3 Jan’26

TAVISHEE AGRWAL
23119050 (PI-3rd yr)

SATELLITE IMAGERY BASED PROPERTY EVALUATION

OVERVIEW

Problem Statement

The goal of this project is to predict residential house prices using:

- Structured tabular features (size, location, quality, condition)
- Unstructured satellite imagery capturing neighbourhood and environmental context

Traditional models rely heavily on tabular data. This project primarily explores the need for satellite imagery data for housing price prediction and also more importantly whether satellite imagery can actually add any complementary spatial information to improve predictive performance.

Instead of straight away jumping to multimodal pipelines I decided to first carry out a detailed analysis of tabular features and build models solely on tabular data features , assess their performance on validation data and get sort of a baseline score to compare my future models against it.

APPROACHES

I implemented mainly three modeling pipelines:

1. Tabular-only baseline models
    - Linear Regression
    - Gradient Boosting
    - Feedforward Neural Network
2. Image-only feature extraction
    - Pretrained ResNet-50 CNN (ImageNet weights)
3. Multimodal fusion model

- CNN for images
- MLP for tabular data
- Late fusion through feature concatenation

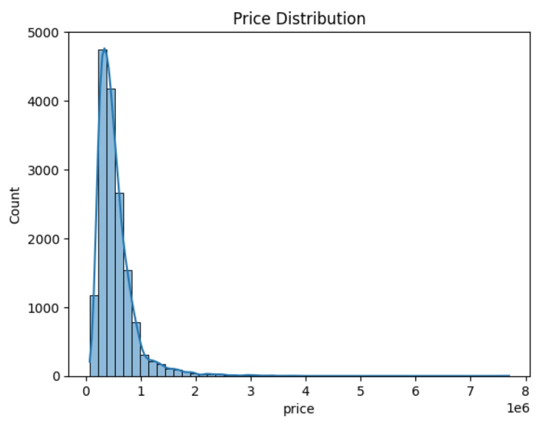
Key design principles:

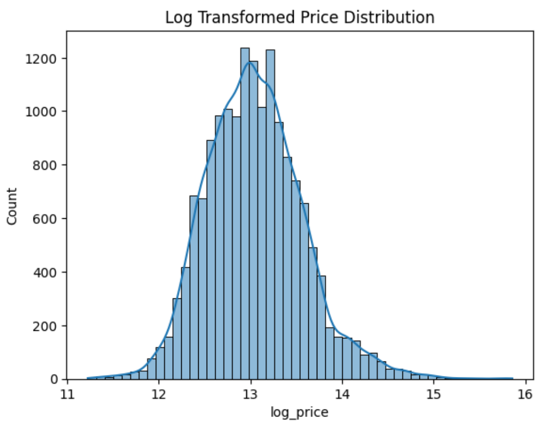
- I used log-transformed prices for stability.
- I decided to initially freeze the resnet weights in order to avoid overfitting.
- I used late fusion to allow each modality to learn independently.
- Then ultimately comparing multimodal performance directly against tabular baselines.

EDA

- Price Distribution

The prices graph was right skewed, with a small number of very high-value properties. as mostly seen with housing datasets so in order to stabilize variance and improve model learning, the price variable was log transformed .

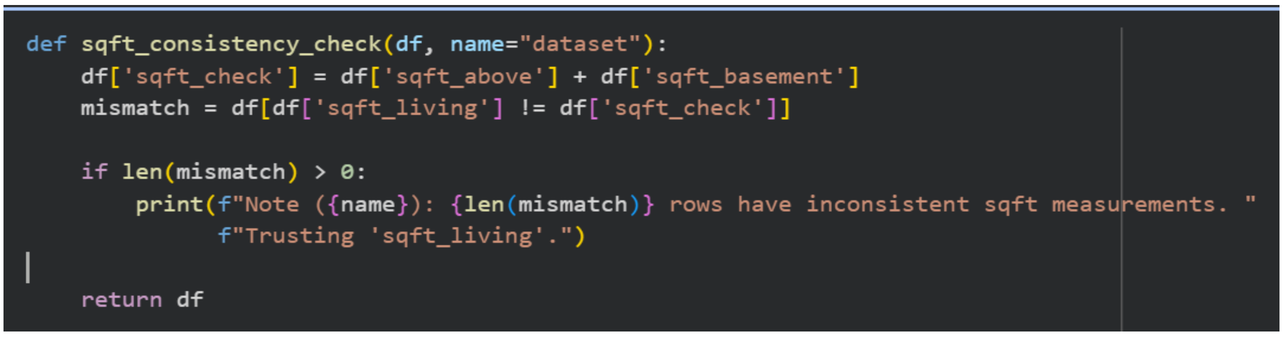




- There were no missing values or duplicate values in the dataset.

- According to the dataset ‘sqft_living’ = ’sqft_basement’ + ‘sqft_above’ ,

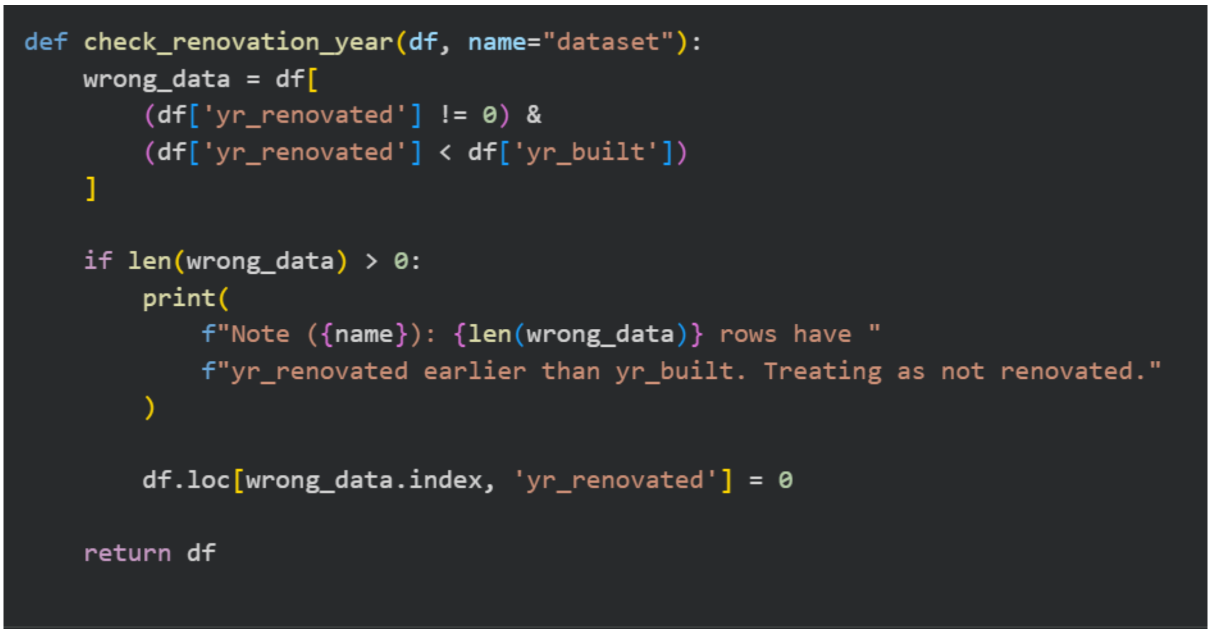
  So I created a function



```python
def sqft_consistency_check(df, name="dataset"):
    df['sqft_check'] = df['sqft_above'] + df['sqft_basement']
    mismatch = df[df['sqft_living'] != df['sqft_check']]

    if len(mismatch) > 0:
        print(f"Note ({name}): {len(mismatch)} rows have inconsistent sqft measurements. "
              f"Trusting 'sqft_living'.")

    return df
```

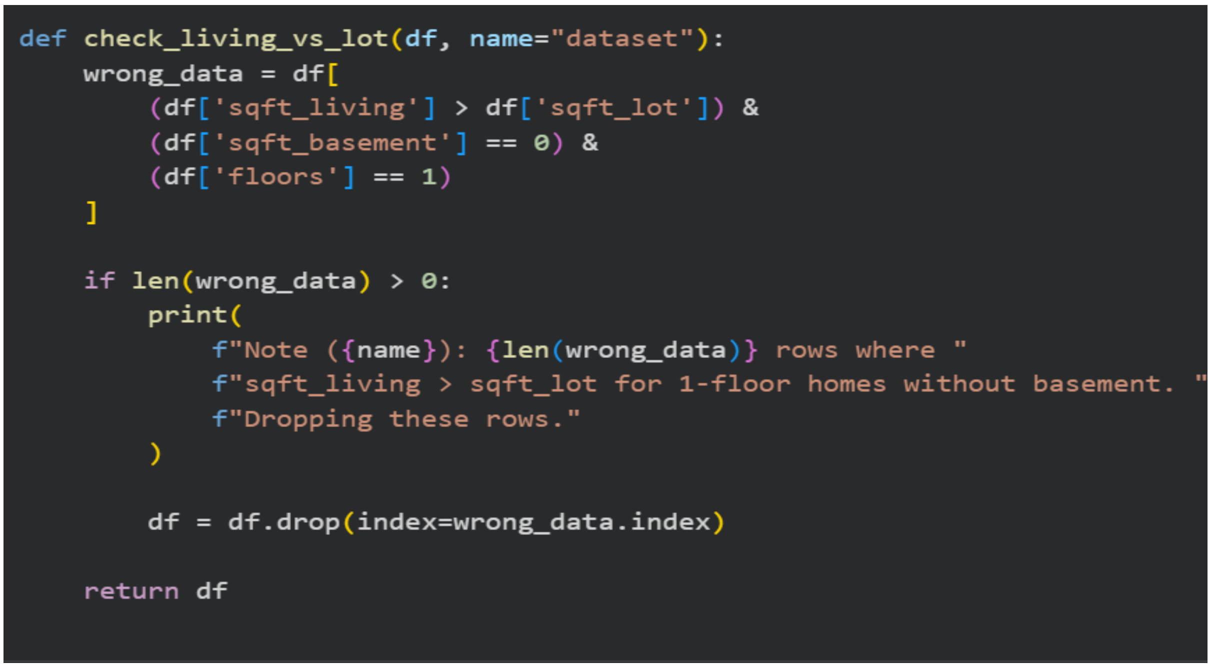
  To check for inconsistencies but found none.

- Then the year in which the house was renovated cannot be earlier than the year in which the house was built hence , I created an additional function to check this but didn’t found any error.



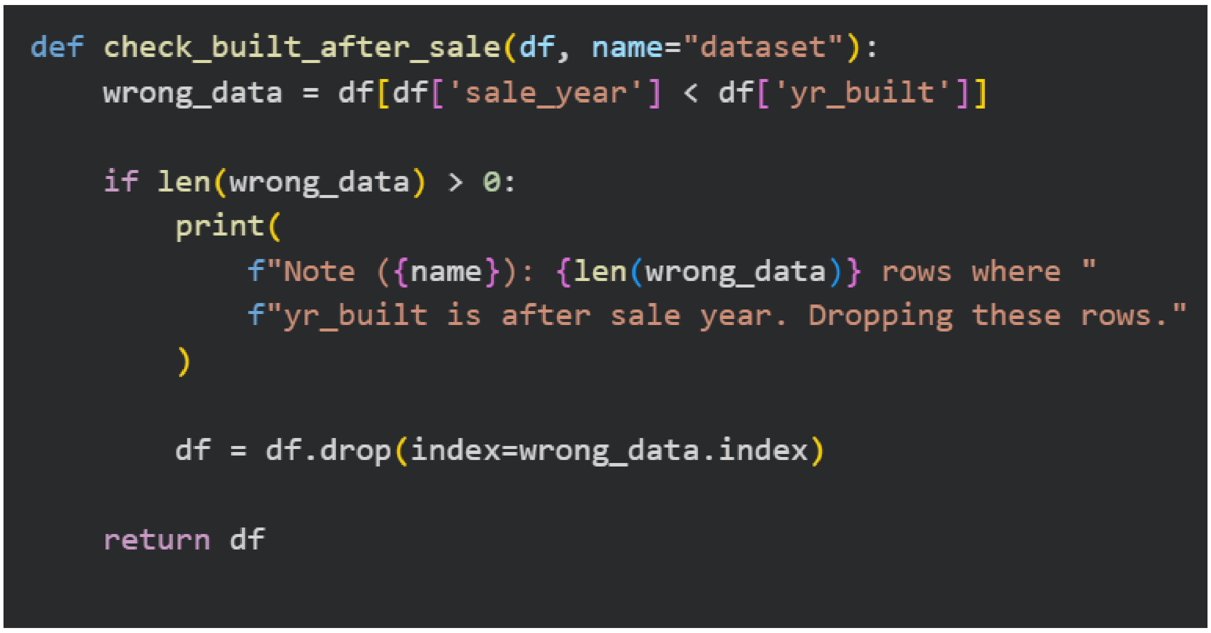
```python
def check_renovation_year(df, name="dataset"):
    wrong_data = df[
        (df['yr_renovated'] != 0) &
        (df['yr_renovated'] < df['yr_built'])
    ]

    if len(wrong_data) > 0:
        print(
            f"Note ({name}): {len(wrong_data)} rows have "
            f"yr_renovated earlier than yr_built. Treating as not renovated."
        )

        df.loc[wrong_data.index, 'yr_renovated'] = 0

    return df
```

- Then for houses of single floor with no basement , sqft_living cant be greater than sqft_lot . No errors found.



```python
def check_living_vs_lot(df, name="dataset"):
    wrong_data = df[
        (df['sqft_living'] > df['sqft_lot']) &
        (df['sqft_basement'] == 0) &
        (df['floors'] == 1)
    ]

    if len(wrong_data) > 0:
        print(
            f"Note ({name}): {len(wrong_data)} rows where "
            f"sqft_living > sqft_lot for 1-floor homes without basement. "
            f"Dropping these rows."
        )

        df = df.drop(index=wrong_data.index)

    return df
```

- The built year cannot be after the sale year .



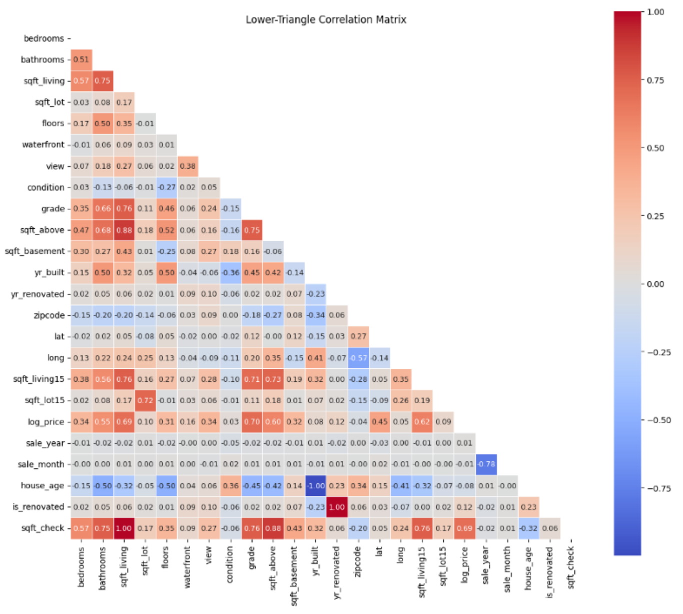
```python
def check_built_after_sale(df, name="dataset"):
    wrong_data = df[df['sale_year'] < df['yr_built']]

    if len(wrong_data) > 0:
        print(
            f"Note ({name}): {len(wrong_data)} rows where "
            f"yr_built is after sale year. Dropping these rows."
        )

        df = df.drop(index=wrong_data.index)

    return df
```

I found 11 rows in the train dataset where such anomalies were present hence I dropped them.

- I engineered some new features like house_age which is essentially sale_year – yr_built . (sale_year was extracted from the date column).

Then instead of using the yr_renovated column I decided to create is_renovated which simply tells whether the house was renovated or not.

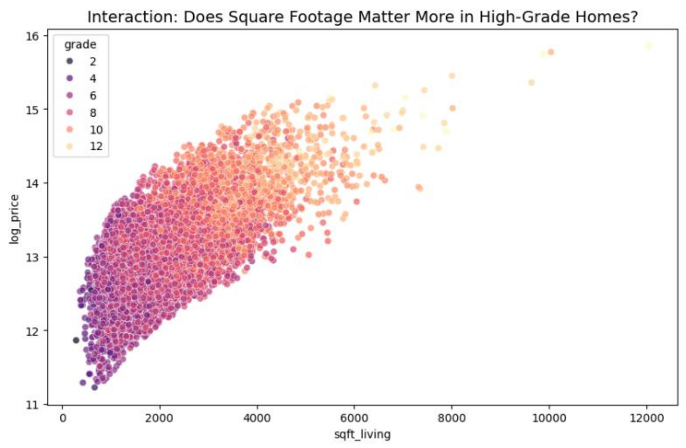
- The correlation feature map showed this .



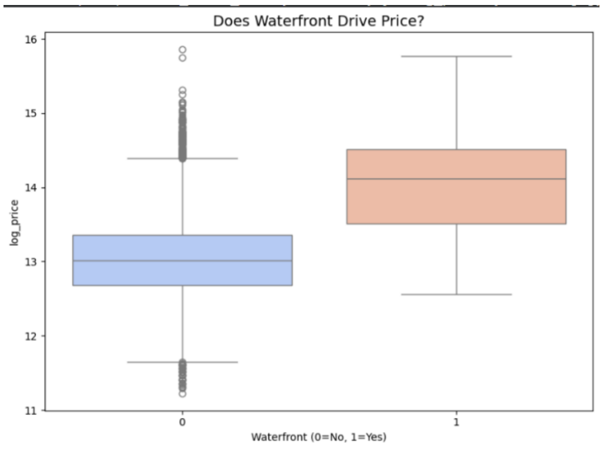
The map showed a very strong correlation between sqft_above and sqft_living as it should be so I planned on not using this later on , similarly during training I will be dropping yr_renovated , yr_built as I now have house_age and is_renovated.

Financial/Visual Insights

- I constructed plots to see how grade of the property affects prices over the sqft_living range.It revealed that square footage contributes more strongly to price in higher-grade homes. While lower-grade properties show diminishing returns to size, high-grade properties exhibit a steeper price increase as square footage grows. This indicates a strong nonlinear interaction between size and quality, motivating the use of nonlinear models such as gradient boosting and neural networks rather than simple linear regression.

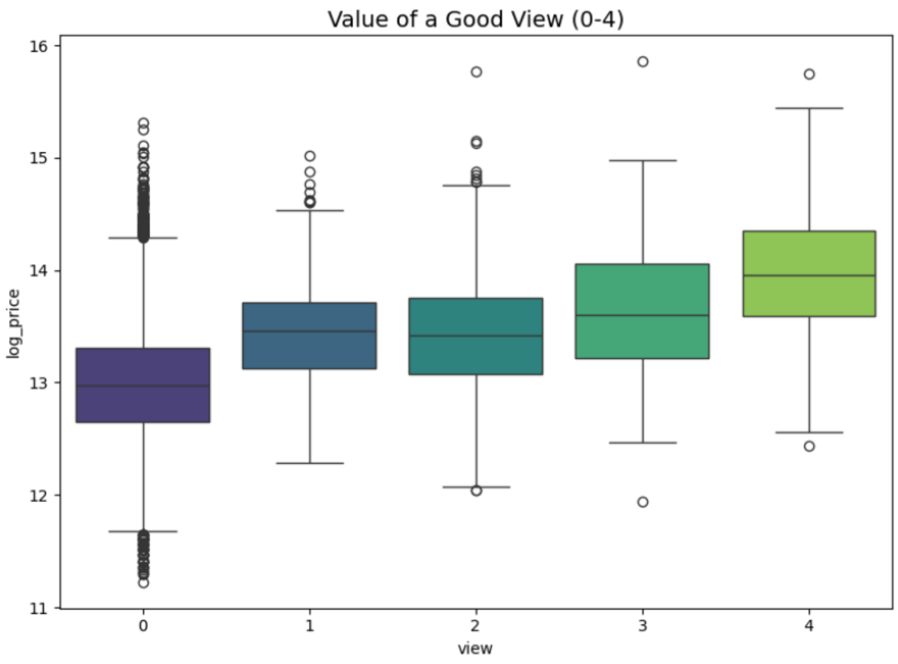


- Waterfront can be a driving factor for prices i.e presence of a waterfront can indicate high prices to observe this I plotted a boxplot between waterfront and log_price.

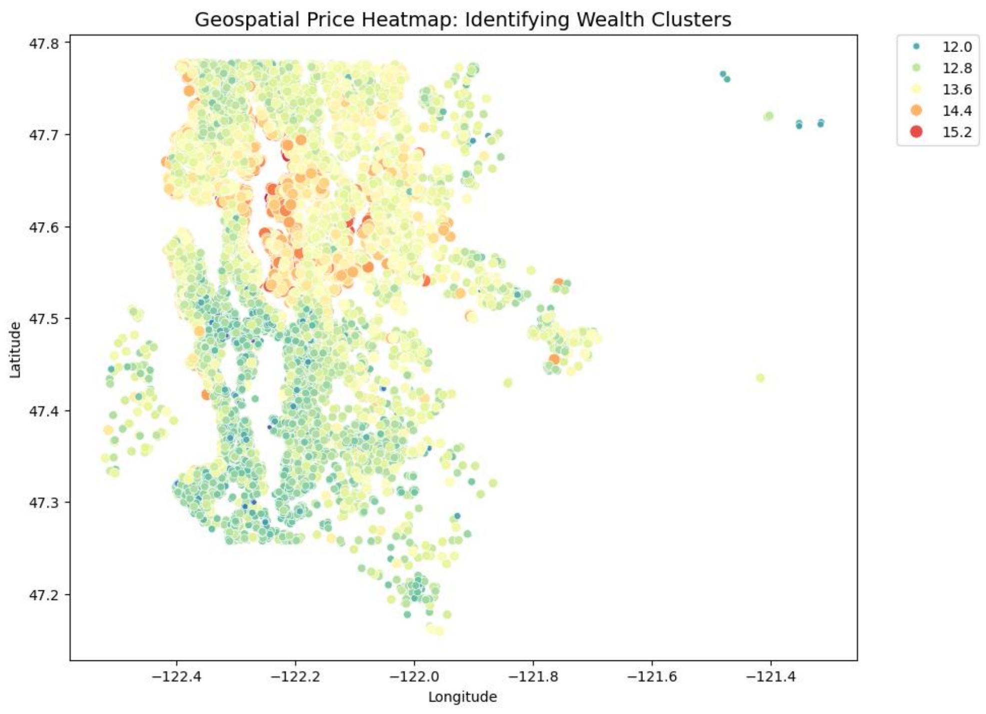


We can see here how properties with a waterfront have on an average higher prices than properties without a waterfront . Though we can see many outliers which don’t have a waterfront but have prices higher , this means that there are other factors which are influencing the prices.

- Same observations can be said for view .



- This plot shows that we can observe clusters of houses in certain price ranges .

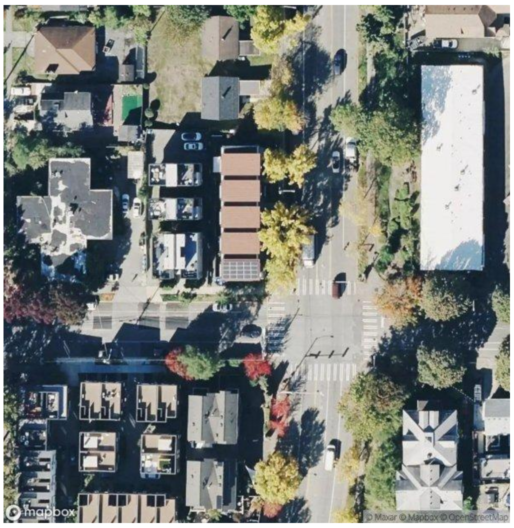


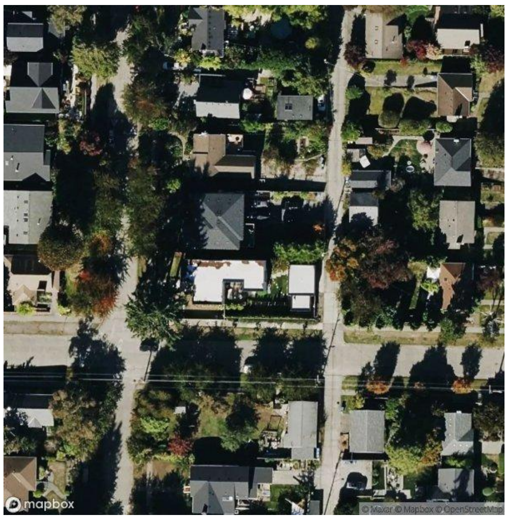
- This means that there are certain regions(marked by yellow and red) where propertied are expensive

All these recent visualizations show that we can benefit from satellite images of houses . The images can capture neighbourhood and environmental factors that otherwise aren't captured in tabular data.

So I extracted the images using the latitude and longitude data given in the tabular dataset through the mapbox api by setting the zoom level to 18 and extracted 600 by 600 pixel images.

Some of the sample images are here :





Satellite images provide overhead views of properties and surrounding areas. Visual patterns observed include:

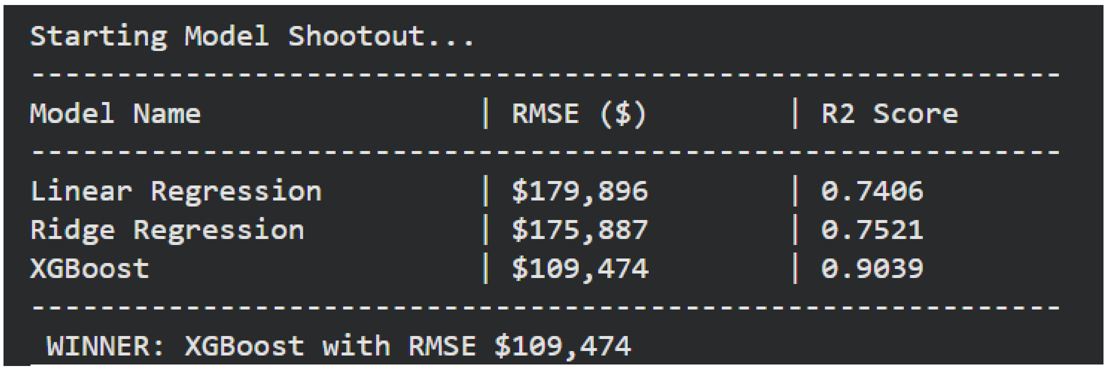
- Dense urban vs. suburban layouts
- Presence of greenery and tree cover
- Proximity to water bodies
- Road density and neighbourhood structure

These visual cues motivated the exploration of a multimodal approach.

Modelling Strategy

I adopted a progressive modeling approach, beginning with strong baselines and gradually increasing complexity.

1. First I decided to train models on solely tabular data features so that we have something to compare against.
   i) Keeping things simple I trained my tabular only data on three different models – Linear Regression , Ridge Regression (for which I scaled the numerical features) and XG Boost .
   Here are the results.



```
Starting Model Shootout...
------------------------------------------------------------
Model Name                | RMSE ($)         | R2 Score
------------------------------------------------------------
Linear Regression         | $179,896         | 0.7406
Ridge Regression          | $175,887         | 0.7521
XGBoost                   | $109,474         | 0.9039
------------------------------------------------------------
 WINNER: XGBoost with RMSE $109,474
```

We got the best results with XG Boost model , as it is also very known to perform well on tabular data. Poor performance of linear and ridge regression shows that we cant simply fit a linear function to our dataset its has to learn non linearity .

ii) Next I thought to experiment a bit more with tabular data and this time with deep learning approaches , I constructed a very simple neural network with this structure.

```
class SimpleTabularNN(nn.Module):
    def __init__(self, input_dim):
        super(SimpleTabularNN, self).__init__()

        self.fc1 = nn.Linear(input_dim, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x.squeeze(1)
```

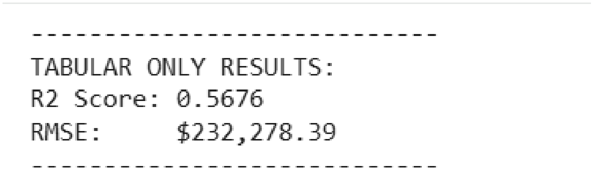
I ran the training for 200 epochs , due to early stopping strategy the training stopped at 78 epoch and gave the following scores.

```
----------------------------
TABULAR ONLY RESULTS:
R2 Score: 0.8500
RMSE:     $136,789.82
----------------------------
```

We can see there is a huge improvement from linear models but it slightly lags behind xg boost.
Later on I tried to make slightly complicated neural network

```
class TabularNN(nn.Module):
    def __init__(self, input_dim):
        super(TabularNN, self).__init__()

        self.network = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.BatchNorm1d(128),
            nn.Dropout(0.3),

            nn.Linear(128, 64),
            nn.ReLU(),
            nn.BatchNorm1d(64),
            nn.Dropout(0.2),

            nn.Linear(64, 32),
            nn.ReLU(),

            nn.Linear(32, 1)
        )
```

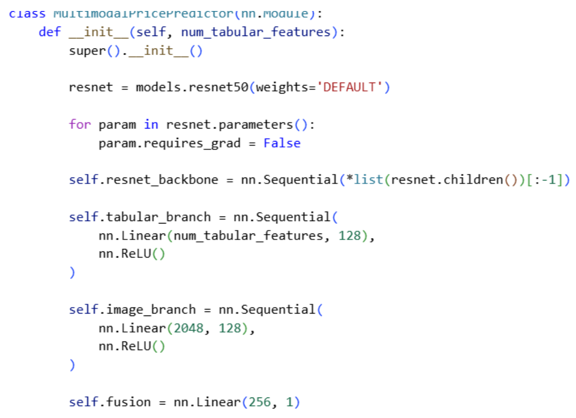
But the results were very poor



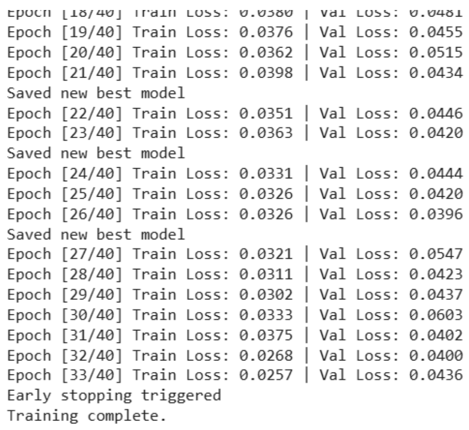
```
----------------------------
TABULAR ONLY RESULTS:
R2 Score: 0.5676
RMSE:     $232,278.39
----------------------------
```

Hence I came to the conclusion that we should not add batch normalization to our tabular data as it is hurting more than helping. The first model fits the signal better whereas the second one over-regularizes and distorts the data.

2. Then I moved onto the multimodal approach where we will be using ResNet-50 for image to vector conversion (feature extraction) , pass the tabular data through a MLP and then fusion both the tabular and imagery features and pass it to a regressor head .
   i. So since our tabular data is already capturing a lot of details, I decided to balance the tabular and image signal.

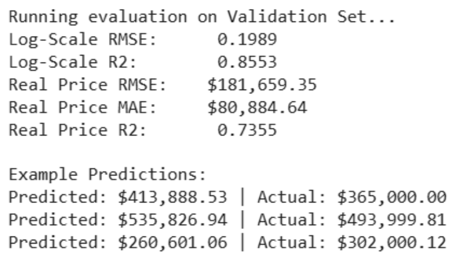


```
class MultimodalPricePredictor(nn.Module):
    def __init__(self, num_tabular_features):
        super().__init__()

        resnet = models.resnet50(weights='DEFAULT')

        for param in resnet.parameters():
            param.requires_grad = False

        self.resnet_backbone = nn.Sequential(*list(resnet.children())[:-1])

        self.tabular_branch = nn.Sequential(
            nn.Linear(num_tabular_features, 128),
            nn.ReLU()
        )

        self.image_branch = nn.Sequential(
            nn.Linear(2048, 128),
            nn.ReLU()
        )

        self.fusion = nn.Linear(256, 1)
```



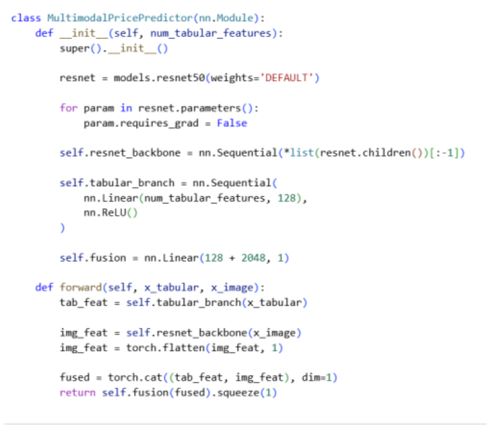
```
Epoch [18/40] Train Loss: 0.0380 | Val Loss: 0.0481
Epoch [19/40] Train Loss: 0.0376 | Val Loss: 0.0455
Epoch [20/40] Train Loss: 0.0362 | Val Loss: 0.0515
Epoch [21/40] Train Loss: 0.0398 | Val Loss: 0.0434
Saved new best model
Epoch [22/40] Train Loss: 0.0351 | Val Loss: 0.0446
Epoch [23/40] Train Loss: 0.0363 | Val Loss: 0.0420
Saved new best model
Epoch [24/40] Train Loss: 0.0331 | Val Loss: 0.0444
Epoch [25/40] Train Loss: 0.0326 | Val Loss: 0.0420
Epoch [26/40] Train Loss: 0.0326 | Val Loss: 0.0396
Saved new best model
Epoch [27/40] Train Loss: 0.0321 | Val Loss: 0.0547
Epoch [28/40] Train Loss: 0.0311 | Val Loss: 0.0423
Epoch [29/40] Train Loss: 0.0302 | Val Loss: 0.0437
Epoch [30/40] Train Loss: 0.0333 | Val Loss: 0.0603
Epoch [31/40] Train Loss: 0.0375 | Val Loss: 0.0402
Epoch [32/40] Train Loss: 0.0268 | Val Loss: 0.0400
Epoch [33/40] Train Loss: 0.0257 | Val Loss: 0.0436
Early stopping triggered
Training complete.
```

This gave me:

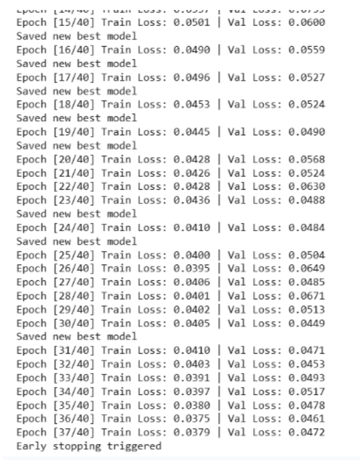


```
Running evaluation on Validation Set...
Log-Scale RMSE:      0.1989
Log-Scale R2:        0.8553
Real Price RMSE:    $181,659.35
Real Price MAE:     $80,884.64
Real Price R2:       0.7355

Example Predictions:
Predicted: $413,888.53 | Actual: $365,000.00
Predicted: $535,826.94 | Actual: $493,999.81
Predicted: $260,601.06 | Actual: $302,000.12
```

   ii. This bottlenecking operation didn’t give much benefit , I decided to pass the features as it is.

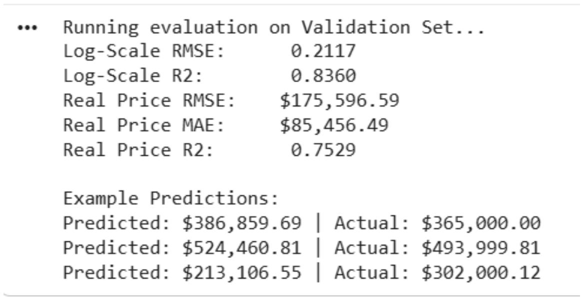


```python
class MultimodalPricePredictor(nn.Module):
    def __init__(self, num_tabular_features):
        super().__init__()

        resnet = models.resnet50(weights='DEFAULT')

        for param in resnet.parameters():
            param.requires_grad = False

        self.resnet_backbone = nn.Sequential(*list(resnet.children())[:-1])

        self.tabular_branch = nn.Sequential(
            nn.Linear(num_tabular_features, 128),
            nn.ReLU()
        )

        self.fusion = nn.Linear(128 + 2048, 1)

    def forward(self, x_tabular, x_image):
        tab_feat = self.tabular_branch(x_tabular)

        img_feat = self.resnet_backbone(x_image)
        img_feat = torch.flatten(img_feat, 1)

        fused = torch.cat((tab_feat, img_feat), dim=1)
        return self.fusion(fused).squeeze(1)
```



```
Epoch [14/40] Train Loss: [illegible] | Val Loss: [illegible]
Epoch [15/40] Train Loss: 0.0501 | Val Loss: 0.0600
Saved new best model
Epoch [16/40] Train Loss: 0.0490 | Val Loss: 0.0559
Saved new best model
Epoch [17/40] Train Loss: 0.0496 | Val Loss: 0.0527
Saved new best model
Epoch [18/40] Train Loss: 0.0453 | Val Loss: 0.0524
Saved new best model
Epoch [19/40] Train Loss: 0.0445 | Val Loss: 0.0490
Saved new best model
Epoch [20/40] Train Loss: 0.0428 | Val Loss: 0.0568
Epoch [21/40] Train Loss: 0.0426 | Val Loss: 0.0524
Epoch [22/40] Train Loss: 0.0428 | Val Loss: 0.0630
Epoch [23/40] Train Loss: 0.0436 | Val Loss: 0.0488
Saved new best model
Epoch [24/40] Train Loss: 0.0410 | Val Loss: 0.0484
Saved new best model
Epoch [25/40] Train Loss: 0.0400 | Val Loss: 0.0504
Epoch [26/40] Train Loss: 0.0395 | Val Loss: 0.0649
Epoch [27/40] Train Loss: 0.0406 | Val Loss: 0.0485
Epoch [28/40] Train Loss: 0.0401 | Val Loss: 0.0671
Epoch [29/40] Train Loss: 0.0402 | Val Loss: 0.0513
Epoch [30/40] Train Loss: 0.0405 | Val Loss: 0.0449
Saved new best model
Epoch [31/40] Train Loss: 0.0410 | Val Loss: 0.0471
Epoch [32/40] Train Loss: 0.0403 | Val Loss: 0.0453
Epoch [33/40] Train Loss: 0.0391 | Val Loss: 0.0493
Epoch [34/40] Train Loss: 0.0397 | Val Loss: 0.0517
Epoch [35/40] Train Loss: 0.0380 | Val Loss: 0.0478
Epoch [36/40] Train Loss: 0.0375 | Val Loss: 0.0461
Epoch [37/40] Train Loss: 0.0379 | Val Loss: 0.0472
Early stopping triggered
```

The result improved slightly:



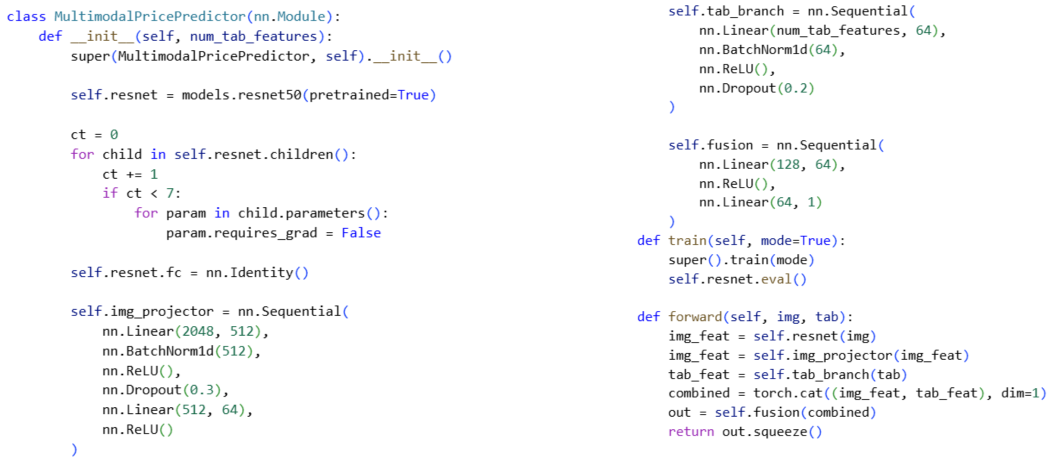
```
••• Running evaluation on Validation Set...
    Log-Scale RMSE:      0.2117
    Log-Scale R2:        0.8360
    Real Price RMSE:    $175,596.59
    Real Price MAE:     $85,456.49
    Real Price R2:       0.7529

    Example Predictions:
    Predicted: $386,859.69 | Actual: $365,000.00
    Predicted: $524,460.81 | Actual: $493,999.81
    Predicted: $213,106.55 | Actual: $302,000.12
```

This shows that the imagery signal is very noisy and our earlier bottlenecking operation was very wide ,instead of reducing noise we intertwined the noise and signal which also disrupted our good tabular signal. Hence we might try adding fine tuning to the resnet and do a more gradual image feature reduction.

I ran some other models where results were very bad and I realised the following reasons for that:

- Unfreezing the CNN layer upto layer 3 is helpful but beyond that we start to loose the important basic features since our dataset is not that humongous so we need to retain those features extracted by resnet-50
- The tabular branch should be rather simple.
- We need to disable the Batch Normalisation statistics (mean and variance) during training especially in case of frozen resnets.
- The fusion layer should also be rather simple. As more complex architecture means more parameters which can lead to issues due to out dataset size.

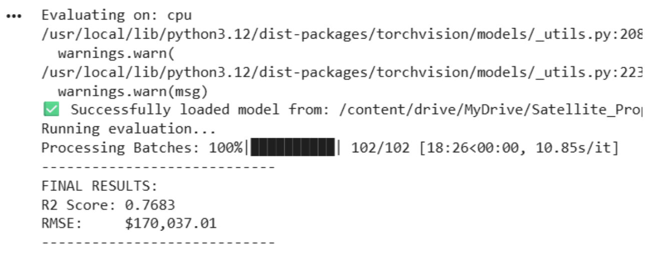
iii. I tried to implement a very detailed structure this time.



```
class MultimodalPricePredictor(nn.Module):
    def __init__(self, num_tab_features):
        super(MultimodalPricePredictor, self).__init__()

        self.resnet = models.resnet50(pretrained=True)

        ct = 0
        for child in self.resnet.children():
            ct += 1
            if ct < 7:
                for param in child.parameters():
                    param.requires_grad = False

        self.resnet.fc = nn.Identity()

        self.img_projector = nn.Sequential(
            nn.Linear(2048, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 64),
            nn.ReLU()
        )
```

```
        self.tab_branch = nn.Sequential(
            nn.Linear(num_tab_features, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Dropout(0.2)
        )

        self.fusion = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )
    def train(self, mode=True):
        super().train(mode)
        self.resnet.eval()

    def forward(self, img, tab):
        img_feat = self.resnet(img)
        img_feat = self.img_projector(img_feat)
        tab_feat = self.tab_branch(tab)
        combined = torch.cat((img_feat, tab_feat), dim=1)
        out = self.fusion(combined)
        return out.squeeze()
```

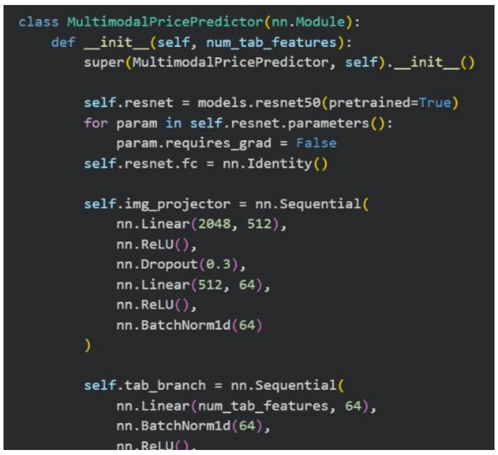
I see a slight improvement with this architecture :



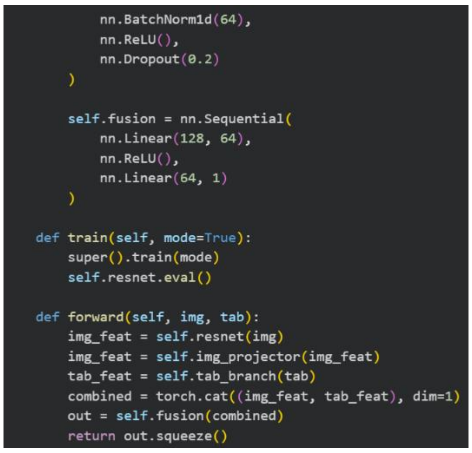
```
...  Evaluating on: cpu
     /usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208
       warnings.warn(
     /usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223
       warnings.warn(msg)
     ✅ Successfully loaded model from: /content/drive/MyDrive/Satellite_Pro
     Running evaluation...
     Processing Batches: 100%|██████████| 102/102 [18:26<00:00, 10.85s/it]
     ----------------------------
     FINAL RESULTS:
     R2 Score: 0.7683
     RMSE:     $170,037.01
     ----------------------------
```

The R2 score has improved but still the result is not able to exceed the performance of XG boost on tabular only data. This points to that our image signal is still noisy.

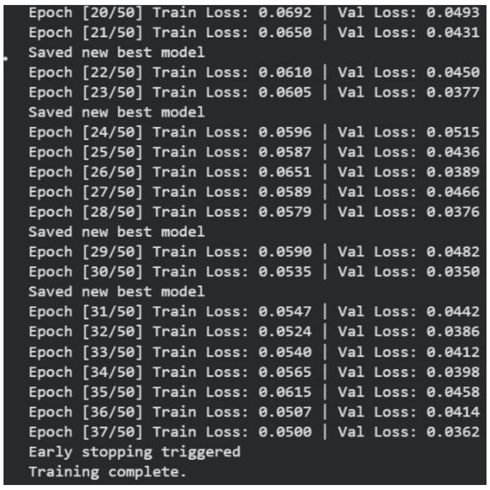
iv. Finally I decided to freeze the resnet again and introduce complexity in the image feature pipeline .



```
class MultimodalPricePredictor(nn.Module):
    def __init__(self, num_tab_features):
        super(MultimodalPricePredictor, self).__init__()

        self.resnet = models.resnet50(pretrained=True)
        for param in self.resnet.parameters():
            param.requires_grad = False
        self.resnet.fc = nn.Identity()

        self.img_projector = nn.Sequential(
            nn.Linear(2048, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 64),
            nn.ReLU(),
            nn.BatchNorm1d(64)
        )

        self.tab_branch = nn.Sequential(
            nn.Linear(num_tab_features, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
```

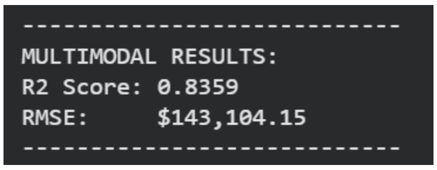


```
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Dropout(0.2)
        )

        self.fusion = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

    def train(self, mode=True):
        super().train(mode)
        self.resnet.eval()

    def forward(self, img, tab):
        img_feat = self.resnet(img)
        img_feat = self.img_projector(img_feat)
        tab_feat = self.tab_branch(tab)
        combined = torch.cat((img_feat, tab_feat), dim=1)
        out = self.fusion(combined)
        return out.squeeze()
```



```
Epoch [20/50] Train Loss: 0.0692 | Val Loss: 0.0493
Epoch [21/50] Train Loss: 0.0650 | Val Loss: 0.0431
Saved new best model
Epoch [22/50] Train Loss: 0.0610 | Val Loss: 0.0450
Epoch [23/50] Train Loss: 0.0605 | Val Loss: 0.0377
Saved new best model
Epoch [24/50] Train Loss: 0.0596 | Val Loss: 0.0515
Epoch [25/50] Train Loss: 0.0587 | Val Loss: 0.0436
Epoch [26/50] Train Loss: 0.0651 | Val Loss: 0.0389
Epoch [27/50] Train Loss: 0.0589 | Val Loss: 0.0466
Epoch [28/50] Train Loss: 0.0579 | Val Loss: 0.0376
Saved new best model
Epoch [29/50] Train Loss: 0.0590 | Val Loss: 0.0482
Epoch [30/50] Train Loss: 0.0535 | Val Loss: 0.0350
Saved new best model
Epoch [31/50] Train Loss: 0.0547 | Val Loss: 0.0442
Epoch [32/50] Train Loss: 0.0524 | Val Loss: 0.0386
Epoch [33/50] Train Loss: 0.0540 | Val Loss: 0.0412
Epoch [34/50] Train Loss: 0.0565 | Val Loss: 0.0398
Epoch [35/50] Train Loss: 0.0615 | Val Loss: 0.0458
Epoch [36/50] Train Loss: 0.0507 | Val Loss: 0.0414
Epoch [37/50] Train Loss: 0.0500 | Val Loss: 0.0362
Early stopping triggered
Training complete.
```

The results were a considerable improvement from previous models:



```
--------------------------
MULTIMODAL RESULTS:
R2 Score: 0.8359
RMSE:      $143,104.15
--------------------------
```

So I am choosing this as my final multimodal architecture with an R2 of 0.8359.

Fusion Architecture Diagram

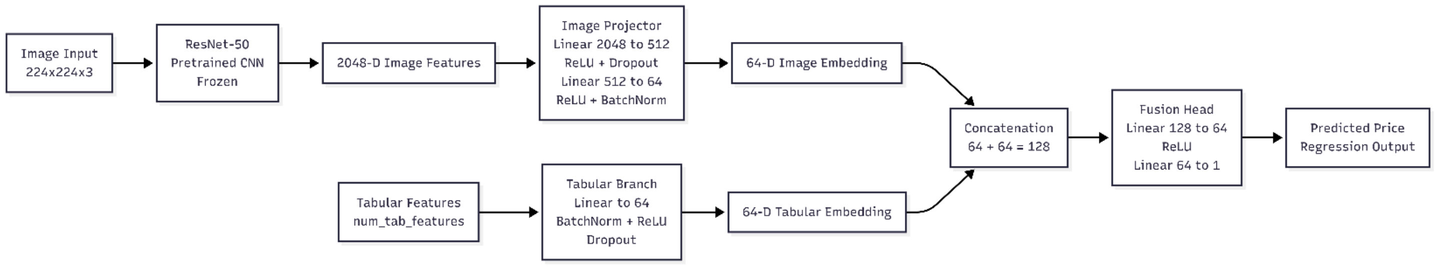
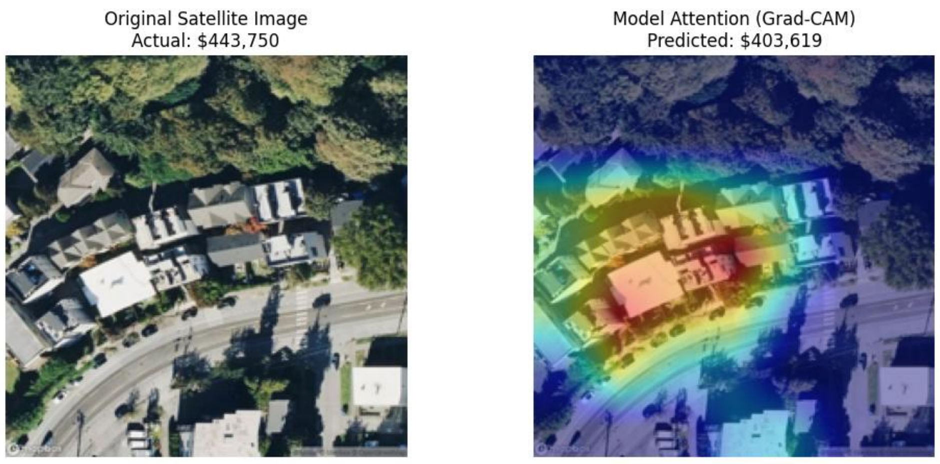
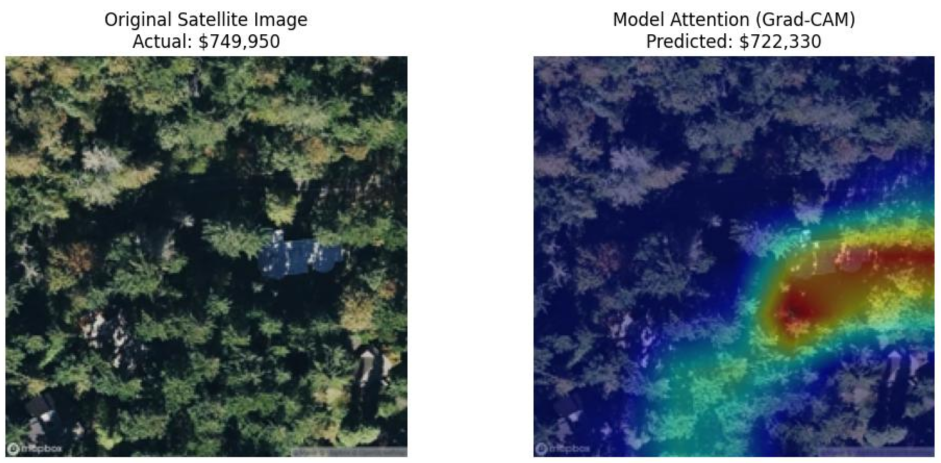


Image Input 224x224x3
ResNet-50 Pretrained CNN Frozen
2048-D Image Features
Image Projector Linear 2048 to 512 ReLU + Dropout Linear 512 to 64 ReLU + BatchNorm
64-D Image Embedding
Tabular Features num_tab_features
Tabular Branch Linear to 64 BatchNorm + ReLU Dropout
64-D Tabular Embedding
Concatenation 64 + 64 = 128
Fusion Head Linear 128 to 64 ReLU Linear 64 to 1
Predicted Price Regression Output

Grad Cam Explainability





The Grad-CAM visualization shows that the model primarily focuses on the building structure itself, particularly rooftops and the immediate surrounding area. This suggests that the model is learning meaningful visual cues such as building size, density, and proximity to roads, rather than irrelevant background features.

Results

Though I have already describe my tabular vs multimodal approach comparison results in the MODEL STRATEGY , I would like to summarize my findings in a few points:

- Nonlinear tabular models outperform linear baselines.
- XG Boost perfomed lot better than linear and ridge regression.
- Multimodal pipelines comprising of neural networks couldn't beat XG Boost tabular only model but they did exceeded linear tabular models and I believe if I had more gpu resources I could have trained my model longer and my results would have beaten the tabular only neural networks
- Satellite imagery contributes information about: neighbourhood density, Urban vs rural context, Road access and surrounding infrastructure
  However:
  Core price determinants (area, location, amenities) remain tabular which limits the standalone predictive power of images.
  THANK YOU