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

# Project A2: Predicting Housing Prices in Cook County

## Due Date: Thursday, October 31th 11:59 PM PDT

You must submit this assignment to Gradescope by the on-time deadline, Thursday, October 31th, 11:59 PM. Please read the syllabus for the Slip Day policy. No late submissions beyond what is outlined in the Slip Day policy will be accepted. We strongly encourage you to plan to submit your work to Gradescope several hours before the stated deadline. This way, you will have ample time to reach out to staff for support if you encounter difficulties with submission. While course staff is happy to help guide you with submitting your assignment ahead of the deadline, we will not respond to last-minute requests for assistance (TAs need to sleep, after all!).

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

## Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the project, we ask that you **write your solutions individually**. If you discuss the assignments with others, please **include their names** below.

**Collaborators:** *list names here*

---

## Mid-Semester Survey

Please fill out the mid-semester survey here. We appreciate your feedback!

Note: The public test only checks if you enter the password in the correct format; it does not check if the password is correct.

```
In [69]:  qsurvey = "aGlkYXRhMTAw"
```

```
In [70]:  grader.check("qsurvey")
```

Out[70]:

**qsurvey** passed! 🎉

# Introduction

In Project A1, you performed some basic Exploratory Data Analysis (EDA), laying out the thought process that leads to certain modeling decisions. Then, you added a few new features to the dataset and cleaned the data in the process.

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

After this part of the project, you should be comfortable with:

- Implementing a data processing pipeline using `pandas`.
- Using `scikit-learn` to build and fit linear models.

# Score Breakdown

| Question | Manual | Points |
| --- | --- | --- |
| survey | No | 1 |
| 1a | Yes | 1 |
| 1b | Yes | 1 |
| 1c | No | 1 |
| 1d | Yes | 1 |
| 1e | Yes | 1 |
| 2 | No | 2 |
| 3a | No | 2 |

| Question | Manual | Points |
|---|---|---|
| 3b | No | 3 |
| 3c | No | 2 |
| 4a | Yes | 2 |
| 4b | No | 1 |
| 5a | No | 0 |
| 5b | No | 0 |
| 5c | No | 0 |
| 5d | No | 3 |
| 5e | No | 0 |
| 5f | No | 0 |
| 5g | No | 0 |
| 6a | No | 1 |
| 6b | No | 2 |
| 6c | Yes | 2 |
| 7a | Yes | 1 |
| 7b | Yes | 2 |
| Test Prediction | No | 3 |
| Total | 8 | 32 |

# Before You Start

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

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

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

# Debugging Guide

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

```python
In [71]: import numpy as np

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

         %matplotlib inline
         import matplotlib.pyplot as plt
         import seaborn as sns
         from sklearn import linear_model as lm

         import warnings
         warnings.filterwarnings("ignore")

         import zipfile
         import os

         from ds100_utils import *
         from feature_func import *

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

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

```python
In [72]: with zipfile.ZipFile('cook_county_data.zip') as item:
             item.extractall()
```

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

```python
In [73]: training_val_data = pd.read_csv("cook_county_train_val.csv", index_col='Unnamed: 0'
         test_data = pd.read_csv("cook_county_contest_test.csv", index_col='Unnamed: 0')
```

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

```python
In [74]: # 204792 observations and 62 features in training data
         assert training_val_data.shape == (204792, 62)
         # 55311 observations and 61 features in test data
         assert test_data.shape == (55311, 61)
         # Sale Price is provided in the training/validation data
         assert 'Sale Price' in training_val_data.columns.values
```

```
# Sale Price is hidden in the test data
assert 'Sale Price' not in test_data.columns.values
```

Let's remind ourselves of the data available to us in the Cook County dataset. Remember, a more detailed description of each variable is included in `codebook.txt`, which is in the same directory as this notebook.

In [75]: `training_val_data.columns.values`

Out[75]:
```
array(['PIN', 'Property Class', 'Neighborhood Code', 'Land Square Feet',
       'Town Code', 'Apartments', 'Wall Material', 'Roof Material',
       'Basement', 'Basement Finish', 'Central Heating', 'Other Heating',
       'Central Air', 'Fireplaces', 'Attic Type', 'Attic Finish',
       'Design Plan', 'Cathedral Ceiling', 'Construction Quality',
       'Site Desirability', 'Garage 1 Size', 'Garage 1 Material',
       'Garage 1 Attachment', 'Garage 1 Area', 'Garage 2 Size',
       'Garage 2 Material', 'Garage 2 Attachment', 'Garage 2 Area',
       'Porch', 'Other Improvements', 'Building Square Feet',
       'Repair Condition', 'Multi Code', 'Number of Commercial Units',
       'Estimate (Land)', 'Estimate (Building)', 'Deed No.', 'Sale Price',
       'Longitude', 'Latitude', 'Census Tract',
       'Multi Property Indicator', 'Modeling Group', 'Age', 'Use',
       "O'Hare Noise", 'Floodplain', 'Road Proximity', 'Sale Year',
       'Sale Quarter', 'Sale Half-Year', 'Sale Quarter of Year',
       'Sale Month of Year', 'Sale Half of Year', 'Most Recent Sale',
       'Age Decade', 'Pure Market Filter', 'Garage Indicator',
       'Neigborhood Code (mapping)', 'Town and Neighborhood',
       'Description', 'Lot Size'], dtype=object)
```

# Question 1: Human Context and Ethics

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

## Question 1a

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

1. An individual home owner: This depends entirely on a person's income. If one does not earn too much (but owns a home) and they intent to stay at that location for sufficinetly

long period of time, then they would want the price of their home to not be over-estimated (until they are ready to sell their home), as that would translate to higher property taxes. If one does generate a lot of income and/or is looking to sell or rent thier home, then they would want the price of their home to be evaluated greater than what is already is because they can demand a higher selling price or have more equity.

2. Law Firms: This depends on the party and the context. In particular, a homeowner is incentivised to sell their home for the highest price possible and therefore if there is an intermediatry party such as a law firm, then the firm would like to negotiate for a higher evaluation price (both the firm and seller share the same incentive). On the other hand, if the home owner wants to minimze property taxes, then the firm can argue for a lower selling/home evaluation price.

3. County Office (Local Tax Assessor): Ideally they should lean moderately or for a slightly higher home evaluation (depending on the neighborhood, proximity to road, quality of schools, home charactersitics, etc). Having a "fair" approach (this is explaind in later sections) ensures each home is accurately assessed, each person pays an amount proportional to their income, and in general this minimizes disputes over home prices.

---

## Question 1b

Which of the following scenarios strike you as unfair, and why? You can choose more than one. There is no single right answer, but you must explain your reasoning. Would you consider some of these scenarios more (or less) fair than others? Why?

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

All of the above (depending on the context). With regard to scenario (A), if a person intends on staying at their home for some duration of time and it's assessed at a higher price than what it would sell for, then that (generally) means they pay more property taxes. If said person does not earn a lot, then this is unfair. Likewise, if the person earn a lot then it's also unfair as they have a tax break and people who earn less have to pay disproportionately more in taxes. Similar reasoning can be applied to (B - D). This is also described in further context above.

---

## Question 1c

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

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

![No description has been provided for this image]

```
In [76]:  q1c = 'B'
```

```
In [77]:  grader.check("q1c")
```

Out[77]:

**q1c** passed! 🙌

# The CCAO Dataset

You'll work with the dataset from the Cook County Assessor's Office (CCAO) in Illinois. This government institution determines property taxes across most of Chicago's metropolitan areas and nearby suburbs. In the United States, all property owners must pay property taxes, which are then used to fund public services, including education, road maintenance, and sanitation. These property tax assessments are based on property values estimated using statistical models considering multiple factors, such as real estate value and construction cost.

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

The Chicago Tribune's four-part series, "The Tax Divide," delves into how this was uncovered. After "compiling and analyzing more than 100 million property tax records from the years 2003 through 2015, along with thousands of pages of documents, then vetting the findings

with top experts in the field," they discovered that "residential assessments had been so far off the mark for so many years." You can read more about their investigation here.

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

---

## Question 1d

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

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

The primary cause of these problems (in Cook County for example) was flawed assesment practices which were used since 1930s (which albiet had some adjustments) but home prices were evaluated such that those who came from non-white communities had to pay disproportionately more in property taxes than their white counter-parts and were systemically effected by such a system. In fact, racial bias was built into their flawed statisitcal models for estimating the home price. This also caused segregation of white vs non-white communities (discussed below).

---

## Question 1e

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

In addition to being regressive, the propety tax system in Cook County placed a disproportionate tax burden on non-white property owners because (as analysis is shown and what will be confirmed below) is that home prices were consistently over or under evaluated. In particular, neighborhoods with predominately white homeowners had under-evaluated home prices and therefore paid less in taxes than their non-white counterparts (who in general) earned less but with an over-evaluated home price, therefore paying disproportioantely more in taxes. An in direct implication of this is segregation: where communities with lower home price evaluations vs higher home price evaluations had different (more or less) concentrations of white vs non-white homeowners.

# Question 2: Preparing Data

Let's split the dataset into a training set and a validation set. We will use the training set to fit our model's parameters and the validation set to evaluate how well our model will perform on unseen data drawn from the same distribution. If we used all the data to fit our model, we would not have a way to estimate model performance on **unseen data** such as the test set in `cook_county_contest_test.csv` .

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

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

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

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

```python
In [78]:  # This makes the train-validation split in this section reproducible across differe
          # of the notebook. You do not need this line to run train_val_split in general.

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

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

              data_len = data.shape[0]
              shuffled_indices = np.random.permutation(data_len)

              # define two numpy arrrays to split first 80% (training) and remaining 20% for
              # make sure the indices are integers (so concat result)
              train_indices = shuffled_indices[: int(data.shape[0] * 0.8)]
              validation_indices = shuffled_indices[int(data.shape[0] * 0.8) :]

              train = data.iloc[train_indices]
```

```
    validation = data.iloc[validation_indices]

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

In [79]: `grader.check("q2")`

Out[79]:
**q2** passed! 🍀

---

## Question 3: Fitting a Simple Model

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

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

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

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

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

---

## Question 3a

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

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

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

In [80]: `q3a = '<='`

In [81]: `grader.check("q3a")`

**q3a** passed! 🎉

## Pipeline Function

In Project A1, you wrote a few functions that added features to the dataset. Instead of calling them manually one by one each time, it is best practice to encapsulate all of this feature engineering into one "pipeline" function. Defining and using a pipeline reduces all the feature engineering to just one function call and ensures that the same transformations are applied to all data. Below, we combined some functions into a single helper function that outputs `X` and `Y` for the first model above. Try to understand what this function does!

**Note 1:** We have automatically imported staff implementations of the functions you wrote in Project A1. These functions are `remove_outliers`, `add_total_bedrooms`, `find_expensive_neighborhoods`, `add_in_expensive_neighborhood`, and `ohe_roof_material`. You are welcome to copy over your own implementations if you would like.

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

```python
In [82]:   from feature_func import *      # Import functions from Project A1

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


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

           # Reload the data
           full_data = pd.read_csv("cook_county_train.csv")

           # Process the data using the pipeline for the first model.
```

```
np.random.seed(1337)
train_m1, valid_m1 = train_val_split(full_data)
X_train_m1_simple, Y_train_m1_simple = feature_engine_simple(train_m1)
X_valid_m1_simple, Y_valid_m1_simple = feature_engine_simple(valid_m1)

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

|  | Bedrooms |
|---|---|
| **130829** | 4 |
| **193890** | 2 |
| **30507** | 2 |
| **91308** | 2 |
| **131132** | 3 |

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

## `.pipe`

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

The following function `feature_engine_pipe` takes in a `DataFrame` `data`, a list `pipeline_functions` containing 3-element tuples `(function, arguments, keyword_arguments)` that will be called on `data` in the pipeline, and the label `prediction_col` that represents the column of our target variable (`Sale Price` in this case). You can use this function with each of the tuples passed in through `pipeline_functions`.

```
In [83]:  # Run this cell to define feature_engine_pipe; no further action is needed.
          def feature_engine_pipe(data, pipeline_functions, prediction_col):
              """Process the data for a guided model."""
              for function, arguments, keyword_arguments in pipeline_functions:
                  if keyword_arguments and (not arguments):
                      data = data.pipe(function, **keyword_arguments)
                  elif (not keyword_arguments) and (arguments):
                      data = data.pipe(function, *arguments)
                  else:
                      data = data.pipe(function)
              X = data.drop(columns=[prediction_col])
              Y = data.loc[:, prediction_col]
              return X, Y
```

# Question 3b

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

- Perform a `train_val_split` on the original dataset, loaded as the `DataFrame` `full_data`. Let 80% of the set be training data, and 20% of the set be validation data.
- For both the training and validation set,
    1. Remove outliers in `Sale Price` so that we consider households with a price that is greater than 499 dollars (or equivalently, a price that is 500 dollars or greater).
    2. Apply log transformations to the `Sale Price` and the `Building Square Feet` columns to create two new columns, `Log Sale Price` and `Log Building Square Feet`.
    3. Extract the total number of bedrooms into a new column `Bedrooms` from the `Description` column.
    4. Select the columns `Log Sale Price` and `Bedrooms` (and `Log Building Square Feet` if this is the second model). We have implemented the helper function `select_columns` for you.
    5. Return the design matrix $\mathbb{X}$ and the observed vector $\mathbb{Y}$. Note that $\mathbb{Y}$ refers to the transformed `Log Sale Price`, not the original `Sale Price`. **Your design matrix and observed vector should be `NumPy` arrays or `pandas` `DataFrame`s**.

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

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

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

As a refresher, the equations model 1 and model 2, respectively, are: $$ \text{Log Sale Price} = \theta_0 + \theta_1 \cdot (\text{Bedrooms}) $$

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

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

```
In [84]:  # Reload the data
          full_data = pd.read_csv("cook_county_train.csv")

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

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

          # Pipelines, a list of tuples
          m1_pipelines = [
              (remove_outliers, None, {
                  'variable': 'Sale Price',
                  'lower': 499,
              }),
              (log_transform, None, {'col': 'Sale Price'}),
              (add_total_bedrooms, None, None),
              (select_columns, ['Log Sale Price', 'Bedrooms'], None)
          ]

          X_train_m1, Y_train_m1 = feature_engine_pipe(train_m1, m1_pipelines, 'Log Sale Pric
          X_valid_m1, Y_valid_m1 = feature_engine_pipe(valid_m1, m1_pipelines, 'Log Sale Pric

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

| | Bedrooms |
|---|---|
| **130829** | 4 |
| **193890** | 2 |
| **30507** | 2 |
| **91308** | 2 |
| **131132** | 3 |

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

In [85]:
```
# DO NOT CHANGE THIS LINE
np.random.seed(1337)
# DO NOT CHANGE THIS LINE

# Process the data using the pipeline for the second model
train_m2, valid_m2 = train_val_split(full_data)

m2_pipelines = [
    (remove_outliers, None, {'variable': 'Sale Price', 'lower': 499}),
    (log_transform, None, {'col': 'Sale Price'}),
    (log_transform, None, {'col': 'Building Square Feet'}),
    (add_total_bedrooms, None, None),
    (select_columns, ['Log Sale Price', 'Bedrooms', 'Log Building Square Feet'], No
]

X_train_m2, Y_train_m2 = feature_engine_pipe(train_m2, m2_pipelines, 'Log Sale Pric
X_valid_m2, Y_valid_m2 = feature_engine_pipe(valid_m2, m2_pipelines, 'Log Sale Pric

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

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

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

In [86]:
```
grader.check("q3b")
```

Out[86]:

**q3b** passed! 🚀

## Question 3c

Finally, let's do some regression!

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

```
In [87]:  linear_model_m1 = lm.LinearRegression(fit_intercept=True)
          linear_model_m2 = lm.LinearRegression(fit_intercept=True)
```

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

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

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

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

```
In [88]:  # Fit the 1st model
          linear_model_m1.fit(X_train_m1, Y_train_m1)
          # Compute the fitted and predicted values of Log Sale Price for 1st model
          Y_fitted_m1 = linear_model_m1.predict(X_train_m1)
          Y_predicted_m1 = linear_model_m1.predict(X_valid_m1)

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

```
In [89]:  grader.check("q3c")
```

```
Out[89]:
```

**q3c** passed! 🌈

---

# Question 4: Evaluate Our Simple Model

---

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

```python
In [90]: def rmse(predicted, actual):
             """
             Calculates RMSE from actual and predicted values.
             Input:
               predicted (1D array): Vector of predicted/fitted values
               actual (1D array): Vector of actual values
             Output:
               A float, the RMSE value.
             """
             return np.sqrt(np.mean((actual - predicted)**2))
```

---

# Question 4a

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

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

```python
In [91]: plt.scatter(Y_valid_m2, Y_valid_m2 - Y_predicted_m2, s = 20, alpha = 0.5)
         plt.title('Residual Plot: Log Sale Price')
         plt.xlabel('Log Sale Price')
         plt.ylabel('Residual: y - hat(y)')
```

```
Out[91]: Text(0, 0.5, 'Residual: y - hat(y)')
```

Residual Plot: Log Sale Price
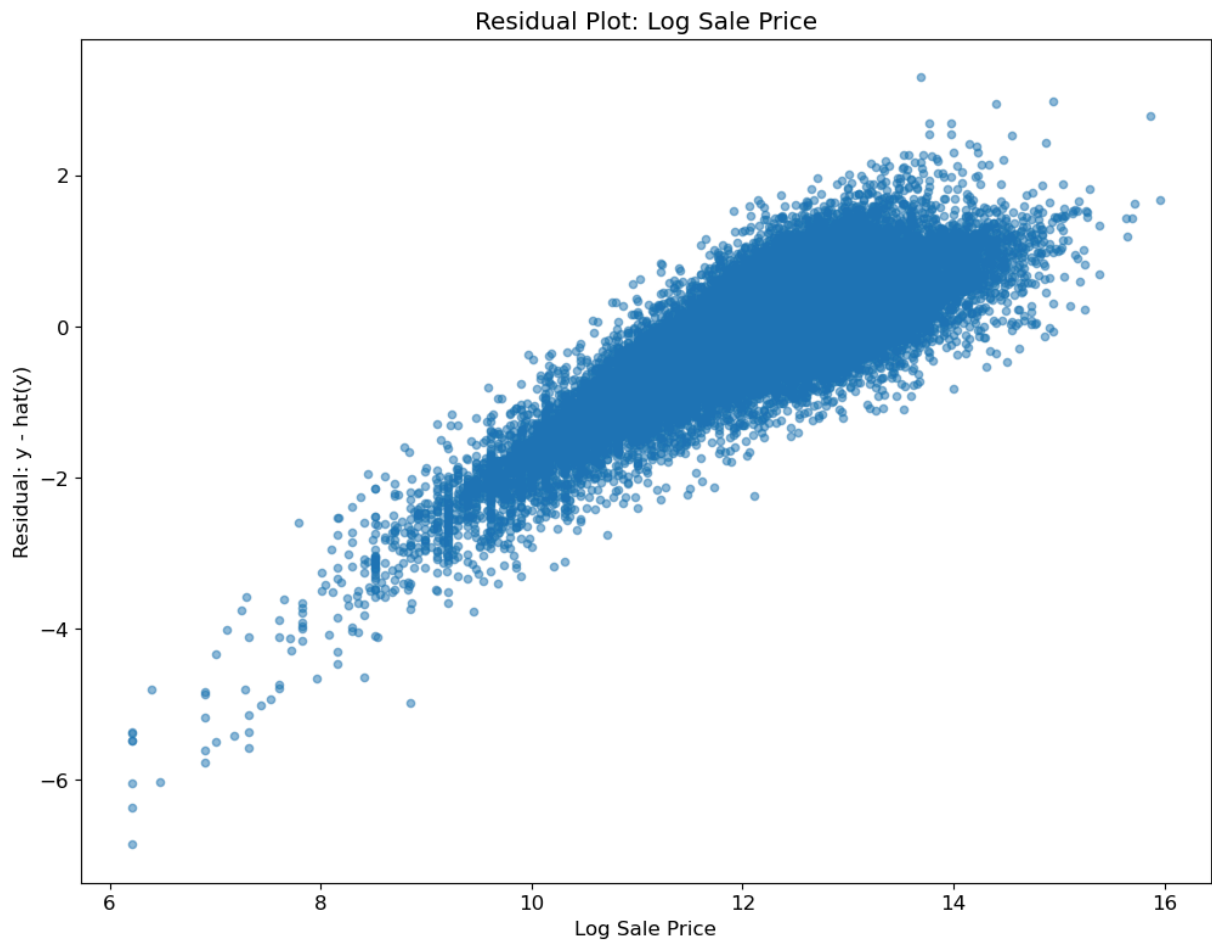
## Question 4b

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

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

```
In [92]:  q4b = "progressive"
```

```
In [93]:  grader.check("q4b")
```

Out[93]:

**q4b** passed! 💯

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

# Question 5

It is time to build your own model!

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

Your goal in Question 5 is to:

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

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

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

## Grading Scheme

Your grade for Question 5 will be based on your model's RMSE when making predictions on the training set, as well as your model's RMSE when making predictions on the test set. The

tables below provide scoring guidelines. If your RMSE lies in a particular range, you will receive the number of points associated with that range.

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

The thresholds are as follows:

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

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

---

# Some notes before you start

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

- You MUST remove any additional new cells you add before submitting to Gradescope to avoid any autograder errors.
- **You can only submit your test set prediction CSV file to Gradescope up to 4 times per day. Start early!** In the case that you are approved for an extension, you will be granted 4 more submissions for each day the deadline has been extended.

**PLEASE READ THE ABOVE MESSAGE CAREFULLY!**

In [94]:
```python
# The 3 lines below to clean up memory from previous questions and reinitialize Ott
# If you want to refer to any functions or variables you defined at any point earli
# Place them in the cell under Question 5c so that you can access them after the me
# If you think you will not run into any memory issues, you are free to comment out

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

# Imports all the necessary libraries again

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

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model as lm

import warnings
warnings.filterwarnings("ignore")

import zipfile
import os

from ds100_utils import *
from feature_func import *

from sklearn.preprocessing import OneHotEncoder
```

---

# Question 5a: Finding Potential Features

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

# Question 5b: More EDA

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

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

In [95]:
```python
# Add any EDA code below
#data_corr = training_val_data.corr()
#corr_sp = data_corr['Sale Price'].abs.sort_values(ascending = False)
#print(corr_sp)

data = training_val_data.copy()
# extract numerical data
filtered_data = data.select_dtypes(include = ['int64', 'float64']).columns.tolist()
# filter numerical data
transform_filtered = data[filtered_data].fillna(data[filtered_data].median())
corr_data = transform_filtered.corr()['Sale Price'].drop('Sale Price')
# set coorelation threshold to determine which columns are best correlated w/Sales
tol = 0.3
# extract all data based on threshold
rel_data = corr_data[corr_data.abs() > tol].index.tolist()
print(rel_data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[95], line 6
      1 # Add any EDA code below
      2 #data_corr = training_val_data.corr()
      3 #corr_sp = data_corr['Sale Price'].abs.sort_values(ascending = False)
      4 #print(corr_sp)
----> 6 data = training_val_data.copy()
      7 # extract numerical data
      8 filtered_data = data.select_dtypes(include = ['int64', 'float64']).columns.t
olist()

NameError: name 'training_val_data' is not defined
```

# Question 5c: Defining Helper Function or Helper Variables

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

In [96]:
```python
# Define any additional helper functions or variables you need here
ohe_enc = OneHotEncoder()

# define a function ohe_column that one-hot encodes for any particular column
# the purpose for this is: previously there were probelms with the matrix conversio
# there were/could have been cases where it found additional features in the test s
# in the training set. hence we included this third parameter
def ohe_column(data, columns, fit = False):
    # fit encoder for specified columns
    # otherwise transform the data as needed
    if fit:
        enc_data = ohe_enc.fit_transform(data[columns])
    else:
        enc_data = ohe_enc.transform(data[columns])

    # used similar structure/formatting to convert sparse matrix to numpy array
    enc_df = pd.DataFrame(enc_data.toarray(), columns = ohe_enc.get_feature_names_o
    return data.drop(columns = columns).join(enc_df)

# define function clean_data to clean/filter out certain values
def clean_data(data, columns):
    for c in columns:
        # ensure we are working with numerical variables
        if data[c].dtype in ['int64', 'float64']:
            # if entry is missing, impute with median
            data[c] = data[c].fillna(data[c].median())
            # replace infinite values with nan; then fill with 1 to ensure no more
            data[c] = data[c].replace([np.inf, -np.inf], np.nan)
            data[c] = data[c].fillna(1)
            # ensure all values are strictly positive; if not replace with 1
            data[c] = data[c].apply(lambda val: val if val > 0 else 1)
        else:
            # otherwise replacing missing values with mode
            data[c] = data[c].fillna(data[c].mode()[0])
    return data
```

# Question 5d: Defining The Pipeline Function

Just as in the guided model from the previous question, you should encapsulate as much of your workflow into functions as possible. Your job is to select better features and define your own feature engineering pipeline inside the function `feature_engine_final` in the following cell. Use of `.pipe` is not required, but you are welcome to incorporate it! **You must not change the parameters inside `feature_engine_final`. Do not edit the two**

**lines at the end of the question cell below. They are helper functions that define a linear model, fit your data, and compute RMSE. If you do, you will receive no credit for this question.**

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

**Hints:**

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

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

```
In [97]:   # Please include all of your feature engineering processes inside this function.
           # Do not modify the parameters of this function.
           def feature_engine_final(data, is_test_set = False):
               # Whenever you access 'Log Sale Price' or 'Sale Price', make sure to use the
               # condition is_test_set like this:
               if not is_test_set:
                   # Processing for the training set (i.e. not the test set)
                   # CAN involve references to sale price!
                   # CAN involve filtering certain rows or removing outliers
                   data = remove_outliers(data, 'Sale Price', lower = 499)
                   data['Log Sale Price'] = np.log(data['Sale Price'])

                   # define feature space
                   features = ['Building Square Feet', 'Estimate (Land)', 'Estimate (Building)',
                               'Lot Size', 'Fireplaces', 'Age', 'Road Proximity',
                               'Repair Condition', 'Property Class', 'Neighborhood Code']
```

```python
    # clean and transform data as needed
    data = clean_data(data, features)

    # apply log transformation to select features
    for i in range(0, 4):
        data['Log' + features[i]] = np.log(data[features[i]])
    for i in range(0, 4):
        data = data.drop(columns = [features[i]])
        features[i] = 'Log' + features[i]

    # define subset of feature space (categorical variables) to apply OHE
    cat_features = ['Repair Condition', 'Property Class', 'Neighborhood Code']
    if not is_test_set:
        data = ohe_column(data, cat_features, fit=True)
    else:
        data = ohe_column(data, cat_features, fit=False)

    # remove certain elements from original features and then
    # add the OHE as defined above
    features = [f for f in features if f not in cat_features]
    features.extend(ohe_enc.get_feature_names_out(cat_features))

    # Processing for both test and training set
    # CANNOT involve references to sale price!
    # CANNOT involve removing any rows
    # we should impute missing values with median since average is sensitive to out
    # data['Lot Size'] = data['Lot Size'].fillna(data['Lot Size'].median())
    # data['Road Proximity'] = data['Lot Size'].fillna(data['Lot Size'].median())
    # data['Land Square Feet'] = data['Land Square Feet'].fillna(data['Land Square
    # data['Building Square Feet'] = data['Building Square Feet'].fillna(data['Buil

    # apply transformations
    # referenced: https://stackoverflow.com/questions/49575897/cant-replace-0-to-na
    # data['Log Land Square Feet'] = np.log(data['Land Square Feet'])
    # data['Log Lot Size'] = np.log(data['Building Square Feet'])
    # data['Log Building Square Feet'] = np.log(data['Building Square Feet'])

    # Create Bedroom column
    # data = add_total_bedrooms(data)
    # One-Hot Encode Features
    #data = ohe_column(data, 'Garage Indicator')
    # data = ohe_column(data, 'Property Class')
    # data = ohe_column(data, 'Construction Quality')
    # data = ohe_column(data, 'Age')
    # data = ohe_column(data, 'Repair Condition')

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

```
        return X, Y

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

Current training RMSE: 261951.12548957177
You can check your grade for your prediction as per the grading scheme outlined at t
he start of Question 5

In [98]: 
```
grader.check("q5d")
```

**q5d results:**

**q5d - 1 result:**

❌ Test case failed

Trying:
    bool(check_rmse_threshold(200000))
Expecting:
    True
**********************************************************************
Line 1, in q5d 0
Failed example:
    bool(check_rmse_threshold(200000))
Expected:
    True
Got:
    False

**q5d - 2 result:**

❌ Test case failed

Trying:
    bool(check_rmse_threshold(240000))
Expecting:
    True
**********************************************************************
Line 1, in q5d 1
Failed example:
    bool(check_rmse_threshold(240000))
Expected:
    True
Got:
    False

**q5d - 3 result:**

✅ Test case passed

**q5d - 4 result:**

✅ Test case passed

# Question 5e: Fit and Evaluate your Model

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

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

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

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

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

```
In [99]:  def rmse(predicted, actual):
              """
              Calculates RMSE from actual and predicted values.
              Input:
                predicted (1D array): Vector of predicted/fitted values
                actual (1D array): Vector of actual values
              Output:
                A float, the RMSE value.
              """
              return np.sqrt(np.mean((actual - predicted)**2))
```

▶ [**Click to Expand**] Hints:

```
In [100…  # Use this space to evaluate your model
          # if you reset your memory, you need to define the functions again
```

---

# Question 5f Submission

Recall that the test set given to you in this assignment does not contain values for the true `Sale Price` of each house. You will be predicting `Log Sale Price` on the data stored in `cook_county_contest_test.csv`. To determine your model's RMSE on the test set, you will submit the predictions made by your model to Gradescope. There, we will run checks to see what your test RMSE is by considering (hidden) true values for the `Sale Price`. We will delog/exponentiate your prediction on Gradescope to compute RMSE and use this to score

your model. Before submitting to Gradescope, make sure that your predicted values can all be delogged (i.e., if one of your `Log Sale Price` predictions is 60, it is too large; $e^{60}$ is too big!)

Your score on this section will be determined by the grading scheme outlined at the start of Question 5. **Remember that you can only submit your test set predictions to Gradescope up to 4 times per day. Plan your time to ensure that you can adjust your model as necessary, and please test your model's performance using cross-validation before making any submissions.** For more on cross-validation, check Lecture 16. In particular, the Lecture 16 notebook may be helpful here. **Furthermore, feel free to use the cross validation implementation done in Lab 8 to test your model.** You can also reference what you did in previous questions when creating training and validation sets and seeing how your model performs.

To determine the error on the test set, please submit your predictions on the test set to the Gradescope assignment **Project A2 Test Set Predictions**. The CSV file to submit is generated below, and you should not modify the cell below. Simply download the CSV file, and submit it to the appropriate Gradescope assignment.

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

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

```
In [101… from datetime import datetime
         from IPython.display import display, HTML

         Y_test_pred = run_linear_regression_test(lm.LinearRegression(fit_intercept=True), f
                                                  is_test = True, is_ranking = False, return
                                                  )

         # Construct and save the submission:
         submission_df = pd.DataFrame({
             "Id": pd.read_csv('cook_county_contest_test.csv')['Unnamed: 0'],
             "Value": Y_test_pred,
         }, columns=['Id', 'Value'])
         timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
         filename = "submission_{}.csv".format(timestamp)
         submission_df.to_csv(filename, index=False)

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

Download your test prediction here.
  You may now upload this CSV file to Gradescope for scoring.

```
In [102… # Scratch space to check if your prediction is reasonable. See 5e for hints.
         # We will not reset the submission count for mis-submission issues.
```

```
submission_df["Value"].describe()
```

Out[102...
```
count    55311.000000
mean        12.188446
std          0.777232
min          3.159515
25%         11.702271
50%         12.128164
75%         12.625053
max         16.314497
Name: Value, dtype: float64
```

Congratulations on finishing your prediction model for home sale prices in Cook County! In the following section, we'll delve deeper into the implications of predictive modeling within the CCAO case study, especially because statistical modeling is how the CCAO valuates properties.

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

---

## Question 6: Exploring RMSE

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

---

In [103...
```python
# Run the cell below; no further action is needed
train_df = pd.read_csv('cook_county_train.csv')
X, Y_true = feature_engine_final(train_df)
model = lm.LinearRegression(fit_intercept=True)
model.fit(X, Y_true)
Y_pred = model.predict(X)
```

In [104...
```python
preds_df = pd.DataFrame({'True Log Sale Price' : Y_true, 'Predicted Log Sale Price'
                         'True Sale Price' : np.e**Y_true, 'Predicted Sale Price' :
preds_df.head()
```

| | True Log Sale Price | Predicted Log Sale Price | True Sale Price | Predicted Sale Price |
|---|---|---|---|---|
| **1** | 12.560244 | 11.652607 | 285000.0 | 114990.749400 |
| **2** | 9.998798 | 11.044170 | 22000.0 | 62578.082729 |
| **3** | 12.323856 | 11.528904 | 225000.0 | 101610.683502 |
| **4** | 10.025705 | 10.992432 | 22600.0 | 59422.703173 |
| **6** | 11.512925 | 11.448208 | 100000.0 | 93733.211946 |

---

## Question 6a

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

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

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

```
The lower interval contains houses with true sale price $493.0 to $219696.0
The higher interval contains houses with true sale price $219696.0 to $72565488.0
```

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

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

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

```
In [106... rmse_cheap = rmse(cheap_df['Predicted Sale Price'], cheap_df['True Sale Price'])
          rmse_expensive = rmse(expensive_df['Predicted Sale Price'], expensive_df['True Sale

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

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

The RMSE for properties with log sale prices in the interval (6.2, 12.3) is 100258.0
The RMSE for properties with log sale prices in the interval (12.3, 18.1) is 359319.
0

The percentage of overestimated values for properties with log sale prices in the in
terval (6.2, 12.3) is 60.05%
The percentage of overestimated values for properties with log sale prices in the in
terval (12.3, 18.1) is 27.34%

In [107... `grader.check("q6a")`

Out[107... **q6a** passed! 🌈

---

## Question 6b

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

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

```
In [108... def rmse_interval(df, start, end):
              '''
              Given a design matrix X and response vector Y, computes the RMSE for a subset o
              wherein the corresponding Log Sale Price lies in the interval [start, end].

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

```python
    subset_df = df[(df['True Log Sale Price'] >= start) & (df['True Log Sale Price']

    rmse_subset = np.sqrt(np.mean((subset_df['True Sale Price'] - subset_df['Predic
    return rmse_subset

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

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

    subset_df = subset_df = df[(df['True Log Sale Price'] >= start) & (df['True Log

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

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

In [109… `grader.check("q6b")`

Out[109…

**q6b** passed! 🙌

---

## Question 6c

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

In [110…
```python
# RMSE plot
plt.figure(figsize = (8,5))
plt.subplot(1, 2, 1)
rmses = []
for i in np.arange(8, 14, 0.5):
    rmses.append(rmse_interval(preds_df, i, i + 0.5))
plt.bar(x = np.arange(8.25, 14.25, 0.5), height = rmses, edgecolor = 'black', width
plt.title('RMSE Over Different Intervals\n of Log Sale Price', fontsize = 10)
plt.xlabel('Log Sale Price')
plt.yticks(fontsize = 10)
plt.xticks(fontsize = 10)
plt.ylabel('RMSE')
```

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

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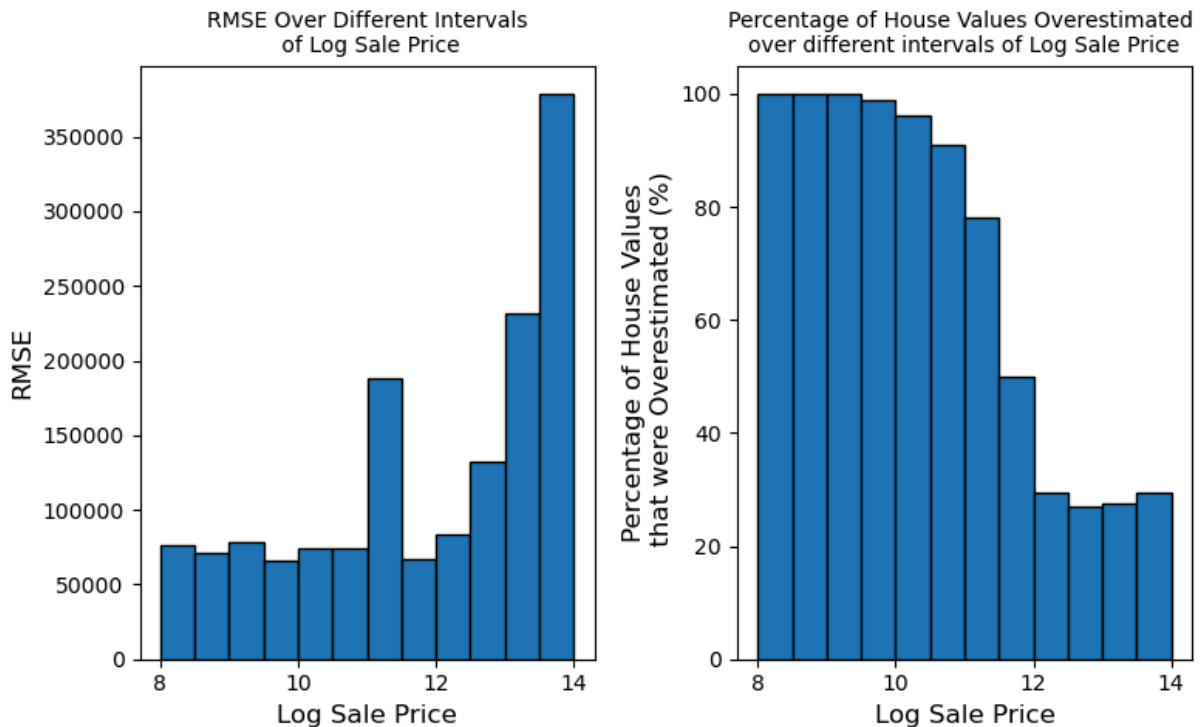


Explicitly referencing **ONE** of the plots above (using `props` and `rmses` ), explain whether the assessments your model predicts more closely aligns with scenario C or scenario D that we discussed back in `q1b` . Which of the two plots would be more useful in ascertaining whether the assessments tended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

```
C. An assessment process that systematically overvalues inexpensive
properties and undervalues expensive properties.
D. An assessment process that systematically undervalues
inexpensive properties and overvalues expensive properties.
```

Scenario (C) e.g., an assessment process that systematically overvalues inexpensive properties and undervalues expensive properties. Looking at the graph to the right (props),

we can see that lower values (of sales price) tend to be overestimated by a large margin compared to homes with a higher value.

---

# Question 7: Evaluating the Model in Context

---

## Question 7a

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

In context for estimating the value of houses, the residual is defined as: resid = true home price - predicited home price. The residual value is correlated to the property tax/mortgage for a home. In particular, if residual > 0, then true home price > predicted home price (so the home is under-estimated). This means the homeowner would pay less in propety taxes than they would normally which contributes to a tax beneift. Under a progressive tax system, wealtheir homeowners who have a positve residual would not pay their (home) taxes that is proportional to their income.

On the other hand if residual < 0, then the true home price < predicited home price (so the home is over-estimated). This means the homeowner pays more in property taxes compared to what their home is actually worth (and is a financial burden). This would lead to a regressive system in which homeowners who don't earn as much are paying an amount that is also not proportional to their income (which in this context is a net-negative).

In both cases, homeowners (in general) are not paying an amount in proerty taxes that's proportional to their incomes which can be a net-positive gain or a net-negative depending socio-economic factors.

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

When the use of algorithms and statistical modeling has real-world consequences, we often refer to the idea of fairness as a measurement of how socially responsible our work is.

Fairness is incredibly multifaceted: Is a fair model one that minimizes loss - one that generates accurate results? Is it one that utilizes "unbiased" data? Or is fairness a broader goal that takes historical contexts into account?

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

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

---

# Question 7b

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

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

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

Possible definition of "fairness": The evaluation (predicted) home price of a is strongly coorelated with an indivual's income. In particular, consider an estimator $\theta$ that predicts home prices/market evaluation. If Bias($\theta$) = 0, then the system (whichever metrics Cook's County Office for example, uses) systematically does not overvalue or undervalue a home and therefore the homeowner pays an amount that is porportional to their income which is what should happen if the system is truely fair.

An RMSE is a metric that allows us to quantify the average difference between the predicted and observed values. If the RMSE is low, then on average, the prediction(s) of the model is close to the true (observed) values (home evaluation). This indicates less error in the

predictions. However, we don't know if the model's predictions are consistently related (close to) the observed values. Also, if it's the case the residuals 'cancel' out (and have a low RMSE) for estimated home values, we don't know necessairly how accurate the model is on average (and thus RMSE alone should not be the only indicator) e.g. there could still be some underlying systemic bias which we don't know based on this calcualtion alone. My answers to the previous two qeustions remained consistent, which was there is underlying systemic bias (or at least some indication of it) and homeowners with lower (actual) sales prices conistently are over-evlaluated and thus have to deal with a heavier tax burden.

---

# Ayga and Beck congratulate you on finishing Project A2!
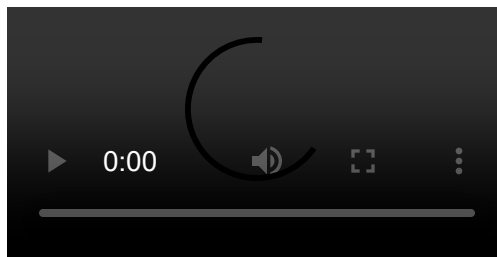


## Course Content Feedback

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

## Submission Instructions

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

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

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

- **Project A2 Coding:** Submit the zip file generated by using the `grader.export()` cell provided below.
- **Project A2 Written:** Gradescope will automatically submit the PDF from the zip file submitted earlier. You do not need to submit anything to this assignment yourself, but *please check that the submission went through properly and that all plots rendered correctly*.
- **Project A2 Test Set Predictions:** Submit the CSV file generated in `q5f`.

**You are responsible for ensuring your submission follows our requirements and that everything was generated and submitted correctly. We will not be granting regrade requests nor extensions to submissions that don't follow instructions.** If you encounter any difficulties with submission, please don't hesitate to reach out to staff prior to the deadline.

## Submission

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

In [112... 
```
# Save your notebook first, then run this cell to export your submission.
grader.export(run_tests=True)
```

Running your submission against local test cases...

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

    qsurvey results: All test cases passed!

    q1c results: All test cases passed!

    q2 results: All test cases passed!

    q3a results: All test cases passed!

    q3b results: All test cases passed!

    q3c results: All test cases passed!

    q4b results: All test cases passed!

    q5d results:
        q5d - 1 result:
           ❌ Test case failed
           Trying:
               bool(check_rmse_threshold(200000))
           Expecting:
               True
           **********************************************************************
           Line 1, in q5d 0
           Failed example:
               bool(check_rmse_threshold(200000))
           Expected:
               True
           Got:
               False

        q5d - 2 result:
           ❌ Test case failed
           Trying:
                bool(check_rmse_threshold(240000))
           Expecting:
                True
            **********************************************************************
           Line 1, in q5d 1
            Failed example:
                bool(check_rmse_threshold(240000))
            Expected:
                True
            Got:
                False

        q5d - 3 result:
           ✅ Test case passed

        q5d - 4 result:
           ✅ Test case passed

```
q6a results: All test cases passed!

q6b results: All test cases passed!
```
Your submission has been exported. Click here to download the zip file.

In [ ]: