

proj1b

November 25, 2024

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

1 Project 1B: Predicting Housing Prices in Cook County

1.1 Due Date: Thursday, October 27th, 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: *list names here*

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	1
10	2
11	1
12	2
Total	33

```
[132]: 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 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.

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

```
[134]: 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.

```
[135]: # 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.

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

```
[136]: 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.

Buyer - wants the value to be low
 Seller - wants the value to be high
 Government who collect property taxes as a percentage of the value of the house - wants the value to be high

1.4.2 Question 0b

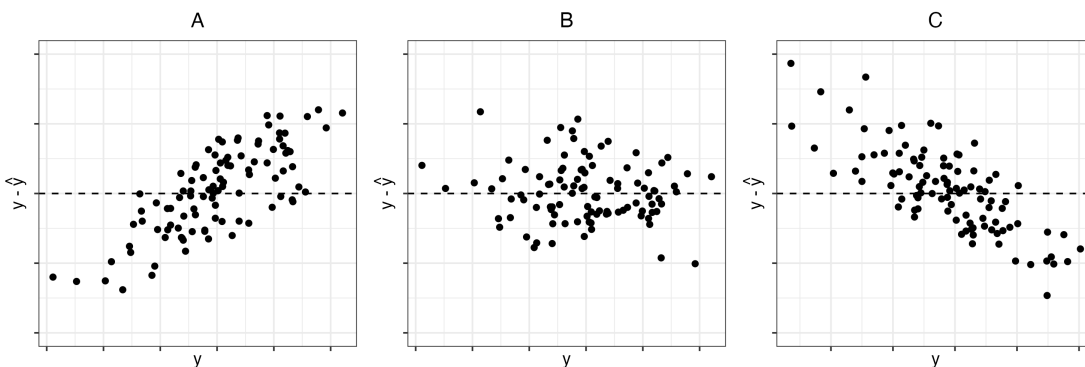
Which of the following scenarios strike you as unfair and why? You can choose more than one. There is no single right answer but you must explain your reasoning.

- 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.

I think all of the scenarios are unfair, with C and D being more unfair since there's a systemic flaw, meaning the impact of such system is more widespread. I consider it unfair because it there's a misestimation of the true value. There's a certain bias, and that by definition isn't impartial or fair.

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?



```
[137]: q0c = 'a'
```

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

```
[138]: 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)

The central problems with the earlier property tax system was that it undervalued expensive properties and overvalued inexpensive properties. This resulted in wealthier people, who were usually white, to pay less in property tax, while poorer people, who were usually black, had to pay more. The primary cause of these problems was the appeals system, in which people who had the knowledge and time to appeal their original property assessment were able to lower their property value and thus their taxes. The head of the property assessment group also had deep connections with people who were apart of the appeals business, so there was personal financial incentive to systemically overvalue properties so there was an opportunity to appeal it.

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?

The property tax system has a quota of amount of taxes it recieves from property taxes, meaning even if the inexpensive properties were fairly assessed, the people who lived in those properties would have to pay more taxes if the richer white property owners were still able to get their appeals accepted.

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.

```
[139]: np.random.seed(1337)
        shuffled_indices = np.random.permutation(10)
        display(shuffled_indices)
        train_indices = shuffled_indices[:int(len(shuffled_indices)*.8)]
        test_indices = shuffled_indices[int(len(shuffled_indices)*.8) + 1:]
        display(train_indices, test_indices)
```

```
array([6, 3, 0, 4, 5, 2, 1, 9, 8, 7])
```

```
array([6, 3, 0, 4, 5, 2, 1, 9])
```

```
array([7])
```

```
[140]: # 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(len(shuffled_indices)*.8)]
            test_indices = shuffled_indices[int(len(shuffled_indices)*.8):]
            return data.iloc[train_indices], data.iloc[test_indices]
        train, test = train_test_split(training_data)
        print(train.shape)
        print(test.shape)
```

```
(163833, 62)
```

```
(40959, 62)
```

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

[141]: 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

greater than or equal to

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`.

```
[142]: 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

[143]: 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).

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

```
[144]:
```

	PIN	Property Class	Neighborhood Code	Land Square Feet	\
0	17294100610000	203	50	2500.0	
1	13272240180000	202	120	3780.0	

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

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

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

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

	Description	Lot Size
0	This property, sold on 09/14/2015, is a one-st...	2500.0
1	This property, sold on 05/23/2018, is a one-st...	3780.0

[2 rows x 62 columns]

```
[145]: # 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')

```

```

[146]: # 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_m1, test_m1

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

```

```

[147]: grader.check("q3")

```

[147]: 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).

```
[148]: 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.

```
[149]: # Fit the 1st model
# Compute the fitted and predicted values of Log Sale Price for 1st model
linear_model_m1.fit(X_train_m1, y_train_m1)
y_fitted_m1 = linear_model_m1.predict(X_train_m1)
y_predicted_m1 = linear_model_m1.predict(X_test_m1)

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

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

[150]: 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.

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

The function is provided below.

```
[151]: 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`

```

[152]: # 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: {}".format(training_error_m1,
↳ test_error_m1))
print("1st Model (no log transform)\nTraining RMSE: {}\nTest RMSE: {}".
↳ 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

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

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

1.11 Question 6

Let's compare the actual parameters (θ_0 and θ_1) from both of our models. As a quick reminder, for the 1st model,

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

for the 2nd model,

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

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

That parameter gives the best fit line with a model with 2 features. It being negative makes sense, since it suggests the log sale price of the property goes down with the same number log square footage and higher bedrooms. This makes sense because with a fixed square footage the bedrooms are more cramped.

```
[154]: # 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
```

1.12 Question 7

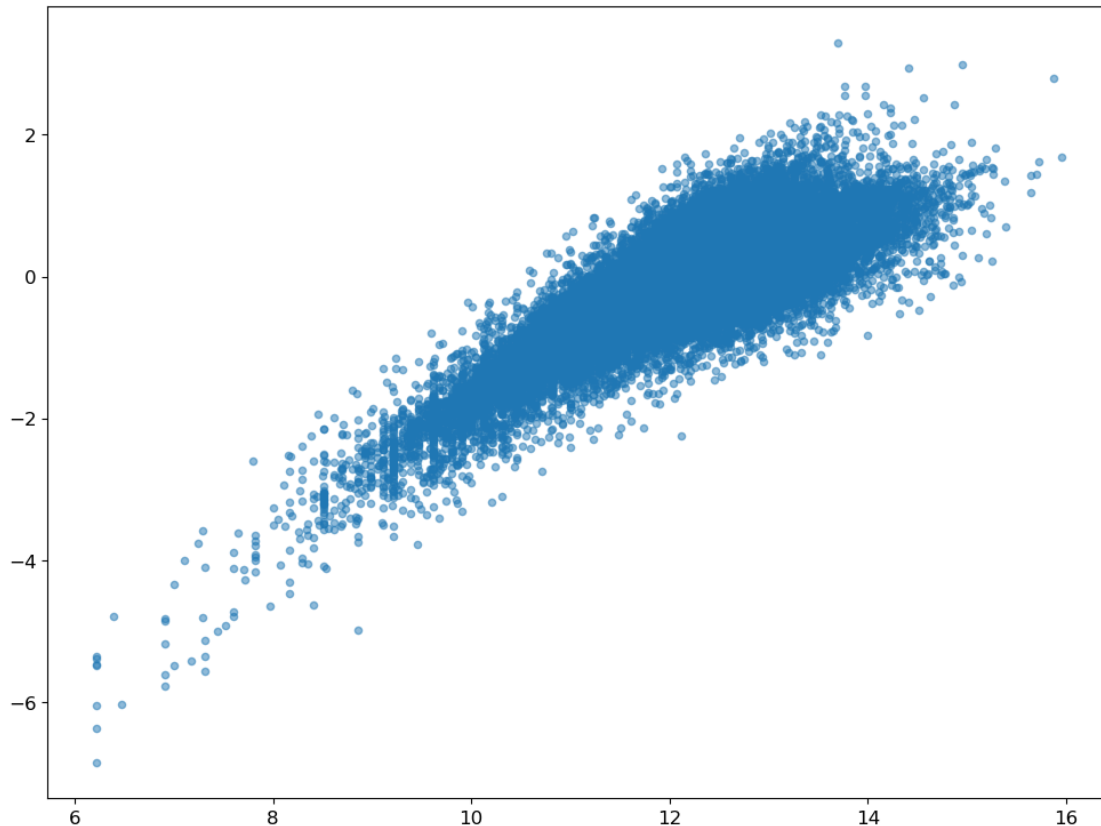
1.12.1 Question 7a

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.

```
[155]: plt.scatter(y_test_m2, y_test_m2-y_predicted_m2, s=20, alpha=.5)
```

```
[155]: <matplotlib.collections.PathCollection at 0x7fcd71eee0a0>
```



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?

```
[156]: q7b = 'regressive'
```

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

[79]: 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. Then, we will process training data using your `process_data_fm`, fit the model with this training data, and compute the training RMSE. Then, we will process some test data with your `process_data_fm`, 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.

Notes: - **If you are running into memory issues, restart 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 (almost) independently code-wise from the remainder of the project. The autograder will have more than 2 GB memory, so you will not lose credit as long as your solution to Question 8 is within the total memory limits of DataHub. Alternatively, you can delete variables you are not using through `del` or `%reset -f`. For example, this will free up memory from data used for older models: `del training_data, test_data, train, test, X_train_m1, X_test_m1, X_train_m2, X_test_m1`. Our staff solution (Summer 2022) can be run independently from all other questions, so we encourage you to do the same to make debugging easier. - `delog` is a function we will run to undo the log transformation on your predictions/original sale prices. Before submitting to Gradescope, make sure that your predicted values can all be delogged (i.e. if the value is 100, it is too large - e^{100} is too big!) - 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`. - You **MUST remove any additional new cells you add below the current one before submitting to Gradescope** to avoid any autograder errors. - Do **not** edit the two lines at the end of the question cell below - if you do, you will receive no credit for this question. - You could only submit the csv file to gradescope up to **3 times** per day.

Hints: - 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! - Pay a *lot* of attention to how you filter your outliers. Treat your upper outlier percentile as a hyperparameter. How can we filter the optimal number of outliers to obtain the best possible test RMSE? - Using the pipeline as we have seen in Question 3 might not work if you are doing one-hot encoding. Consider writing general function to process training and test set.

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


```

[127]: # Define any additional helper functions you need here
from sklearn.model_selection import KFold

# Uncomment the line below to clean up memory from previous questions and
# ↪reinitialize Otter!
# MAKE SURE TO COMMENT THE NEXT 3 LINES OUT BEFORE SUBMITTING!
# %reset -f
# import otter
# grader = otter.Notebook("proj1b.ipynb")

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 *

# def compute_CV_error(model, X_train, Y_train):
#     '''
#     # Split the training data into 4 subsets.
#     # For each subset,
#     #     fit a model holding out that subset
#     #     compute the MSE on that subset (the validation set)
#     # You should be fitting 4 models total.
#     # Return the average MSE of these 4 folds.

#     Args:
#     #     model: an sklearn model with fit and predict functions
#     #     X_train (data_frame): Training data
#     #     Y_train (data_frame): Label

#     Return:
#     #     the average validation MSE for the 4 splits.
#     '''

```

```

#     kf = KFold(n_splits=4)
#     validation_errors = []
#     for train_idx, valid_idx in kf.split(X_train):
#         # split the data
#         split_X_train, split_X_valid = X_train.iloc[train_idx], X_train.
#         ↪iloc[valid_idx]
#         split_Y_train, split_Y_valid = Y_train.iloc[train_idx], Y_train.
#         ↪iloc[valid_idx]

#         # Fit the model on the training split
#         model.fit(split_X_train, split_Y_train)

#         # Compute the RMSE on the validation split
#         error = rmse(split_Y_valid, model.predict(split_X_valid))

#         validation_errors.append(error)

#     return np.mean(validation_errors)

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

def process_common(data):
    data['Log Building Square Feet'] = np.log(data['Building Square Feet'])
    data = add_total_bedrooms(data)
    # data['Property Class Squared'] = data['Property Class']**2
    # data['Age Squared'] = data['Age']**2

    return data
...
...

# Please include all of your feature engineering process inside this function.
# Do not modify the parameters of this function.
def process_data_fm(data, is_test_set=False):

```

```

# 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:
    # do your processing for the training set (i.e. not the test set)
    # this can involve references to sale price!
    data['Log Sale Price'] = np.log(data['Sale Price'])
    ...
    data = remove_outliers(data, 'Sale Price', lower=500)
    data = remove_outliers(data, 'Building Square Feet', upper=800)
    data = process_common(data)

else:
    ...
    # do your processing for the test set
    # this CANNOT involve references to sale price!
    data = process_common(data)

...
# Return predictors and response variables separately
if is_test_set:
    X = data
    ... # any other processing you wish to do
    X = data[['Log Building Square Feet', 'Bedrooms']]

    return X
else:
    X = data.drop(['Log Sale Price'], axis = 1)
    y = data.loc[:, 'Log Sale Price']
    ... # any other processing you wish to do
    X = data[['Log Building Square Feet', 'Bedrooms']]
    return X, y

# DO NOT EDIT THESE TWO LINES!
check_rmse_threshold = run_linear_regression_test_optim(lm.
↳LinearRegression(fit_intercept=True), process_data_fm, 'cook_county_train.
↳csv', None, False)
print("Current training RMSE:", check_rmse_threshold.loss)

```

Current training RMSE: 113798.44354744775

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

[128]: 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!

```
[129]: from datetime import datetime

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

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

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

Created a CSV file: submission_2022-10-28T06:18:16.csv.

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

1.14 Question 9

In building your model in question 8, what different models have you tried? What worked and what did not? Briefly discuss your modeling process.

Note: We are looking for a single correct answer. Explain what you did in question 8 and you will get point.

I had a somewhat naive approach, in which I used natural intuition and looked at the features in the dataset, and thought about which features would most likely have a linear relationship to the sale price of the home. These features ranged from rooms to squarefootage to age to property class. I then tried a handful of combinations, some with featured engineering (i.e. property class squared, with my reasoning being that the property classes are too bunched up in a small range, so I thought squaring them would help spread out the values and would yield better predictions overall). I noticed that when I got to four or so features, my train rsme couldn't beat 113-114k, but my test rsme was the worst it's ever been. I made the naive assumption to keep the number of features below 4, with the assumption being that I'm not missing a hugely important feature, that most features don't contribute significantly more to the sales price than another feature. Since each feature is around the same "value" in terms of having it in the model, and with 4 features leading to overfitting, I decided to use less than 4 features in total.

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!

1.15 Question 10

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?

RSME in the context of estimating values of houses are how much the model was off by in terms of dollars. This greatly affects homeowners, because if they were overvalued, they would have to pay more property taxes. Additionally, the rsme of my model was 113-114k, which is a lot of extra money to be taxed on.

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](#).

1.16 Question 11

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

In terms of property assessments and taxes, I believe accurate assessments do equate significantly to fairness. Keep in mind, I'm not talking about the merits of the overall structure and system of the tax system in place (i.e. I don't know how property taxes work super well. Is it a flat percentage? Should there even be property taxes?). My focus is ensuring that such system is realized in its thought about form. With accurate assessments, homeowners would pay taxes that they deserve

to pay under the system. A big underlying principle under my belief is that there isn't some sort of systemic bias in valuation of homes, in that if the model uses the values of homes with similar characteristics, the evaluation of that training set is accurate and fair. I suppose my belief of this comes from the free market, in that if there were an unnatural valuation in certain homes, then the market would correct itself. Of course, this assumption, and thus the model built on top of it, could be debated.

Additionally, I only focused on a vacuum situation. In the grander scale of fairness in property taxes, there are larger historical trends and inequalities to be addressed outside the modeling predictions. Things like reparations for citizens who were historically robbed is what I consider fair, because they were overcharged in the first place. Fairness to me, as a general term, means abiding to the agreed upon system. And in the case of property taxes, poorer families had to act in a different way (pay more) to the system, and thus that should be corrected.

1.17 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.

1.18 Question 12

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

I do feel like there's sufficient enough documentation to be able to decipher and dissect the model. I would say, however, that almost every part of the documentation is tough for nontechnical people to understand. There's too many file pathways, with too foreign of naming to the general audience. The figures in the docs folder are too sparse, and the README is still really long and somewhat technical. I don't blame CCAO though. I don't believe there's a way to dumb down and summarize the inner workings of modeling without losing a ton of important information, to the point where that summary would be useless and “magical/black box” which is the opposite of what something that's transparent should be.

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...

1.19 Congratulations! You have finished Project 1B!

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

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

```
[157]: 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!
```

1.20 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!**

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

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