

# proj1b

May 25, 2022

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

## 1 Project 1B: Predicting Housing Prices in Cook County

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

#### 1.1.1 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:** Nazanin Malekzada, Harleen Gill

### 1.2 Introduction

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

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

After this part of the project, you should be comfortable with: - Implementing a data processing pipeline using **pandas** - Using **scikit-learn** to build and fit linear models

### 1.3 Score Breakdown

Question	Points
0	5
1	2

Question	Points
2	2
3	3
4	2
5	2
6	1
7	4
8	6
9	2
10	1
11	2
Total	32

```
[2]: import numpy as np

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

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

import warnings
warnings.filterwarnings("ignore")

import zipfile
import os

from ds100_utils import run_linear_regression_test

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

Let's load the training and test data.

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

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

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

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

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

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

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

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

## 1.4 Question 0

### 1.4.1 Question 0a

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

Three parties interested in seeing the value of a home are homeowners, homebuyers, and the government. For homeowners specifically, these individuals would be interested in seeing the value be lower so they could pay lower taxes for the home. Homebuyers would be interested in seeing the value be lower in order to maximize their money used and invest in homes that are not overly inflated. And for the government, this group of individuals has an interest in seeing the value of homes to be higher in order to collect more taxes from the homeowners and homebuyers.

### 1.4.2 Question 0b

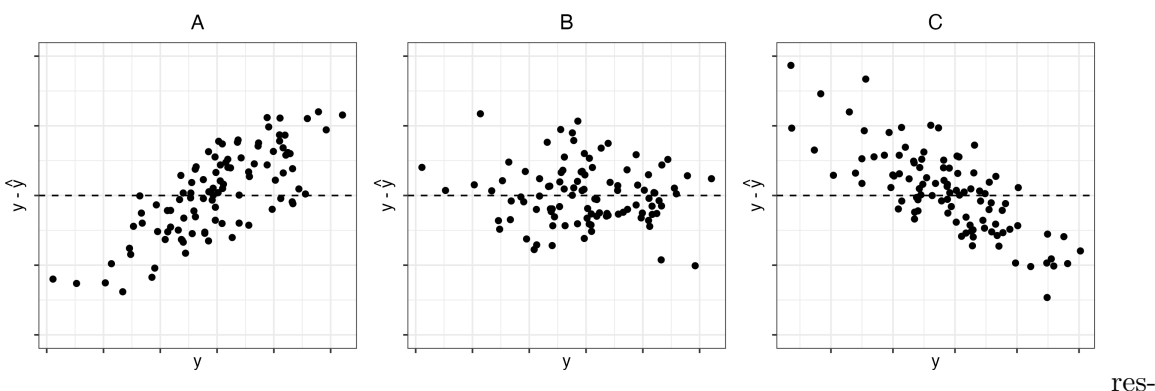
Which of the following scenarios strike you as unfair and why? You can choose more than one. There is no single right answer but you must explain your reasoning.

- A. A homeowner whose home is assessed at a higher price than it would sell for.
- B. A homeowner whose home is assessed at a lower price than it would sell for.
- C. An assessment process that systematically overvalues inexpensive properties and undervalues expensive properties.
- D. An assessment process that systematically undervalues inexpensive properties and overvalues expensive properties.

A strikes me as unfair because the homeowners will pay higher taxes than they should. B strikes me as unfair because the homeowner has smaller value in their assets than they should. C strikes me as unfair because more the more privileged, rich individuals will pay less in taxes in comparison to the unprivileged, poorer individuals, who will pay more in taxes for cheaper homes. D strikes me as unfair because the unprivileged, poorer individuals have homes that are more inexpensive in comparison to the more privileged, richer individuals who pay more taxes.

### 1.4.3 Question 0c

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



```
[7]: q0c = "B"
```

```
[8]: grader.check("q0c")
```

```
[8]: q0c results: All test cases passed!
```

## 1.5 The CCAO Dataset

The dataset you'll be working with comes from the Cook County Assessor's Office (CCAO) in Illinois, a government institution that determines property taxes across most of Chicago's metropolitan area and its nearby suburbs. In the United States, all property owners are required to pay property taxes, which are then used to fund public services including education, road maintenance, and

sanitation. These property tax assessments are based on property values estimated using statistical models that consider multiple factors, such as real estate value and construction cost.

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

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

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

### 1.5.1 Question 0d

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

*Type your answer here, replacing this text.*

### 1.5.2 Question 0e

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

*Type your answer here, replacing this text.*

## 1.6 Question 1

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

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

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

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

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

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

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

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

      def train_test_split(data):
          data_len = data.shape[0]
          shuffled_indices = np.random.permutation(data_len)
          train_indices = shuffled_indices[:int(data_len * 0.8)]
          test_indices = shuffled_indices[int(data_len * 0.8):]
          train = data.iloc[train_indices]
          test = data.iloc[test_indices]
          return train, test
      train, test = train_test_split(training_data)
```

```
[10]: grader.check("q1")
```

[10]: q1 results: All test cases passed!

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

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

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

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

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

## 1.7 Question 2

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

$\geq$

$\leq$

$=$

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

Training Loss of the 1st Model \_\_\_\_\_ Training Loss of the 2nd Model

$\geq$

## 1.8 Question 3

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



```
]

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

```
[14]: # DO NOT CHANGE THIS LINE
np.random.seed(1337)
# DO NOT CHANGE THIS LINE

# Process the data using the pipeline for the second model
train_m2, test_m2 = train_test_split(full_data)
m2_pipelines = [(remove_outliers, None, {'variable': "Sale Price", "lower":_
↪499}),
                (log_transform, None, {'col': "Sale Price"}),
                (log_transform, None, {'col': "Building Square Feet"}),
                (add_total_bedrooms, None, None),
                (select_columns, ['Log Sale Price', 'Bedrooms', 'Log Building_
↪Square Feet'], None)]

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

```
[15]: grader.check("q3")
```

[15]: q3 results: All test cases passed!

## 1.9 Question 4

Finally, let's do some regression!

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

```
[16]: from sklearn import linear_model as lm

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

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

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

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

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

```
[17]: # 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_test_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_test_m2)
```

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

[18]: q4 results: All test cases passed!

## 1.10 Question 5

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

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

The function is provided below.

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

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

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

- 1st model: training\_error\_m1, test\_error\_m1
- 2nd model: training\_error\_m2, test\_error\_m2

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

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

- 1st model: training\_error\_m1\_delog, test\_error\_m1\_delog
- 2nd model: training\_error\_m2\_delog, test\_error\_m2\_delog

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

# Training and test errors for the 1st model (in its original values before the
↳log transform)
training_error_m1_delog = rmse(np.exp(y_fitted_m1), np.exp(y_train_m1))
test_error_m1_delog = rmse(np.exp(y_predicted_m1), np.exp(y_test_m1))

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

# Training and test errors for the 2nd model (in its original values before the
↳log transform)
training_error_m2_delog = rmse(np.exp(y_fitted_m2), np.exp(y_train_m2))
test_error_m2_delog = rmse(np.exp(y_predicted_m2), np.exp(y_test_m2))

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

1st Model

Training RMSE: 0.9025651719699077

Test RMSE: 0.9068644732045896

1st Model (no log transform)

Training RMSE: 382697.78149699024

Test RMSE: 310679.2486611569

2nd Model  
Training RMSE: 0.8042009333446841  
Test RMSE: 0.8113963052434995

2nd Model (no log transform)  
Training RMSE: 325716.40819160367  
Test RMSE: 254880.42228506133

```
[21]: grader.check("q5")
```

[21]: q5 results: All test cases passed!

### 1.11 Question 6

Let's compare the actual parameters ( $\theta_0$  and  $\theta_1$ ) from both of our models. As a quick reminder, for the 1st model,

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

for the 2nd model,

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

Run the following cell and compare the values of  $\theta_1$  from both models. Why does  $\theta_1$  change from positive to negative when we introduce an additional feature in our 2nd model?

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

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

print("1st Model\n 0: {}\n 1: {}".format(theta0_m1, theta1_m1))
print("2nd Model\n 0: {}\n 1: {}\n 2: {}".format(theta0_m2, theta1_m2,
↪theta2_m2))
```

1st Model  
0: 10.571725401040084  
1: 0.4969197463141442  
2nd Model  
0: 1.9339633173823696  
1: -0.030647249803554506  
2: 1.4170991378689644

We can see that the Log Building Square Feet has a large, positive value, whereas Bedrooms has a small, positive value. In other words, this just means that the Log Building Square Feet variable is more significant and adds more value to the model than Building does.

## 1.12 Question 7

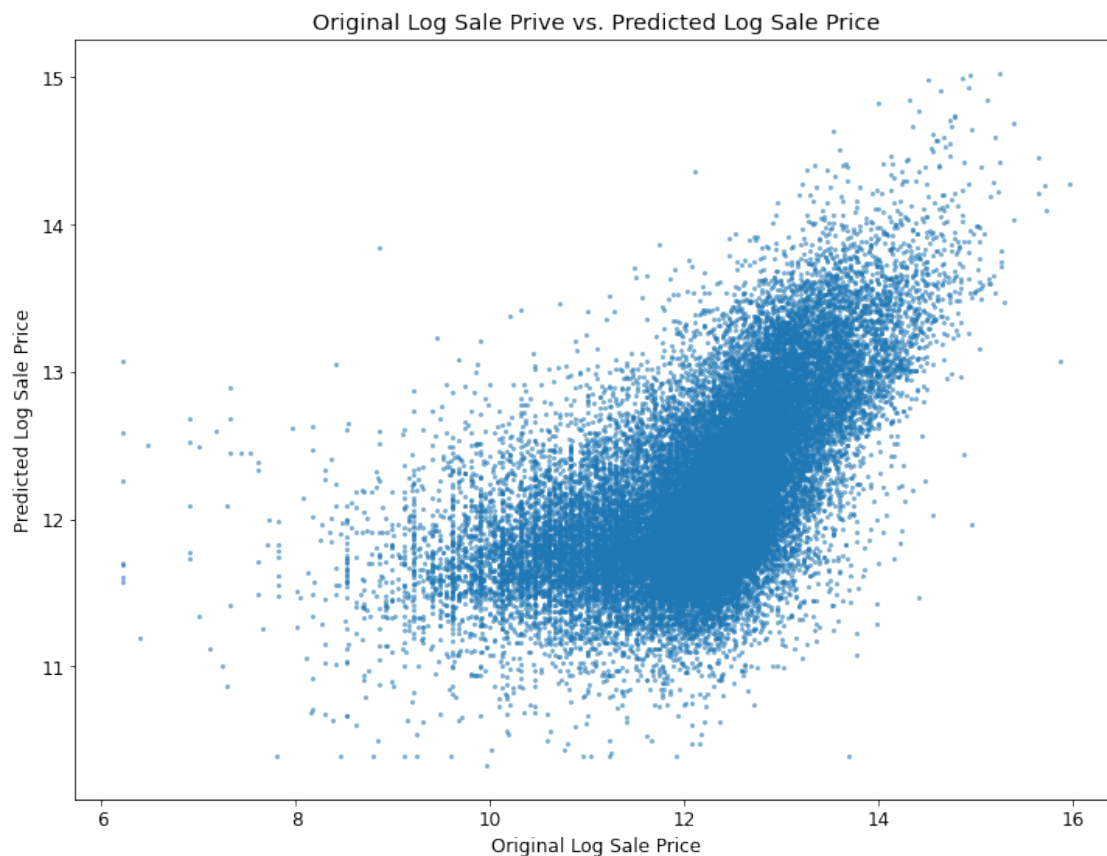
### 1.12.1 Question 7a

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

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

```
[23]: plt.scatter(x = y_test_m2, y = y_predicted_m2, s = 4, alpha = 0.50)
plt.xlabel("Original Log Sale Price")
plt.ylabel("Predicted Log Sale Price")
plt.title("Original Log Sale Price vs. Predicted Log Sale Price")
```

```
[23]: Text(0.5, 1.0, 'Original Log Sale Price vs. Predicted Log Sale Price')
```



### 1.12.2 Question 7b

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

```
[24]: q7b = "progressive"
```

```
[25]: grader.check("q7b")
```

[25]: q7b results: All test cases passed!

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

### 1.13 Question 8

It is time to build your own model!

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

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

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

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

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

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

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

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

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

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

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

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

### 1.13.1 Grading Scheme

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

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

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

```
[26]: # Define any additional helper functions you need here
def one_hot_encoding(data, lbl):
    ohe = OneHotEncoder()
    ohe.fit(data[[lbl]])
    dummies = pd.DataFrame(ohe.transform(data[[lbl]]).todense(), columns = ohe.
        get_feature_names(), index = data.index)
    return data.join(dummies)

def add_total_rooms(data):
    with_rooms = data.copy()
    r = r'has a total of (\d+) rooms'
    with_rooms["Rooms"] = with_rooms["Description"].str.extract(r).astype(int)
    return with_rooms

# Please include all of your feature engineering process inside this function.
# Do not modify the parameters of this function.
def process_data_fm(data, is_test_set=False):
    data = remove_outliers(data, variable = "Sale Price", lower = np.
        percentile(data["Sale Price"], 25), upper = np.percentile(data["Sale_
        Price"], 75))
    data = add_total_bedrooms(data)
    data = add_total_rooms(data)
```

```

data["Log Sale Price"] = np.log(data["Sale Price"])
data["Log Building Square Feet"] = np.log(data["Building Square Feet"])
data["Root Garage 1 Area"] = np.sqrt(data["Garage 1 Area"])
data["Root Garage 2 Area"] = np.sqrt(data["Garage 2 Area"])
expensive_neighborhoods = find_expensive_neighborhoods(data, 3, np.median)
data = add_in_expensive_neighborhood(data, expensive_neighborhoods)
data = one_hot_encoding(data, "Fireplaces")
ohe_cols = data.filter(regex = "^Ox").columns.tolist()
data = select_columns(data, "Log Building Square Feet", "Log Sale Price",
↪ "Root Garage 1 Area", "Root Garage 2 Area")

# Return predictors and response variables separately
if is_test_set:
    return X
else:
    X = data.drop(['Log Sale Price'], axis = 1)
    y = data.loc[:, 'Log Sale Price']
    return X, y

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

```

```

[26]: (
      Log Building Square Feet  Root Garage 1 Area  Root Garage 2 Area
1                6.904751                1.414214                0.0
3                7.068172                1.414214                0.0
6                7.458186                1.414214                0.0
9                7.343426                1.414214                0.0
10               6.841615                1.414214                0.0
...
204785            6.761573                1.414214                0.0
204786            6.948897                1.414214                0.0
204788            7.603399                1.000000                0.0
204789            6.815640                1.414214                2.0
204791            6.946976                0.000000                0.0

```

```

[102320 rows x 3 columns],
1      12.560244
3      12.323856
6      11.512925
9      12.506177
10     11.695247
...
204785  11.736069
204786  12.568978
204788  12.323856
204789  11.813030
204791  11.736069

```



Name: Log Sale Price, Length: 102320, dtype: float64)

```
[27]: grader.check("q8")
```

[27]: q8 results: All test cases passed!

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

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

```
[28]: from datetime import datetime

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

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

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

```
-----
KeyError                                Traceback (most recent call last)
File /opt/conda/lib/python3.9/site-packages/pandas/core/indexes/base.py:3361, in Index.get_loc(self, key, method, tolerance)
    ↳ Index.get_loc(self, key, method, tolerance)
    3360 try:
-> 3361     return self._engine.get_loc(casted_key)
    3362 except KeyError as err:

File /opt/conda/lib/python3.9/site-packages/pandas/_libs/index.pyx:76, in pandas._libs.index.IndexEngine.get_loc()
    ↳ pandas._libs.index.IndexEngine.get_loc()

File /opt/conda/lib/python3.9/site-packages/pandas/_libs/index.pyx:108, in pandas._libs.index.IndexEngine.get_loc()
    ↳ pandas._libs.index.IndexEngine.get_loc()
```

```
File pandas/_libs/hashtable_class_helper.pxi:5198, in pandas._libs.hashtable.  
↳PyObjectHashTable.get_item()
```

```
File pandas/_libs/hashtable_class_helper.pxi:5206, in pandas._libs.hashtable.  
↳PyObjectHashTable.get_item()
```

KeyError: 'Sale Price'

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

KeyError Traceback (most recent call last)

Input In [28], in <cell line: 3>()

```
1 from datetime import datetime
```

```
----> 3 Y_test_pred =
```

```
↳run_linear_regression_test(lm.LinearRegression(fit_intercept=True), process_data_fm, None,
```

```
4
```

```
↳ is_test = True, is_ranking = False, return_predictions)
```

```
5 )
```

```
7 # Construct and save the submission:
```

```
8 submission_df = pd.DataFrame({
```

```
9     "Id": pd.read_csv('cook_county_contest_test.csv')['Unnamed: 0'],
```

```
10     "Value": Y_test_pred,
```

```
11 }, columns=['Id', 'Value'])
```

File ~/sp22/proj/proj1b/ds100\_utils.py:93, in

```
↳run_linear_regression_test(final_model, process_data_fm, threshold,
```

```
↳train_data_path, test_data_path, is_test, is_ranking, return_predictions)
```

```
91 if is_test:
```

```
92     test_data = pd.read_csv(test_data_path, index_col='Unnamed: 0')
```

```
---> 93     X_test = process_data_fm(test_data, is_test_set = True)
```

```
94     assert len(test_data) == len(X_test), 'You may not remove data
```

```
↳points from the test set!'
```

```
96 final_model.fit(X_train, y_train)
```

Input In [26], in process\_data\_fm(data, is\_test\_set)

```
16 def process_data_fm(data, is_test_set=False):
```

```
---> 17     data = remove_outliers(data, variable = "Sale Price", lower = np.
```

```
↳percentile(data["Sale Price"], 25), upper = np.percentile(data["Sale Price"],
```

```
↳75))
```

```
18     data = add_total_bedrooms(data)
```

```
19     data = add_total_rooms(data)
```

File /opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3458, in

```
↳DataFrame._getitem__(self, key)
```

```
3456 if self.columns.nlevels > 1:
```

```
3457     return self._getitem_multilevel(key)
```

```
-> 3458 indexer = self.columns.get_loc(key)
```

```
3459 if is_integer(indexer):
```

```

3460     indexer = [indexer]

File /opt/conda/lib/python3.9/site-packages/pandas/core/indexes/base.py:3363, in Index.get_loc(self, key, method, tolerance)
    3361         return self._engine.get_loc(casted_key)
    3362     except KeyError as err:
-> 3363         raise KeyError(key) from err
    3365 if is_scalar(key) and isna(key) and not self.hasnans:
    3366     raise KeyError(key)

KeyError: 'Sale Price'

```

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 values properties.

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

## 1.14 Question 9

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

In the context of estimating the values of houses, specifically for individual homeowners, error means how far the original sale price of the homes deviates from the predicted sale prices. In terms of property taxes, these values depend on the assessment accuracy of the property, so the error does play into the value of the home (whether it is overvalued, or undervalued). In terms of RMSE specifically, a higher value signifies that the mean value homeowners are actually paying deviate significantly from the value homeowners "should" be paying. So, a lower RMSE value is more desirable.

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

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

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

taxes?

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

### 1.15 Question 10

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

In regards to property assessment and taxes, I believe that the term fairness signifies how individual’s ownership of their properties are assessed equally, regardless of the individuals’ personal attributes (i.e. race, gender, sexual orientation, ethnicity, etc.). As we know, individuals who are considered to be a minority are generally placed into and experience more poverty and struggle than individuals part of the “majority” groups.

### 1.16 The CCAO and Transparency

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

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

### 1.17 Question 11

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

After looking at the Residential Automated Valuation Model files, I don’t believe that the documentation sufficiently explains how the model works. The technical terminology employed in this documentation was quite difficult for me to understand, and I’ve been more exposed to the basics of modeling in data science, so I can’t imagine how the general society would be able to understand the documentation. Specifically, the Run Outputs table would be difficult for nontechnical audiences to understand as individuals may not know what the difference between limited and full runs is, or what either of these terms mean.

You might feel that the model’s inner workings are beyond your pay grade - it’s far more complex than the model you built in this assignment, after all! Though we won’t delve further into the role

of transparency in the broader CCAO case study, consider its effectiveness and/or ineffectiveness: Is the system truly transparent if it's inaccessible to Cook County constituents? Do transparency measures actually bolster the accuracy of a model - or do they only affect the *perceived* accuracy of a model?

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

## 1.18 Congratulations! You have finished Project 1B!

---

To double-check your work, the cell below will rerun all of the autograder tests.

```
[29]: grader.check_all()
```

```
[29]: q0c results: All test cases passed!
```

```
q1 results: All test cases passed!
```

```
q3 results: All test cases passed!
```

```
q4 results: All test cases passed!
```

```
q5 results: All test cases passed!
```

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

```
q8 results: All test cases passed!
```

## 1.19 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!**

```
[30]: # Save your notebook first, then run this cell to export your submission.
grader.export()
```

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