proj1b

February 15, 2023

```
[1]: # 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 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 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
2	2
3	3
4	2
5	2
6	1
7	4
8	6
9	1
10	2
11	1
12	2
Total	33

[2]:
```python
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.

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

[4]:
```python
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',
       'Neigborhood 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.

People who might be interested in an answer to this question would be: 1. homeowner: if the owner is trying to sell the property, then a higher value would benefit them more 2. buyer: a lower house value would be preferable for someone looking to buy the property 3. government party: if the house value is high, then so is the property tax which would benefit the state

1.4.2 Question 0b

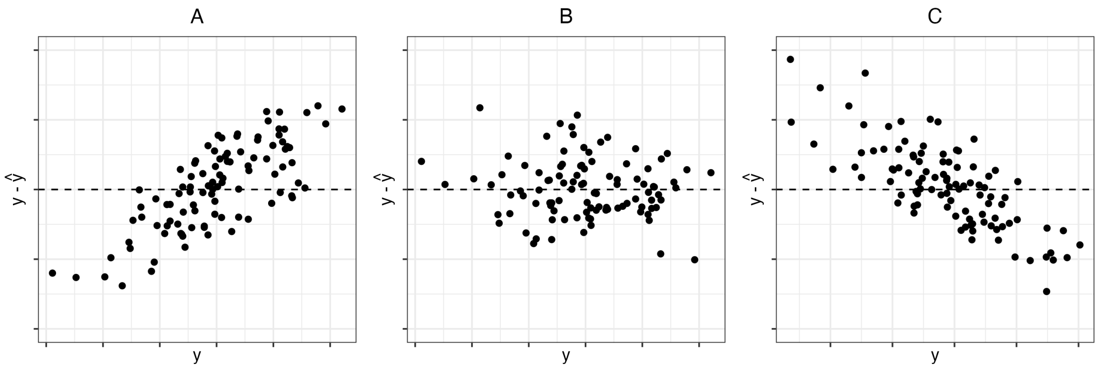
Which of the following scenarios strike you as unfair and why? You can choose more than one. There is no single right answer but you must explain your reasoning.

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.

Each scenario could be considered unfair, but I personally believe that option C is the most unfair because it is the same underlying system for regressive taxation. This can result in minority populations being disporportionally taxed while wealthier populations only benefit even more.

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)

The central problems with the property tax system in Cook County included: 1. redlining: the process of making it difficult or impossible to get a federally-backed mortgage to buy a house in specific neighborhoods coded as "risky" (red) which disproportionally affects minorities. 2. property value deductions based off of race statistics: 1937 Appraising Manual had "having foreign or racial neighbors" as a property value deduction 3. non-standardized taxation: as income-level rose, taxation decreased which resulted in regressive taxation

These issues all have underlying roots of racism and elitist ideals.

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 in Cook County placed a disproportionate tax burden on non-white property owners due to over assessment of lower valued properties.

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
from sklearn import model_selection

def train_test_split(data):
    data_len = data.shape[0]
    shuffled_indices = np.random.permutation(data_len)
    train_i = shuffled_indices[:int(data_len * 0.8)]
    test_i = shuffled_indices[int(data_len * 0.8):]
    train = data.iloc[train_i]
    test = data.iloc[test_i]
    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:

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

$$\geq$$

Since we are adding more features, the 2nd model is more complex than the 1st linear model and as complexity increases, the training loss either stays the same or decreases.

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]:
```python
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 `ohe_roof_material`. You are welcome to copy over your own implementations if you like.

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

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

[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]:
```
full_data["Building Square Feet"]
```

[14]:
```
0          1280.0
1           997.0
2           907.0
3          1174.0
4           949.0
            …
204787      910.0
204788     2005.0
204789      912.0
204790     1203.0
204791     1040.0
Name: Building Square Feet, Length: 204792, dtype: float64
```

[15]:
```
# DO NOT CHANGE THIS LINE
np.random.seed(1337)
# DO NOT CHANGE THIS LINE

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

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

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

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

[16]:
```
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).

[17]:
```
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.

[18]:
```
# Fit the 1st model
reg_1 = 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 = reg_1.predict(X_train_m1)
y_predicted_m1 = reg_1.predict(X_test_m1)

# Fit the 2nd model
reg_2 = 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 = reg_2.predict(X_train_m2)
y_predicted_m2 = reg_2.predict(X_test_m2)
```

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

[19]:
```
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.

[20]:
```
def rmse(predicted, actual):
    """
    Calculates RMSE from actual and predicted values
    Input:
      predicted (1D array): vector of predicted/fitted values
      actual (1D array): vector of actual values
    Output:
      a float, the root-mean square error
    """
    return np.sqrt(np.mean((actual - predicted)**2))
```

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

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

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

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

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

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

[21]:
```python
# 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
```

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

[22]:
```
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?

Theta 1 changes from positive to negative when we introduce an additional feature, Log Building Square Feet, in our 2nd model because the added feature is better for prediction than Bedrooms.

[23]:
```
# Parameters from 1st model
theta0_m1 = linear_model_m1.intercept_
theta1_m1 = linear_model_m1.coef_[0]

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

print("1st Model\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

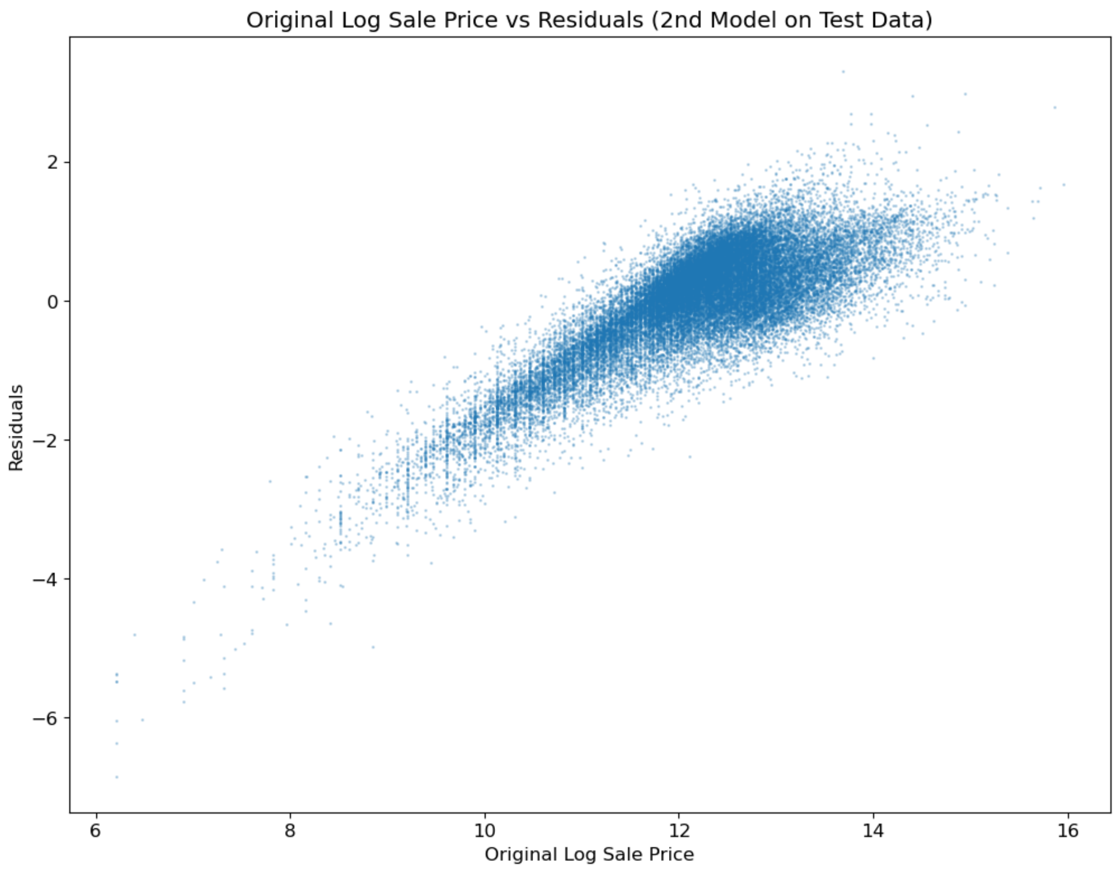
1.12.1 Question 7a

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.

```
[24]: residuals = (y_test_m2 - y_predicted_m2)
plt.scatter(y_test_m2, residuals, s = 1, alpha = 0.25)
plt.xlabel("Original Log Sale Price")
plt.ylabel("Residuals")
plt.title("Original Log Sale Price vs Residuals (2nd Model on Test Data)")
```

```
[24]: Text(0.5, 1.0, 'Original Log Sale Price vs Residuals (2nd Model on Test Data)')
```



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?

[25]:
```
q7b = "regressive"
```

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

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

While our simple model explains some of the variability in price, there is certainly still a lot of room for improvement to be made – one reason is we have been only utilizing 1 or 2 features (out of a total of 70+) so far! Can you engineer and incoporate 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 cound 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 pipline 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

[27]:
```
training_data.corr()["Sale Price"].sort_values(ascending = False)
```

[27]:
```
Sale Price                 1.000000
Estimate (Building)        0.609286
Estimate (Land)            0.523583
Building Square Feet       0.520472
Fireplaces                 0.395262
Pure Market Filter         0.314310
Latitude                   0.311347
Property Class             0.244005
Central Air                0.234717
Roof Material              0.195681
Cathedral Ceiling          0.194849
Garage 1 Size              0.182971
Design Plan                0.180211
Garage 1 Material          0.171696
Lot Size                   0.114456
Land Square Feet           0.114456
Most Recent Sale           0.104401
Wall Material              0.070290
Garage Indicator           0.070130
Garage 2 Material          0.046376
Attic Type                 0.040380
Garage 2 Attachment        0.034886
Other Heating              0.034196
Porch                      0.025360
```

```
Floodplain                     0.017337
Other Improvements             0.017241
Sale Half of Year              0.016788
Garage 2 Area                  0.016270
Sale Quarter of Year           0.013531
Sale Month of Year             0.011575
Central Heating                0.010067
Road Proximity                 0.008984
O'Hare Noise                   0.004408
Number of Commercial Units     0.004063
Garage 1 Area                  0.002316
Apartments                    -0.000399
Multi Code                    -0.002529
Multi Property Indicator      -0.008140
Garage 2 Size                 -0.024392
Attic Finish                  -0.036228
Basement                      -0.038668
Garage 1 Attachment           -0.042414
Deed No.                      -0.047138
Sale Half-Year                -0.047139
Sale Quarter                  -0.047404
Sale Year                     -0.049317
Site Desirability             -0.065321
Census Tract                  -0.066156
Repair Condition              -0.086370
Town Code                     -0.089018
Longitude                     -0.110532
Neighborhood Code             -0.118884
Neigborhood Code (mapping)    -0.118884
Basement Finish               -0.129076
Construction Quality          -0.132195
Town and Neighborhood         -0.144724
Age Decade                    -0.180765
Age                           -0.180765
PIN                           -0.299936
Use                                 NaN
Name: Sale Price, dtype: float64
```

[28]:
```
# 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
import re

%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 *

# Please include all of your feature engineering process inside this function.
# Do not modify the parameters of this function.

#1. choose top predictor columns (based off of my own intuition)
#2. remove outliers
#3. add column that has total number of rooms/bedrooms in house
#4. apply transformations to top 8 columns

# Define any additional helper functions you need here
# potential helper functions:

def add_total_rooms(data):
    with_rooms = data.copy()
    with_rooms["Total Rooms"] = with_rooms["Description"].str.
 ↪extract(r'[of]\s(\d+)\s[room]').replace(to_replace = "", value = 0).
 ↪astype(int)
    return with_rooms

def add_total_bedrooms(data):
    with_rooms = data.copy()
    with_rooms["Total Bedrooms"] = with_rooms["Description"].str.extract(r',␣
 ↪(\d+) of which are bedrooms').replace(to_replace = "", value = 0).astype(int)
    return with_rooms

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

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:

    data = add_total_rooms(data)
    data = add_total_bedrooms(data)

    # adding in other transformations
    data = log_transform(data, 'Land Square Feet')
    data = log_transform(data, 'Building Square Feet')
    data['Log Estimate (Land)'] = np.log(data['Estimate (Land)'] + 1)
    data['Log Estimate (Building)'] = np.log(data['Estimate (Building)'] + 1)
    data['Squared Total Rooms'] = np.square(data['Total Rooms'])
    data['Squared Total Bedrooms'] = np.square(data['Total Bedrooms'])

    if not is_test_set: # this is the set used to train the model
        # do your processing for the training set (i.e. not the test set)
        # this can involve references to sale price!
        data = remove_outliers(data, 'Sale Price', lower = data['Sale Price'].
 ↪quantile(q = 0.05), upper = data['Sale Price'].quantile(q = 0.95))
        training = log_transform(data, 'Sale Price')

    data = select_columns(data, 'Log Land Square Feet',
                          'Log Building Square Feet',
                          'Log Estimate (Land)',
                          'Log Estimate (Building)',
                          'Squared Total Rooms')

    # Return predictors and response variables separately
    if is_test_set:
        X = data.replace(np.nan, 0)
        return X

    else:
        y = training['Log Sale Price']
        X = data.replace(np.nan, 0)
        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: 132906.18683674803
```

[29]:
```
grader.check("q8")
```

[29]:
```
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!**

[30]:
```
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_2022-10-28T04:40:05.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? Brief 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.

First, I chose features that I thought would have high predictive power based off of my own intuition and then later on used .corr() to find the correlation between each variable and “Sale Price” when I realized that was a much more efficient method. I also used two helper functions to determine the total number of rooms and bedrooms in each house and saved it as a feature to use later. I then chose 4 highly correlated variables, applied transformations (logging very high values and squaring smaller values), and finally for the training set, I also removed outliers in the '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 valuates 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?

In the context of estimating the value of houses, for an individual homeowner, error would most likely mean the difference between the property value determined by the Cook County Assessor's Office and how much the house actually sold for. As discussed during lecture, if a property is overvalued, then the owner may need to pay higher taxes than they actually should and if a property is undervalued, then the owner would pay disproportionately low taxes.

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.

I believe that fairness in terms of property assessments and taxes would be a system that has a standardardized tax rate and that property values are determined by size, and the integrity of the property itself and not where it is located.

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 believe that the documentation is well organized and it is easy to navigate, but it would be difficult for a general audience to understand technical terms describing the model or the mathematics/statistical background behind the inner workings of the model.

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.

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

```
[31]: 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!**

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

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