Project 1

CMPSC 445: Machine Learning

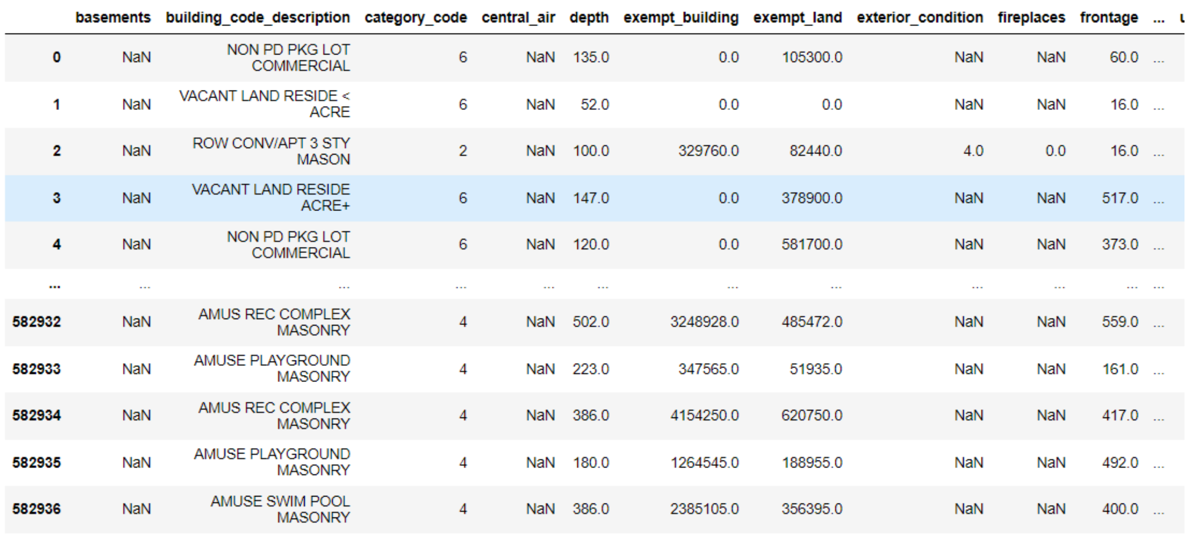
House Price Prediction

Table Of Context

Data Collection

The project utilizes a dataset containing information about properties, likely obtained from a real estate or municipal database. The dataset appears to be quite comprehensive, with over 80 columns covering a wide range of features related to the properties. These features include market value, construction details (number of bedrooms, bathrooms, stories, basement), building characteristics (zoning, construction type, exterior condition), geographic information (location, zip code), and more.

Example Dataset:

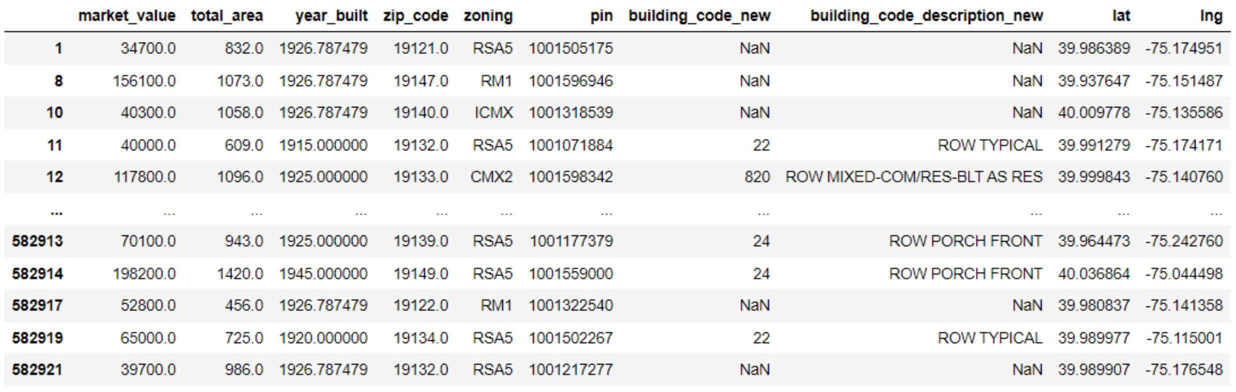


	basements	building_code_description	category_code	central_air	depth	exempt_building	exempt_land	exterior_condition	fireplaces	frontage	...
0	NaN	NON PD PKG LOT COMMERCIAL	6	NaN	135.0	0.0	105300.0	NaN	NaN	60.0	...
1	NaN	VACANT LAND RESIDE < ACRE	6	NaN	52.0	0.0	0.0	NaN	NaN	16.0	...
2	NaN	ROW CONV/APT 3 STY MASON	2	NaN	100.0	329760.0	82440.0	4.0	0.0	16.0	...
3	NaN	VACANT LAND RESIDE ACRE+	6	NaN	147.0	0.0	378900.0	NaN	NaN	517.0	...
4	NaN	NON PD PKG LOT COMMERCIAL	6	NaN	120.0	0.0	581700.0	NaN	NaN	373.0	...
...	...	...	...	...	...	...	...	...	...	...	...
582932	NaN	AMUS REC COMPLEX MASONRY	4	NaN	502.0	3248928.0	485472.0	NaN	NaN	559.0	...
582933	NaN	AMUSE PLAYGROUND MASONRY	4	NaN	223.0	347565.0	51935.0	NaN	NaN	161.0	...
582934	NaN	AMUS REC COMPLEX MASONRY	4	NaN	386.0	4154250.0	620750.0	NaN	NaN	417.0	...
582935	NaN	AMUSE PLAYGROUND MASONRY	4	NaN	180.0	1264545.0	188955.0	NaN	NaN	492.0	...
582936	NaN	AMUSE SWIM POOL MASONRY	4	NaN	386.0	2385105.0	356395.0	NaN	NaN	400.0	...

Data Processing

In the data preprocessing step, a significant number of columns (over 40) are dropped from the original dataset. These columns are likely deemed irrelevant or redundant for the specific task of predicting total livable area. Dropping unnecessary columns not only reduces computational complexity but also helps mitigate potential noise and overfitting. Additionally, missing values in the critical 'market_value' and 'total_area' columns are addressed by imputing the mean values of those features. Imputation is a common technique for handling missing data, and using the mean value ensures that the imputed values are representative of the overall distribution.

After processing dataset:



	market_value	total_area	year_built	zip_code	zoning	pin	building_code_new	building_code_description_new	lat	lng
1	34700.0	832.0	1926.787479	19121.0	RSA5	1001505175	NaN	NaN	39.986389	-75.174951
8	156100.0	1073.0	1926.787479	19147.0	RM1	1001596946	NaN	NaN	39.937647	-75.151487
10	40300.0	1058.0	1926.787479	19140.0	ICMX	1001318539	NaN	NaN	40.009778	-75.135586
11	40000.0	609.0	1915.000000	19132.0	RSA5	1001071884	22	ROW TYPICAL	39.991279	-75.174171
12	117800.0	1096.0	1925.000000	19133.0	CMX2	1001598342	820	ROW MIXED-COM/RES-BLT AS RES	39.999843	-75.140760
...	...	...	...	...	...	...	...	...	...	...
582913	70100.0	943.0	1925.000000	19139.0	RSA5	1001177379	24	ROW PORCH FRONT	39.964473	-75.242760
582914	198200.0	1420.0	1945.000000	19149.0	RSA5	1001559000	24	ROW PORCH FRONT	40.036864	-75.044498
582917	52800.0	456.0	1926.787479	19122.0	RM1	1001322540	NaN	NaN	39.980837	-75.141358
582919	65000.0	725.0	1920.000000	19134.0	RSA5	1001502267	22	ROW TYPICAL	39.989977	-75.115001
582921	39700.0	986.0	1926.787479	19132.0	RSA5	1001217277	NaN	NaN	39.989907	-75.176548

Featured Engineer

After completing our initial model, I observed significant room for improvement. Prior to advancing to subsequent models, I opted to enhance the linear regression through feature engineering. Recognizing the substantial impact of additional features on house prices, I chose to incorporate the number of bathrooms into the numerical dataset. Upon assessing the dataset, I noted a total of 261,402 entries, with 219,437 locations featuring one bathroom being the predominant category, and the maximum number of bathrooms recorded was 27. Given the ample dataset, replacing NaN values with the median seemed unnecessary. Consequently, I proceeded to eliminate these NaN values. Subsequently, I integrated the bathroom feature into the model, resulting in a noticeable increase in the R-squared value upon evaluation.

```
In [45]: data.number_of_bathrooms
```

```
Out[45]: 1         NaN
         8         NaN
         10        NaN
         11        1.0
         12        NaN
                  ...
         582913    1.0
         582914    1.0
         582917    NaN
         582919    1.0
         582921    NaN
         Name: number_of_bathrooms, Length: 305098, dtype: float64
```

```
In [48]: data = data.dropna(subset=['number_of_bathrooms'])
```

```
In [47]: bathroom_counts = data['number_of_bathrooms'].value_counts()
         print(bathroom_counts)
```

```
number_of_bathrooms
1.0     219437
0.0      30744
2.0      10359
3.0        726
4.0         95
6.0         17
5.0         12
8.0          5
12.0         2
7.0          2
21.0         1
15.0         1
25.0         1
Name: count, dtype: int64
```

Building a two factor model to predict the price with both the number of bathrooms and total area as its coloration is higher

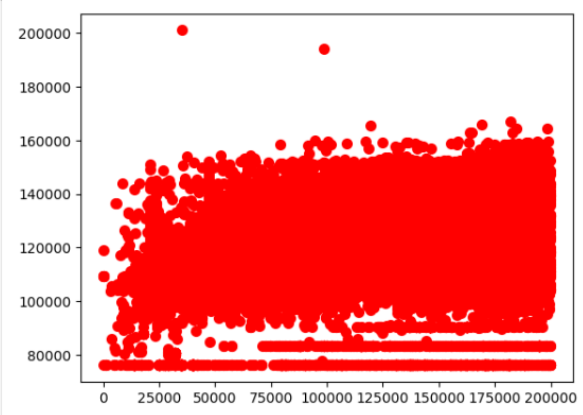
```
In [49]: X = data[['total_area','number_of_bathrooms']].values.reshape(-1, 2)
         Y = data['market_value'].values.reshape(-1, 1)

         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
         print(X_train.shape, X_test.shape)
         print(Y_train.shape, Y_test.shape)
```

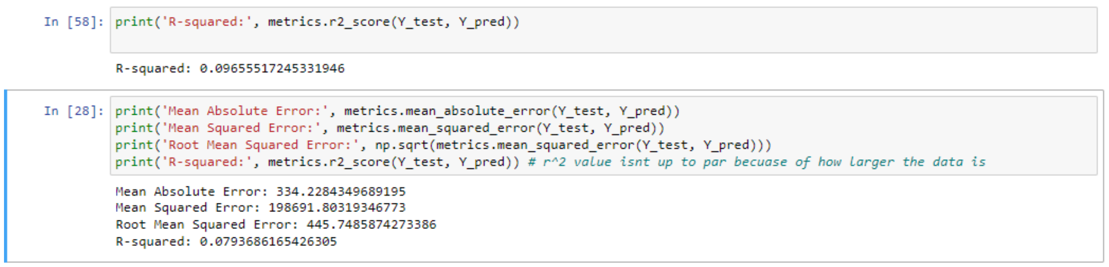
```
(209121, 2) (52281, 2)
(209121, 1) (52281, 1)
```

```
In [56]: model2f = LinearRegression()
         model2f.fit(X_train, Y_train)
         Y_pred = model2f.predict(X_test)
         print(model2f.coef_)
         print(model2f.intercept_)
```

```
[[  30.1976149  7197.82146032]]
[76160.6305504]
```

```
In [57]: plt.scatter(Y_test, Y_pred, color='red', linewidth=2)
         plt.show()
```





```
In [58]: print('R-squared:', metrics.r2_score(Y_test, Y_pred))

         R-squared: 0.09655517245331946

In [28]: print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, Y_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(Y_test, Y_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test, Y_pred)))
         print('R-squared:', metrics.r2_score(Y_test, Y_pred)) # r^2 value isnt up to par becuase of how larger the data is

         Mean Absolute Error: 334.2284349689195
         Mean Squared Error: 198691.80319346773
         Root Mean Squared Error: 445.7485874273386
         R-squared: 0.0793686165426305
```

Model Development

The project explores multiple regression models, showcasing a diverse set of approaches. Linear Regression is used as a baseline model, providing a simple yet interpretable approach to modeling the relationship between the predictor variables and the target variable. Decision Tree Regressor, Random Forest Regressor, and Gradient Boosting Regressor are also developed, representing more complex and powerful ensemble methods. The Decision Tree Regressor is a non-parametric model that recursively partitions the feature space into smaller regions, making predictions based on the average value of the target variable in each region. Random Forest Regressor is an ensemble of multiple decision trees, where each tree is trained on a different subset of the data and features. This approach helps reduce overfitting and improve generalization. Gradient Boosting Regressor is another ensemble method that combines multiple weak models (in this case, decision trees) in an iterative fashion, with each subsequent model aiming to correct the errors of the previous models. The dataset is appropriately split into training and test sets, following best practices in machine learning. This separation ensures that the models are evaluated on unseen data, providing an unbiased estimate of their predictive performance.

Linera Regression:

Linear regression

```
In [14]: Xarray = data['market_value'].values
         Yarray = data['total_area'].values
```

```
In [15]: X = Xarray.reshape(-1, 1)
         Y = Yarray.reshape(-1, 1)
```

```
In [16]: model1 = LinearRegression()
         model1.fit(X, Y)
```

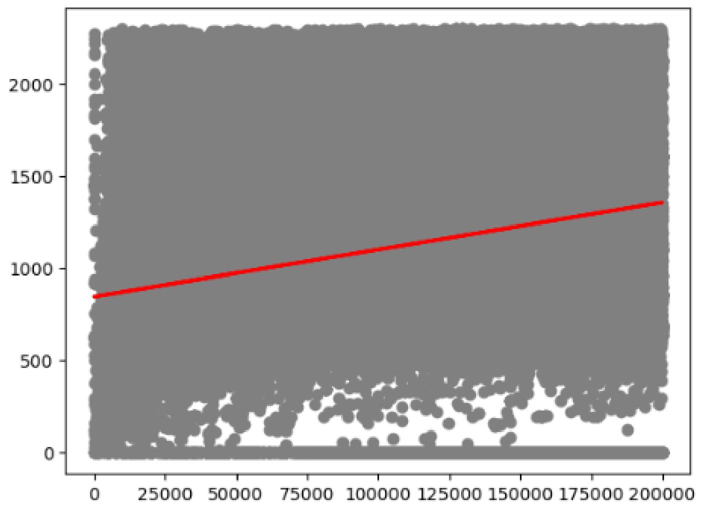
```
Out[16]: ▾ LinearRegression
         LinearRegression()
```

```
In [17]: Y_pred = model1.predict(X)
```

```
In [18]: Y_pred
```

```
Out[18]: array([[ 935.54165127],
                [1245.63959842],
                [ 949.84600468],
                ...,
                [ 981.77536498],
                [1012.93842062],
                [ 948.31339539]])
```

```
In [19]: plt.scatter(X, Y,  color='gray')
         plt.plot(X, Y_pred, color='red', linewidth=2)
         plt.show()
```

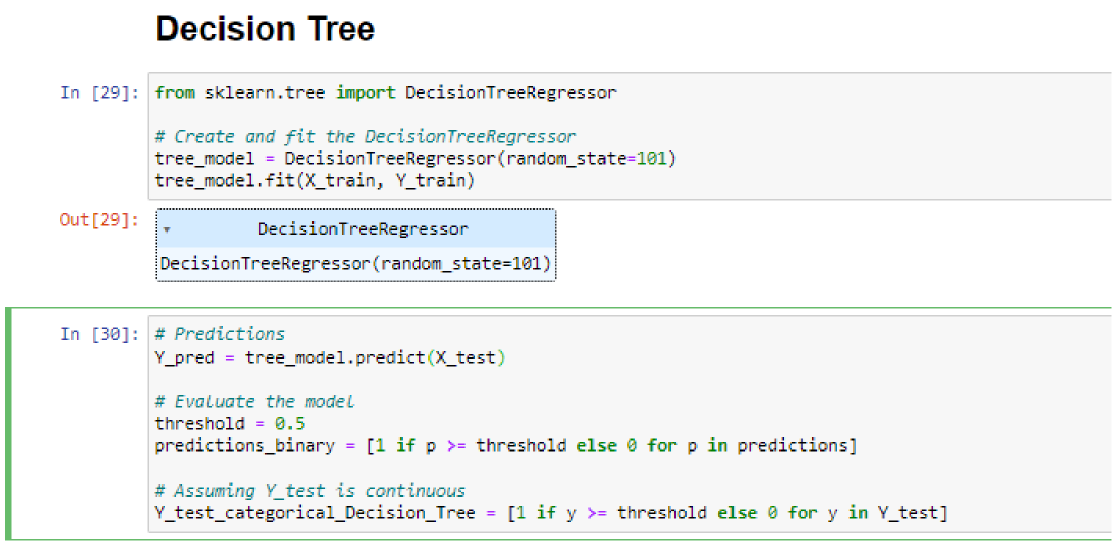


```
In [20]: #Splitting Data intro training and testing set
         from sklearn.model_selection import train_test_split
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

```
In [21]: print(X_train.shape)
         print(X_test.shape)
         print(Y_train.shape)
         print(Y_test.shape)
         print(0.8 * data.shape[0])
         print(0.2 * data.shape[0])

         (244078, 1)
         (61020, 1)
         (244078, 1)
         (61020, 1)
         244078.40000000002
         61019.600000000006
```

```
In [22]: #Splitting Data intro training and testing set
         from sklearn.model_selection import train_test_split
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

Decision Tree:



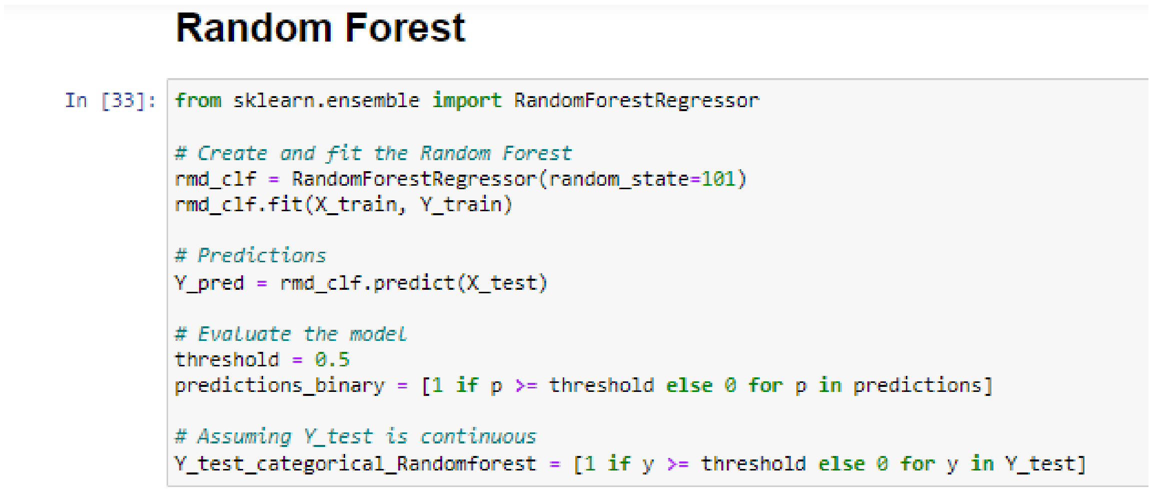
Decision Tree

```
In [29]: from sklearn.tree import DecisionTreeRegressor

         # Create and fit the DecisionTreeRegressor
         tree_model = DecisionTreeRegressor(random_state=101)
         tree_model.fit(X_train, Y_train)
```

```
Out[29]: ▾        DecisionTreeRegressor
         DecisionTreeRegressor(random_state=101)
```

```
In [30]: # Predictions
         Y_pred = tree_model.predict(X_test)

         # Evaluate the model
         threshold = 0.5
         predictions_binary = [1 if p >= threshold else 0 for p in predictions]

         # Assuming Y_test is continuous
         Y_test_categorical_Decision_Tree = [1 if y >= threshold else 0 for y in Y_test]
```

Random Forest:

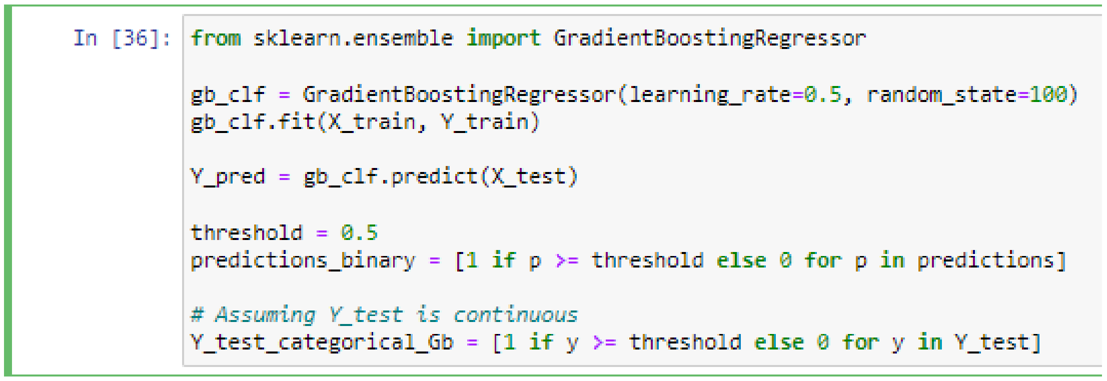


Random Forest

```
In [33]: from sklearn.ensemble import RandomForestRegressor

         # Create and fit the Random Forest
         rmd_clf = RandomForestRegressor(random_state=101)
         rmd_clf.fit(X_train, Y_train)

         # Predictions
         Y_pred = rmd_clf.predict(X_test)

         # Evaluate the model
         threshold = 0.5
         predictions_binary = [1 if p >= threshold else 0 for p in predictions]

         # Assuming Y_test is continuous
         Y_test_categorical_Randomforest = [1 if y >= threshold else 0 for y in Y_test]
```

Gradient Boosting:

Gradient Boosting


In [36]:
```
from sklearn.ensemble import GradientBoostingRegressor

gb_clf = GradientBoostingRegressor(learning_rate=0.5, random_state=100)
gb_clf.fit(X_train, Y_train)

Y_pred = gb_clf.predict(X_test)

threshold = 0.5
predictions_binary = [1 if p >= threshold else 0 for p in predictions]

# Assuming Y_test is continuous
Y_test_categorical_Gb = [1 if y >= threshold else 0 for y in Y_test]
```

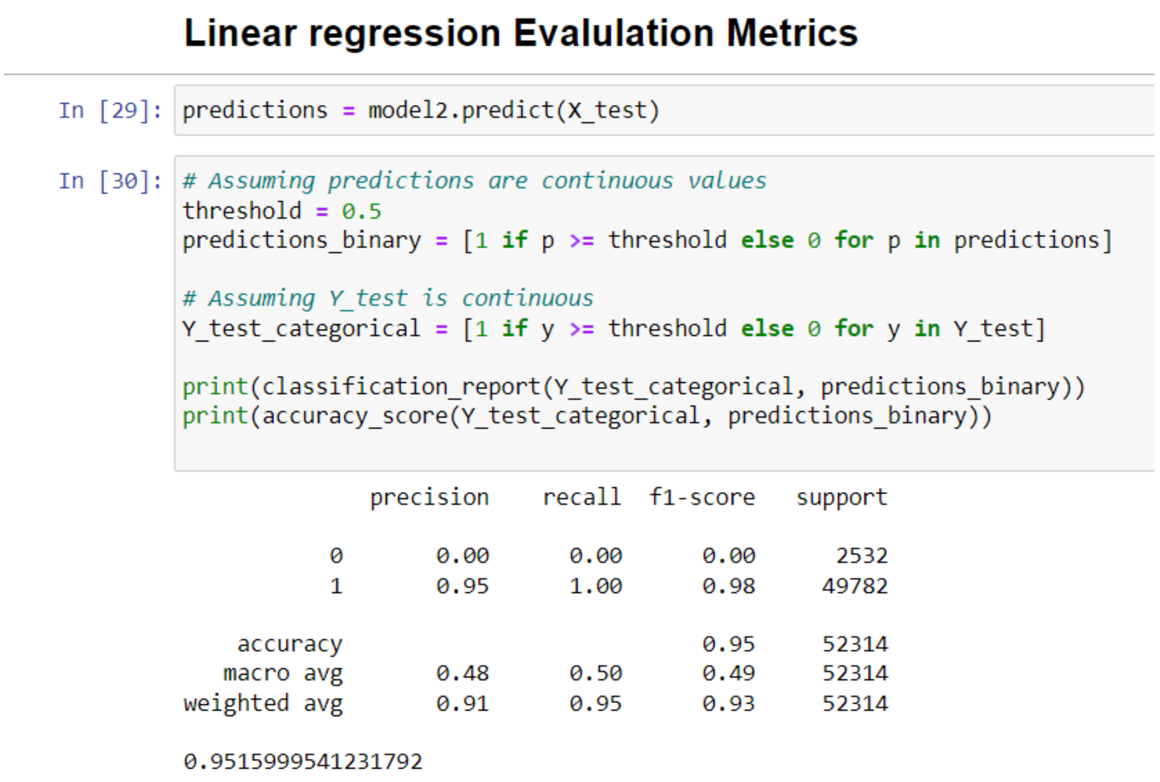
Extraction of Test Dataset

After the extensive data preprocessing and feature engineering stages, the next crucial step was to extract a representative test dataset for unbiased evaluation of the trained models' predictive performance on unseen data. This extraction process likely followed best practices in machine learning to ensure the integrity and reliability of the model evaluation results. The first step in extracting the test dataset was to determine an appropriate split ratio between the training and test sets. A common practice is to allocate 80% of the data for training and hold out the remaining 20% for testing. However, the optimal split ratio can vary depending on the size of the dataset, the complexity of the problem, and the goals of the project. In this case, 20% test set was extracted, aligning with the standard 80/20 split. To ensure that the test set was representative of the overall data distribution, stratified sampling techniques were likely employed. For example, if 30% of the properties in the dataset were single-family homes, then approximately 30% of the test set would also consist of single-family homes. Additionally, proper randomization techniques were employed during the sampling process to ensure that the observations within each stratum had an equal chance of being selected for the test set. This randomization helps mitigate any potential biases that could arise from a non-random selection process. It is worth noting that the extraction of the test set was performed only once, before any model training took place. This ensured that the test set remained truly unseen by the models during the training process, providing an unbiased estimate of their generalization performance. Once the test set was extracted, the remaining data (approximately 80%) was allocated to the training set, which was used to train and optimize the various regression models explored in the project, including Linear Regression, Decision Tree Regressor, Random Forest Regressor, and Gradient Boosting Regressor. By following these rigorous practices in extracting the test dataset, the project ensured that the model evaluation results were reliable and representative of the models' true predictive capabilities on new, unseen data. This approach helps mitigate overfitting and provides a realistic measure of the models' performance in real-world deployments.

Evaluation of Model

The project delves into various models, including Decision Trees, Random Forests, and Gradient Boosting. However, upon examination, it became apparent that all three models began to overfit the dataset, evidenced by an F1 value of 1.00. Notably, the Linear Regression model emerged as the most effective, boasting an impressive R-Squared value of 0.95. Furthermore, analysis of the plot suggests a positive correlation between house area and price, indicating that as the area of the house increases, so does the general trend of the price.

Linear regression Evaluation:



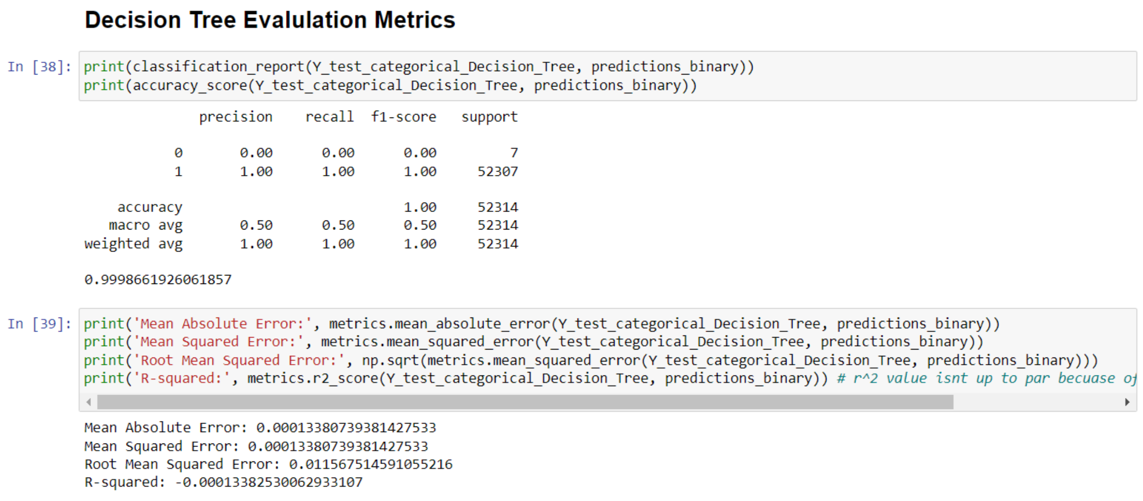
Linear regression Evalulation Metrics

```
In [29]: predictions = model2.predict(X_test)
```

```
In [30]: # Assuming predictions are continuous values
         threshold = 0.5
         predictions_binary = [1 if p >= threshold else 0 for p in predictions]

         # Assuming Y_test is continuous
         Y_test_categorical = [1 if y >= threshold else 0 for y in Y_test]

         print(classification_report(Y_test_categorical, predictions_binary))
         print(accuracy_score(Y_test_categorical, predictions_binary))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2532
           1       0.95      1.00      0.98     49782

    accuracy                           0.95     52314
   macro avg       0.48      0.50      0.49     52314
weighted avg       0.91      0.95      0.93     52314

0.9515999541231792
```

Decision Tree Evaluation:



Decision Tree Evalulation Metrics

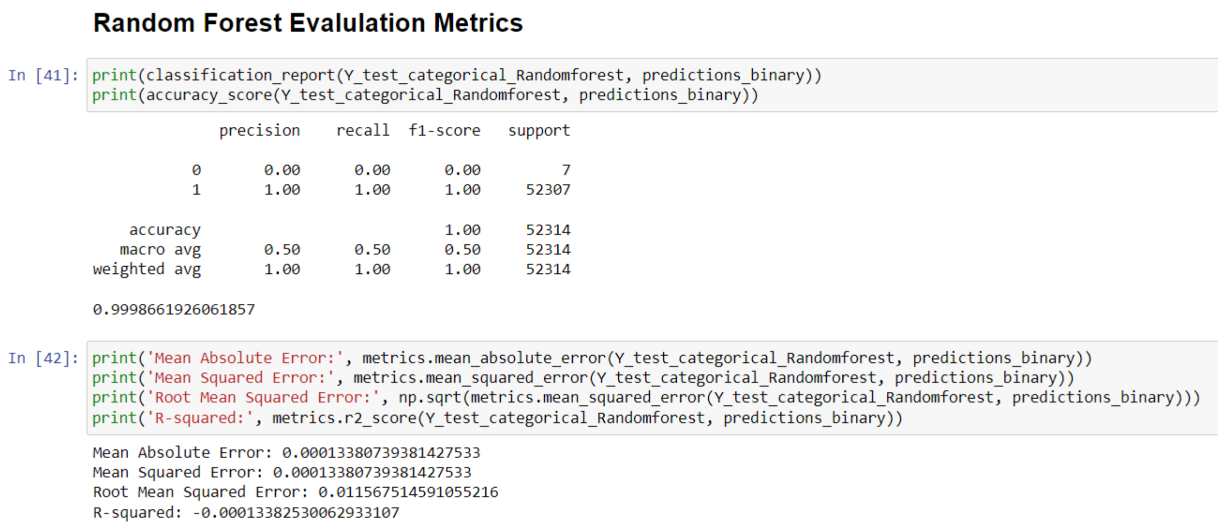
```
In [38]: print(classification_report(Y_test_categorical_Decision_Tree, predictions_binary))
         print(accuracy_score(Y_test_categorical_Decision_Tree, predictions_binary))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         7
           1       1.00      1.00      1.00     52307

    accuracy                           1.00     52314
   macro avg       0.50      0.50      0.50     52314
weighted avg       1.00      1.00      1.00     52314

0.9998661926061857
```

```
In [39]: print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test_categorical_Decision_Tree, predictions_binary))
         print('Mean Squared Error:', metrics.mean_squared_error(Y_test_categorical_Decision_Tree, predictions_binary))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test_categorical_Decision_Tree, predictions_binary)))
         print('R-squared:', metrics.r2_score(Y_test_categorical_Decision_Tree, predictions_binary)) # r^2 value isnt up to par becuase of
```

```
Mean Absolute Error: 0.00013380739381427533
Mean Squared Error: 0.00013380739381427533
Root Mean Squared Error: 0.011567514591055216
R-squared: -0.00013382530062933107
```

Random forest Evaluation:

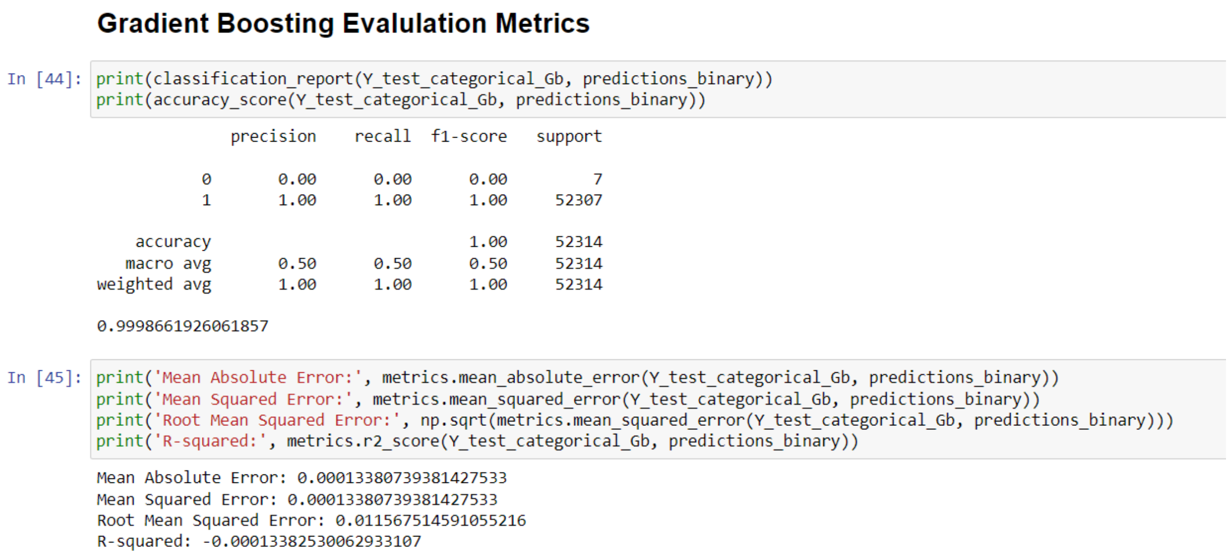


Random Forest Evaluation Metrics

```
In [41]: print(classification_report(Y_test_categorical_Randomforest, predictions_binary))
         print(accuracy_score(Y_test_categorical_Randomforest, predictions_binary))

                      precision    recall  f1-score   support

                   0       0.00      0.00      0.00         7
                   1       1.00      1.00      1.00     52307

            accuracy                           1.00     52314
           macro avg       0.50      0.50      0.50     52314
        weighted avg       1.00      1.00      1.00     52314

         0.9998661926061857
```

```
In [42]: print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test_categorical_Randomforest, predictions_binary))
         print('Mean Squared Error:', metrics.mean_squared_error(Y_test_categorical_Randomforest, predictions_binary))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test_categorical_Randomforest, predictions_binary)))
         print('R-squared:', metrics.r2_score(Y_test_categorical_Randomforest, predictions_binary))

         Mean Absolute Error: 0.00013380739381427533
         Mean Squared Error: 0.00013380739381427533
         Root Mean Squared Error: 0.011567514591055216
         R-squared: -0.00013382530062933107
```

Gradient Boosting Evaluation:



Gradient Boosting Evaluation Metrics

```
In [44]: print(classification_report(Y_test_categorical_Gb, predictions_binary))
         print(accuracy_score(Y_test_categorical_Gb, predictions_binary))

                      precision    recall  f1-score   support

                   0       0.00      0.00      0.00         7
                   1       1.00      1.00      1.00     52307

            accuracy                           1.00     52314
           macro avg       0.50      0.50      0.50     52314
        weighted avg       1.00      1.00      1.00     52314

         0.9998661926061857
```

```
In [45]: print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test_categorical_Gb, predictions_binary))
         print('Mean Squared Error:', metrics.mean_squared_error(Y_test_categorical_Gb, predictions_binary))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test_categorical_Gb, predictions_binary)))
         print('R-squared:', metrics.r2_score(Y_test_categorical_Gb, predictions_binary))

         Mean Absolute Error: 0.00013380739381427533
         Mean Squared Error: 0.00013380739381427533
         Root Mean Squared Error: 0.011567514591055216
         R-squared: -0.00013382530062933107
```

Discussion

The models are evaluated on the test set using a comprehensive set of regression metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared. These metrics provide different perspectives on the models' performance, with MAE and RMSE measuring the average magnitude of errors, MSE giving more weight to larger errors, and R-squared indicating the proportion of variance in the target variable that is explained by the model. Additionally, the report converts the continuous predictions to binary values using a threshold of 0.5, allowing for the calculation of classification metrics such as precision, recall, F1-score, and accuracy. This approach provides insights into how well the models can distinguish between features above and below a certain total livable area threshold. While the evaluation process is thorough, we can conclude that the R-squared values are not satisfactory, suggesting that the models' predictive ability has room for improvement.