `data` named `in_expensive_neighborhood` that takes on the **integer** value of 1 if a property is part of a neighborhood in `expensive_neighborhoods` and the integer value of 0 if it is not. This type of variable is known as an **indicator variable**.

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

In [78]:
```
def add_in_expensive_neighborhood(data, expensive_neighborhoods):
    """
    Input:
      data (DataFrame): a DataFrame containing a 'Neighborhood Code' column with values
        found in the codebook
      expensive_neighborhoods (list of ints): ints should be the neighborhood codes of
        neighborhoods pre-identified as expensive
    Output:
      DataFrame identical to the input with the addition of a binary
      in_expensive_neighborhood column
    """
    data['in_expensive_neighborhood'] = data["Neighborhood Code"].isin(expensive_neighborhoods)
    return data

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

In [79]:
```
grader.check("q6d")
```

Out[79]:

q6d passed! 🚀

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.

Question 7a

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.

```
Roof Material (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`. If you modify the existing `DataFrame` by accident, you can load `training_data` again in `q2b`.

Hint: the `DataFrame.replace` (documentation) method may be useful here.

In [93]:
```
def substitute_roof_material(data):
    """
    Input:
      data (DataFrame): a DataFrame containing a 'Roof Material' column.  Its values
                         should be limited to those found in the codebook
    Output:
      new DataFrame identical to the input except with a refactored 'Roof Material' column
    """
    dict = {
        1: "Shingle/Asphalt",
        2:    "Tar & Gravel",
        3:   "Slate",
        4:  "Shake",
        5: "Tile",
        6:"Other"
    }
    new_data = data.copy()
    new_data["Roof Material"] = new_data["Roof Material"].replace(dict)
    return new_data

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

Out[93]:

	PIN	Property Class	Neighborhood Code	Land Square Feet	Town Code	Apartments	Wall Material	Roof Material	Base
1	13272240180000	202	120	3780.0	71	0.0	2.0	Shingle/Asphalt	
2	25221150230000	202	210	4375.0	70	0.0	2.0	Shingle/Asphalt	
3	10251130030000	203	220	4375.0	17	0.0	3.0	Shingle/Asphalt	
4	31361040550000	202	120	8400.0	32	0.0	3.0	Shingle/Asphalt	
6	30314240080000	203	181	10890.0	37	0.0	1.0	Shingle/Asphalt	

5 rows × 66 columns

In [94]:
```
grader.check("q7a")
```

Out[94]:

q7a passed! 🌈

Question 7b

An Important Note on One-Hot-Encoding

Unfortunately, simply replacing the integers with the appropriate strings 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. 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` and **should not modify the existing `DataFrame`**.

You should use `scikit-learn`'s `OneHotEncoder` to perform the one-hot-encoding. `OneHotEncoder` will automatically generate column names of the form `Roof Material_MATERIAL`. Refer back to the video walkthrough for Question 1 of Lab 7 for an example of its use. Unlike in the lab example however, in this problem we only wish to construct the one-hot-encoding columns **without removing any columns**.

In [120…

```
from sklearn.preprocessing import OneHotEncoder

def ohe_roof_material(data):
    """
    One-hot-encodes roof material. New columns are of the form "Roof Material_MATERIAL".
    """
    ohe = OneHotEncoder()
    ohe.fit(data[["Roof Material"]])
    oheok = ohe.transform(data[["Roof Material"]]).toarray()
    oheRM = pd.DataFrame(data = oheok, columns = ohe.get_feature_names_out(), index = data.index
    return oheRM.join(data)
training_data_ohe = ohe_roof_material(training_data_mapped)
# This line of code will display only the one-hot-encoded columns in training_data_ohe that
# have names that begin with "Roof Material_"
training_data_ohe.filter(regex='^Roof Material_').head(10)
```

Out[120…

	Roof Material_Other	Roof Material_Shake	Roof Material_Shingle/Asphalt	Roof Material_Slate	Roof Material_Tar & Gravel	Roof Material_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 [121…

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

Out[121…

q7b passed! 🍀

Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

In [123…

```
# Save your notebook first, then run this cell to export your submission.
grader.export(run_tests=True)
```

```
Running your submission against local test cases...


Your submission received the following results when run against available test cases:

    qsurvey results: All test cases passed!

    q2b results: All test cases passed!

    q3a results: All test cases passed!

    q3b results: All test cases passed!

    q4 results: All test cases passed!

    q5a results: All test cases passed!

    q5b results: All test cases passed!

    q6a results: All test cases passed!

    q6b results: All test cases passed!

    q6c results: All test cases passed!

    q6d results: All test cases passed!

    q7a results: All test cases passed!

    q7b results: All test cases passed!
```

Your submission has been exported. Click here to download the zip file.

In [1]:
```python
# Initialize Otter
import otter
grader = otter.Notebook("projA2.ipynb")
```

Project A2: Predicting Housing Prices in Cook County

Due Date: Thursday, March 21st, 11:59 PM

You must submit this assignment to Gradescope by the on-time deadline, Thursday, March 21st, 11:59 PM. Please read the syllabus for the grace period policy. No late submissions beyond the grace period will be accepted. **We strongly encourage you to plan to submit your work to Gradescope several hours before the stated deadline.** This way, you will have ample time to reach out to staff for submission support. While course staff is happy to help you if you encounter difficulties with submission, we may not be able to respond to last-minute requests for assistance (TAs need to sleep, after all!).

Please read the instructions carefully when submitting your work to Gradescope.

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 Project A1, 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 and cleaned the data in the process.

In this project, you will specify and fit a linear model to a few features of the housing data to predict house 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 valuates 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 project, 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	Manual	Points
1a	Yes	1
1b	Yes	1
1c	No	1
1d	Yes	1
1e	Yes	1
2	No	2
3a	No	2
3b	No	3
3c	No	2
4a	Yes	2
4b	No	1
5a	No	0
5b	No	0
5c	No	0
5d	No	3
5e	No	0
5f	No	0
5g	No	0
6a	No	1
6b	No	2
6c	Yes	2
7a	Yes	1
7b	Yes	2
Test Prediction	No	3
Total	8	31

Before You Start

For each question in the assignment, please write down your answer in the answer cell(s) right below the question.

We understand that it is helpful to have extra cells breaking down the process towards reaching your final answer. If you happen to create new cells below your answer to run code, **NEVER** add cells between a question cell and the answer cell below it. It will cause errors when we run the autograder, and it will sometimes cause a failure to generate the PDF file.

Important note: The local autograder tests will not be comprehensive. You can pass the automated tests in your notebook but still fail tests in the autograder. Please be sure to check your results carefully.

Debugging Guide

If you run into any technical issues, we highly recommend checking out the Data 100 Debugging Guide. In this guide, you can find general questions about Jupyter notebooks / Datahub, Gradescope, common `pandas` errors, RegEx, visualizations, and Proj. A1 and A2 common questions.

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
from sklearn import linear_model as lm

import warnings
warnings.filterwarnings("ignore")

import zipfile
import os

from ds100_utils import *
from feature_func import *

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

Let's load the training, validation, and test data.

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

This dataset is split into a training set, a validation set, and a test set. Importantly, the test set does not contain values for our target variable, `Sale Price`. In this project, you will train a model on the training and validation sets and then use this model to predict the `Sale Price`s of the test set. In the cell below, we load the training and validation sets into the `DataFrame` `training_val_data` and the test set into the `DataFrame` `test_data`.

In [4]:
```
training_val_data = pd.read_csv("cook_county_train_val.csv", index_col='Unnamed: 0')
test_data = pd.read_csv("cook_county_contest_test.csv", index_col='Unnamed: 0')
```

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

In [5]:
```
# 204792 observations and 62 features in training data
assert training_val_data.shape == (204792, 62)
```

```
# 55311 observations and 61 features in test data
assert test_data.shape == (55311, 61)
# Sale Price is provided in the training/validation data
assert 'Sale Price' in training_val_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.

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

Question 1: Human Context and Ethics

In this part of the project, we will explore the human context of our housing dataset. **You should watch Lecture 15 before attempting this question.**

Question 1a

"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 housing price be low or high.**

Home buyers such as a family looking to relocate, sellers such as a team that buys a house, fixes it and sells immediately, as well as government agencies looking to collect tax revenue on the value of the home.

Question 1b

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. Would you consider some of these scenarios more (or less) fair than others? Why?

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.

They are all unfair but I would argue D is the most unfair because using a systemically flawed model that biases against the lower class and favors the upper class does the most net damage to society. A and B are presumably common occurances that just occur to chance or a misunderstanding of that specific home's value, and C is also unfair but causes less net negative to society I think, because we can think of it as wealth redistribution and slightly closing a wealth gap rather than widening it like D would.

Question 1c

Consider a model that is fit to $n = 50$ training observations. We denote the response as y (Log Sale Price), the prediction as $\hat{y}$, and the corresponding residual to be $y - \hat{y}$. Which residual plot corresponds to a model that might make property assessments that result in regressive taxation? Recall from Lecture 15 that regressive taxation overvalues inexpensive properties and undervalues expensive properties. Assume that all three plots use the same vertical scale and that the horizontal line marks $y - \hat{y} = 0$. Assign `q1c` to the string letter corresponding to your plot choice.

Hint: When a model overvalues a property (predicts a `Sale Price` greater than the actual `Sale Price`), what are the relative sizes of y and $\hat{y}$? What about when a model undervalues a property?

No description has been provided for this image

```
In [7]: q1c = "A"
```

```
In [8]: grader.check("q1c")
```

Out[8]:

q1c passed! 💯

The CCAO Dataset

You'll work with the dataset from the Cook County Assessor's Office (CCAO) in Illinois. This government institution determines property taxes across most of Chicago's metropolitan areas and nearby suburbs. In the United States, all property owners must 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 considering multiple factors, such as real estate value and construction cost.

However, this system 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.

Make sure to watch Lecture 15 before answering the following questions!

Question 1d

What were the central problems with the earlier property tax system in Cook County as reported by the Chicago Tribune? What were the primary causes of these problems?

Note: Along with reading the paragraph above, you will need to watch Lecture 15 to answer this question.

Expensive homes were undervalued by the model used by Cook County to appraise homes for tax purposes, meaning rich people were paying less in tax than some people thought they should, and the inverse was happening with inexpensive homes. The poor paid more tax than they should have been paying. It was to be noted that as income rose, proportionally less tax was paid by the upper class. Some of the primary causes included factoring race into value as a tradition, not accepting appeals from areas that were the most overvalued, as seen in the heat maps where the part of the city with the longest running overvaluing systems had a 0-20% appeals rate where other areas had appeals rate much higher. Also, the process of redlining is still embedded in the tax and appraisal system despite efforts to make the process more scientific.

Question 1e

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

The proportion of nonwhite people in inexpensive homes was higher than the proportion of white people with inexpensive homes, meaning the county's flawed model was actively harming richer and therefore white people less than everyone else on average. We also saw white people living in areas that had access to appeals much more frequently, meaning they would have an easier time fixing their rates if they were overvalued. Black and minority areas were given appeal access much less as denoted by the 0-20% appeal by neighborhood area on the map in lecture slides.

Question 2: Preparing Data

Let's split the dataset into a training set and a validation set. We will use the training set to fit our model's parameters and the validation set to evaluate 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** such as the test set in `cook_county_contest_test.csv`.

In the cell below, complete the function `train_val_split` that splits `data` into two smaller `DataFrame`s named `train` and `validation`. Let `train` contain 80% of the data, and let `validation` contain the remaining 20%. **You should not import any additional libraries for this question.**

You should only use `NumPy` functions to generate randomness! Your answer should use the variable `shuffled_indices` defined for you. Take a look at the documentation for `np.permutation` to better understand what `shuffled_indices` contains.

Hint: While there are multiple solutions, one way is to create two `NumPy` arrays named `train_indices` and `validation_indices` (or any variable names of your choice) that contain a *random* 80% and 20% of the indices, respectively. Then, use these arrays to index into `data` to create your final `train` and `validation` `DataFrame`s. To ensure that your code matches the solution, use the first 80% as the training set and the last 20% as the validation set. Remember, the values you use to partition `data` must be integers!

The provided tests check that you not only answered correctly but ended up with the same train/validation split as our reference implementation. Testing later on is easier this way.

In [9]:
```python
# This makes the train-validation split in this section reproducible across different runs
# of the notebook. You do not need this line to run train_val_split in general.

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

def train_val_split(data):
    """
    Takes in a DataFrame `data` and randomly splits it into two smaller DataFrames
    named `train` and `validation` with 80% and 20% of the data, respectively.
    """

    data_len = data.shape[0]
    shuffled_indices = np.random.permutation(data_len)
    indexSplit = int(data_len * 0.8)
    train =  data.iloc[shuffled_indices[:indexSplit]]
    validation = data.iloc[shuffled_indices[indexSplit:]]

    return train, validation
train, validation = train_val_split(training_val_data)
```

```
In [10]: grader.check("q2")
```

```
Out[10]:
```

q2 passed! 🍀

Question 3: Fitting a Simple Model

Let's fit our linear regression model using the ordinary least squares estimator! We will start with something simple by using only two 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 first 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 second 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 3a

Without running any calculation or code, assign `q3a` to be the comparator ('>=', '=', '<=') that fills the blank in the following statement:

We quantify the loss on our linear models using MSE (Mean Squared Error). Consider the training loss of the first model and the training loss of the second model. We are guaranteed that:

$$\text{Training Loss of the 1st Model} \_\_\_\_ \ \text{Training Loss of the 2nd Model}$$

```
In [11]: q3a = ">="
```

```
In [12]: grader.check("q3a")
```

```
Out[12]:
```

q3a passed! 🍀

Pipeline Function

In Project A1, you wrote a few functions that added features to the dataset. Instead of calling them manually one by one each time, 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. Below, we combined some functions into a single helper function that outputs `X` and `Y` for the first model above. Try to understand what this function does!

Note 1: We have automatically imported staff implementations of the functions you wrote in Project A1. 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 would like.

Note 2: The staff implementation provided for `remove_outliers` is slightly different from what you did in Project A1. Here `remove_outliers` is exclusive for the bounds whereas in Project A1, it was inclusive for the bounds. `remove_outliers` will only output values strictly greater than the lower bound and strictly smaller than the upper bound. Feel free to still use your original implementation of the function; it shouldn't affect your score if it was done correctly but may slightly change your approach to `q5f`.

In [13]:
```
from feature_func import *    # Import functions from Project A1

###### Copy any function you would like to below ######
...
#######################################################



def feature_engine_simple(data):
    # Remove outliers
    data = remove_outliers(data, 'Sale Price', lower=499)
    # Create Log Sale Price column
    data = log_transform(data, 'Sale Price')
    # Create Bedroom column
    data = add_total_bedrooms(data)
    # Select X and Y from the full data
    X = data[['Bedrooms']]
    Y = data['Log Sale Price']
    return X, Y

# 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, valid_m1 = train_val_split(full_data)
X_train_m1_simple, Y_train_m1_simple = feature_engine_simple(train_m1)
X_valid_m1_simple, Y_valid_m1_simple = feature_engine_simple(valid_m1)

# Take a look at the result
display(X_train_m1_simple.head())
display(Y_train_m1_simple.head())
```

	Bedrooms
130829	4
193890	2
30507	2
91308	2
131132	3

```
130829    12.994530
193890    11.848683
30507     11.813030
91308     13.060488
131132    12.516861
Name: Log Sale Price, dtype: float64
```

`.pipe`

Alternatively, we can build the pipeline using `pd.DataFrame.pipe` (documentation). Take a look at our use of `pd.DataFrame.pipe` below.

The following function `feature_engine_pipe` 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). You can use this function with each of the tuples passed in through `pipeline_functions`.

In [14]:
```python
# Run this cell to define feature_engine_pipe; no further action is needed.
def feature_engine_pipe(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])
    Y = data.loc[:, prediction_col]
    return X, Y
```

Question 3b

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

- Perform a `train_val_split` on the original dataset, loaded as the `DataFrame` `full_data`. Let 80% of the set be training data, and 20% of the set be validation data.
- For both the training and validation set,
  1. Remove outliers in `Sale Price` so that we consider households with a price that is greater than 499 dollars (or equivalently, a price that is 500 dollars or greater).
  2. Apply log transformations to the `Sale Price` and the `Building Square Feet` columns to create two 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` if this is the second model). We have implemented the helper function `select_columns` for you.

5. Return the design matrix $\mathbb{X}$ and the observed vector $\mathbb{Y}$. Note that $\mathbb{Y}$ refers to the transformed `Log Sale Price`, not the original `Sale Price`. **Your design matrix and observed vector should be `NumPy` arrays or `pandas` `DataFrame`s.**

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

- First Model: `X_train_m1`, `Y_train_m1`, `X_valid_m1`, `Y_valid_m1`. This is already implemented for you.
- Second Model: `X_train_m2`, `Y_train_m2`, `X_valid_m2`, `Y_valid_m2`. Please implement this in the second cell below. You may use the first model as an example.

For an example of how to work with pipelines, we have processed model 1 for you using `m1_pipelines` by passing in the corresponding pipeline functions as a list of tuples in the below cell. Your task is to do the same for model 2 in the cell after —— that is, save your pipeline functions as a list of tuples and assign it to `m2_pipelines` for model 2.

As a refresher, the equations model 1 and model 2, respectively, are:

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

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

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

In [15]:
```
# Reload the data
full_data = pd.read_csv("cook_county_train.csv")

# Apply feature engineering to the data using the pipeline for the first model
np.random.seed(1337)
train_m1, valid_m1 = train_val_split(full_data)

# Helper function
def select_columns(data, *columns):
    """Select only columns passed as arguments."""
    return data.loc[:, columns]

# Pipelines, a list of tuples
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 = feature_engine_pipe(train_m1, m1_pipelines, 'Log Sale Price')
X_valid_m1, Y_valid_m1 = feature_engine_pipe(valid_m1, m1_pipelines, 'Log Sale Price')

# Take a look at the result
# It should be the same above as the result returned by feature_engine_simple
display(X_train_m1.head())
display(Y_train_m1.head())
```

	Bedrooms
130829	4
193890	2
30507	2
91308	2
131132	3

```
130829    12.994530
193890    11.848683
30507     11.813030
91308     13.060488
131132    12.516861
Name: Log Sale Price, dtype: float64
```

In [16]:
```
# 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, valid_m2 = train_val_split(full_data)

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

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 = feature_engine_pipe(train_m2, m2_pipelines, 'Log Sale Price')
X_valid_m2, Y_valid_m2 = feature_engine_pipe(valid_m2, m2_pipelines, 'Log Sale Price')


# Take a look at the result
display(X_train_m2.head())
display(Y_train_m2.head())
```

	Bedrooms	Log Building Square Feet
130829	4	7.870166
193890	2	7.002156
30507	2	6.851185
91308	2	7.228388
131132	3	7.990915

```
130829    12.994530
193890    11.848683
30507     11.813030
91308     13.060488
131132    12.516861
Name: Log Sale Price, dtype: float64
```

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

Out[17]:

q3b passed! 🌟

Question 3c

Finally, let's do some regression!

We first initialize a `sklearn.linear_model.LinearRegression` object (documentation) 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 [18]:
```python
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 validation data.

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

- First Model: predicted values on **training set**: `Y_fitted_m1`, predicted values on **validation set**: `Y_predicted_m1`
- Second Model: predicted values on **training set**: `Y_fitted_m2`, predicted values on **validation set**: `Y_predicted_m2`

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

In [19]:
```python
# Fit the 1st model
linear_model_m1.fit(X_train_m1,Y_train_m1)
# Compute the fitted and predicted values of Log Sale Price for 1st model
Y_fitted_m1 = linear_model_m1.predict(X_train_m1)
Y_predicted_m1 = linear_model_m1.predict(X_valid_m1)

# Fit the 2nd model
linear_model_m2.fit(X_train_m2,Y_train_m2)
# Compute the fitted and predicted values of Log Sale Price for 2nd model
Y_fitted_m2 = linear_model_m2.predict(X_train_m2)
Y_predicted_m2 = linear_model_m2.predict(X_valid_m2)
```

In [20]:
```python
grader.check("q3c")
```

Out[20]: **q3c** passed! 🎉

Question 4: Evaluate Our Simple Model

Let's now move into the analysis of our two models!

In [21]:
```
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 RMSE value.
    """
    return np.sqrt(np.mean((actual - predicted)**2))
```

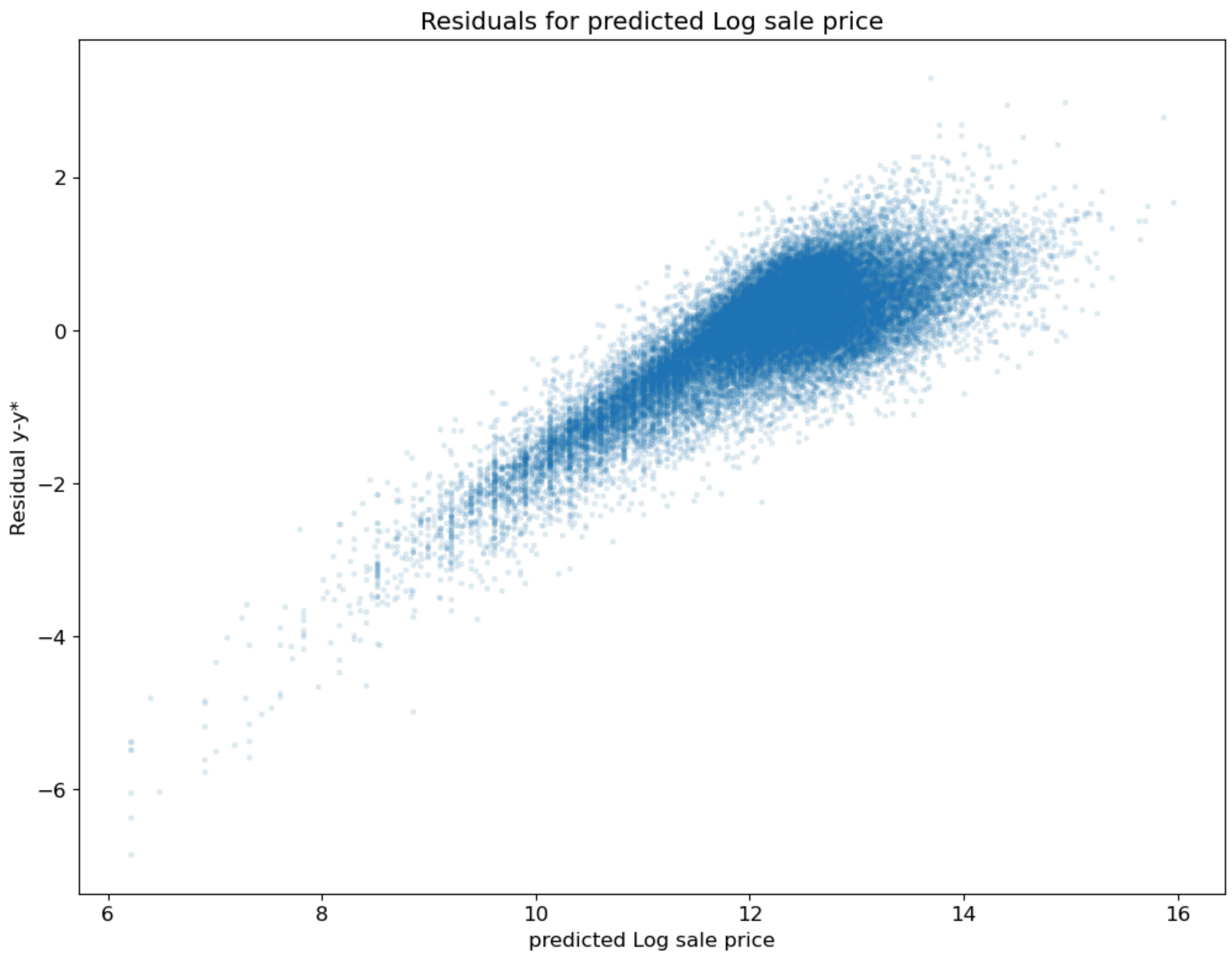
Question 4a

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

In the cell below, use `plt.scatter` (documentation) to plot the residuals from predicting `Log Sale Price` using **only the second model** against the original `Log Sale Price` for the **validation data**. With such a large dataset, it is difficult to avoid overplotting entirely. You should also **ensure that the dot size and opacity in the scatter plot are set appropriately** to reduce the impact of overplotting as much as possible.

In [22]:
```
plt.scatter(x = Y_valid_m2, y= Y_valid_m2 - Y_predicted_m2, s = 6, alpha = 0.1)
plt.xlabel("predicted Log sale price")
plt.ylabel("Residual y-y*")
plt.title("Residuals for predicted Log sale price")
```

Out[22]:
```
Text(0.5, 1.0, 'Residuals for predicted Log sale price')
```



Question 4b

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

Assign "regressive", "fair" or "progressive" to `q4b` in the cell below accordingly.

```
In [23]: q4b = "progressive"
```

```
In [24]: grader.check("q4b")
```

Out[24]: **q4b** passed! 💯

While our simple model explains some of the variability in price, there is certainly still a lot of room for improvement —— 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 will be important going into the next part of this project.

Question 5

It is time to build your own model!

You will conduct feature engineering on your training data using the `feature_engine_final` function (you will define this in `q5d`), fit the model with this training data, and compute the training Root Mean Squared Error (RMSE). Then, we will process our test data with `feature_engine_final`, use the model to predict `Log Sale Price` for the test data, transform the predicted and original log values back into their original forms (by using `delog`), and compute the test RMSE.

Your goal in Question 5 is to:

- Define a function to perform feature engineering and produce a design matrix for modeling.
- Apply this feature engineering function to the training data and use it to train a model that can predict the `Log Sale Price` of houses.
- Use this trained model to predict the `Log Sale Price`s of the test set. Remember that our test set does not contain the true `Sale Price` of each house –– your model is trying to guess them!
- Submit your predicted `Log Sale Price`s on the test set to Gradescope.

Right under the grading scheme, we will outline some important Datahub logistics. **Please make sure you read this carefully to avoid running into memory issues later!**

- In Question 5a, you can explore possible features for your model. This portion is **not graded**.
- In Question 5b, you can perform EDA on the dataset. This portion is **not graded**.
- In Question 5c, you can define feature engineering helper functions. This portion is **not graded**.
- In Question 5d, you will create your design matrix and train a model. This portion is **is graded**.
- In Question 5e, you can fit and evaluate your model. This portion is **not graded**.
- In Question 5f, you will generate the predictions for the test set. This portion is **is graded**.

Grading Scheme

Your grade for Question 5 will be based on your model's RMSE when making predictions on the training set, as well as your model's RMSE when making predictions on the test set. The tables below provide scoring guidelines. If your RMSE lies in a particular range, you will receive the number of points associated with that range.

Important: while your training RMSE can be checked at any time in this notebook, your test RMSE can only be checked by submitting your model's predictions to Gradescope. **You will only be able to submit your test set predictions to Gradescope up to 4 times per day**. Attempts will not carry over across days, so we recommend planning ahead to make sure you have enough time to finetune your model!

The thresholds are as follows:

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

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

---

Some notes before you start

- **If you are running into memory issues, restart the kernel and only run the cells you need to.** The cell below (question cell) contains most to all of the imports necessary to successfully complete this portion of the project, so it can be completed independently code-wise from the remainder of the project, and you do not need to rerun the cell at the top of this notebook. The autograder will have more than 4GB of memory, so you will not lose credit as long as your solution to Question 5 is within the total memory (4GB) limits of Datahub. By default, we reset the memory and clear all variables using `%reset -f`. If you want to delete specific variables, you may also use `del` in place of `%reset -f%`. For example, the following code will free up memory from data used for older models: `del training_val_data, test_data, train, validation, X_train_m1, X_valid_m1, X_train_m2, X_valid_m1`. Our staff solution can be run independently from all other questions, so we encourage you to do the same to make debugging easier.
- **If you need the data again after deleting the variables or resetting, you must reload them again from earlier in the notebook.**
- You will be predicting `Log Sale Price` on the data stored in `cook_county_contest_test.csv`. We will delog/exponentiate your prediction on Gradescope to compute RMSE and use this to score your model. Before submitting to Gradescope, make sure that your predicted values can all be delogged (i.e., if one of your `Log Sale Price` predictions is 60, it is too large; e^{60} is too big!)
- You MUST remove any additional new cells you add before submitting to Gradescope to avoid any autograder errors.
- **You can only submit your test set prediction CSV file to Gradescope up to 4 times per day. Start early!** In the case that you are approved for an extension, you will be granted 4 more submissions for each day the deadline has been extended.

PLEASE READ THE ABOVE MESSAGE CAREFULLY!

In [25]:
```
# The 3 lines below to clean up memory from previous questions and reinitialize Otter!
# If you want to refer to any functions or variables you defined at any point earlier in the proj
# Place them in the cell under Question 5c so that you can access them after the memory is reset
# If you think you will not run into any memory issues, you are free to comment out the next 3 l

%reset -f
import otter
grader = otter.Notebook("projA2.ipynb")

# Imports all the necessary libraries again

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
from sklearn import linear_model as lm

import warnings
warnings.filterwarnings("ignore")

import zipfile
import os

from ds100_utils import *
from feature_func import *

from sklearn.preprocessing import OneHotEncoder
```

Question 5a: Finding Potential Features

This question is not graded – it is intended to give helpful guidance on how to get started with feature engineering in `q5d`. You may write as little or as much as you would like here; it will not factor into your grade. Read the documentation about the dataset in `codebook.txt`, located in this directory. Is there any data you think may be related to housing prices? Include them below for future reference.

Road proximity, bedrooms, land square feet, estimate(land), repair condition

---

Question 5b: More EDA

This question is not graded – it is intended to give helpful guidance on how to get started with feature engineering. You may write as little or as much as you would like here; it will not factor into your grade. Use the scratch space below to conduct any additional EDA you would like to see. You may use this space to make additional plots to help you visualize the relationship between any variables or compute any relevant statistics. You are free to add any number of cells as needed below and before the next question. You may find it helpful to review Project A1 and the techniques we explore there.

▶ [**Click to Expand**] Some potential ideas.

In [ ]:

```
```

Question 5c: Defining Helper Function or Helper Variables

This question is not graded, but we suggest that you put all your helper functions below for readability and ease of testing. Use this space below to define any additional helper functions you may use in your final model. These can be transformation functions you identified in the optional question above.

In [26]:
```python
def logTransform(data, listOfColumns):
    for column in listOfColumns:
        data["Log " + column] = np.log(data[column] + 1)
    return data


def outlierRange(data, column, lower, upper):
    return data[(data[column] > lower) & (data[column] < upper)]
```

Question 5d: Defining The Pipeline Function

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 `feature_engine_final` in the following cell. Use of `.pipe` is not required, but you are welcome to incorporate it! **You must not change the parameters inside `feature_engine_final`. Do not edit the two lines at the end of the question cell below. They are helper functions that define a linear model, fit your data, and compute RMSE. If you do, you will receive no credit for this question.**

- Any feature engineering techniques that involve referencing `Sale Price` (for example, removing outlying `Sale Price` values from the training data) should be performed under the condition `if not is_test_set:`.
- All other feature engineering techniques should be applied to both the training and test sets. This means that you should perform them under the condition `else:`.
- When `is_test_set` is `True`, your function should return only the design matrix, `X`.
- When `is_test_set` is `False`, your function should return both the design matrix and the response variable `Y` (the `Log Sale Price` column).

Hints:

- Some features may have missing values in the test set but not in the training/validation set. Make sure `feature_engine_final` handles missing values appropriately for each feature.
- We have imported all feature engineering functions from Project A1 for you. You do not have access to the `feature_func.py` file with the function body and definitions, but they work as defined in Project A1. Feel free to use them as you see fit!
- You may wish to consider removing outlying datapoints from the training set before fitting your model. You may not, however, remove any datapoints from the test set (after all, the CCAO could not simply "refuse" to make predictions for a particular house!)
- As you finetune your model, you may unintentionally consume too much Datahub memory, causing your kernel to crash. See `q5a` for guidance on how to resolve this!!

Note: If you run into any errors, the Proj. A2 Common Mistakes section of the Data 100 Debugging Guide may be a helpful resource.

In [27]:
```python
# Please include all of your feature engineering processes inside this function.
# Do not modify the parameters of this function.
def feature_engine_final(data, is_test_set=False):
    # Whenever you access 'Log Sale Price' or 'Sale Price', make sure to use the
    # condition is_test_set like this:
    data = logTransform(data, ["Building Square Feet", "Age Decade", "Estimate (Building)"])

    if not is_test_set:
        # Processing for the training set (i.e. not the test set)
        # CAN involve references to sale price!
        # CAN involve filtering certain rows or removing outliers
        data['Log Sale Price'] = np.log(data['Sale Price'])
        data = outlierRange(data, "Log Sale Price", 5, 15 )
        data = outlierRange(data, "Age Decade", 0, 7)
        data = outlierRange(data, "Log Estimate (Building)", 3, 13)

    features = ["Log Building Square Feet", "Age Decade", "Log Estimate (Building)"]

    # Return predictors (X) and response (Y) variables separately
    if is_test_set:
        # Predictors
        X = data[features]
        return X
    else:
        # Predictors. Your X should not include Log Sale Price!
        X = data[features]
        # Response variable
        Y = data["Log Sale Price"]

        return X, Y

# DO NOT EDIT THESE TWO LINES!
check_rmse_threshold = run_linear_regression_test_optim(lm.LinearRegression(fit_intercept=True),
print("Current training RMSE:", check_rmse_threshold.loss)
print("You can check your grade for your prediction as per the grading scheme outlined at the sta
```

```
Current training RMSE: 126685.79383359094
You can check your grade for your prediction as per the grading scheme outlined at the start of Q
uestion 5
```

In [28]:
```
grader.check("q5d")
```

Out[28]:

q5d passed! 🌈

Question 5e: Fit and Evaluate your Model

This question is not graded. Use this space below to evaluate your models. Some ideas are listed below.

Note: While we have a grader function that checks RMSE for you, it is best to define and create your own model object and fit on your data. This way, you have access to the model directly to help you evaluate/debug if needed. For this project, you should use a `sklearn` default `LinearRegression()` model with intercept term for grading purposes. Do not modify any hyperparameter in `LinearRegression()`, and focus on feature selection or hyperparameters of your own feature engineering function.

It may also be helpful to calculate the RMSE directly as follows:

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

A function that computes the RMSE is provided below. Feel free to use it if you would like calculate the RMSE for your training set.

In [29]:
```
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 RMSE value.
    """
    return np.sqrt(np.mean((actual - predicted)**2))
```

In [ ]:
```
```

▶ [**Click to Expand**] Hints:

Question 5f Submission

Recall that the test set given to you in this assignment does not contain values for the true `Sale Price` of each house. You will be predicting `Log Sale Price` on the data stored in `cook_county_contest_test.csv`. To determine your model's RMSE on the test set, you will submit the predictions made by your model to Gradescope. There, we will run checks to see what your test RMSE is by considering (hidden) true values for the `Sale Price`. We will delog/exponentiate your prediction on Gradescope to compute RMSE and use this to score your model. Before submitting to Gradescope, make sure that your predicted values can all be delogged (i.e., if one of your `Log Sale Price` predictions is 60, it is too large; e^{60} is too big!)

Your score on this section will be determined by the grading scheme outlined at the start of Question 5. **Remember that you can only submit your test set predictions to Gradescope up to 4 times per day. Plan your time to ensure that you can adjust your model as necessary, and please test your model's performance using cross-validation before making any submissions.** For more on cross-validation, check Lecture 16. In particular, the Lecture 16 notebook may be helpful here. You can also feel free to reference what you did in previous questions when creating training and validation sets and seeing how your model performs.

To determine the error on the test set, please submit your predictions on the test set to the Gradescope assignment **Project A2 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.

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

Note: If you run into any errors, the Proj. A2 Common Mistakes section of the Data 100 Debugging Guide may be a helpful resource.

```
In [30]: from datetime import datetime
         from IPython.display import display, HTML

         Y_test_pred = run_linear_regression_test(lm.LinearRegression(fit_intercept=True), feature_engine
                                                  is_test = True, is_ranking = False, return_predictions
                                                  )

         # 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.now().strftime("%Y%m%d_%H%M%S")
         filename = "submission_{}.csv".format(timestamp)
         submission_df.to_csv(filename, index=False)

         #print('Created a CSV file: {}.'.format("submission_{}.csv".format(timestamp)))
         display(HTML("Download your test prediction <a href='" + filename + "' download>here</a>."))
         print('You may now upload this CSV file to Gradescope for scoring.')#
```

Download your test prediction here.

```
You may now upload this CSV file to Gradescope for scoring.
```

```
In [31]: # Scratch space to check if your prediction is reasonable. See 5e for hints.
         # We will not reset the submission count for mis-submission issues.
         submission_df["Value"].describe()
```

```
Out[31]: count    55311.000000
         mean        12.115893
         std          1.208307
         min          1.047067
         25%         11.683874
         50%         12.135728
         75%         12.631910
         max         16.141858
         Name: Value, dtype: float64
```

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 valuates properties.

Refer to Lecture 15 if you're having trouble getting started!

Question 6: Exploring RMSE

Let's delve a bit deeper into what RMSE means in the context of predicting house prices. We will go through different ways of visualizing the performance of the model you created and see how that ties into questions about property taxes. To this end, we'll create the `preds_df` `DataFrame` below that will prove useful for the later questions.

In [32]:
```
# Run the cell below; no further action is needed
train_df = pd.read_csv('cook_county_train.csv')
X, Y_true = feature_engine_final(train_df)
model = lm.LinearRegression(fit_intercept=True)
model.fit(X, Y_true)
Y_pred = model.predict(X)
```

In [33]:
```
preds_df = pd.DataFrame({'True Log Sale Price' : Y_true, 'Predicted Log Sale Price' : Y_pred,
                         'True Sale Price' : np.e**Y_true, 'Predicted Sale Price' : np.e**Y_pred
preds_df.head()
```

Out[33]:

	True Log Sale Price	Predicted Log Sale Price	True Sale Price	Predicted Sale Price
3	12.323856	12.283256	225000.0	216048.016531
4	10.025705	10.513990	22600.0	36827.133015
9	12.506177	12.420189	270000.0	247753.307632
11	11.184421	11.658679	72000.0	115691.126602
13	13.096019	13.016034	487000.0	449564.233852

Question 6a

Let's examine how our model performs on two halves of our data: `cheap_df` which contains the rows of `preds_df` with prices below or equal to the median sale price, and `expensive_df` which has rows of `preds_df` with true sale prices above the median. Take a moment to understand what is happening in the cell below, as it will also prove useful in `q6b`.

In [34]:
```
# Run the cell below to obtain the two subsets of data; no further action is needed.
min_Y_true, max_Y_true = np.round(np.min(Y_true), 1) , np.round(np.max(Y_true), 1)
median_Y_true = np.round(np.median(Y_true), 1)
cheap_df = preds_df[(preds_df['True Log Sale Price'] >= min_Y_true) & (preds_df['True Log Sale Pr
expensive_df = preds_df[(preds_df['True Log Sale Price'] > median_Y_true) & (preds_df['True Log S

print(f'\nThe lower interval contains houses with true sale price ${np.round(np.e**min_Y_true)} 
print(f'The higher interval contains houses with true sale price ${np.round(np.e**median_Y_true)
```

```
The lower interval contains houses with true sale price $245.0 to $198789.0
The higher interval contains houses with true sale price $198789.0 to $3269017.0
```

Compute the RMSE of your model's predictions of `Sale Price` on each subset separately, and assign those values to `rmse_cheap` and `rmse_expensive` respectively.

Separately, we also want to understand whether the proportion of houses in each interval that the model overestimates the value of the actual `Sale Price`. To that end, **compute the proportion of predictions strictly greater than the corresponding true price in each subset**, and assign it to `prop_overest_cheap` and `prop_overest_expensive` respectively. For example, if we were working with a dataset of 3 houses where the actual `Log Sale Price`s were [10, 11, 12] and the model predictions were [5, 15, 13], then the proportion of houses with overestimated values would be 2/3.

Note: When calculating `prop_overest_cheap` and `prop_overest_expensive`, you could use either `Log Sale Price` or `Sale Price`. Take a second to think through why this metric is unchanged under a log transformation.

In [35]:
```python
yHat,Y= cheap_df["Predicted Sale Price"], cheap_df["True Sale Price"]
yHat2, Y2 = expensive_df["Predicted Sale Price"], expensive_df["True Sale Price"]
rmse_cheap = rmse(yHat, Y)
rmse_expensive = rmse(yHat2, Y2)

prop_overest_cheap = cheap_df.query("`Predicted Sale Price` > `True Sale Price`").shape[0] / pred
prop_overest_expensive = expensive_df.query("`Predicted Sale Price` > `True Sale Price`").shape[(

print(f"The RMSE for properties with log sale prices in the interval {(min_Y_true, median_Y_true
print(f"The RMSE for properties with log sale prices in the interval {(median_Y_true, max_Y_true
print(f"The percentage of overestimated values for properties with log sale prices in the interva
print(f"The percentage of overestimated values for properties with log sale prices in the interva
```

```
The RMSE for properties with log sale prices in the interval (5.5, 12.2) is 56536.0
The RMSE for properties with log sale prices in the interval (12.2, 15.0) is 166777.0

The percentage of overestimated values for properties with log sale prices in the interval (5.5,
12.2) is 29.26%
The percentage of overestimated values for properties with log sale prices in the interval (12.2,
15.0) is 14.82%
```

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

Out[36]:

q6a passed! ✨

Question 6b

The intervals we defined above were rather broad. Let's try and take a more fine-grained approach to understand how RMSE and proportion of houses overestimated vary across different intervals of `Log Sale Price`. Complete the functions `rmse_interval` and `prop_overest_interval` to allow us to compute the appropriate values for any given interval. Pay close attention to the function description, and feel free to reuse and modify the code you wrote in the previous part as needed.

Note: The autograder tests provided for each of the functions are **not** comprehensive as the outputs of the function are highly dependent on your model. Make sure that the values you obtain are interpretable and that the plots that follow look right.

In [37]:
```python
def rmse_interval(df, start, end):
    '''
    Given a design matrix X and response vector Y, computes the RMSE for a subset of values
    wherein the corresponding Log Sale Price lies in the interval [start, end].

    Input:
    df : pandas DataFrame with columns 'True Log Sale Price',
        'Predicted Log Sale Price', 'True Sale Price', 'Predicted Sale Price'
    start : A float specifying the start of the interval (inclusive)
    end : A float specifying the end of the interval (inclusive)
    '''
    subset_df = df.query("`True Log Sale Price` >= @start and `True Log Sale Price` <= @end")

    rmse_subset = rmse(subset_df["Predicted Sale Price"], subset_df["True Sale Price"])
    return rmse_subset

def prop_overest_interval(df, start, end):
    '''
    Given a DataFrame df, computes prop_overest for a subset of values
    wherein the corresponding Log Sale Price lies in the interval [start, end].

    Input:
    df : pandas DataFrame with columns 'True Log Sale Price',
        'Predicted Log Sale Price', 'True Sale Price', 'Predicted Sale Price'
    start : A float specifying the start of the interval (inclusive)
    end : A float specifying the end of the interval (inclusive)
    '''

    subset_df = df.query("@end >=`True Log Sale Price` and `True Log Sale Price` >= @start")

    # DO NOT MODIFY THESE TWO LINES
    if subset_df.shape[0] == 0:
        return -1

    prop_subset = subset_df.query("`Predicted Sale Price` > `True Sale Price`").shape[0] / subse
    return prop_subset
```

In [38]:
```
grader.check("q6b")
```

Out[38]:

q6b passed! 🎉

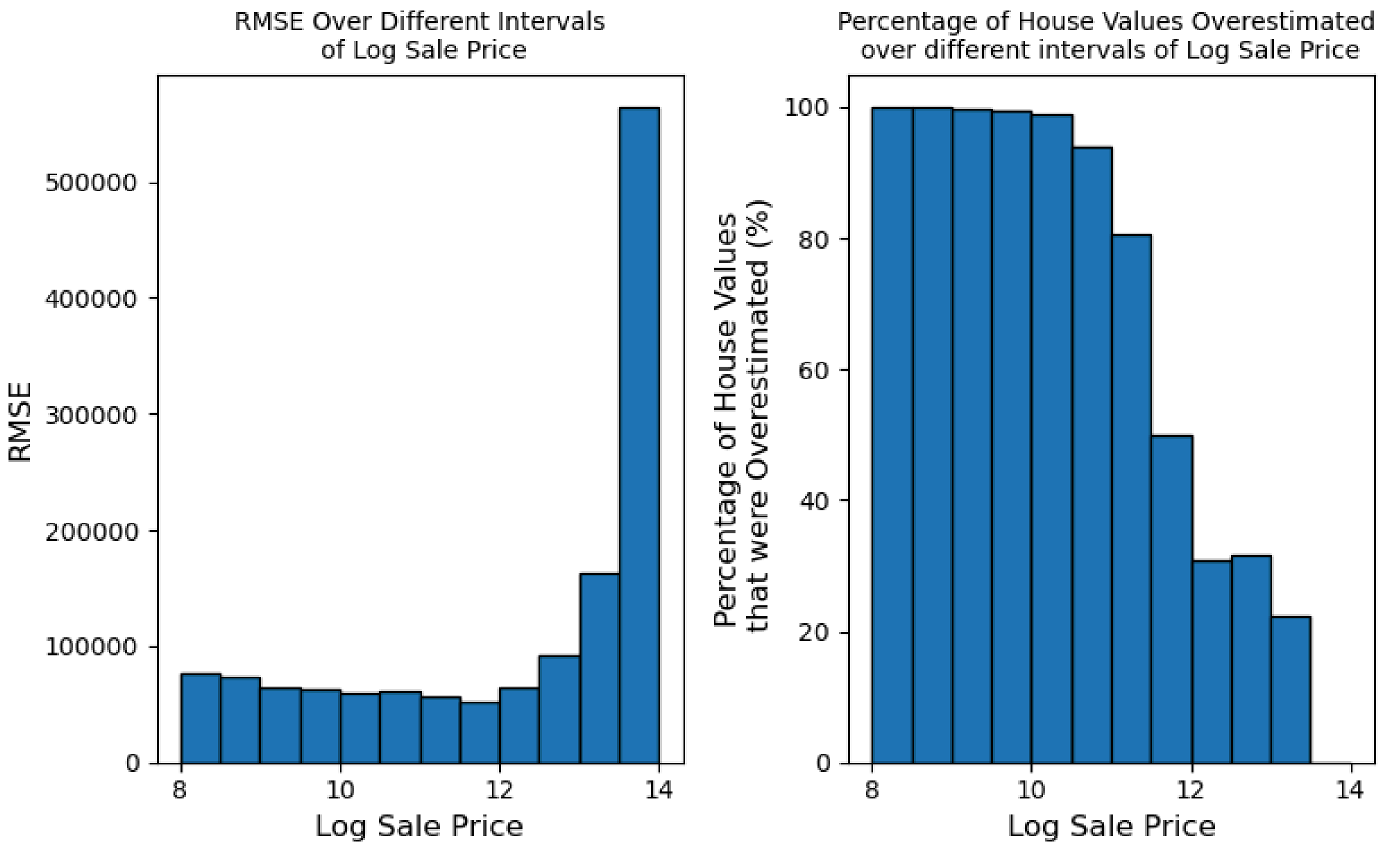
Question 6c

Now that you've defined these functions, let's put them to use and generate some interesting visualizations of how the RMSE and proportion of overestimated houses vary for different intervals.

In [39]:
```
# RMSE plot
plt.figure(figsize = (8,5))
plt.subplot(1, 2, 1)
rmses = []
for i in np.arange(8, 14, 0.5):
    rmses.append(rmse_interval(preds_df, i, i + 0.5))
plt.bar(x = np.arange(8.25, 14.25, 0.5), height = rmses, edgecolor = 'black', width = 0.5)
plt.title('RMSE Over Different Intervals\n of Log Sale Price', fontsize = 10)
plt.xlabel('Log Sale Price')
plt.yticks(fontsize = 10)
plt.xticks(fontsize = 10)
plt.ylabel('RMSE')

# Overestimation plot
plt.subplot(1, 2, 2)
props = []
for i in np.arange(8, 14, 0.5):
    props.append(prop_overest_interval(preds_df, i, i + 0.5) * 100)
plt.bar(x = np.arange(8.25, 14.25, 0.5), height = props, edgecolor = 'black', width = 0.5)
plt.title('Percentage of House Values Overestimated \nover different intervals of Log Sale Price
plt.xlabel('Log Sale Price')
plt.yticks(fontsize = 10)
plt.xticks(fontsize = 10)
plt.ylabel('Percentage of House Values\n that were Overestimated (%)')

plt.tight_layout()
plt.show()
```



Explicitly referencing **ONE** of the plots above (using `props` and `rmses`), explain whether the assessments your model predicts more closely aligns with scenario C or scenario D that we discussed back in `q1b`. Which of the two plots would be more useful in ascertaining whether the assessments tended to result in progressive or regressive taxation? Provide a brief explanation to support your choice of plot. For your reference, the scenarios are also shown below:

```
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.
```

Looking at `props`, I notice that overestimates are occuring nearly all of the time until log sale price appraoches values greater than 11, which translates to home values less than ~ $60,000 getting systematically over valued all the time. Looking at home values greater than log == 13.5, we see a staggering 0%, meaning our model either predicted these homes perfectly, or, undervalued them, which is actually the case here. So, my model here alligns with C. I am curious as to why these scenarios are so cut and dry, and why overvaluing both class of property or undervaluing both isnt a possibility, and i would like to understand why these scenarios laid out so perfectly reflect what is happening in my results.

Question 7: Evaluating the Model in Context

Question 7a

When evaluating your model, we used RMSE. In the context of estimating the value of houses, what does the residual mean for an individual homeowner? How does it affect them in terms of property taxes? Discuss the cases where the residual is positive and negative separately.

In terms of property taxes, a residual between the predicted home price and the true home price will tell you if that individual is paying more or less than they should in taxes. A positive residual, where home is overvalued, means they will pay more tax than is fair, and a negative residual will imply the opposite- if the county thinks your home is worth less than it "truly" is, you will pay less in taxes per year, and this a problem because when it comes time to sell your home, people aren't gonna pay the ammount the government/IRS thought it was worth, they will pay what they think it is worth.

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. 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 7b

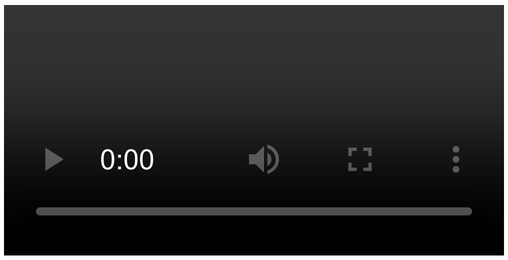
Reflecting back on your exploration in Questions 6 and 7a, in your own words, what makes a model's predictions of property values for tax assessment purposes "fair"?

This question is open-ended and part of your answer may depend on your specific model; we are looking for thoughtfulness and engagement with the material, not correctness.

Hint: Some guiding questions to reflect on as you answer the question above: What is the relationship between RMSE, accuracy, and fairness as you have defined it? Is a model with a low RMSE necessarily accurate? Is a model with a low RMSE necessarily "fair"? Is there any difference between your answers to the previous two questions? And if so, why?

I think the most apparent definition of fair here is fine tuning a model that maintains close to a 0% overestimation/underestimation rate for property values across the board, but in my experience, this is easier said than done. Lowering rmse does not necessarily make our model more fair, it might just mean that one side is skewed more than the other. Originally, my model used different outlier range cutoffs, and my rmse was higher, but the graph produced in prop overestimates from #6c was more balanced than it is now. I think my tweaks lowered rmse by overvaluing inexpensive homes while undervaluing expensive ones in a way that across the board was less far off but changed the proportions per log sale price bins a lot in favor of the less than upper class.

Ayga and Beck congratulate you on finishing Project A2!



Course Content Feedback

If you have any feedback about this assignment or about any of our other weekly, weekly assignments, lectures, or discussions, please fill out the Course Content Feedback Form. Your input is valuable in helping us improve the quality and relevance of our content to better meet your needs and expectations!

Submission Instructions

Below, you will see a cell. Running this cell will automatically generate a zip file with your autograded answers. Once you submit this file to the Project A2 Coding assignment on Gradescope, Gradescope will automatically submit a PDF file with your written answers to the Project A2 Written assignment. If you run into any issues when running this cell, feel free to check this section in the Data 100 Debugging Guide.

If there are issues with automatically generating the PDF, you can try downloading the notebook as a PDF by clicking on `File -> Save and Export Notebook As... -> PDF`. If that doesn't work either, you can manually take screenshots of your answers to the manually graded questions and submit those.

Please make sure you submit the following to the right assignments: