0.1 Question 1: Human Context and Ethics

In this part of the project, we will explore the human context of our housing dataset.

You should read the Project_CaseStudy.pdf on Canvas explaining the context and history surrounding this dataset before attempting this section.

0.1.1 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 to be high or low.**

1. Owner of the home wants the house price to be high because it increases return on their original investment
2. Potential buyers want the price to be low so their purchase will be more affordable and they'll have lower mortgage
3. Real Estate agents want the price to be high because their commission is usually a percentage of the sale price.

0.1.2 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.

A. If assessed higher than sale price, the property taxes would be higher, since they are often a percentage of the property's value, so that would hurt the homeowner by costing them higher taxes

B. If assessed lower and the homeowner is trying to sell the home, this would lower their asking price, hurting their profits from sale and overall return on investment

C. Like A, if the inexpensive properties were overvalued, it would raise their property taxes when they should be lower, but it would help them ask for me if they were selling the house. If expensive properties are undervalued, it would benefit the owners by lowering their property taxes when they should be higher, but hurt their sell price.

D. This would help the owner of the inexpensive properties when it comes to taxes, but hurt them if they were trying to sell the home. For expensive properties, the opposite is true. They would have to pay higher property taxes, but if they were selling the home, they could ask a higher price due to higher value of assessment.

0.1.3 Question 1d

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

The tax code was regressive, meaning wealthier homeowners paid less and working class homeowners paid more, taxing cheaper homes at a higher rate than it taxed the expensive homes. The primary cause was based in redlinging due to race and foreign neighbor deductions being included in the home's valuation rating.

0.2 Question 2a: More EDA

In good news you have already done a lot of EDA with this dataset in Project 1.

Before fitting any model, we should check for any missing data and/or unusual outliers.

Since we're trying to predict `Sale Price`, we'll start with that field.

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

- 2ai). 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.
- 2aii). 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).
- 2aiii). 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.

2ai) There are over 35000 properties with a Sale Price of 1. We could remove all of the values of 1 by using this code: training_val_data = training_val_data[training_val_data["Sale Price"] != 1]

2aii) Yes, it has 843 properties over 2,000,000. We could remove all of those outliers and that's still less than half of 1 percent of the total number of properties.

2aiii) Yes, it has 2201 properties under 12000 (not including all the 1s), so we could remove all of those outliers and still not hit 1 percent of the total number of properties.

```
In [9]: ones = training_val_data[training_val_data["Sale Price"] == 1]["Sale Price"].count()
        training_val_data[training_val_data["Sale Price"] < 12000]["Sale Price"].count() - ones
        # your code exploring Sale Price above this line
```

```
Out[9]: 2201
```

```
In [10]: ...
         # optional extra cell for exploring code
```

```
Out[10]: Ellipsis
```

0.3 Question 5: Improving the Model

0.3.1 Question 5a: 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). 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.

Note: There is not one single right answer as to which feature to add, however 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!)
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.

0.3.2 Question 5a Answer Cell:

After comparing Land Square Feet, Lot Size, Age, and Estimate (Land), I settled on using Estimate (Land) since it's box plots seemed to have the most variation from zero, although Land Square Feet seemed to be a pretty valid option as well.

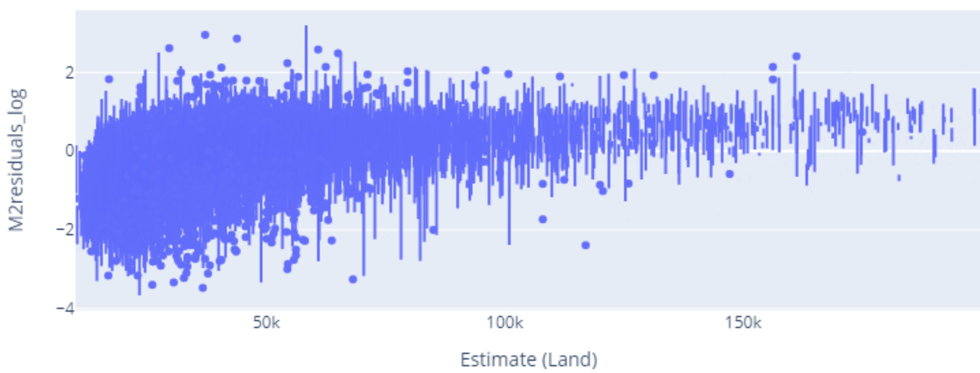
Model 3:

Log Sale Price = θ_1(Log Building Square Feet)+θ_2(Estimate (Land))+θ_3(Shingle/Asphalt)+θ_4(Tar/Gravel)+θ_5(Tile)+θ_6(Sl

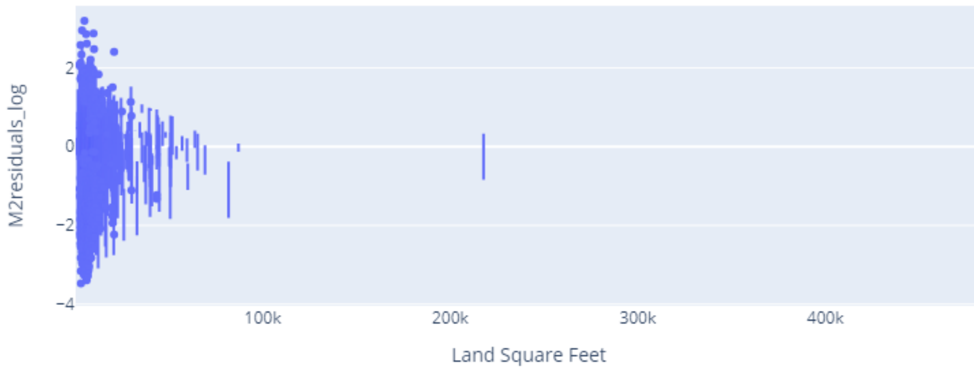
```
In [41]: def process_data_candidates2(data):

             # Remove Non-Market Sales
             data = data[data["Pure Market Filter"]==1]

             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 Bedrooms
             data = add_total_bedrooms(data)
```

```
    # Update Roof Material feature with names
    data = substitute_roof_material(data)

    # Select columns for comparing residuals
    data = data[['Log Building Square Feet',  'Land Square Feet', 'Lot Size', 'Age', 'Estimate

    return data

valid_comp = process_data_candidates2(valid)
valid_comp = valid_comp.assign(M2residuals_log=Y_valid_m2 - Y_predict_valid_m2)
# Show work in this cell exploring data to determine which feature to add
```

```
In [42]: comp = valid_comp[valid_comp["Estimate (Land)"] < 200000]
         comp = comp[comp["Estimate (Land)"] > 10000]
         px.box(comp, x='Estimate (Land)', y='M2residuals_log')


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




```
In [43]: px.box(valid_comp, x='Land Square Feet', y='M2residuals_log')

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




```
In [44]: valid.columns
         valid["Estimate (Land)"].max()
         # Optional code cell for additional work exploring data/ explaining which feature you chose.
```

```
Out[44]: 3716680
```

0.3.3 Question 5b: Create Model 3

In the cells below fill in the code to create and analyze Model 3 (follow the Modeling steps outlined in Questions 3 and 4).

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

```
In [45]: # Modeling Step 1:  Process the Data

         # Hint: You can either use your implementation of the One Hot Encoding Function from Project P

         from feature_func import *

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

         def process_data_m3(data):

             # Remove Non-Market Sales
             data = data[data["Pure Market Filter"]==1]

             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"])

             data = data[data["Estimate (Land)"] < 200000]
             data = data[data["Estimate (Land)"] > 10000]

             # Update Roof Material feature with names
             data = substitute_roof_material(data)

             data = ohe_roof_material(data)

             data = data[['Log Building Square Feet', 'Log Sale Price', 'Estimate (Land)'] + [col for co

             return data


         # Process the data for Model 3
         processed_train_m3 = process_data_m3(train)
         processed_val_m3 = process_data_m3(valid)

         # Create X (dataframe) and Y (series) to use in 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"]
```

```python
# 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 Building Square Feet  Estimate (Land)  Roof Material_Other  \
130829                  7.870166            40930                  0.0
193890                  7.002156            23000                  0.0
30507                   6.851185            43100                  0.0
91308                   7.228388            56400                  0.0
131132                  7.990915            60260                  0.0

        Roof Material_Shake  Roof Material_Shingle/Asphalt  \
130829                  0.0                            1.0
193890                  0.0                            1.0
30507                   0.0                            1.0
91308                   0.0                            1.0
131132                  0.0                            1.0

        Roof Material_Slate  Roof Material_Tar&Gravel  Roof Material_Tile
130829                  0.0                       0.0                 0.0
193890                  0.0                       0.0                 0.0
30507                   0.0                       0.0                 0.0
91308                   0.0                       0.0                 0.0
131132                  0.0                       0.0                 0.0
```

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

```
        Log Building Square Feet  Estimate (Land)  Roof Material_Other  \
50636                   7.310550            56250                  0.0
82485                   7.325808            42160                  0.0
193966                  7.090077            20590                  0.0
160612                  7.281386            38590                  0.0
7028                    7.118016            35340                  0.0

        Roof Material_Shake  Roof Material_Shingle/Asphalt  \
50636                   0.0                            1.0
82485                   0.0                            1.0
193966                  0.0                            1.0
160612                  0.0                            1.0
7028                    0.0                            1.0
```

```
        Roof Material_Slate  Roof Material_Tar&Gravel  Roof Material_Tile
50636                   0.0                       0.0                 0.0
82485                   0.0                       0.0                 0.0
193966                  0.0                       0.0                 0.0
160612                  0.0                       0.0                 0.0
7028                    0.0                       0.0                 0.0
```

```
50636     11.682668
82485     12.820655
193966     9.825526
160612    12.468437
7028      12.254863
Name: Log Sale Price, dtype: float64
```

In [46]:
```
# Modeling STEP 2:  Create a Multiple Linear Regression Model

# Be sure to set fit_intercept to False, since we are incorporating one-hot-encoded data
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 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)
```

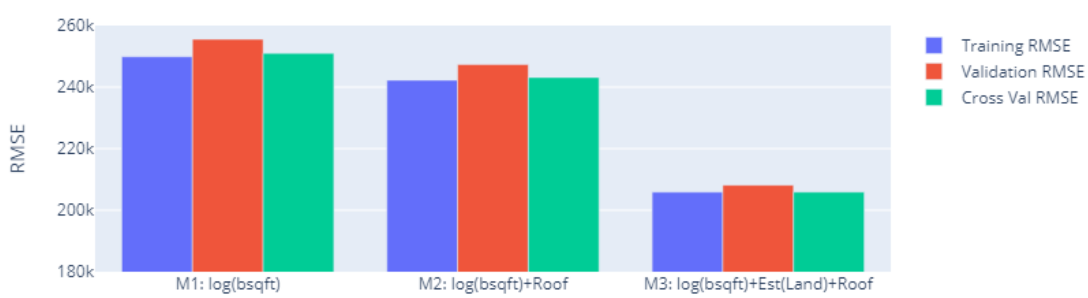
In [47]:
```
# MODELING STEP 3:  Evaluate the RMSE for your model

# Training and test errors for the model (in its units of Log Sale Price)

training_error_log[2] = rmse(Y_train_m3, Y_predict_train_m3)
validation_error_log[2]= rmse(Y_valid_m3, Y_predict_valid_m3)

# Training and test errors for the model (in its original values before the log transform)
training_error[2] = rmse(np.exp(Y_train_m3), np.exp(Y_predict_train_m3))
validation_error[2] = rmse(np.exp(Y_valid_m3), np.exp(Y_predict_valid_m3))

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

```
3rd Model
Training RMSE (log): 0.6582055898103141
Validation RMSE (log): 0.6590140311365982

3rd Model
Training RMSE: 205871.37132454585
Validation RMSE: 208131.31102327758
```

```
In [48]: # MODELING STEP 4:  Conduct 5-fold cross validation for model and output RMSE

         linear_model_m3_cv = lm.LinearRegression(fit_intercept=False)
         processed_full_m3 = process_data_m3(training_val_data)

         X_full_m3 = processed_full_m3.drop(columns="Log Sale Price")
         Y_full_m3 = processed_full_m3["Log Sale Price"]

         np.random.seed(1330)

         # your code above this line to use 5-fold cross-validation and output RMSE (in units of dollar

         cv_error[2] = cross_validate_rmse(linear_model_m3_cv, X_full_m3, Y_full_m3)

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

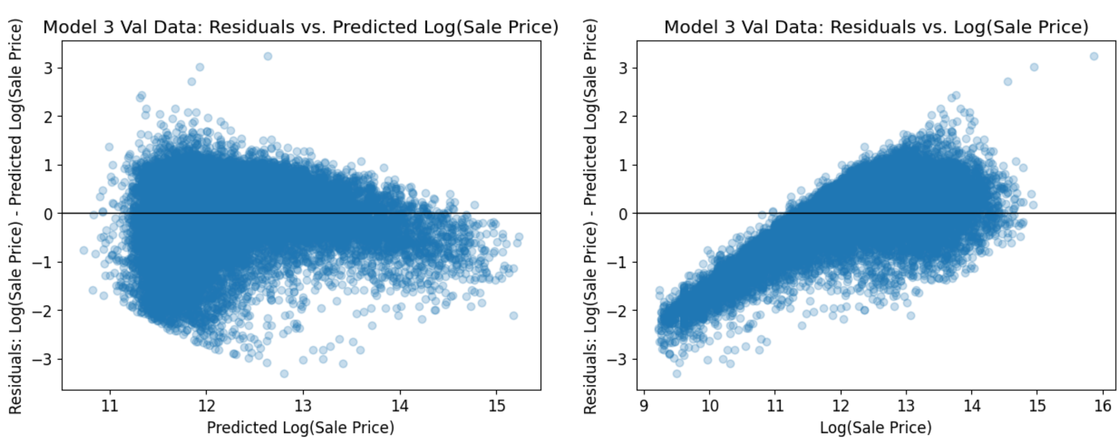
```
3rd Model Cross Validation RMSE: 205877.36763002165
```

```
In [49]: # MODELING STEP 5:  Add a name for your 3rd model describing the features and run this cell to

         model_names[2] = "M3: log(bsqft)+Est(Land)+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 [50]:
```
# MODELING STEP 5 cont'd:  Plot 2 side-by-side residual plots (similar to Question 3, for vali

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

res3 = Y_valid_m3 - Y_predict_valid_m3

x_plt1 = Y_predict_valid_m3
y_plt1 = res3

x_plt2 = Y_valid_m3
y_plt2 = res3


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[50]:
```
Text(0.5, 1.0, 'Model 3 Val Data: Residuals vs. Log(Sale Price)')
```



0.3.4 Question 5c

i). Comment on your RMSE and residual plots from Model 3 compared to the first 2 models.

ii). 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?

iii). If you had more time to improve your model, what would your next steps be?

i) the RMSE from model 3 is substantially lower than models 1 and 2 and the residual plots are starting to tighten up instead of being so spread out like models 1 and 2.

ii) They are sadly still showing the same trend. In the next round of modeling, I could continue adding in variables like the one I was saying was also a decent candidate: Land Square Feet, Lot Size, and/or Age. I could also try taking the log of Estimate (Land) to see how that changes things.

iii) First I would try the log of Estimate (Land) to see if that helps. If it doesn't help, or doesn't help enough, I would start adding in more Variables. Honestly, if we really wanted to get a good fit, wouldn't more variables be better, so maybe add all of them and see. Hopefully it wouldn't become overfit though.

0.4 Question 6: Evaluating the Model in Context

0.5 Question 6a

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.

If their residual is positive, it means their house value has been undervalued by the model, so actual value (y) is greater than predicted value (yhat), and the owner pays less in property taxes than they actually should be paying. If their residual is negative, it means their house value has been overvalued by the model, so actual value (y) is less than predicted value (yhat), and the owner pays more in property taxes than they should be paying.

0.6 Question 6b

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 model's predictions are accurate when it can correctly predict values close to the actual property values, but a low RMSE won't guarantee that EVERY individual prediction is precise and it doesn't account for systemic biases. For true fairness, we would have to ensure that no specific group of homeowners (race, income, location, etc.) is systematically over or undervalued. Even an accurate model can reinforce systemic bias if it doesn't account for the historical and social contexts of the areas it assesses. Undervalueing properties in white neighborhoods and overvalueing properties in black/hispanic neighborhoods would only exacerbate the present wealth disparities. True fairness wouldn't only include accuracy, it would ideally include justice measures like reparations, actively working to mitigate and correct historical/systemic biases.

0.7 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)

In [54]:
```
# Modeling Step 1:  Process the Data




# Hint: You can either use your implementation of the One Hot Encoding Function from Project P
#from feature_func import *


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


def process_data_ec(data):

    # You should start by only keeping values with Pure Market Filter = 1
    ...

    return data


# Process the data
processed_train_ec = ...

processed_val_ec = ...


X_train_ec = ...
Y_train_ec = ...

X_valid_ec = ...
Y_valid_ec = ...


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

#display(X_valid_m3.head())
#display(Y_valid_m3.head())
```

In [55]:
```
# Modeling STEP 2:  Create a Multiple Linear Regression Model

# If you are are incorporating one-hot-encoded data, set the fit_intercept to False
```

```
...
# your code above this line to create regression model for Model 2

Y_predict_train_ec = ...

Y_predict_valid_ec = ...
```

In [56]:
```
# MODELING STEP 3:  Evaluate the RMSE for your model


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


print("Extra Credit Model\nTraining RMSE (log): {}\nValidation RMSE (log): {}\n".format(traini
print("Extra Credit \nTraining RMSE: {}\nValidation RMSE: {}\n".format(training_error_ec, vali
```

```
Extra Credit Model
Training RMSE (log): Ellipsis
Validation RMSE (log): Ellipsis

Extra Credit
Training RMSE: Ellipsis
Validation RMSE: Ellipsis
```

In [57]:
```
# Optional: Run this cell to visualize

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")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File /opt/conda/lib/python3.10/site-packages/IPython/core/formatters.py:920, in IPython.DisplayFormatte
    918 method = get_real_method(obj, self.print_method)
    919 if method is not None:
--> 920     method()
    921     return True
```

```
File /opt/conda/lib/python3.10/site-packages/plotly/basedatatypes.py:834, in BaseFigure._ipython_displ
    831 import plotly.io as pio
    833 if pio.renderers.render_on_display and pio.renderers.default:
--> 834     pio.show(self)
    835 else:
    836     print(repr(self))

File /opt/conda/lib/python3.10/site-packages/plotly/io/_renderers.py:388, in show(fig, renderer, valid
    385 fig_dict = validate_coerce_fig_to_dict(fig, validate)
    387 # Mimetype renderers
--> 388 bundle = renderers._build_mime_bundle(fig_dict, renderers_string=renderer, **kwargs)
    389 if bundle:
    390     if not ipython_display:

File /opt/conda/lib/python3.10/site-packages/plotly/io/_renderers.py:296, in RenderersConfig._build_mi
    293             if hasattr(renderer, k):
    294                 setattr(renderer, k, v)
--> 296         bundle.update(renderer.to_mimebundle(fig_dict))
    298 return bundle

File /opt/conda/lib/python3.10/site-packages/plotly/io/_base_renderers.py:95, in PlotlyRenderer.to_mim
     91 if config:
     92     fig_dict["config"] = config
     94 json_compatible_fig_dict = json.loads(
---> 95     to_json(fig_dict, validate=False, remove_uids=False)
     96 )
     98 return {"application/vnd.plotly.v1+json": json_compatible_fig_dict}

File /opt/conda/lib/python3.10/site-packages/plotly/io/_json.py:199, in to_json(fig, validate, pretty,
    196     for trace in fig_dict.get("data", []):
    197         trace.pop("uid", None)
--> 199 return to_json_plotly(fig_dict, pretty=pretty, engine=engine)

File /opt/conda/lib/python3.10/site-packages/plotly/io/_json.py:123, in to_json_plotly(plotly_object,
    119         opts["separators"] = (",", ":")
    121     from _plotly_utils.utils import PlotlyJSONEncoder
--> 123     return json.dumps(plotly_object, cls=PlotlyJSONEncoder, **opts)
    124 elif engine == "orjson":
    125     JsonConfig.validate_orjson()

File /opt/conda/lib/python3.10/json/__init__.py:238, in dumps(obj, skipkeys, ensure_ascii, check_circu
    232 if cls is None:
    233     cls = JSONEncoder
    234 return cls(
    235     skipkeys=skipkeys, ensure_ascii=ensure_ascii,
    236     check_circular=check_circular, allow_nan=allow_nan, indent=indent,
    237     separators=separators, default=default, sort_keys=sort_keys,
--> 238     **kw).encode(obj)

File /opt/conda/lib/python3.10/site-packages/_plotly_utils/utils.py:59, in PlotlyJSONEncoder.encode(se
     52 """
     53 Load and then dump the result using parse_constant kwarg
     54
```

```
     55 Note that setting invalid separators will cause a failure at this step.
     56 
     57 """
     58 # this will raise errors in a normal-expected way
---> 59 encoded_o = super(PlotlyJSONEncoder, self).encode(o)
     60 # Brute force guessing whether NaN or Infinity values are in the string
     61 # We catch false positive cases (e.g. strings such as titles, labels etc.)
     62 # but this is ok since the intention is to skip the decoding / reencoding
     63 # step when it's completely safe
     65 if not ("NaN" in encoded_o or "Infinity" in encoded_o):

File /opt/conda/lib/python3.10/json/encoder.py:199, in JSONEncoder.encode(self, o)
    195         return encode_basestring(o)
    196 # This doesn't pass the iterator directly to ''.join() because the
    197 # exceptions aren't as detailed.  The list call should be roughly
    198 # equivalent to the PySequence_Fast that ''.join() would do.
--> 199 chunks = self.iterencode(o, _one_shot=True)
    200 if not isinstance(chunks, (list, tuple)):
    201     chunks = list(chunks)

File /opt/conda/lib/python3.10/json/encoder.py:257, in JSONEncoder.iterencode(self, o, _one_shot)
    252 else:
    253     _iterencode = _make_iterencode(
    254         markers, self.default, _encoder, self.indent, floatstr,
    255         self.key_separator, self.item_separator, self.sort_keys,
    256         self.skipkeys, _one_shot)
--> 257 return _iterencode(o, 0)

File /opt/conda/lib/python3.10/site-packages/_plotly_utils/utils.py:136, in PlotlyJSONEncoder.default(
    134     except NotEncodable:
    135         pass
--> 136 return _json.JSONEncoder.default(self, obj)

File /opt/conda/lib/python3.10/json/encoder.py:179, in JSONEncoder.default(self, o)
    160 def default(self, o):
    161     """Implement this method in a subclass such that it returns
    162     a serializable object for ``o``, or calls the base implementation
    163     (to raise a ``TypeError``).
   (…)
    177 
    178     """
--> 179     raise TypeError(f'Object of type {o.__class__.__name__} '
    180                     f'is not JSON serializable')

TypeError: Object of type ellipsis is not JSON serializable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File /opt/conda/lib/python3.10/site-packages/IPython/core/formatters.py:972, in MimeBundleFormatter.__
    969     method = get_real_method(obj, self.print_method)
    971     if method is not None:
```

```
--> 972         return method(include=include, exclude=exclude)
    973     return None
    974 else:

File /opt/conda/lib/python3.10/site-packages/plotly/basedatatypes.py:825, in BaseFigure._repr_mimebund
    822 from plotly.io._utils import validate_coerce_fig_to_dict
    824 fig_dict = validate_coerce_fig_to_dict(self, validate)
--> 825 return renderers._build_mime_bundle(fig_dict, renderer_str, **kwargs)

File /opt/conda/lib/python3.10/site-packages/plotly/io/_renderers.py:296, in RenderersConfig._build_mi
    293             if hasattr(renderer, k):
    294                 setattr(renderer, k, v)
--> 296         bundle.update(renderer.to_mimebundle(fig_dict))
    298 return bundle

File /opt/conda/lib/python3.10/site-packages/plotly/io/_base_renderers.py:95, in PlotlyRenderer.to_mim
     91 if config:
     92     fig_dict["config"] = config
     94 json_compatible_fig_dict = json.loads(
---> 95     to_json(fig_dict, validate=False, remove_uids=False)
     96 )
     98 return {"application/vnd.plotly.v1+json": json_compatible_fig_dict}

File /opt/conda/lib/python3.10/site-packages/plotly/io/_json.py:199, in to_json(fig, validate, pretty,
    196     for trace in fig_dict.get("data", []):
    197         trace.pop("uid", None)
--> 199 return to_json_plotly(fig_dict, pretty=pretty, engine=engine)

File /opt/conda/lib/python3.10/site-packages/plotly/io/_json.py:123, in to_json_plotly(plotly_object,
    119         opts["separators"] = (",", ":")
    121     from _plotly_utils.utils import PlotlyJSONEncoder
--> 123     return json.dumps(plotly_object, cls=PlotlyJSONEncoder, **opts)
    124 elif engine == "orjson":
    125     JsonConfig.validate_orjson()

File /opt/conda/lib/python3.10/json/__init__.py:238, in dumps(obj, skipkeys, ensure_ascii, check_circu
    232 if cls is None:
    233     cls = JSONEncoder
    234 return cls(
    235     skipkeys=skipkeys, ensure_ascii=ensure_ascii,
    236     check_circular=check_circular, allow_nan=allow_nan, indent=indent,
    237     separators=separators, default=default, sort_keys=sort_keys,
--> 238     **kw).encode(obj)

File /opt/conda/lib/python3.10/site-packages/_plotly_utils/utils.py:59, in PlotlyJSONEncoder.encode(se
     52 """
     53 Load and then dump the result using parse_constant kwarg
     54 
     55 Note that setting invalid separators will cause a failure at this step.
     56 
     57 """
     58 # this will raise errors in a normal-expected way
---> 59 encoded_o = super(PlotlyJSONEncoder, self).encode(o)
     60 # Brute force guessing whether NaN or Infinity values are in the string
```

```
     61 # We catch false positive cases (e.g. strings such as titles, labels etc.)
     62 # but this is ok since the intention is to skip the decoding / reencoding
     63 # step when it's completely safe
     65 if not ("NaN" in encoded_o or "Infinity" in encoded_o):

File /opt/conda/lib/python3.10/json/encoder.py:199, in JSONEncoder.encode(self, o)
    195         return encode_basestring(o)
    196 # This doesn't pass the iterator directly to ''.join() because the
    197 # exceptions aren't as detailed.  The list call should be roughly
    198 # equivalent to the PySequence_Fast that ''.join() would do.
--> 199 chunks = self.iterencode(o, _one_shot=True)
    200 if not isinstance(chunks, (list, tuple)):
    201     chunks = list(chunks)

File /opt/conda/lib/python3.10/json/encoder.py:257, in JSONEncoder.iterencode(self, o, _one_shot)
    252 else:
    253     _iterencode = _make_iterencode(
    254         markers, self.default, _encoder, self.indent, floatstr,
    255         self.key_separator, self.item_separator, self.sort_keys,
    256         self.skipkeys, _one_shot)
--> 257 return _iterencode(o, 0)

File /opt/conda/lib/python3.10/site-packages/_plotly_utils/utils.py:136, in PlotlyJSONEncoder.default(
    134     except NotEncodable:
    135         pass
--> 136 return _json.JSONEncoder.default(self, obj)

File /opt/conda/lib/python3.10/json/encoder.py:179, in JSONEncoder.default(self, o)
    160 def default(self, o):
    161     """Implement this method in a subclass such that it returns
    162     a serializable object for ``o``, or calls the base implementation
    163     (to raise a ``TypeError``).
   (…)
    177
    178     """
--> 179     raise TypeError(f'Object of type {o.__class__.__name__} '
    180                     f'is not JSON serializable')

TypeError: Object of type ellipsis is not JSON serializable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File /opt/conda/lib/python3.10/site-packages/IPython/core/formatters.py:342, in BaseFormatter.__call__
    340     method = get_real_method(obj, self.print_method)
    341     if method is not None:
--> 342         return method()
    343     return None
    344 else:

File /opt/conda/lib/python3.10/site-packages/plotly/basedatatypes.py:808, in BaseFigure._repr_html_(se
    804 def _repr_html_(self):
```

```
    805     """
    806     Customize html representation
    807     """
--> 808     bundle = self._repr_mimebundle_()
    809     if "text/html" in bundle:
    810         return bundle["text/html"]

File /opt/conda/lib/python3.10/site-packages/plotly/basedatatypes.py:825, in BaseFigure._repr_mimebund
    822 from plotly.io._utils import validate_coerce_fig_to_dict
    824 fig_dict = validate_coerce_fig_to_dict(self, validate)
--> 825 return renderers._build_mime_bundle(fig_dict, renderer_str, **kwargs)

File /opt/conda/lib/python3.10/site-packages/plotly/io/_renderers.py:296, in RenderersConfig._build_mi
    293             if hasattr(renderer, k):
    294                 setattr(renderer, k, v)
--> 296         bundle.update(renderer.to_mimebundle(fig_dict))
    298 return bundle

File /opt/conda/lib/python3.10/site-packages/plotly/io/_base_renderers.py:95, in PlotlyRenderer.to_mim
     91 if config:
     92     fig_dict["config"] = config
     94 json_compatible_fig_dict = json.loads(
---> 95     to_json(fig_dict, validate=False, remove_uids=False)
     96 )
     98 return {"application/vnd.plotly.v1+json": json_compatible_fig_dict}

File /opt/conda/lib/python3.10/site-packages/plotly/io/_json.py:199, in to_json(fig, validate, pretty,
    196     for trace in fig_dict.get("data", []):
    197         trace.pop("uid", None)
--> 199 return to_json_plotly(fig_dict, pretty=pretty, engine=engine)

File /opt/conda/lib/python3.10/site-packages/plotly/io/_json.py:123, in to_json_plotly(plotly_object,
    119         opts["separators"] = (",", ":")
    121     from _plotly_utils.utils import PlotlyJSONEncoder
--> 123     return json.dumps(plotly_object, cls=PlotlyJSONEncoder, **opts)
    124 elif engine == "orjson":
    125     JsonConfig.validate_orjson()

File /opt/conda/lib/python3.10/json/__init__.py:238, in dumps(obj, skipkeys, ensure_ascii, check_circu
    232 if cls is None:
    233     cls = JSONEncoder
    234 return cls(
    235     skipkeys=skipkeys, ensure_ascii=ensure_ascii,
    236     check_circular=check_circular, allow_nan=allow_nan, indent=indent,
    237     separators=separators, default=default, sort_keys=sort_keys,
--> 238     **kw).encode(obj)

File /opt/conda/lib/python3.10/site-packages/_plotly_utils/utils.py:59, in PlotlyJSONEncoder.encode(se
     52 """
     53 Load and then dump the result using parse_constant kwarg
     54 
     55 Note that setting invalid separators will cause a failure at this step.
     56 
     57 """
```

```
     58 # this will raise errors in a normal-expected way
---> 59 encoded_o = super(PlotlyJSONEncoder, self).encode(o)
     60 # Brute force guessing whether NaN or Infinity values are in the string
     61 # We catch false positive cases (e.g. strings such as titles, labels etc.)
     62 # but this is ok since the intention is to skip the decoding / reencoding
     63 # step when it's completely safe
     65 if not ("NaN" in encoded_o or "Infinity" in encoded_o):

File /opt/conda/lib/python3.10/json/encoder.py:199, in JSONEncoder.encode(self, o)
    195         return encode_basestring(o)
    196 # This doesn't pass the iterator directly to ''.join() because the
    197 # exceptions aren't as detailed.  The list call should be roughly
    198 # equivalent to the PySequence_Fast that ''.join() would do.
--> 199 chunks = self.iterencode(o, _one_shot=True)
    200 if not isinstance(chunks, (list, tuple)):
    201     chunks = list(chunks)

File /opt/conda/lib/python3.10/json/encoder.py:257, in JSONEncoder.iterencode(self, o, _one_shot)
    252 else:
    253     _iterencode = _make_iterencode(
    254         markers, self.default, _encoder, self.indent, floatstr,
    255         self.key_separator, self.item_separator, self.sort_keys,
    256         self.skipkeys, _one_shot)
--> 257 return _iterencode(o, 0)

File /opt/conda/lib/python3.10/site-packages/_plotly_utils/utils.py:136, in PlotlyJSONEncoder.default(
    134     except NotEncodable:
    135         pass
--> 136 return _json.JSONEncoder.default(self, obj)

File /opt/conda/lib/python3.10/json/encoder.py:179, in JSONEncoder.default(self, o)
    160 def default(self, o):
    161     """Implement this method in a subclass such that it returns
    162     a serializable object for ``o``, or calls the base implementation
    163     (to raise a ``TypeError``).
   (…)
    177
    178     """
--> 179     raise TypeError(f'Object of type {o.__class__.__name__} '
    180                     f'is not JSON serializable')

TypeError: Object of type ellipsis is not JSON serializable
```

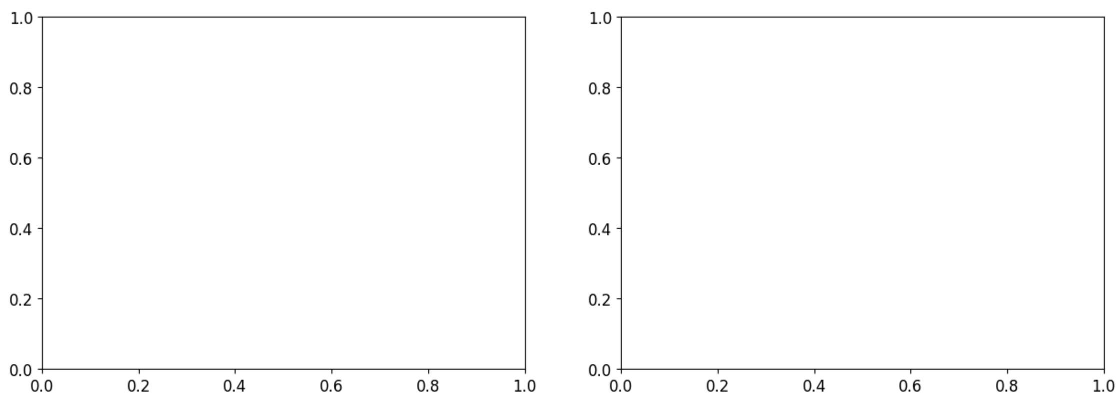
```
Out[57]: Figure({
             'data': [{'name': 'Training RMSE', 'type': 'bar', 'x': ['Extra Credit Model'], 'y': [Ellips
                       {'name': 'Validation RMSE', 'type': 'bar', 'x': ['Extra Credit Model'], 'y': [Ell
             'layout': {'template': '…', 'yaxis': {'range': [140000, 260000], 'title': {'text': 'RMSE'}
         })
```

```
In [58]: # MODELING STEP 5: Plot 2 side-by-side residual plots for validation data
```

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


x_plt1 = ...
y_plt1 = ...

x_plt2 = ...
y_plt2 = ...


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)")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[58], line 13
      9 x_plt2 = ...
     10 y_plt2 = ...
---> 13 ax[0].scatter(x_plt1, y_plt1, alpha=.25)
     14 ax[0].axhline(0, c='black', linewidth=1)
     15 ax[0].set_xlabel(r'Predicted Log(Sale Price)')

File /opt/conda/lib/python3.10/site-packages/matplotlib/__init__.py:1442, in _preprocess_data.<locals>
   1439 @functools.wraps(func)
   1440 def inner(ax, *args, data=None, **kwargs):
   1441     if data is None:
-> 1442         return func(ax, *map(sanitize_sequence, args), **kwargs)
   1444     bound = new_sig.bind(ax, *args, **kwargs)
   1445     auto_label = (bound.arguments.get(label_namer)
   1446                   or bound.kwargs.get(label_namer))

File /opt/conda/lib/python3.10/site-packages/matplotlib/axes/_axes.py:4673, in Axes.scatter(self, x, y
   4667         linewidths = [
   4668             lw if lw is not None else mpl.rcParams['lines.linewidth']
   4669             for lw in linewidths]
   4671 offsets = np.ma.column_stack([x, y])
-> 4673 collection = mcoll.PathCollection(
   4674     (path,), scales,
   4675     facecolors=colors,
   4676     edgecolors=edgecolors,
   4677     linewidths=linewidths,
   4678     offsets=offsets,
   4679     offset_transform=kwargs.pop('transform', self.transData),
```

```
   4680     alpha=alpha,
   4681 )
   4682 collection.set_transform(mtransforms.IdentityTransform())
   4683 if colors is None:

File /opt/conda/lib/python3.10/site-packages/matplotlib/collections.py:994, in PathCollection.__init__
    980 def __init__(self, paths, sizes=None, **kwargs):
    981     """
    982     Parameters
    983     ----------
   (…)
    991         Forwarded to `.Collection`.
    992     """
--> 994     super().__init__(**kwargs)
    995     self.set_paths(paths)
    996     self.set_sizes(sizes)

File /opt/conda/lib/python3.10/site-packages/matplotlib/_api/deprecation.py:454, in make_keyword_only.
    448 if len(args) > name_idx:
    449     warn_deprecated(
    450         since, message="Passing the %(name)s %(obj_type)s "
    451         "positionally is deprecated since Matplotlib %(since)s; the "
    452         "parameter will become keyword-only %(removal)s.",
    453         name=name, obj_type=f"parameter of {func.__name__}()")
--> 454 return func(*args, **kwargs)

File /opt/conda/lib/python3.10/site-packages/matplotlib/collections.py:192, in Collection.__init__(sel
    189     self._joinstyle = None
    191 if offsets is not None:
--> 192     offsets = np.asanyarray(offsets, float)
    193     # Broadcast (2,) -> (1, 2) but nothing else.
    194     if offsets.shape == (2,):

TypeError: float() argument must be a string or a real number, not 'ellipsis'
```



0.8 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?

Type your answer here, replacing this text.