Using Machine Learning to Create Price Prediction for Tokyo Based Real Estate Company with Python

Part B: Correlation Analysis and Machine Learning Modeling

By Shahin Karami

This Notebook contains the second part of this project. The previous part consisted of data prep and exploratory data analysis (EDA). The focus of this Notebook is to conduct Correlation Analysis using variables outlined in the previous part, along with building and running a Supervised ML model using `PyCaret`. Once the ML model is created, model analysis is conducted. As with the previous notebook, this notebook will also be split up into 2 steps: Correlation Analysis and Machine Learning Modeling

---

Step 1: Correlation Analysis

After importing the necessary packages and the data, I am going to explore the relationship between the variables and the price. Since price is arguably the key factor for each piece of real estate, it makes sense to examine how many other factors are correlated with it. To accomplish this, I will consider two different methods. The first method involves analyzing the correlation between numeric variables and `price`, while the second method involves exploring the correlation between categorical variables and `price`. For numerical variables, I will use Pearson correlation, and for categorical variables, I will use ANOVA.

1 - Imports and Data Import via `pickle`

In [1]:
```
import pickle
import matplotlib
import json

import pandas as pd
import numpy as np
import seaborn as sns

import statsmodels.api as sm
from statsmodels.formula.api import ols

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from matplotlib.pyplot import figure

plt.style.use('ggplot')
%matplotlib inline

from pycaret.regression import *
```

In [2]:
```
with open('TokyoSub_Final.pickle', 'rb') as file:
    df = pickle.load(file)

df
```

Out[2]:

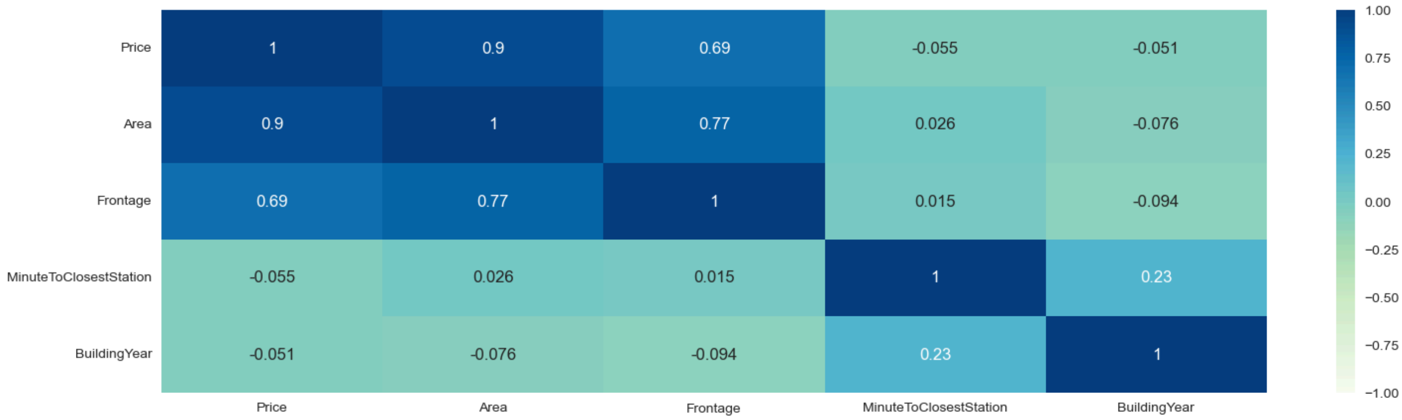
	Type	Municipality	MinuteToClosestStation	Price	FloorPlan	Area	LandShape	Fronta
5	Residential Land(Land and Building)	Chiyoda Ward	4.0	120000000	1K	65	Semi-rectangular Shaped	
18	Residential Land(Land and Building)	Chiyoda Ward	3.0	240000000	1K	80	Semi-trapezoidal Shaped	
30	Residential Land(Land and Building)	Chiyoda Ward	5.0	300000000	1K	210	Semi-rectangular Shaped	1
31	Residential Land(Land and Building)	Chiyoda Ward	4.0	170000000	1K	105	Semi-rectangular Shaped	
90	Residential Land(Land and Building)	Chiyoda Ward	4.0	1600000000	1K	640	Trapezoidal Shaped	3
...	...	...	...	...	...	...	...	
19954	Residential Land(Land and Building)	Bunkyo Ward	4.0	9000000	1K	35	Rectangular Shaped	
19956	Residential Land(Land and Building)	Minato Ward	1.0	530000000	1K	230	Semi-trapezoidal Shaped	1
19975	Residential Land(Land and Building)	Minato Ward	16.0	110000000	1K	85	Semi-square Shaped	
19988	Residential Land(Land and Building)	Bunkyo Ward	6.0	40000000	1K	30	Rectangular Shaped	
19989	Residential Land(Land and Building)	Bunkyo Ward	6.0	36000000	1K	30	Rectangular Shaped	

2108 rows × 13 columns

2 - Pearson Correlation

In [3]:

```
pearson_cor = df[['Price','Area','Frontage','MinuteToClosestStation','BuildingYear']].corr(metho

plt.figure(figsize = (18,5))

sns.heatmap(pearson_cor,
            vmin = -1,
            vmax = 1,
            cmap = "GnBu",
            annot = True)
plt.show()
```



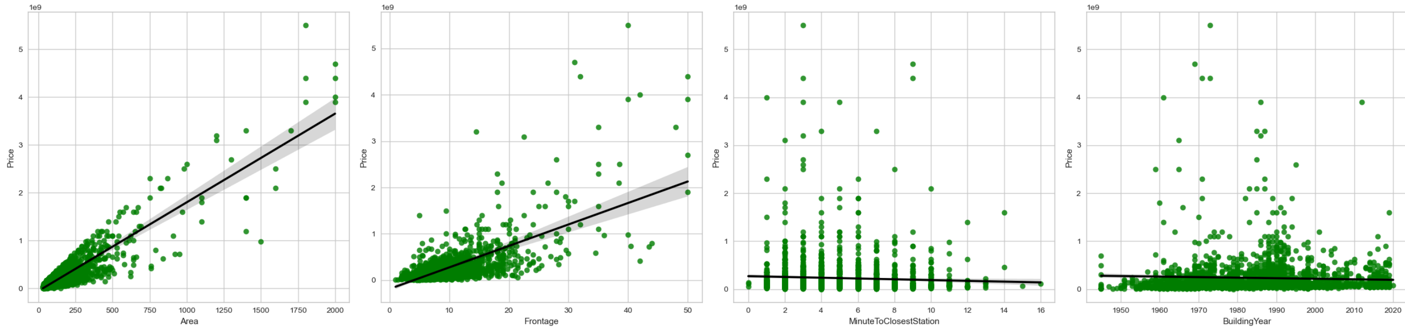
Above is the Pearson Correlation matrix for all the numerical variables. Note that `Price` is positively correlated with `Area` and `Frontage`. This is to be expected since properties with more land tend to be more valuable. The most interesting correlation—or rather, the lack thereof—is between the distance to the nearest station and the age of the building. One would naturally assume that property owners, especially homeowners, in Tokyo would value properties near a train station, thus inflating their prices. However, there is almost no correlation between price and distance. Similarly, the age of a property does not correlate with its price. This suggests that property age and distance to the nearest station have no relationship with the price

To visualize the correlations, I created four plots, each representing a variable with respect to price. For each plot, I fitted an OLS regression line to the data. The correlation coefficient determines the slope of the fitted line. When the correlation coefficient is closer to 1.0, the slope of the line is steeper. Conversely, when the correlation coefficient is closer to 0.0, the slope is flatter. The plots below visualize the results of the Pearson Correlation matrix above.

In [4]:

```
#declare numeric variable
numeric = ['Area','Frontage','MinuteToClosestStation','BuildingYear']

# Create a grid of subplots
fig, axes = plt.subplots(1, 4, figsize=(25, 6))

# Flatten the axes array to make it 1D
axes = axes.ravel()

# Loop through each subplot and plot sns.regplot
for i, col in enumerate(numeric):
    sns.regplot(x = col, y='Price', data = df ,ax = axes[i], scatter_kws = {"color": "green"}, l
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Price')

# Adjust spacing between subplots
plt.tight_layout()
```

```python
# Show the plot
plt.show()
```



---

3 - ANOVA Test

For categorical variables, my first step is to check whether a particular categorical variable impacts the price significantly or not. To do this, I run an ANOVA test to compare the differences between means. After that, for significant variables, I conduct a descriptive analysis for all labels within the particular categorical variable and then compare their impacts on the price. I will omit `Renovation`, `FloorPlan`, and `Type` from the pairwise descriptive analysis as each of these variables only has one unique value.

In [5]:
```python
cat_list = ['Municipality', 'LandShape', 'Structure', 'Use', 'Direction']

for i in cat_list:
    formula = 'Price ~ {}'.format(i)
    model = ols(formula, data=df).fit()
    anova = sm.stats.anova_lm(model, typ=2, robust = 'hc3')
    p_value = anova.iloc[0,3]

    print('P-value for Price ~ {}: {}'.format(i , p_value))
```

```
P-value for Price ~ Municipality: 1.244738946240639e-10
P-value for Price ~ LandShape: 1.5745395046513997e-17
P-value for Price ~ Structure: 1.0219039791033004e-77
P-value for Price ~ Use: 8.04871268698268e-26
P-value for Price ~ Direction: 3.7488561007703056e-80
```

The results above show that all categorical variables are significant in explaining the price of real estate (P-value < 0.05), leading to the rejection of the null hypothesis. In the next step, I aim to determine which label in the above categorical variables has the greatest impact on price. To achieve this, I create a function to calculate the mean for each label within every single categorical variable. This approach will provide a clear picture of how categorical characteristics affect the price of real estate.

In [6]:
```python
def mean_pairwise(cat_var):
    mean_by = df.groupby(cat_var)['Price'].mean()
    mean_by = pd.DataFrame(mean_by)
    mean_by = mean_by.sort_values(by=['Price'], inplace=False, ascending=False)

    return mean_by
```

In [7]:
```python
mean_pairwise('Municipality')
```

Out[7]:

Municipality	Price
Minato Ward	3.435867e+08
Chiyoda Ward	2.814054e+08
Chuo Ward	2.189682e+08
Bunkyo Ward	1.864118e+08
Shinjuku Ward	1.706380e+08

In [8]:
```
mean_pairwise('LandShape')
```

Out[8]:

LandShape	Price
Irregular Shaped	4.086514e+08
Semi-shaped	3.117000e+08
Trapezoidal Shaped	2.869474e+08
Other LandShape	2.400000e+08
Semi-trapezoidal Shaped	2.209007e+08
Semi-rectangular Shaped	2.034132e+08
Semi-square Shaped	1.981786e+08
Square Shaped	1.762941e+08
Rectangular Shaped	1.758670e+08
Flag-shaped etc.	8.532500e+07

In [9]:
```
mean_pairwise('Structure')
```

Out[9]:

Structure	Price
SRC	6.090839e+08
RC	3.503991e+08
S	1.641401e+08
LS	1.251042e+08
W	8.214610e+07
B	5.766667e+07

In [10]:
```
mean_pairwise('Use')
```

Out[10]:

Use	Price
Other	5.792000e+08
Warehouse	4.950833e+08
Office	3.877884e+08
Housing Complex	2.924059e+08
Shop	2.808986e+08
Factory	1.777500e+08
House	1.381165e+08
Workshop	1.302500e+08

In [11]:

```
mean_pairwise('Direction')
```

Out[11]:

Direction	Price
Northwest	2.732897e+08
Southeast	2.608280e+08
Northeast	2.320756e+08
West	2.315943e+08
North	2.297664e+08
Southwest	2.256248e+08
East	2.063345e+08
South	1.975000e+08
No facing road	2.478750e+07

Step 2: Building the Regresion Model

Now I will use build a regression model using the `PyCaret` package.

1 - Feature Declaration

I begin with declaring all features based on their type.

In [12]:

```
num_f = ['MinuteToClosestStation', 'Area' , 'Frontage', 'BuildingYear']

cat_f = ['Type', 'Municipality', 'FloorPlan', 'LandShape', 'Structure', 'Use', 'Direction', 'Ren
```

---

2 - Model Setup

I set up the regressor, model target, and data split.

In [15]:
```
reg = setup(
    #Model Selection
    data = df,    target = 'Price',    train_size = 0.75,    fold = 10,    session_id = 19907,

    #Data Types
    numeric_features = num_f,    categorical_features = cat_f,    ignore_features = None)
```

	Description	Value
0	Session id	19907
1	Target	Price
2	Target type	Regression
3	Original data shape	(2108, 13)
4	Transformed data shape	(2108, 46)
5	Transformed train set shape	(1581, 46)
6	Transformed test set shape	(527, 46)
7	Numeric features	4
8	Categorical features	8
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Maximum one-hot encoding	25
14	Encoding method	None
15	Fold Generator	KFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	reg-default-name
21	USI	7183

3 - Model Comparison, Selection, and Training

Now, I run multiple different Machine Leraning models to determine the optimal model--based on RSME. I find that a Gradiant Boosting Regressor Model will be the most optimal model for the real estate data.

In [16]:
```
compare_models(sort = 'RMSE')
```

	Model	MAE	MSE	RMSE	R2
gbr	Gradient Boosting Regressor	68911659.8578	19259832008996756.0000	137360564.2537	0.850
rf	Random Forest Regressor	70697907.0516	19627170822730428.0000	139406800.6194	0.850
et	Extra Trees Regressor	72799348.4237	20354123367315708.0000	141698068.0860	0.847
xgboost	Extreme Gradient Boosting	72680613.5936	22571305760493688.0000	149168312.2651	0.829
ridge	Ridge Regression	88732875.8318	24950862610899216.0000	155660656.3434	0.815
llar	Lasso Least Angle Regression	88871519.6259	25004960999073504.0000	155823540.7895	0.815
lasso	Lasso Regression	88879323.0676	25006160077303712.0000	155827165.5617	0.815
en	Elastic Net	88938523.8367	25249900418238568.0000	156439842.3381	0.812
omp	Orthogonal Matching Pursuit	91662406.4268	25351970008235276.0000	157001579.0608	0.811
huber	Huber Regressor	88870193.5624	29671431287072508.0000	168754889.7742	0.797
par	Passive Aggressive Regressor	92165870.9509	31375335841402092.0000	173259113.5187	0.788
ada	AdaBoost Regressor	129017992.3013	30472120610804012.0000	173911780.2154	0.773
knn	K Neighbors Regressor	91763342.0110	30883343203631876.0000	174355441.0153	0.766
lightgbm	Light Gradient Boosting Machine	78010545.5972	37055097398948488.0000	188405368.3611	0.715
dt	Decision Tree Regressor	96294663.6414	38011590593503696.0000	193268595.7485	0.718
br	Bayesian Ridge	192830497.2367	156364063301113920.0000	386697674.5981	-0.00
dummy	Dummy Regressor	192830497.3052	156364063406369216.0000	386697674.7297	-0.00
lar	Least Angle	11374041916.2810	262132730254072899 1744.0000	16358569491.6800	-152

	Model	MAE	MSE	RMSE	R2
	Regression				
lr	Linear Regression	6664263884585.6016	70169744811202209289320005632.0000	83767522190684.5156	-770

```
Processing:   0%|          | 0/81 [00:00<?, ?it/s]
```

Out[16]:
```
GradientBoostingRegressor
GradientBoostingRegressor(random_state=19907)
```

Now I create the Gradient Boosting Regressor model:

In [18]:
```python
reg_gbf = create_model('gbr')
```

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	62006234.2573	19427766992284996.0000	139383524.8237	0.8260	0.3915	0.3166
1	64449622.2801	13740488909276138.0000	117219831.5528	0.8303	0.4594	0.4252
2	69085764.3071	23962343086408148.0000	154797748.9707	0.8688	0.4481	0.3955
3	68653152.9192	21844780506250792.0000	147799798.7355	0.7601	0.5254	0.6255
4	71230386.2928	26490352712437312.0000	162758571.8555	0.8961	0.3890	0.3691
5	68019683.8437	15243548424339896.0000	123464765.9227	0.9442	0.4734	0.4409
6	75477279.9737	21107898196078244.0000	145285574.6318	0.8772	0.4865	0.4797
7	79891084.7308	25324743292267292.0000	159137498.0709	0.6623	0.6093	1.0034
8	72408980.4230	15980414100936862.0000	126413662.6356	0.9044	0.4550	0.3988
9	57894409.5505	9475983869687832.0000	97344665.3376	0.9375	0.4453	0.4210
Mean	68911659.8578	19259832008996756.0000	137360564.2537	0.8507	0.4683	0.4876
Std	6092045.0366	5238078338355431.0000	19796651.1539	0.0818	0.0607	0.1885

```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

---

4 - Hyper-parameter Tunning

Now I can hyper tune and optimze the Gradiant Boosting Regressor to find a potential stronger model.

RMSE (Mean Squared Error) *The Root Mean Squared Error measures the average root squared difference between the predicted and actual values. It is calculated by taking the square of the difference between the predicted and actual values for each data point, and then taking the mean of these root squared differences. A smaller value of RMSE indicates better performance, as it means the model's predictions are closer to the actual values on average. In the table, the model with the lowest RMSE (135,193,045.1611) indicates that the model's predictions are off by around 135,193,045 Yen root squared from the actual values.*

R2 (R-squared) *The coefficient of determination, or R-squared, measures the proportion of variance in the target variable that can be explained by the independent variables in the model. An R-squared value of 1 indicates a perfect fit, meaning all the variance in the target variable is explained by the independent variables in the model. On the other hand, an R-squared value of 0 indicates that the model does not explain any of the variance in the target variable. In the table, the model with the highest R-squared (0.8181) indicates that around 81.81% of the variance in the target variable can be explained by the independent variables in the model.*

In [19]:
```
tuned_reg_gbf = tune_model(reg_gbf)
```

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	64795263.7668	22909130484171188.0000	151357624.4666	0.7948	0.4115	0.3309
1	66697820.1170	15499245618951110.0000	124495966.2758	0.8086	0.4121	0.3767
2	69595169.4635	25189466915659056.0000	158711899.0992	0.8621	0.4132	0.3690
3	65922587.2660	20510747830340356.0000	143215738.7662	0.7747	0.5185	0.5740
4	71414120.1244	27369502935639644.0000	165437308.1733	0.8926	0.4111	0.3782
5	67617701.6390	14711066261418062.0000	121289184.4371	0.9461	0.4924	0.4298
6	73789379.0185	21156202602230572.0000	145451719.1450	0.8769	0.4749	0.4691
7	78117346.8869	24720172834651032.0000	157226501.6931	0.6704	0.5958	1.0062
8	71067232.2593	16047743201894938.0000	126679687.4084	0.9040	0.4500	0.3888
9	56794095.3728	10217859128937966.0000	101083426.5789	0.9326	0.4623	0.4357
Mean	68581071.5914	19833113781389388.0000	139494905.6044	0.8463	0.4642	0.4759
Std	5459944.7989	5223433676836766.0000	19346449.0752	0.0801	0.0568	0.1882

```
Processing:   0%|          | 0/7 [00:00<?, ?it/s]
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Original model was better than the tuned model, hence it will be returned. NOTE: The display metr
ics are for the tuned model (not the original one).
```

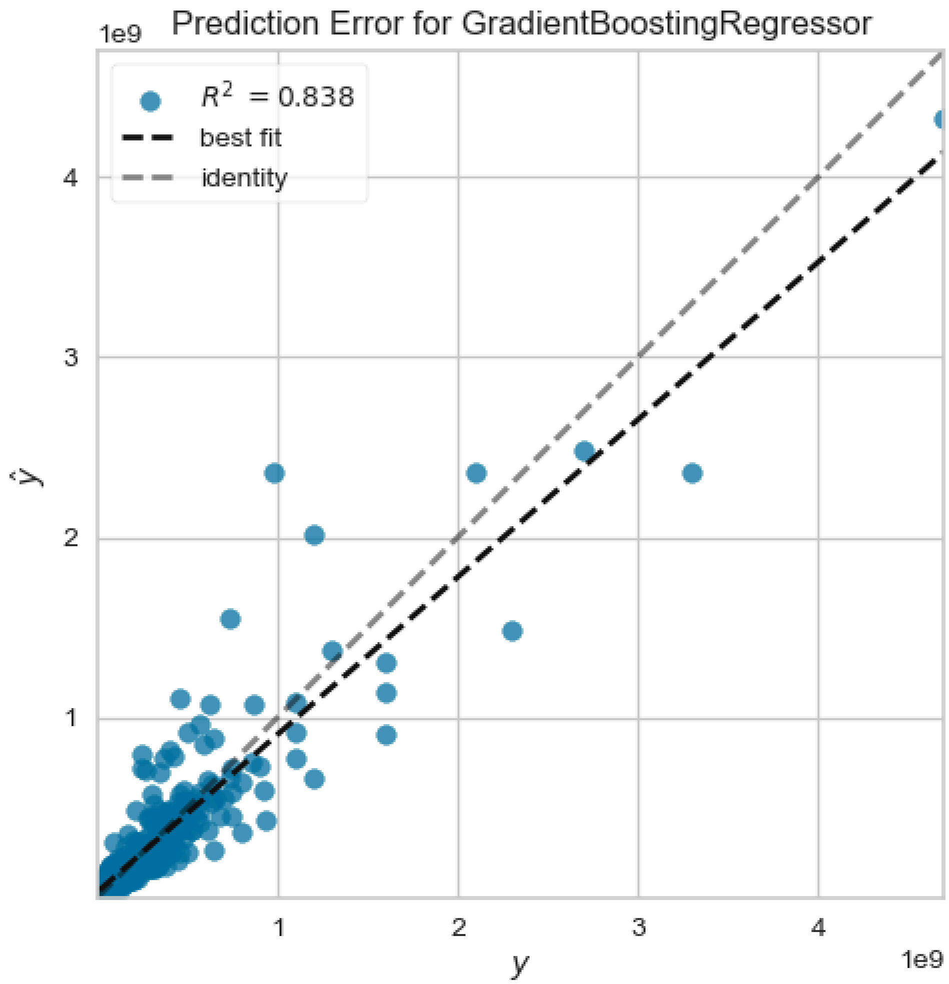
Tuning the Gradient did not yield a stronger model--thus the original model will be used moving forward.

---

5 - Model Analysis

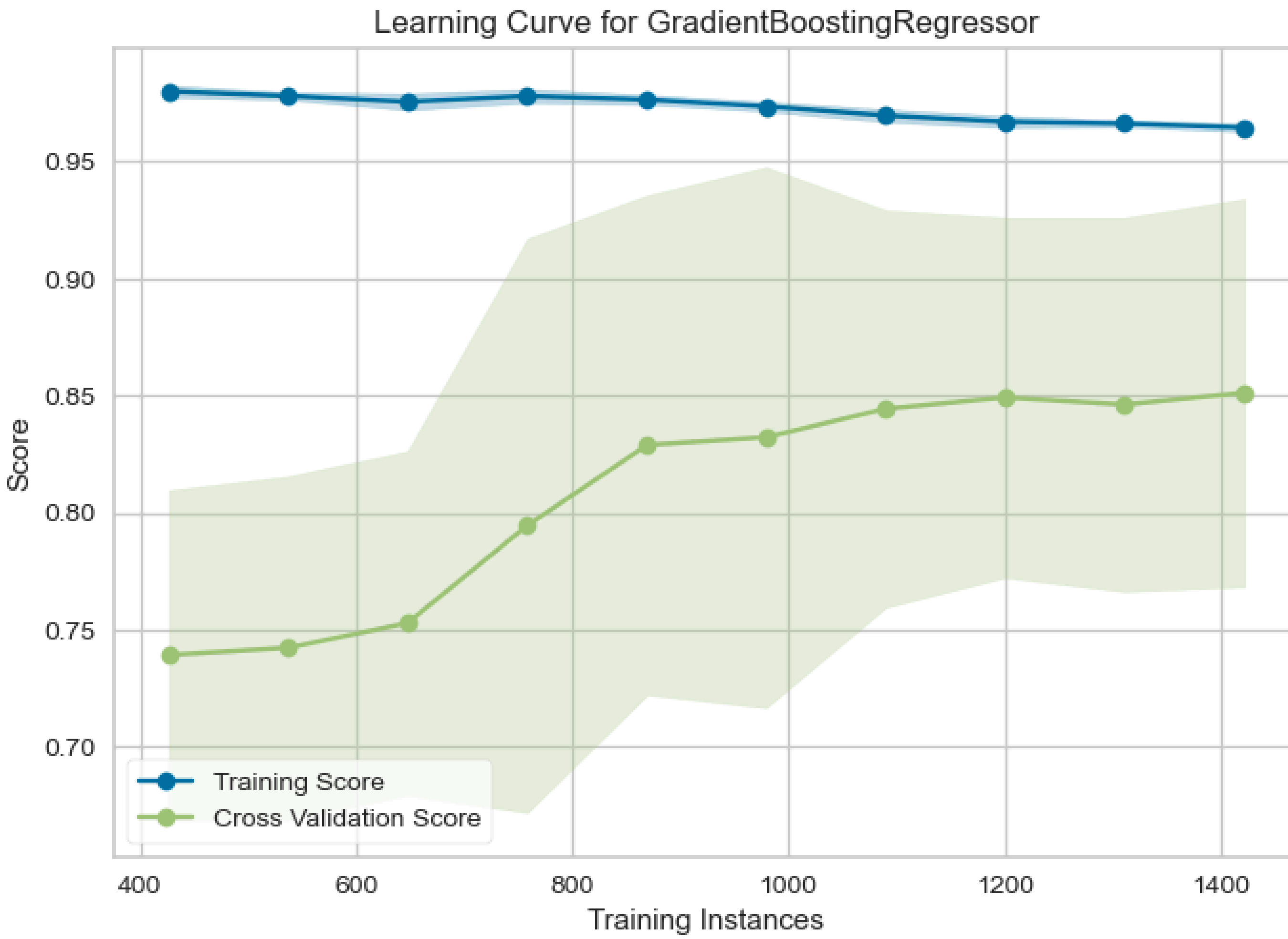
Now, I can analyze the performance of trained model on test set.

In [20]:
```
plot_model(tuned_reg_gbf, plot = 'error')
```



A Prediction Error plot is a graphical representation of the difference between the actual price and the predicted price of a regression model. It is a useful way to understand the distribution and magnitude of the errors made by the model across the range of price values. Data scientists can diagnose regression models using this plot by comparing against the 45 degree line (Identity), where the prediction exactly matches the model. The plot for the GBR model will show the predicted price (x-axis) versus the difference between the actual values (y-axis). Since the errors are evenly distributed around zero and have no apparent pattern or trend, it suggests that the GBR model has accurately predicted the price values.

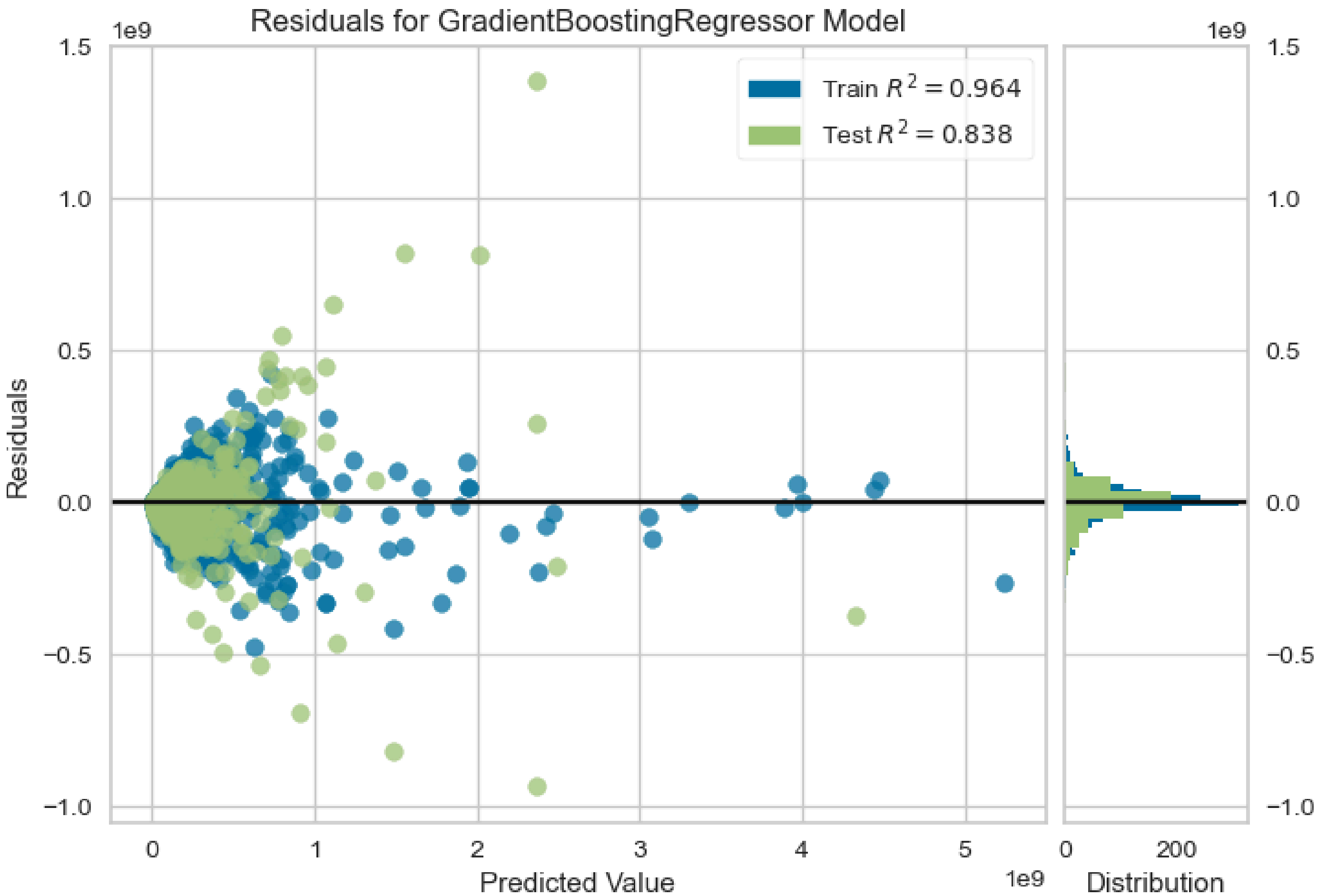
In [21]:
```
plot_model(tuned_reg_gbf, plot = 'learning')
```



Learning curves are plots used to show a model's performance as the training set size increases. They can also display the model's performance over a defined period of time. Learning curves are a useful tool for understanding the bias-variance tradeoff of a model and identifying whether the model is underfitting or overfitting the data.

The validation score typically increases with the number of training samples but eventually reaches a plateau, indicating that the model is not overfitting the data. As the gap between the training and validation scores becomes small and stable, it indicates that the model is achieving a good balance and is neither overfitting nor underfitting the data.

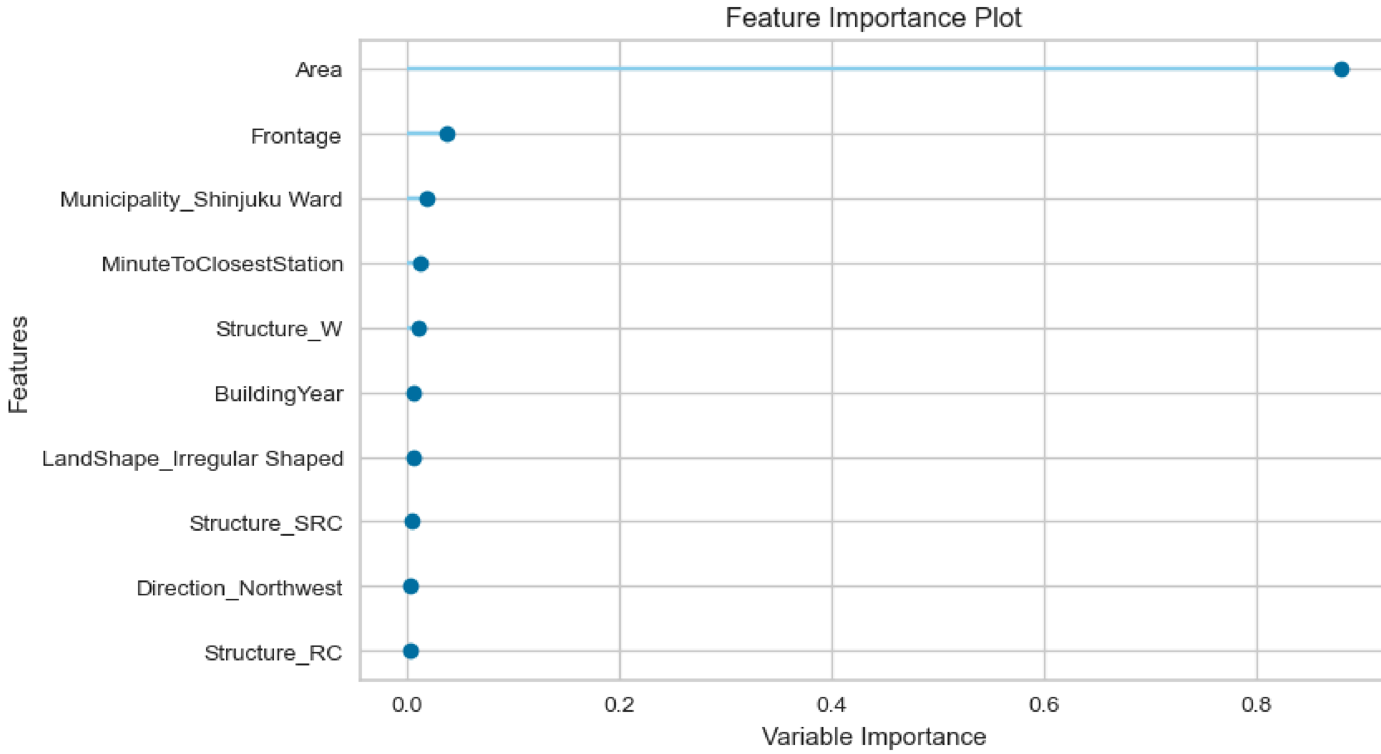
In [22]:
```
plot_model(tuned_reg_gbf, plot = 'residuals')
```



A residual plot is a scatterplot that displays the residuals on the vertical axis and the predicted values on the horizontal axis. Residual plots help determine whether a given model is appropriate for modeling the given data. Ideally, a residual plot should exhibit no clear trend and a lack of heteroskedasticity.

In this plot, both the training and testing data clearly do not exhibit any pattern. However, while the training data shows a rather tight residual spread, the testing set exhibits varying magnitudes as the predicted value increases.

In [23]:
```
plot_model(tuned_reg_gbf, plot = 'feature')
```



A Feature Importance Plot is a graphical representation of the relative importance of the predictor variables in a regression model. It's a useful way to understand the contribution of each variable to the prediction of the price. This plot shows the `Area` is by far the most important feature to predict the price of house. Naturally, this makes sense in the context of the real state industry.

---

6 - Finalize the Model

Now, we run the trained model on test to see the result.

In [26]:
```
predict_model(tuned_reg_gbf)
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Gradient Boosting Regressor	73909492.8351	22221867178801316.0000	149070007.6434	0.8377	0.4820	0.6905

Out[26]:

	Type	Municipality	MinuteToClosestStation	FloorPlan	Area	LandShape	Frontage	Building'
17991	Residential Land(Land and Building)	Chuo Ward	2.0	1K	260	Semi-shaped	17.0	19
30	Residential Land(Land and Building)	Chiyoda Ward	5.0	1K	210	Semi-rectangular Shaped	13.0	19
9846	Residential Land(Land and Building)	Minato Ward	3.0	1K	40	Rectangular Shaped	5.0	19
15645	Residential Land(Land and Building)	Shinjuku Ward	7.0	1K	45	Rectangular Shaped	5.5	20
14786	Residential Land(Land and Building)	Shinjuku Ward	8.0	1K	70	Rectangular Shaped	6.5	19
...	...	...	...	...	...	...	...	
15012	Residential Land(Land and Building)	Minato Ward	4.0	1K	90	Semi-square Shaped	9.0	19
16319	Residential Land(Land and Building)	Shinjuku Ward	9.0	1K	85	Semi-rectangular Shaped	5.5	19
14593	Residential Land(Land and Building)	Shinjuku Ward	6.0	1K	60	Semi-trapezoidal Shaped	2.2	20
16409	Residential Land(Land and Building)	Chuo Ward	4.0	1K	80	Rectangular Shaped	6.2	19
18381	Residential Land(Land and Building)	Shinjuku Ward	5.0	1K	170	Semi-rectangular Shaped	7.6	19

527 rows × 14 columns

Now we finalize the model. The purpose of this function is to train the model on the complete dataset before it is deployed in production. This is done this to train the model on the entire dataset (including the test set) and prepare the model for deployment. The function takes a trained model object and additional parameters and returns a finalized model object that is ready to be used for making predictions on new data.

In [27]:
```
final = finalize_model(tuned_reg_gbf)
final
```

Out[27]:
```
Pipeline
  numerical_imputer: TransformerWrapper
    transformer: SimpleImputer
      SimpleImputer
  categorical_imputer: TransformerWrapper
    transformer: SimpleImputer
      SimpleImputer
  onehot_encoding: TransformerWrapper
    transformer: OneHotEncoder
      OneHotEncoder
  GradientBoostingRegressor
```

Now that the model is finalized, I can rerun it with the complete dataset before saving it for deployment. As expected, the R2 has increased, and the RMSE has decreased. This improvement is because the final model is trained on the complete dataset, which includes the test set.

In [28]:
```
predict_model(final, data = df)
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Gradient Boosting Regressor	52913530.7619	7162347678709238.0000	84630654.4859	0.9527	0.4478	0.5059

Out[28]:

	Type	Municipality	MinuteToClosestStation	FloorPlan	Area	LandShape	Frontage	Building'
5	Residential Land(Land and Building)	Chiyoda Ward	4.0	1K	65	Semi-rectangular Shaped	6.5	19
18	Residential Land(Land and Building)	Chiyoda Ward	3.0	1K	80	Semi-trapezoidal Shaped	6.8	19
30	Residential Land(Land and Building)	Chiyoda Ward	5.0	1K	210	Semi-rectangular Shaped	13.0	19
31	Residential Land(Land and Building)	Chiyoda Ward	4.0	1K	105	Semi-rectangular Shaped	6.8	19
90	Residential Land(Land and Building)	Chiyoda Ward	4.0	1K	640	Trapezoidal Shaped	35.0	19
...	...	...	...	...	...	...	...	
19954	Residential Land(Land and Building)	Bunkyo Ward	4.0	1K	35	Rectangular Shaped	3.7	19
19956	Residential Land(Land and Building)	Minato Ward	1.0	1K	230	Semi-trapezoidal Shaped	10.0	19
19975	Residential Land(Land and Building)	Minato Ward	16.0	1K	85	Semi-square Shaped	8.0	19
19988	Residential Land(Land and Building)	Bunkyo Ward	6.0	1K	30	Rectangular Shaped	3.0	20
19989	Residential Land(Land and Building)	Bunkyo Ward	6.0	1K	30	Rectangular Shaped	3.0	19

2108 rows × 14 columns

---

7 - Final Comments and Saving the Model

In this notebook I have conducted correlation analysis which showed that the area and frontage are correlated with the price of real estate. I also break down the relationship of each categorical features with price. Using `PyCaret` package I identify the approprate ML model for the dataset--a stock Gradiant Boosting Regressor. I explore various aspects of the model and present my findings.

Lastly, I save the final model for any future prediction.

In [149…

```
GBR_Model = save_model(final,'Final Regression Model')
```

```
Transformation Pipeline and Model Successfully Saved
```