0.0.1 Question 1a

In this project we are essentially trying to answer the question. “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 then describe whether each one has an interest in seeing the housing price to be either high or low.**

Parties that may be interested in trying to answer this question are: real estate agents, homeowners, and prospective home buyers. Real estate agents would be interested in seeing and keeping track of housing prices in order to determine when to sell to clients. When housing prices are higher, agents are able to sell for a higher profit which allows them to receive a higher commission. As for homeowners, if they want to sell their existing home, they would seek to sell when housing prices are higher to earn a profit. On the contrary, home buyers would look to purchase when housing prices are low since they’re more affordable and likely have lower interest rates as well.

0.0.2 Question 1b

- 1bi). 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.
- 1bii). Would you consider some of these scenarios more (or less) fair than others? Why?

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

Write your full answers to both parts in the cell below:

All of these scenarios are seemingly unfair due to the fact that they either over or undervalue some aspect of each property. Scenario C is the most unfair due to its systematic bias against lower-value property owners since it places a greater burden on those who are less able to bear it, while Scenario A is also concerning for the financial burden it has on people. Scenario B and Scenario D may seem less unfair, but they're still concerning when it comes to equity and systemic impacts.

0.0.3 Question 1d

- 1di). What were the central problems with the earlier property tax system in Cook County as reported by the Chicago Tribune?
- 1dii). What were the primary causes of these problems? (Note: in addition to reading the paragraph above you will need to **read the Project Case Study.pdf explaining the context and history of this dataset before answering this question).**

1di.) The central problems involved biased property tax assessments, which undervalued high-priced homes and overvaluing low-priced homes. This led to wealthier homeowners paying less than their fair share of property taxes, while working-class homeowners paid more. Similarly, there were racial disparities where white homeowners benefited from the undervaluations, while working-class homeowners (people of color) were unfairly taxed at higher rates. Assessments were also consistently "off the mark" for years, demonstrating a lack of precision and fairness in the valuation process.

1dii.) Primary causes of these problems seemed to stem from the fact that the old system relied on outdated or inconsistent methods that failed to accurately estimate property values. Errors varied across different groups, regions, or property types, leading to unequal treatment. Higher-value properties were also often under-assessed compared to lower-value properties, which resulted in disproportionate tax burdens on lower income homeowners. Additionally, the Board of Review appeals process created opportunities for favoritism and unfair influence. Wealthier property owners could afford to appeal their assessments, which often led to reduced tax burdens, further skewing equity. Existing data was incomplete, outdated, or biased. For instance, missing data was more prevalent in lower-income neighborhoods, and renovations or improvements were inconsistently reported.

0.0.4 Question 2b:

Since we're trying to predict `Sale Price`, next we'll look for missing or unusual outliers in that field.

Examine the `Sale Price` column in the `training_val_data` DataFrame and answer the following questions:

- 2bi). Does the `Sale Price` data have any missing, N/A, negative or 0 values for the data? If so, propose a way to handle this.
- 2bii). Does the `Sale Price` data have any unusually large outlier values? If so, propose a cutoff to use for throwing out large outliers, and justify your reasoning).
- 2biii). Does the `Sale Price` data have any unusually small outlier values? If so, propose a cutoff to use for throwing out small outliers, and justify your reasoning.

Below are three cells. The first is a Markdown cell for you to write up your responses to all 3 parts above. The second two are code cells that are available for you to write code to explore the outliers and/or visualize the Sale Price data.

0.0.5 Question 2b i, ii, iii answer cell: ***Type your responses to all three parts in this cell…***

2bi.) Based on my calculations, it appears that Sale Price does not have any missing or nagative/0 values within the data.

2bii.) Sale Price does have some unusually large outlier values, with prices exceeding $712200. In order to combat this, the rows containing these high sale price can be removed to not interfere with the rest of the dat, or we could limit the data to only examine properties that fall under this sale threshold.

2biii.) Similarly, it appears there are some unusually low prices as $1 was the lower limit threshold, but no house sells for only $1. So similarly, we could just remove all the rows that contain absurdly low sale prices so it doesn't conflict with the general data set.

```
In [12]: missing_values = tr_val_data['Sale Price'].isna().sum()

         negative_or_zero_values = (tr_val_data['Sale Price'] <= 0).sum()

         print("Missing or N/A values:", missing_values)
         print("Negative or 0 values:", negative_or_zero_values)
         # your code exploring Sale Price above this line
```

```
Missing or N/A values: 0
Negative or 0 values: 0
```

In [13]:
```python
Q1 = tr_val_data['Sale Price'].quantile(0.25)
Q3 = tr_val_data['Sale Price'].quantile(0.75)
IQR = Q3 - Q1
upper_limit = Q3 + 1.5 * IQR
large_outliers = tr_val_data[tr_val_data['Sale Price'] > upper_limit]

print("Number of large outliers:", large_outliers.shape[0])
print("Upper limit for outliers:", upper_limit)

# Calculate lower quartile (Q1) and other percentiles
Q1 = tr_val_data['Sale Price'].quantile(0.25)
lower_limit = tr_val_data['Sale Price'].quantile(0.05)

small_outliers = tr_val_data[tr_val_data['Sale Price'] < lower_limit]

print("Number of small outliers:", len(small_outliers))
print("Lower limit for small outliers:", lower_limit)

# optional extra cell for exploring code
```

```
Number of large outliers: 12229
Upper limit for outliers: 712200.0
Number of small outliers: 0
Lower limit for small outliers: 1.0
```

0.0.6 Question 6a: Choose an additional feature

It's your turn to choose another feature to add to the model. Choose one new **quantitative** (not qualitative) feature and create Model 3 incorporating this feature (along with the features we've already chosen in Model 2). Try to choose a feature that will have a large impact on reducing the RMSE and/or will improve your residual plots. This can be a raw feature available in the dataset, or a transformation of one of the features in the dataset, or a new feature that you create from the dataset (see Project 1 for ideas).

Note: There is not one single right answer as to which feature to add, however **to receive credit on this question you should make sure the feature decreases the Cross Validation RMSE compared to Model 2 (i.e. we want to improve the model, not make it worse!)**

In the cell below, explain what additional feature you have chosen and why. Justify your reasoning. There are optional code cells provided below for you to use when exploring the dataset to determine which feature to add.

This problem will be graded based on your reasoning and explanation of the feature you choose, and then on your implementation of incorporating the feature.

NOTE Please don't add additional coding cells below or the Autograder will have issues. You do not need to use all the coding cells provided.

0.0.7 Question 6a Answer Cell:

In this cell, explain what feature you chose to add and why. Then give the equation for your new model (use the LaTeX from Model 2 from above and then add an additional term).

$$\text{Log Sale Price} = \theta_1(\text{Log Building Square Feet}) + \theta_2(\text{Shingle/Asphalt})$$

$$+\theta_3(\text{Tar\&Gravel}) + \theta_4(\text{Tile}) + \theta_5(\text{Shake}) + \theta_6(\text{Other}) + \theta_7(\text{Slate}) + \theta_8(\text{Estimate (Building)})$$

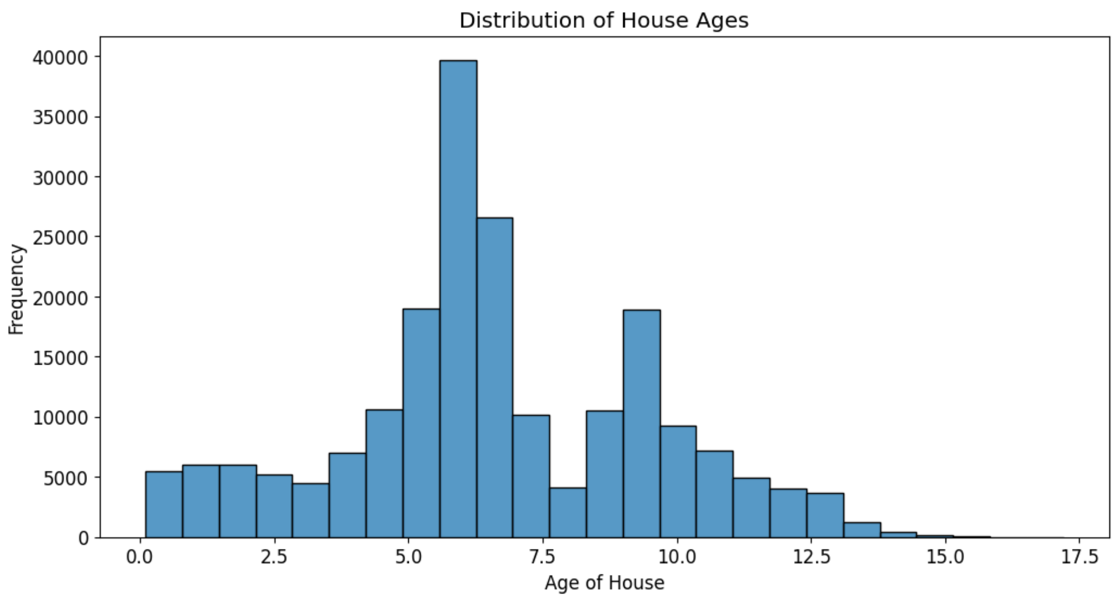
I was originally going to incorporate the Age Decade column as the age of the house usually plays a decent role in determining the property's overall value. Depending on the property, age could either increase of decrease a property's value due to depreciation or vintage styling choices. Similarly, older houses have more wear and tear which may also decrease its overall value, so it seemed like a good choice to help improve the model. However, once I got to calculating the RMSE, the value was higher than before, so I chose the next qualitative column I saw: Neighborhood Code. However, that also was not optimal in improving my RMSE value, which led me to use Building (Estimate). Building (Estimate) seems to be the estimated sale price for the property, so it sounds like it would also be a good feature to add to the model since we're dealing with sale prices.

In [53]:
```python
tr_val_data["Age Decade"].unique()

# Show work in this cell exploring data to determine which feature to add
```

Out[53]:
```
array([13.2,  9.6, 11.2,  6.3,  5.8, 10.9,  1.7, 10. ,  4.8,  7.4,  3.4,
        1.3, 12.2,  1.6,  5.9,  9.4,  8.7,  4.1,  6.5,  6.9,  0.1,  6.4,
        9.5,  2.7,  9.2,  6.2,  7.3,  6.7, 10.7,  9.3,  5.4,  3.3,  4. ,
        0.7,  9.1,  4.2,  1. ,  3.8,  2.8,  2.5, 10.2,  4.9,  5.7,  6.1,
        3. ,  3.9,  6.6, 10.1,  7.1,  4.7,  0.9,  5.1,  6. ,  8.8, 10.5,
        7.5,  8.9,  7.7,  4.4,  4.6, 11.5,  9. ,  6.8, 11.7,  4.3, 10.6,
        5.5, 11. ,  8.5,  3.7,  7.6, 13.4,  8. , 12.9,  4.5,  5. ,  1.9,
        8.4,  5.3,  3.5,  0.8,  0.4, 12.3, 14. ,  5.6,  7.8,  3.6, 12.1,
       11.8,  8.6,  3.1, 12.7, 12.8,  1.5,  2.9,  5.2,  1.8,  8.3,  8.2,
       10.4,  9.8,  9.7, 11.4, 11.3,  9.9,  2.4,  7. , 12. , 11.6, 11.1,
       12.5,  3.2, 12.4, 13.6,  7.2,  2.1, 10.3,  2.2,  2. , 10.8, 13.3,
        0.6, 15. , 13. ,  0.5,  1.2,  7.9, 13.7,  1.4,  2.6, 14.3,  1.1,
       11.9,  8.1, 12.6, 14.7, 14.4, 15.1, 13.5,  2.3, 14.2, 13.1, 13.9,
       13.8, 15.5, 14.8, 14.1, 14.5, 15.9,  0.3, 17.2, 14.6, 15.4, 15.6,
       14.9, 15.2, 15.3,  0.2, 16. , 16.5, 16.1, 16.3, 16.9, 16.4, 15.8,
       15.7])
```

In [54]:
```python
display(tr_val_data['Age Decade'].describe())
plt.figure(figsize=(12, 6))
sns.histplot(tr_val_data['Age Decade'], bins = 25)
plt.title("Distribution of House Ages")
plt.xlabel("Age of House")
plt.ylabel("Frequency")
plt.show()

# Optional code cell for additional work exploring data/ explaining which feature you chose.
```

```
count    204792.000000
mean          6.598121
std           2.900149
min           0.100000
25%           5.100000
50%           6.200000
75%           8.900000
max          17.200000
Name: Age Decade, dtype: float64
```



In [55]:
```
tr_val_data['Neighborhood Code'].unique()

# Optional code cell for additional work exploring data/ explaining which feature you chose.
```

Out[55]:
```
array([ 50, 120, 210, 220, 380, 181,  52,  70,  33,  20,  21,  30, 100,
        40,  61,  24, 282,  11, 112, 130, 240,  60,  10,  41,  92, 330,
        31,  32,  93,  51, 520,  84,  85,  88, 180, 110, 194, 160,  71,
       171,  22, 141, 113,  80,  13, 440, 212, 200,  45, 250, 190, 274,
       150,  35, 422, 162, 142, 260,  81, 314,  82,  42, 361, 430,  74,
       410, 103, 420, 321, 102,  12,  23, 132, 122, 101, 350, 151, 223,
       115,  90, 323,  91,  63, 270, 281,  15, 280, 560,  14, 170, 211,
       111, 191, 271,  55, 251, 104, 221, 241,  53, 371,  72, 461,  75,
       342, 345, 175, 320,  43,  62, 423, 140, 230, 362,  94,  34, 193,
       310, 121,  54, 390,  46, 344, 114,  19,  44,  65, 431, 166, 131,
        47, 312, 275, 161,  39, 257,  83, 255, 183, 192,  73,  36, 222,
       316,  96, 164, 315, 580,  64, 300,  26, 224,  87, 226, 400, 152,
        56,  25, 293, 402, 182, 174, 340, 201, 133, 116, 185,  27, 600,
       432, 165,  48,  37,  67, 360, 262, 232, 134,  99, 117, 227, 290,
       163,  18, 109,  95,  38, 143,  86, 145, 463, 341, 106])
```

In [56]:
```
tr_val_data['Estimate (Building)'].unique()

# Optional code cell for additional work exploring data/ explaining which feature you chose.
```

Out[56]:
```
array([139500, 177500,  63470, …, 635930, 537990, 637720])
```

0.0.8 Question 6b: Create Model 3

In the cells below fill in the code to create and analyze Model 3 (follow the Modeling steps outlined above).

PLEASE DO NOT ADD ANY ADDITIONAL CELLS IN THIS PROBLEM OR IT MIGHT MAKE THE AUTOGRADER FAIL

In [57]:
```python
# Modeling Step 1:  Process the Data

# Hint: You can either use your implementation of the One Hot Encoding Function
#from Project Part 1, or use the staff's implementation

from feature_func import *

...
# Optional:  Define any helper functions you need for one-hot encoding above this line


def process_data_m3(df):

    data = df.copy()

    data = data[data["Pure Market Filter"]==1]

    # Create Log Sale Price column
    data["Log Sale Price"] = np.log(data["Sale Price"])

    # Create Log Building Square Feet column
    data["Log Building Square Feet"] = np.log(data["Building Square Feet"])

    # Create Log Estimate (Building) column
    data["Log Estimate (Building)"] = np.log(data["Estimate (Building)"] + 1)

    # Change Roof Material to names
    data = substitute_roof_material(data)

    # one-hot encode the roof material
    data = ohe_roof_material(data)

    # select only relevant columns
    relevant_columns = ['Log Estimate (Building)', 'Log Building Square Feet', 'Log Sale Price
    data = data[relevant_columns]



    return data



# Process the data for Model 3 (using the same tr and val datatsets we created in Question 3)
processed_train_m3 = process_data_m3(tr)
```

```python
processed_val_m3 = process_data_m3(val)

# Create X (Dataframe) and y (series) to use to train the model
X_train_m3 = processed_train_m3.drop(columns = ['Log Sale Price'])
y_train_m3 = processed_train_m3['Log Sale Price']

X_valid_m3 = processed_val_m3.drop(columns = ['Log Sale Price'])
y_valid_m3 = processed_val_m3['Log Sale Price']


# Take a look at the result
display(X_train_m3.head())
display(y_train_m3.head())

display(X_valid_m3.head())
display(y_valid_m3.head())
```

```
        Log Estimate (Building)  Log Building Square Feet  \
21302                 11.248061                  6.871091
19451                 11.510031                  7.576610
32018                 12.096431                  6.891626
144262                12.042265                  7.186901
197227                12.612640                  7.576610

        Roof Material_Other  Roof Material_Shake  \
21302                   0.0                  0.0
19451                   0.0                  0.0
32018                   0.0                  0.0
144262                  0.0                  0.0
197227                  0.0                  0.0

        Roof Material_Shingle/Asphalt  Roof Material_Slate  \
21302                             1.0                  0.0
19451                             1.0                  0.0
32018                             1.0                  0.0
144262                            1.0                  0.0
197227                            1.0                  0.0

        Roof Material_Tar&Gravel  Roof Material_Tile
21302                        0.0                 0.0
19451                        0.0                 0.0
32018                        0.0                 0.0
144262                       0.0                 0.0
197227                       0.0                 0.0


21302     11.738466
19451     12.100712
32018     12.577291
144262    12.502467
197227    12.449019
Name: Log Sale Price, dtype: float64
```

```
        Log Estimate (Building)  Log Building Square Feet  \
17112                11.882217                  7.487174
189337               12.110118                  7.688913
141725               11.760418                  6.985642
9776                 11.569599                  6.846943
81676                11.272776                  7.096721

        Roof Material_Other  Roof Material_Shake  \
17112                   0.0                  0.0
189337                  0.0                  0.0
141725                  0.0                  0.0
9776                    0.0                  0.0
81676                   0.0                  0.0

        Roof Material_Shingle/Asphalt  Roof Material_Slate  \
17112                             1.0                  0.0
189337                            1.0                  0.0
141725                            1.0                  0.0
9776                              1.0                  0.0
81676                             1.0                  0.0

        Roof Material_Tar&Gravel  Roof Material_Tile
17112                        0.0                 0.0
189337                       0.0                 0.0
141725                       0.0                 0.0
9776                         0.0                 0.0
81676                        0.0                 0.0


17112     12.100707
189337    12.460715
141725    12.165251
9776      11.982929
81676     11.719940
Name: Log Sale Price, dtype: float64
```

```
In [58]: # Modeling STEP 2:  Create and Fit a Multiple Linear Regression Model


         linear_model_m3 = lm.LinearRegression(fit_intercept=False)

         linear_model_m3.fit(X_train_m3, y_train_m3)

         # your code above this line to create and fit regression model for Model 3

         y_predict_train_m3 = linear_model_m3.predict(X_train_m3)

         y_predict_valid_m3 = linear_model_m3.predict(X_valid_m3)
```

```
In [59]: # MODELING STEP 3:  Evaluate the RMSE for your model
```

```
# Training and validation errors for the model (in units dollars, not log(dollars))

training_error[2] = rmse(y_predict_train_m3, y_train_m3)
validation_error[2] = rmse(y_predict_valid_m3, y_valid_m3)

training_error[2] = rmse(np.exp(y_predict_train_m3), np.exp(y_train_m3))
validation_error[2] = rmse(np.exp(y_predict_valid_m3), np.exp(y_valid_m3))

(print("3rd Model \nTraining RMSE: $ {}\nValidation RMSE: {}\n"
       .format(training_error[2], validation_error[2]))
)
```

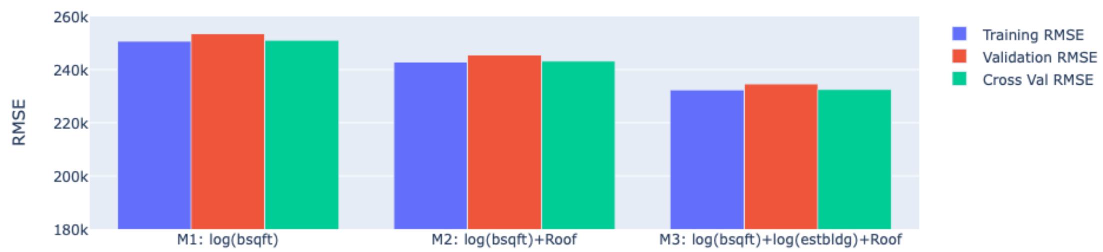
```
3rd Model
Training RMSE: $ 232349.50469106025
Validation RMSE: 234597.11998179482
```

In [60]:
```
# MODELING STEP 4:  Conduct 5-fold cross validation for model and output RMSE

linear_model_m3_cv = lm.LinearRegression(fit_intercept=False)


# Process the entire cleaned training_val dataset using the m3 pipeline
processed_full_m3 = process_data_m3(tr_val_data)

# Split the processed_full_m3 Dataset into X and y to use in models.
X_full_m3 = processed_full_m3.drop(columns="Log Sale Price")
y_full_m3 = processed_full_m3["Log Sale Price"]


# Run cross_validate_rmse function:
cv_error_m3  = cross_validate_rmse(linear_model_m3_cv, X_full_m3, y_full_m3)

# Save the cross validation error for model 3 in our list to compare different models:

cv_error[2] = cv_error_m3

print("3rd Model Cross Validation RMSE: {}".format(cv_error[2]))
```

```
3rd Model Cross Validation RMSE: 232477.29879016159
```

In [61]:
```
# MODELING STEP 5:  Add a name for your 3rd model describing the features
#and run this cell to Plot bar graph all 3 models

model_names[2] = "M3: log(bsqft)+log(estbldg)+Roof"
```

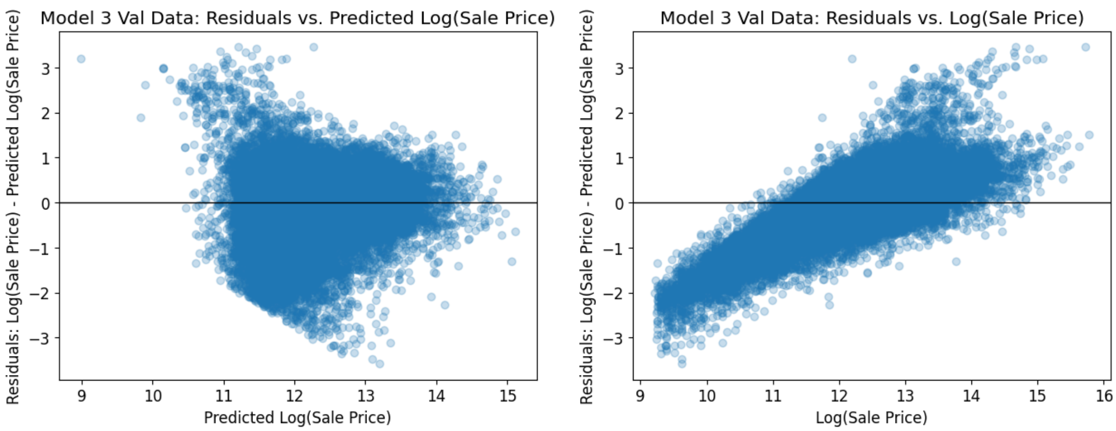
```
fig = go.Figure([
go.Bar(x = model_names, y = training_error, name="Training RMSE"),
go.Bar(x = model_names, y = validation_error, name="Validation RMSE"),
go.Bar(x = model_names, y = cv_error, name="Cross Val RMSE")
])

fig.update_yaxes(range=[180000,260000], title="RMSE")

fig
```



In [62]:
```
# MODELING STEP 5 cont'd:  Plot 2 side-by-side residual plots
#(similar to Question 3, for validation data)

fig, ax = plt.subplots(1,2, figsize=(15, 5))

residuals = y_valid_m3 - y_predict_valid_m3

x_plt1 = y_predict_valid_m3
y_plt1 = residuals

x_plt2 = y_valid_m3
y_plt2 = residuals


ax[0].scatter(x_plt1, y_plt1, alpha=.25)
ax[0].axhline(0, c='black', linewidth=1)
ax[0].set_xlabel(r'Predicted Log(Sale Price)')
ax[0].set_ylabel(r'Residuals: Log(Sale Price) - Predicted Log(Sale Price)');
ax[0].set_title("Model 3 Val Data: Residuals vs. Predicted Log(Sale Price)")

ax[1].scatter(x_plt2, y_plt2, alpha=.25)
```

```
ax[1].axhline(0, c='black', linewidth=1)
ax[1].set_xlabel(r'Log(Sale Price)')
ax[1].set_ylabel(r'Residuals: Log(Sale Price) - Predicted Log(Sale Price)');
ax[1].set_title("Model 3 Val Data: Residuals vs. Log(Sale Price)")
```

Out[62]: Text(0.5, 1.0, 'Model 3 Val Data: Residuals vs. Log(Sale Price)')



In [63]: grader.check("q6b")

Out[63]: q6b results: All test cases passed!

0.0.9 Question 6c

- 6ci). Comment on your RMSE and residual plots from Model 3 compared to the first 2 models.
- 6cii). Are the residuals of your model still showing a trend that overestimates lower priced houses and underestimates higher priced houses? If so, how could you try to address this in the next round of modeling?
- 6ciii). If you had more time to improve your model, what would your next steps be?

6ci). The RMSE and residual plots from Model 3 seem to be better than the first two models. Since the distribution is more evenly distributed around 0, we can tell that the estimation of houses is more accurate to their actual values.

6cii). There is still a slight trend which we can try and accomodate for by adding more features in future models. For instance, we could try and incorporate other features that may be closely correlated with the sale price so as to avoid the over and underestimation.

6ciii). If I had more time to improve my model, I would go back and try to incorporate age as I feel that would be a good factor. I was originally going to incorporate that, but messed up somewhere so with more time, I would try and debug it to see if it would work.

0.1 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 residual is positive and negative separately.

When the residual is positive, the model underestimates the house's value. This might mean lower property taxes for the homeowner, but they might not realize how much their house is really worth if they sell. When it's negative, the model overestimates the house's value, which could lead to higher property taxes. This is especially unfair for people with lower-cost homes since they may end up paying more than they should. Overall, when the model is wrong, it can either save homeowners money or cost them, depending on whether the predicted price is too low or too high.

0.2 Question 7b

Reflecting back on your exploration in Questions 5 and 6a, 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 upon 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?

A fair model should avoid large errors and shouldn't be influenced by homeowners' socio-economic status or legal biases. It's important that the residual plot is evenly and tightly distributed around 0, since unfair taxes can create serious financial troubles for many people.

0.3 Extra Credit Step 1: Creating Your Model

Complete the modeling steps (you can skip the cross validation step to save memory) in the cells below.

DO NOT ADD ANY EXTRA CELLS BELOW (for this part of the problem)

1 Please include all of your feature engineering processes

```
# for the training, validation and test sets.
# dataset_type is a flag to use as follows:
# dataset_type=1  implies the data is the training data
# dataset_type=2 implies the data is the validation data
# dataset_type = 3 imples the data is the test data

#Important Instructions:
# When processing the training data, you CAN drop any rows/data that you deem to be outliers
# When processing the validation data you CANNOT drop any rows
# When processing the test data, you CANNOT drop any rows, and you CANNOT reference "Sale Price", as it
```

In [74]:
```
# Modeling Step 1:  Process the Data


# Hint: You can either use your implementation of the One Hot Encoding Function from
#Project Part 1, or use the staff's implementation
from feature_func import *

def ohe_towncode(data):

    oh_enc = OneHotEncoder()
    oh_enc.fit(data[['Town Code']])
    dummies = pd.DataFrame(oh_enc.transform(data[['Town Code']]).todense(),
                           columns=oh_enc.get_feature_names_out(),
                           index=data.index)
    # Join the dummies to the original DataFrame
    return data.join(dummies)

def ohe_propertyclass(data):
    oh_enc = OneHotEncoder()
    oh_enc.fit(data[['Property Class']])
    dummies = pd.DataFrame(oh_enc.transform(data[['Property Class']]).todense(),
                           columns=oh_enc.get_feature_names_out(),
                           index=data.index)
    # Join the dummies to the original DataFrame
    return data.join(dummies)

# # Optional:  Define any helper functions you need (for example, for one-hot
# #encoding, etc) above this line
```

```python
def process_data_ec(df, dataset_type):

    """
    Function that includes all feature engineering processes
    for the training, validation and test sets for your extra credit model.

    Important Instructions:
    When processing the training data, you CAN drop any rows/data that you deem to be outliers
    When processing the validation data you CANNOT drop any rows (even what you deem to be an
    When processing the test data, you CANNOT drop any rows, and you CANNOT reference "Sale Pr

    dataset_type is a flag to use as follows:
        dataset_type=1  implies the data is the training data
        dataset_type=2 implies the data is the validation data
        dataset_type = 3 imples the data is the test data

    """
    data = df.copy()

    #for test dataset
    if dataset_type == 3:
        relevant_columns = []
    else:

        data = data[(data["Pure Market Filter"] == 1) & (data["Sale Price"] > 100)]

        data["Log Sale Price"] = np.log(data["Sale Price"])

        relevant_columns = ['Log Sale Price']



    # Create Log Building Square Feet column
    data["Log Building Square Feet"] = np.log(data["Building Square Feet"])

    # Create Log Estimate (Building) column
    data["Log Estimate (Building)"] = np.log(data["Estimate (Building)"] + 1)

    data["Log Age"] = np.log(data["Age"])

    data["Log Estimate (Land)"] = np.log(data["Estimate (Land)"] + 1)

    # Change Roof Material to names
    data = substitute_roof_material(data)

    # one-hot encode the roof material
    data = ohe_roof_material(data)

    data = ohe_towncode(data)

    data = ohe_propertyclass(data)
```

```python
    # select only relevant columns
    relevant_columns += ['Log Estimate (Building)', 'Log Building Square Feet', 'Log Age', 'Lo
    relevant_columns += [col for col in data.columns if "Town Code" in col]
    relevant_columns += [col for col in data.columns if "Property Class" in col]
    # relevant_columns += [f'Property_Class_{pc}' for pc in property_classes]
    data = data[relevant_columns]


    return data


# Use the same original train and valid datasets from 3a (otherwise the
# validation errors aren't comparable).  Don't resplit the data.

# Process the data
processed_train_ec = process_data_ec(tr, dataset_type = 1)

processed_val_ec = process_data_ec(val, dataset_type = 2)



X_train_ec = processed_train_ec.drop(columns = ['Log Sale Price'])
y_train_ec = processed_train_ec['Log Sale Price']

X_valid_ec = processed_val_ec.drop(columns = ['Log Sale Price'])
y_valid_ec = processed_val_ec['Log Sale Price']



# Take a look at the result
display(X_train_ec.head())
display(y_train_ec.head())

display(X_valid_ec.head())
display(y_valid_ec.head())
```

```
        Log Estimate (Building)  Log Building Square Feet   Log Age  \
21302                 11.248061                  6.871091  4.276666
19451                 11.510031                  7.576610  4.736198
32018                 12.096431                  6.891626  4.248495
144262                12.042265                  7.186901  4.007333
197227                12.612640                  7.576610  4.465908

        Log Estimate (Land)  Roof Material_Other  Roof Material_Shake  \
21302              9.838469                  0.0                  0.0
19451             10.686270                  0.0                  0.0
32018             10.532123                  0.0                  0.0
144262            10.270593                  0.0                  0.0
197227            11.033615                  0.0                  0.0

        Roof Material_Shingle/Asphalt  Roof Material_Slate  \
```

```
21302                            1.0                 0.0
19451                            1.0                 0.0
32018                            1.0                 0.0
144262                           1.0                 0.0
197227                           1.0                 0.0

        Roof Material_Tar&Gravel  Roof Material_Tile  …  Property Class  \
21302                        0.0                 0.0  …             205
19451                        0.0                 0.0  …             205
32018                        0.0                 0.0  …             202
144262                       0.0                 0.0  …             203
197227                       0.0                 0.0  …             205

        Property Class_202  Property Class_203  Property Class_204  \
21302                  0.0                 0.0                 0.0
19451                  0.0                 0.0                 0.0
32018                  1.0                 0.0                 0.0
144262                 0.0                 1.0                 0.0
197227                 0.0                 0.0                 0.0

        Property Class_205  Property Class_206  Property Class_207  \
21302                  1.0                 0.0                 0.0
19451                  1.0                 0.0                 0.0
32018                  0.0                 0.0                 0.0
144262                 0.0                 0.0                 0.0
197227                 1.0                 0.0                 0.0

        Property Class_208  Property Class_209  Property Class_278
21302                  0.0                 0.0                 0.0
19451                  0.0                 0.0                 0.0
32018                  0.0                 0.0                 0.0
144262                 0.0                 0.0                 0.0
197227                 0.0                 0.0                 0.0

[5 rows x 59 columns]
```

```
21302     11.738466
19451     12.100712
32018     12.577291
144262    12.502467
197227    12.449019
Name: Log Sale Price, dtype: float64
```

```
        Log Estimate (Building)  Log Building Square Feet   Log Age  \
17112                 11.882217                  7.487174  4.043051
189337                12.110118                  7.688913  4.644391
141725                11.760418                  6.985642  4.127134
9776                  11.569599                  6.846943  4.499810
81676                 11.272776                  7.096721  4.158883

        Log Estimate (Land)  Roof Material_Other  Roof Material_Shake  \
```

```
17112           10.357775                 0.0                 0.0
189337          11.077068                 0.0                 0.0
141725          10.434733                 0.0                 0.0
9776             9.918918                 0.0                 0.0
81676           10.008793                 0.0                 0.0

        Roof Material_Shingle/Asphalt  Roof Material_Slate  \
17112                             1.0                  0.0
189337                            1.0                  0.0
141725                            1.0                  0.0
9776                              1.0                  0.0
81676                             1.0                  0.0

        Roof Material_Tar&Gravel  Roof Material_Tile  …  Property Class  \
17112                        0.0                 0.0  …             203
189337                       0.0                 0.0  …             205
141725                       0.0                 0.0  …             203
9776                         0.0                 0.0  …             202
81676                        0.0                 0.0  …             203

        Property Class_202  Property Class_203  Property Class_204  \
17112                  0.0                 1.0                 0.0
189337                 0.0                 0.0                 0.0
141725                 0.0                 1.0                 0.0
9776                   1.0                 0.0                 0.0
81676                  0.0                 1.0                 0.0

        Property Class_205  Property Class_206  Property Class_207  \
17112                  0.0                 0.0                 0.0
189337                 1.0                 0.0                 0.0
141725                 0.0                 0.0                 0.0
9776                   0.0                 0.0                 0.0
81676                  0.0                 0.0                 0.0

        Property Class_208  Property Class_209  Property Class_278
17112                  0.0                 0.0                 0.0
189337                 0.0                 0.0                 0.0
141725                 0.0                 0.0                 0.0
9776                   0.0                 0.0                 0.0
81676                  0.0                 0.0                 0.0

[5 rows x 59 columns]
```

```
17112     12.100707
189337    12.460715
141725    12.165251
9776      11.982929
81676     11.719940
Name: Log Sale Price, dtype: float64
```

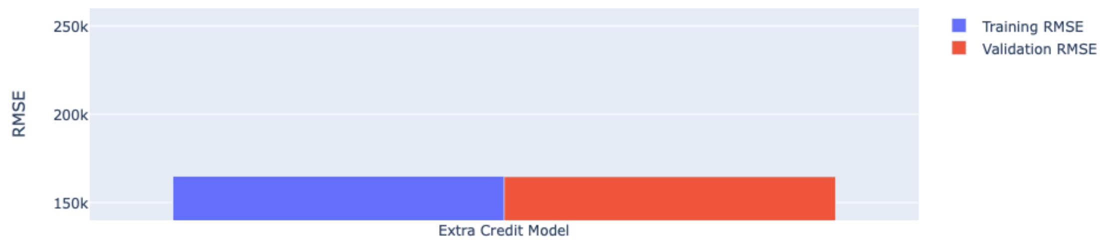
In [75]:
```
# Run this code to make sure you haven't dropped any of the rows in the validation set
```

```
assert X_valid_ec.shape[0] == 33475
```

In [78]:
```
# MODELING STEP 3:  Evaluate the RMSE for your model

linear_model_ec = lm.LinearRegression(fit_intercept=False)

linear_model_ec.fit(X_train_ec, y_train_m3)

y_predict_train_ec = linear_model_ec.predict(X_train_ec)

y_predict_valid_ec = linear_model_ec.predict(X_valid_ec)

# Training and test errors for the model
#(in its original values before the log transform)

training_error_ec = rmse(y_predict_train_ec, y_train_ec)
validation_error_ec = rmse(y_predict_valid_ec, y_valid_ec)

training_error_ec = rmse(np.exp(y_predict_train_ec), np.exp(y_train_ec))
validation_error_ec = rmse(np.exp(y_predict_valid_ec), np.exp(y_valid_ec))

(print("Extra Credit \nTraining RMSE:$ {}\nValidation RMSE:$ {}\n"
       .format(training_error_ec, validation_error_ec))
)
```

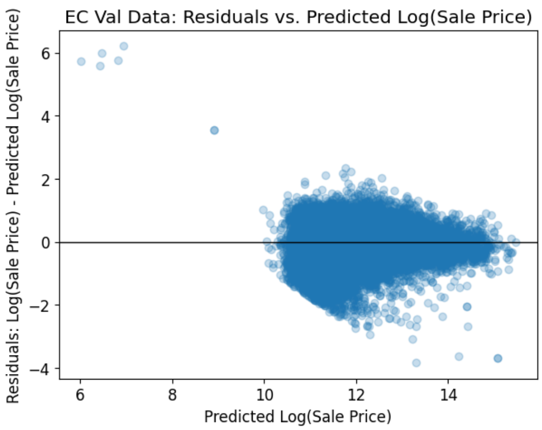
```
Extra Credit
Training RMSE:$ 165037.16802993405
Validation RMSE:$ 164650.58983746613
```

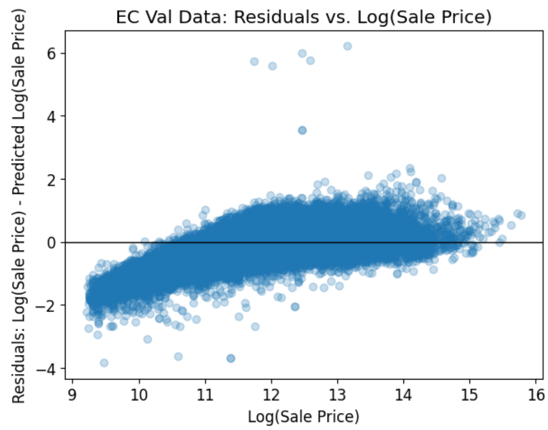
In [79]:
```
# Optional: Run this cell to visualize

#import plotly.graph_objects as go

fig = go.Figure([
go.Bar(x = ["Extra Credit Model"], y = [training_error_ec], name="Training RMSE"),
go.Bar(x = ["Extra Credit Model"], y = [validation_error_ec], name="Validation RMSE"),

])


fig
fig.update_yaxes(range=[140000,260000], title="RMSE")
# Feel free to update the range as needed
```



In [80]:
```
# MODELING STEP 5: Plot 2 side-by-side residual plots for validation data

fig, ax = plt.subplots(1,2, figsize=(15, 5))

residuals = y_valid_ec - y_predict_valid_ec

x_plt1 = y_predict_valid_ec
y_plt1 = residuals

x_plt2 = y_valid_ec
y_plt2 = residuals


ax[0].scatter(x_plt1, y_plt1, alpha=.25)
ax[0].axhline(0, c='black', linewidth=1)
ax[0].set_xlabel(r'Predicted Log(Sale Price)')
ax[0].set_ylabel(r'Residuals: Log(Sale Price) - Predicted Log(Sale Price)');
ax[0].set_title("EC Val Data: Residuals vs. Predicted Log(Sale Price)")

ax[1].scatter(x_plt2, y_plt2, alpha=.25)
ax[1].axhline(0, c='black', linewidth=1)
ax[1].set_xlabel(r'Log(Sale Price)')
ax[1].set_ylabel(r'Residuals: Log(Sale Price) - Predicted Log(Sale Price)');
ax[1].set_title("EC Val Data: Residuals vs. Log(Sale Price)")
```

Out[80]:
```
Text(0.5, 1.0, 'EC Val Data: Residuals vs. Log(Sale Price)')
```


EC Val Data: Residuals vs. Predicted Log(Sale Price)
Residuals: Log(Sale Price) - Predicted Log(Sale Price)
Predicted Log(Sale Price)
6
4
2
0
−2
−4
6
8
10
12
14


EC Val Data: Residuals vs. Log(Sale Price)
Residuals: Log(Sale Price) - Predicted Log(Sale Price)
Log(Sale Price)
6
4
2
0
−2
−4
9
10
11
12
13
14
15
16

1.1 Extra Credit Step 2: Explanation (Required for points on model above):

- Explain what you did to create your model. What versions did you try? What worked and what didn't?
- Comment on the RMSE and residual plots from your model. Are the residuals of your model still showing a trend that overestimates lower priced houses and underestimates higher priced houses?

Write your answers in the text cell below

I had played around a lot with different columns, such as Estimate(Land), Neighborhood Code, and Age Decade; however, those features didn't have a strong effect on the total RMSE. So, I just removed them and continued to add on other columns until I noticed noticeable changes. For instance, what really decreased my RMSE were property class and town codes, as once I one hot encoded and implemented them into my model, I noticed a significant decrease in the total compared to when I used the other previously mentioned features. The RMSE and residual plots definitely appear to be more significant than the ones from the previous models as well. There is still a slight trend seen in the graphs, but the dispersion is getting closer to being around 0 compared to before.