projA2

January 9, 2025

1 Project A2: Predicting Housing Prices in Cook County

1.1 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.

[1]:
```
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.

[2]:
```
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`.

[3]:
```
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.

[4]:
```
# 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.

[5]:
```
training_val_data.columns.values
```

[5]:
```
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)
```

1.2 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.**

1.2.1 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.**

1. Houseowner. They usually have an interest in seeing the housing price be high so that they can have more money if they choose to sell.
2. Buyers. They hope to see housing porices to be lower so that they can buy them more affordably.
3. Tax accessors. They usually have an interest in seeing a higher housing price, beacause higher housing prices usually lead to higher property taxes.

1.2.2 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.

'A' seems to be unfair. When a home is assessed at a higher price that it would sell for, the homeowner would be paying for more in taxes that they should be, which would cause a financial burden.

'C' seems to be more unfair than 'A' since a lower-income homeowner pay more taxes and a high-income homeowner pay less taxes.

'A' is an unfair situation between all homeowners and the government. And 'C' is an unfair situation between low-income and high-income houseowners.

1.2.3 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?

[6]:
```
q1c = "A"
```

1.3 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!

1.3.1 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.

The central problems with the earlier property tax system were inequities in property assessments that led to regressive taxation. Lower-priced properties were consistently overvalued, while 'higher-priced properties were undervalued.

The primary cause of these problems was not due to a faulty model. It was more about the appeals systems, which allowed wealthier homeowners to take advantage of the system and successfully challenge their assessments.

1.3.2 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?

Most low-income homeowners are non-white, and they typically do not have the resources or ability to appeal their home assessments, which puts a higher tax burden on them.

1.4 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 `DataFrames` 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` `DataFrames`. 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.

[7]:
```
# 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)

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

    train = data.iloc[train_indices]
    validation = data.iloc[validation_indices]

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

1.5 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:

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

1.6 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:

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

[8]:
```
q3a = "<="
```

1.6.1 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`.

[9]:
```
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
```

1.6.2 .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`.

[10]:
```
# 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
```

1.7 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 DataFrames`.**

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).

[11]:
```
# 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
```

[12]:
```
# 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)

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
```

1.8 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).

[13]:
```
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.

[14]:
```
# 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)
```

1.9 Question 4: Evaluate Our Simple Model

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

[15]:
```
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))
```

1.10 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

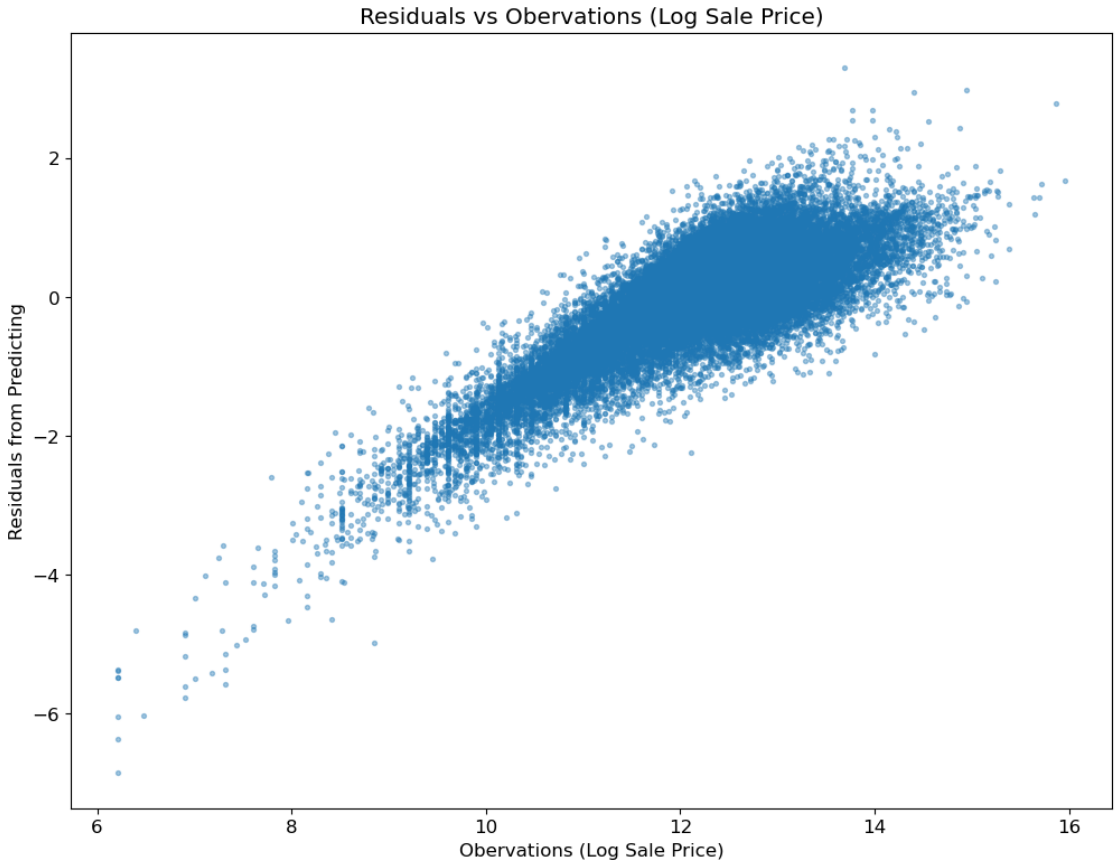
ensure that the dot size and opacity in the scatter plot are set appropriately to reduce the impact of overplotting as much as possible.

[16]:
```
e_m2 = Y_valid_m2 - Y_predicted_m2

plt.scatter(Y_valid_m2, e_m2, s=8, alpha=0.4)

plt.title("Residuals vs Obervations (Log Sale Price)")
plt.xlabel("Obervations (Log Sale Price)")
plt.ylabel("Residuals from Predicting")

plt.show()
```



1.11 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.

[17]:
```
q4b = "regressive"
```

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.

2 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**.

2.0.1 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

---

2.1 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!

[18]:
```
# 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 project,
```

```
# 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 lines as well.

%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
```

2.2 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.

2.3 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.

- Plot the distribution of a variable. Is this variable heavily skewed? Are there any outliers? This can inform how you engineer your features later on.
- Make a scatter plot between a continuous feature and the outcome. Is there a relationship? Is there a transformation that may linearize the relationship?
- Make a plot of a categorical/discrete feature and the outcome. Is there a relationship? How can we transform this categorical data into numerical features that can be useful for OLS?
- Find the correlation coefficient between features and the outcome. Is there a strong relationship between the two? Can you find the correlation coefficient between different transformations of the feature and the outcome?

[19]:
```
# Add any EDA code below
```

2.4 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.

[20]:
```
# Define any additional helper functions or variables you need here
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)
    if prediction_col in data.columns:
        X = data.drop(columns=[prediction_col])
        Y = data.loc[:, prediction_col]
        return X, Y
    else:
        # For test set where `prediction_col` is not present
        X = data
        return X, None

def find_expensive_neighborhoods(data, n=3, metric=np.median):
```

```
    neighborhoods = data.groupby("Neighborhood Code")['Sale Price'].agg(metric).
↪sort_values(ascending=False).head(n).index

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

def ohe(data, columns):
    """
    One-hot-encodes roof material. New columns are of the form "Roof␣
↪Material_MATERIAL".
    """
    new_data = data.copy()

    encoder = OneHotEncoder(dtype=int)

    RM_data = encoder.fit_transform(new_data[[columns]]).toarray()
    RM_df = pd.DataFrame(data=RM_data, columns=encoder.
↪get_feature_names_out([columns]), index=new_data.index)

    return new_data.join(RM_df)
```

2.5 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.

[21]:
```
# 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:
    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
        m3_pipelines = [
        (remove_outliers, None, {
                'variable': 'Sale Price',
            'lower': 499,
            'upper': 2350000,
        }),
        (log_transform, None, {'col': 'Sale Price'}),
        (log_transform, None, {'col': 'Building Square Feet'}),
        (log_transform, None, {'col': 'Age Decade'}),
        (add_total_bedrooms, None, None),
        (substitute_roof_material, None, None),
        (ohe, ['Roof Material'], None),
        (ohe, ['Wall Material'], None),
        (ohe, ['Property Class'], None),
        (ohe, ['Town and Neighborhood'], None),
        (select_columns, ['Log Sale Price', 'Bedrooms', 'Log Building Square␣
 ↪Feet', 'Lot Size', 'Log Age Decade',
                          'Garage 1 Size', 'Garage 2 Size', 'Basement Finish',␣
 ↪'Land Square Feet', 'Roof Material_Other',
                          'Roof Material_Shake', 'Roof Material_Shingle/
 ↪Asphalt', 'Roof Material_Slate', 'Roof Material_Tile', 'Wall Material_1.0',
                         'Wall Material_2.0', 'Wall Material_3.0', 'Wall␣
 ↪Material_4.0', 'Property Class_202', 'Property Class_203', 'Property␣
 ↪Class_204',
                         'Property Class_205', 'Property Class_206', 'Property␣
 ↪Class_207', 'Property Class_208', 'Property Class_209', 'Property Class_278',
                          'Other Improvements', 'Town and Neighborhood_77104',␣
 ↪'Town and Neighborhood_77115', 'Town and Neighborhood_77120', 'Town and␣
 ↪Neighborhood_77131',
```

```
                        'Town and Neighborhood_77132', 'Town and␣
↪Neighborhood_77141', 'Town and Neighborhood_77150', 'Town and␣
↪Neighborhood_77151', 'Town and Neighborhood_77152',
                        'Town and Neighborhood_77170', 'Town and␣
↪Neighborhood_1011', 'Town and Neighborhood_1012', 'Town and␣
↪Neighborhood_1014', 'Town and Neighborhood_1021',
                       'Town and Neighborhood_1022', 'Town and␣
↪Neighborhood_1023', 'Town and Neighborhood_1024', 'Town and␣
↪Neighborhood_1025', 'Town and Neighborhood_1030',
                       'Town and Neighborhood_1031', 'Town and␣
↪Neighborhood_2010', 'Town and Neighborhood_2011', 'Town and␣
↪Neighborhood_2020', 'Town and Neighborhood_2030',
                      'Town and Neighborhood_2040', 'Town and␣
↪Neighborhood_2050', ], None)
  ]


  else:
      # Processing for the test set
      # CANNOT involve references to sale price!
      # CANNOT involve removing any rows
      m3_pipelines = [
      (log_transform, None, {'col': 'Building Square Feet'}),
      (log_transform, None, {'col': 'Age Decade'}),
      (add_total_bedrooms, None, None),
      (substitute_roof_material, None, None),
      (ohe, ['Roof Material'], None),
      (ohe, ['Wall Material'], None),
      (ohe, ['Property Class'], None),
      (ohe, ['Town and Neighborhood'], None),
      (select_columns, ['Bedrooms', 'Log Building Square Feet', 'Lot Size',␣
↪'Log Age Decade',
                        'Garage 1 Size', 'Garage 2 Size', 'Basement Finish',␣
↪'Land Square Feet', 'Roof Material_Other',
                        'Roof Material_Shake', 'Roof Material_Shingle/
↪Asphalt', 'Roof Material_Slate', 'Roof Material_Tile', 'Wall Material_1.0',
                       'Wall Material_2.0', 'Wall Material_3.0', 'Wall␣
↪Material_4.0', 'Property Class_202', 'Property Class_203', 'Property␣
↪Class_204',
                       'Property Class_205', 'Property Class_206', 'Property␣
↪Class_207', 'Property Class_208', 'Property Class_209', 'Property Class_278',
                        'Other Improvements', 'Town and Neighborhood_77104',␣
↪'Town and Neighborhood_77115', 'Town and Neighborhood_77120', 'Town and␣
↪Neighborhood_77131',
```

```
                            'Town and Neighborhood_77132', 'Town and␣
 ↪Neighborhood_77141', 'Town and Neighborhood_77150', 'Town and␣
 ↪Neighborhood_77151', 'Town and Neighborhood_77152',
                            'Town and Neighborhood_77170', 'Town and␣
 ↪Neighborhood_1011', 'Town and Neighborhood_1012', 'Town and␣
 ↪Neighborhood_1014', 'Town and Neighborhood_1021',
                           'Town and Neighborhood_1022', 'Town and␣
 ↪Neighborhood_1023', 'Town and Neighborhood_1024', 'Town and␣
 ↪Neighborhood_1025', 'Town and Neighborhood_1030',
                           'Town and Neighborhood_1031', 'Town and␣
 ↪Neighborhood_2010', 'Town and Neighborhood_2011', 'Town and␣
 ↪Neighborhood_2020', 'Town and Neighborhood_2030',
                          'Town and Neighborhood_2040', 'Town and␣
 ↪Neighborhood_2050', ], None)
        ]

    # Processing for both test and training set
    # CANNOT involve references to sale price!
    # CANNOT involve removing any rows



    # Return predictors (X) and response (Y) variables separately
    if is_test_set:
        # Predictors
        X, _ = feature_engine_pipe(data, m3_pipelines, 'Log Sale Price')
        return X
    else:
        # Predictors. Your X should not include Log Sale Price!
        X, _ = feature_engine_pipe(data, m3_pipelines, 'Log Sale Price')
        # Response variable
        _, Y = feature_engine_pipe(data, m3_pipelines, '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), feature_engine_final,␣
 ↪'cook_county_train.csv', None, False)
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 start of Question 5")
```

```
Current training RMSE: 199061.64677832078
You can check your grade for your prediction as per the grading scheme outlined
at the start of Question 5
```

2.6 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.

[22]:
```
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))
```

[Click to Expand] Hints:

Train set:

- Check your RMSE. Is this a reasonable number? You may use our grading scheme as a reference. Keep in mind that training error is generally less than testing error.

Test set: * Find the original data shape at the beginning of the notebook (in the provided assert statement). What should the output shape be?

- Since test and training/validation sets come from the same population (recall that test and training/validation sets are a random split from larger data), we expect our test prediction to have a similar range as the validation data. Plot the observed training (Log) Sale Price and the predicted (Log) Sale Price. Are the ranges similar? Do you have any unreasonable extreme prediction that cannot be exponentiated?
- We cannot compute test RMSE directly since we do not have the observed values. Perform cross-validation to estimate your test error. Recall that we are treating the validation set as

unseen data.

[23]:
```
# Use this space to evaluate your model
# if you reset your memory, you need to define the functions again
```

2.7 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. **Furthermore, feel free to use the cross validation implementation done in Lab 8 to test your model.** You can also 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.

[24]:
```
from datetime import datetime
from IPython.display import display, HTML

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

# Construct and save the submission:
submission_df = pd.DataFrame({
```

```
    "Id": pd.read_csv('cook_county_contest_test.csv')['Unnamed: 0'],
    "Value": Y_test_pred,
}, columns=['Id', 'Value'])
timestamp = datetime.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.')#
```

```
<IPython.core.display.HTML object>
```

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

[25]:
```
# 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()
```

[25]:
```
count    55311.000000
mean        12.173659
std          0.609936
min         10.385290
25%         11.754816
50%         12.031337
75%         12.499953
max         15.824084
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.

2.8 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.

---

[26]:
```
# 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)
```

[27]:
```
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()
```

[27]:

	True Log Sale Price	Predicted Log Sale Price	True Sale Price	Predicted Sale Price
1	12.560244	11.759013	285000.0	127901.191297
2	9.998798	11.520066	22000.0	100716.593283
3	12.323856	11.789663	225000.0	131882.015104
4	10.025705	11.494287	22600.0	98153.441188
6	11.512925	12.144627	100000.0	188080.858120

2.8.1 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`.

[28]:
```
# 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 Price'] <= median_Y_true)]
expensive_df = preds_df[(preds_df['True Log Sale Price'] > median_Y_true) &␣
↪(preds_df['True Log Sale Price'] <= max_Y_true)]

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

```
The lower interval contains houses with true sale price $493.0 to $219696.0
The higher interval contains houses with true sale price $219696.0 to $2421748.0
```

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.

[29]:
```
rmse_cheap = rmse(cheap_df['True Sale Price'], cheap_df['Predicted Sale Price'])
rmse_expensive = rmse(expensive_df['True Sale Price'], expensive_df['Predicted␣
 ↪Sale Price'])

prop_overest_cheap = np.mean(cheap_df['Predicted Sale Price'] > cheap_df['True␣
 ↪Sale Price'])
prop_overest_expensive = np.mean(expensive_df['Predicted Sale Price'] >␣
 ↪expensive_df['True Sale Price'])

print(f"The RMSE for properties with log sale prices in the interval␣
 ↪{(min_Y_true, median_Y_true)} is {np.round(rmse_cheap)}")
print(f"The RMSE for properties with log sale prices in the interval␣
 ↪{(median_Y_true, max_Y_true)} is {np.round(rmse_expensive)}\n")
print(f"The percentage of overestimated values for properties with log sale␣
 ↪prices in the interval {(min_Y_true, median_Y_true)} is {np.round(100 *␣
 ↪prop_overest_cheap, 2)}%")
print(f"The percentage of overestimated values for properties with log sale␣
 ↪prices in the interval {(median_Y_true, max_Y_true)} is {np.round(100 *␣
 ↪prop_overest_expensive, 2)}%")
```

```
The RMSE for properties with log sale prices in the interval (6.2, 12.3) is
92797.0
The RMSE for properties with log sale prices in the interval (12.3, 14.7) is
268099.0

The percentage of overestimated values for properties with log sale prices in
the interval (6.2, 12.3) is 63.77%
The percentage of overestimated values for properties with log sale prices in
the interval (12.3, 14.7) is 17.51%
```

2.8.2 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.

[30]:
```
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[(df['True Log Sale Price'] >= start) & (df['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[(df['True Log Sale Price'] >= start) & (df['True Log Sale␣
 ↪Price'] <= end)]

    # DO NOT MODIFY THESE TWO LINES
```

```
    if subset_df.shape[0] == 0:
        return -1

    prop_subset = np.mean(subset_df['Predicted Sale Price'] > subset_df['True␣
 ↪Sale Price'])
    return prop_subset
```

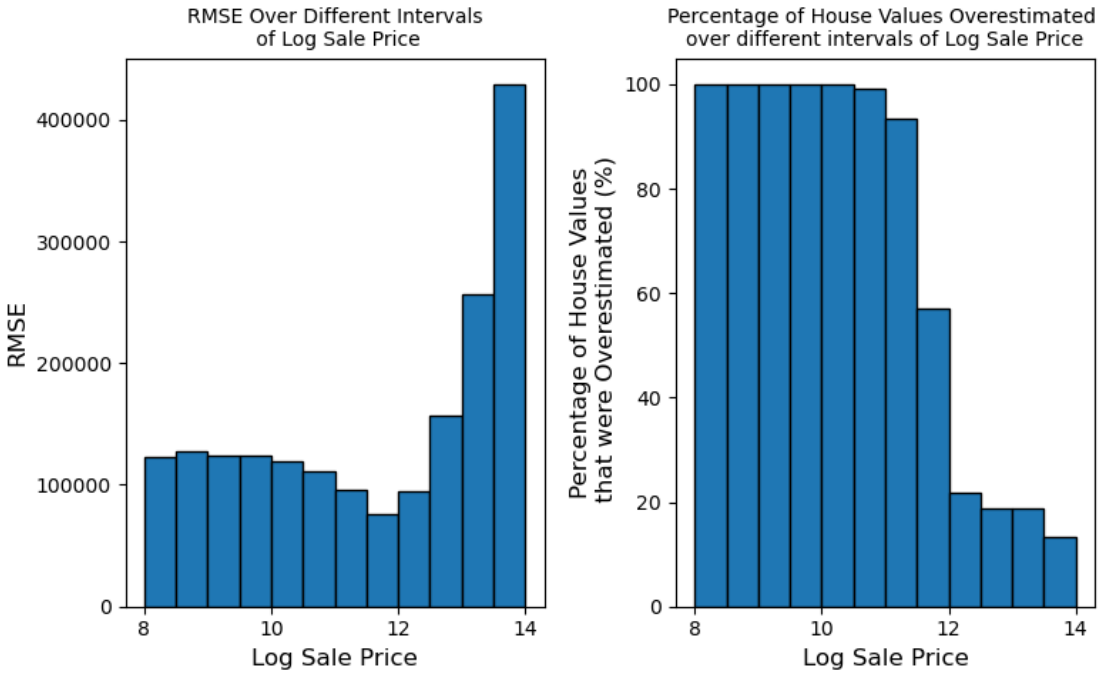
2.8.3 Question 6c

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.

[31]:
```
# 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', fontsize = 10)
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
D. An assessment process that systematically undervalues inexpensive properties and overvalues
```

According to the props plot, the pattern suggests that my model predicts aligns more closely with senario C which overvalues inexpensive properties and undervalues expensice properties.

The props plot is more useful in ascertaining whether the assessments tened to result in progressive or regressive taxation, since it can directly shows whether the predicted values are overestimated over different intervals.

2.9 Question 7: Evaluating the Model in Context

2.10 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.

Answer: The residual represents the difference between the model's predicted house value and the true market value of their property. When the residual is positive, the model has underestimated

the property's value, then the homeowner may pay less in property taxes than they should. When the residual is negative, the model has overestimated the property's value, then the homeowner may pay more in property taxes than they should.

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.

2.11 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?

A "fair" model not only needs to aim for minimizing the residuals and RMSE, but also needs yo recognize and account for systemic inequities. Even if we have an accurate model for property assessments, rich people have more resources to appeal their property assessments, which can lead to unfairness in the process. Therefore, when making a model, it is important to go beyond the data and consider broader society and historical issues to ensure the fairness.

[ ]: