```
    named `train` and `validation` with 80% and 20% of the data, respectively.
    """

    data_len = data.shape[0]


    train_indices = shuffled_indices[:pointeight]



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

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:

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

---

[13]:

```python
from feature_func import *    # Import functions from Project A1

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



def feature_engine_simple(data):
    # Remove outliers

    # Create Log Sale Price column

    # Create Bedroom column

    # Select X and Y from the full data

    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.
```

```python
np.random.seed(1337)
train_m1, valid_m1
X_train_m1_simple,
X_valid_m1_simple,

# 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
```

```
# Helper function
def select_columns(data, *columns):
    """Select only columns passed as arguments."""


# Pipelines, a list of tuples
m1_pipelines = [
    (remove_outliers, None, {


    }),
    (log_transform,
    (add_total_bedrooms,
    (select_columns, ['L
]

X_train_m1, Y_train_m1

X_valid_m1, Y_valid_m1 =


# 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
```

```
m2_pipelines =
    }),
    (log_transform,
]

X_train_m2, Y_train_m2 =
 ↪Price')
X_valid_m2, Y_valid_m2 =
 ↪Price')


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

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

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

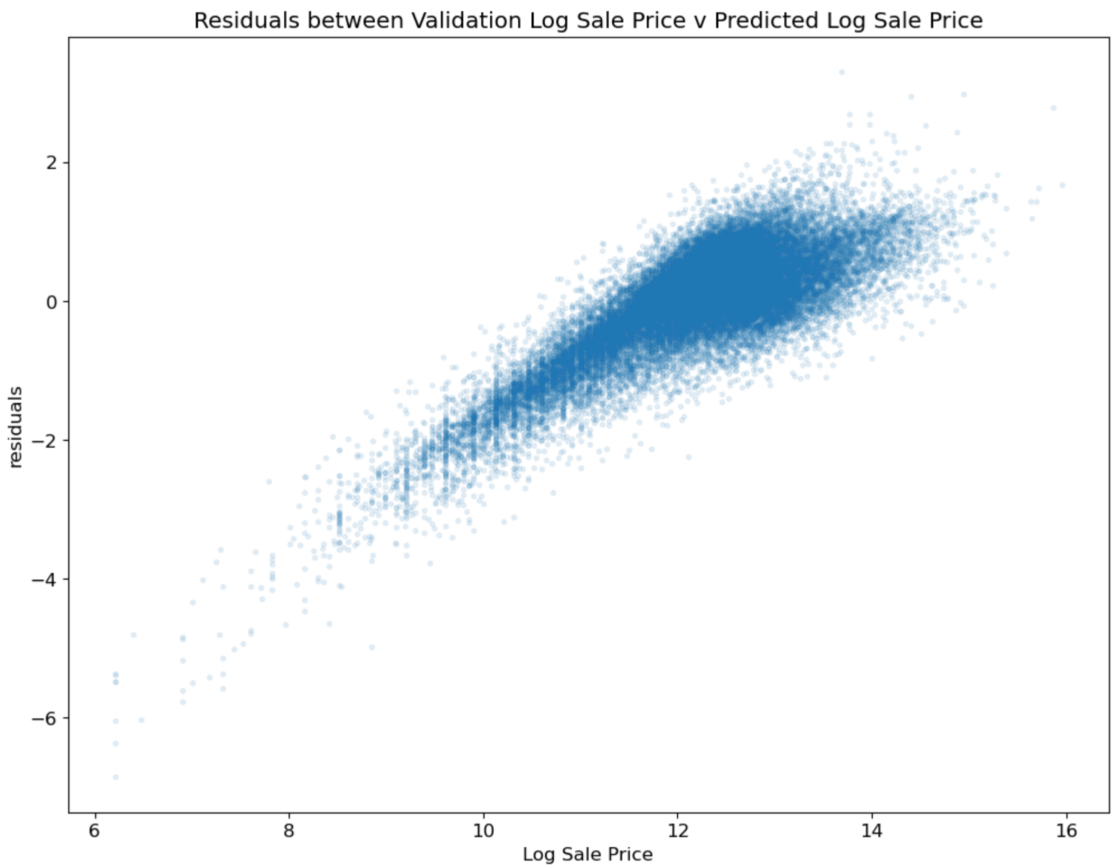
[17]:
```
grader.check("q3b")
```

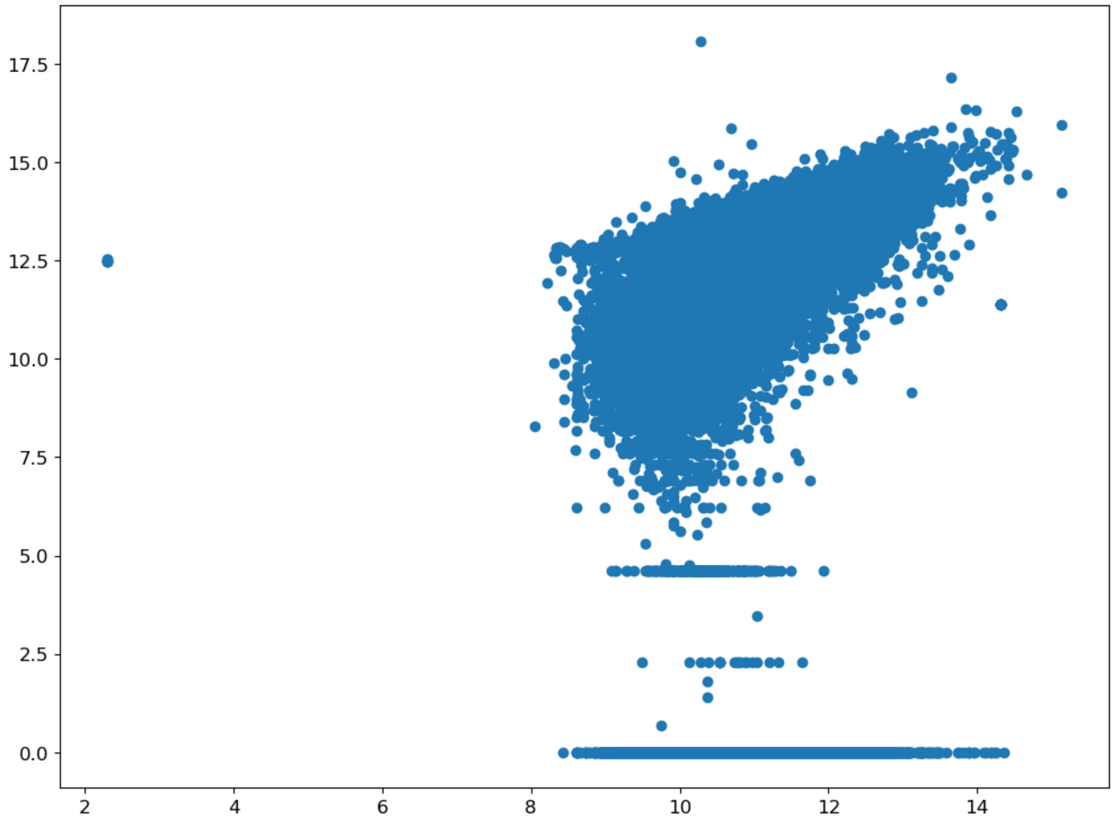
## 1.14 Question 4a

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

[22]:
```
plt.scatter                                                                  8)
plt.xlabel(
plt.ylabel(
plt.title('Residuals between Validation Log Sale Price v Predicted Log Sale␣
 ↪Price')
```

[22]:
```
Text(0.5, 1.0, 'Residuals between Validation Log Sale Price v Predicted Log Sale
Price')
```






[27]:
```
full_data.describe()[["Estimate (Land)"]] #-- high should be around 200,000k
```

[27]:

| | Estimate (Land) |
|---|---|
| count | 2.047920e+05 |
| mean | 5.187066e+04 |
| std | 5.591360e+04 |
| min | 0.000000e+00 |
| 25% | 2.762000e+04 |
| 50% | 3.795000e+04 |
| 75% | 5.580000e+04 |
| max | 3.716680e+06 |

[28]:
```
full_data["Estimate (Land)"].describe(percentiles=[0.95])
```

[28]:
```
count    2.047920e+05
mean     5.187066e+04
std      5.591360e+04
min      0.000000e+00
50%      3.795000e+04
95%      1.320445e+05
max      3.716680e+06
```

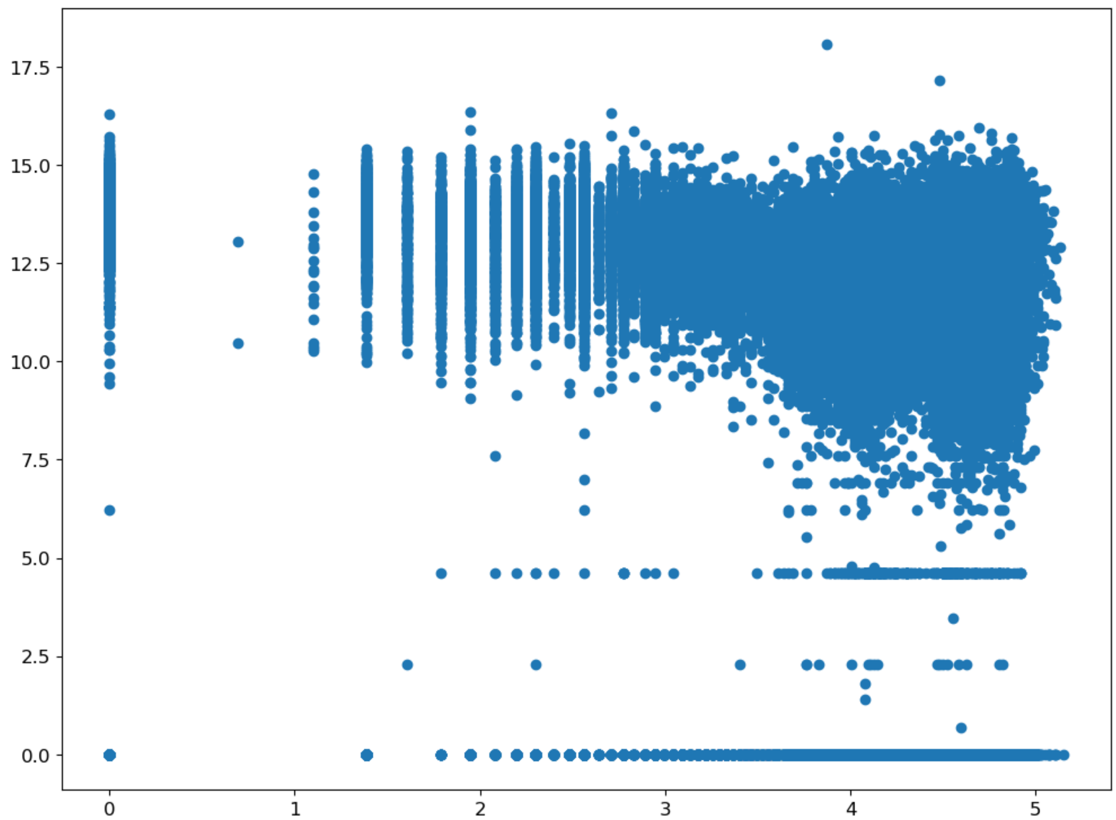
```
Name: Estimate (Land), dtype: float64
```

[29]:
```
full_data["Sale Price"].describe(percentiles=[0.95]) # high should be 700k
```

[29]:
```
count    2.047920e+05
mean     2.451646e+05
std      3.628694e+05
min      1.000000e+00
50%      1.750000e+05
95%      7.750000e+05
max      7.100000e+07
Name: Sale Price, dtype: float64
```

[30]:
```
full_data_cop = full_data.copy()
full_data_cop["Log Sale Price"] = np.log(full_data_cop["Sale Price"])
full_data_cop["ages"] = np.log(full_data_cop["Age"])
plt.scatter(data = full_data_cop, x = "ages", y = "Log Sale Price")
```

[30]:
```
<matplotlib.collections.PathCollection at 0x7fd101641050>
```




[31]:
```
np.percentile
```

`LinearRegression()` model with intercept term for grading purposes. Do not modify any hyperparameter in `LinearRegression()`, and focus on feature selection or hyperparameters of your own feature engineering function.

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

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

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

[35]:
```python
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))
```

[41]:

| | True Log Sale Price | Predicted Log Sale Price | True Sale Price |
|---|---|---|---|
| 1 | 12.560244 | 11.966303 | 285000.0 |
| 2 | 9.998798 | 11.475456 | 22000.0 |
| 3 | 12.323856 | 11.790922 | 225000.0 |
| 4 | 10.025705 | 11.354999 | 22600.0 |
| 6 | 11.512925 | 12.153325 | 100000.0 |

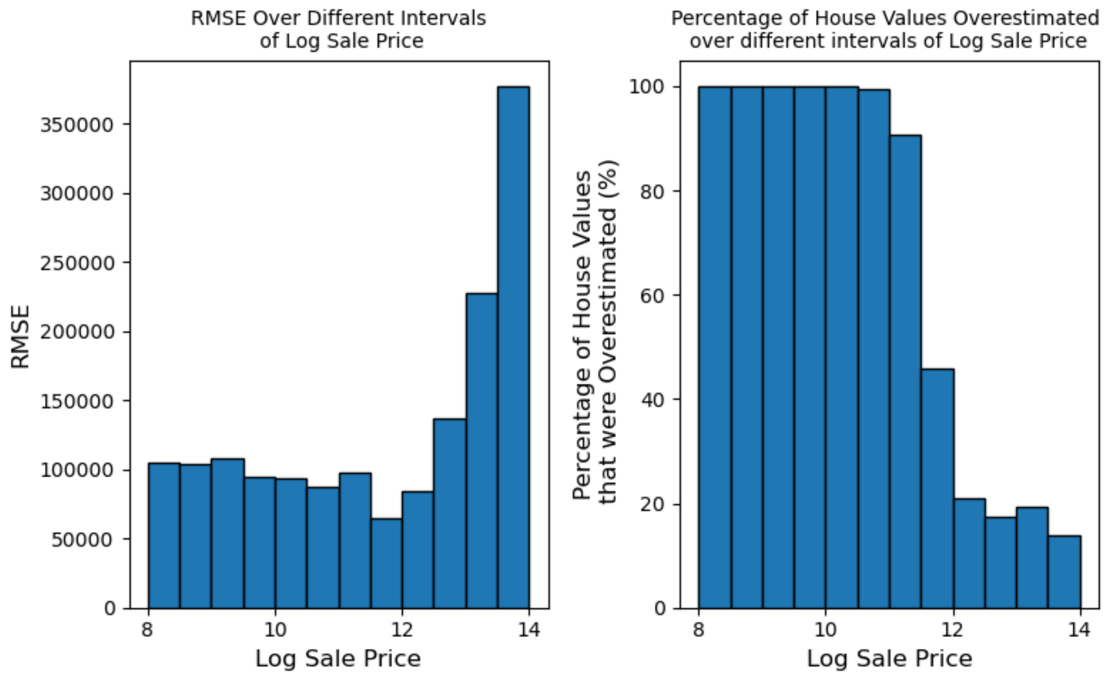
| | Predicted Sale Price |
|---|---|
| 1 | 157361.864168 |
| 2 | 96322.366007 |
| 3 | 132048.214772 |
| 4 | 85391.220473 |
| 6 | 189723.911211 |

---

```
The lower interval contains houses with true sale price $735.0 to $219696.0
The higher interval contains houses with true sale price $219696.0 to $1202604.0
```

[43]:
```
rmse_cheap =
rmse_expensiv


prop_overest_cheap =

prop_overest_expensive =


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

```
The RMSE for properties with log sale prices in the interval              is

The RMSE for properties with log sale prices in the interval             ) is


The percentage of overestimated values for properties with log sale prices in
the interval              is
The percentage of overestimated values for properties with log sale prices in
the interval              is
```


RMSE Over Different Intervals
of Log Sale Price
RMSE
350000
300000
250000
200000
150000
100000
50000
0
8
10
12
14
Log Sale Price
Percentage of House Values Overestimated
over different intervals of Log Sale Price
Percentage of House Values
that were Overestimated (%)
100
80
60
40
20
0
8
10
12
14
Log Sale Price