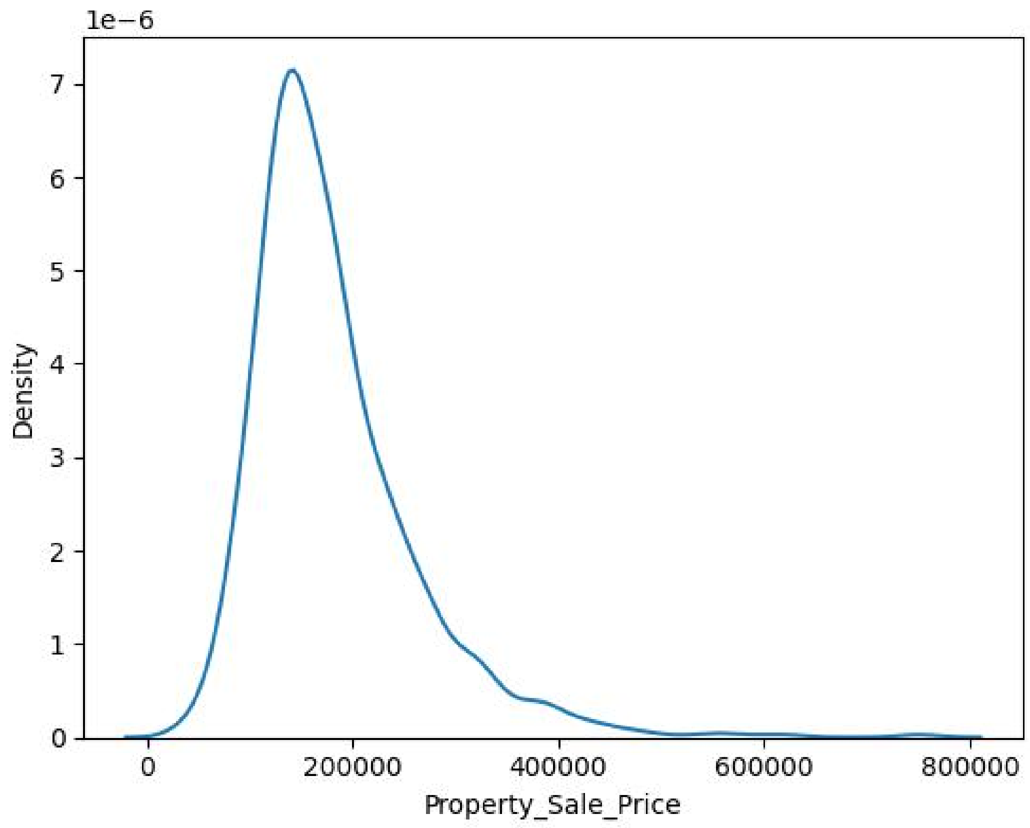
Modeling Report for House Price Prediction Project

The target variable "Property_Sale_Price":

```
sns.kdeplot(df['Property_Sale_Price'])
```

```
<Axes: xlabel='Property_Sale_Price', ylabel='Density'>
```



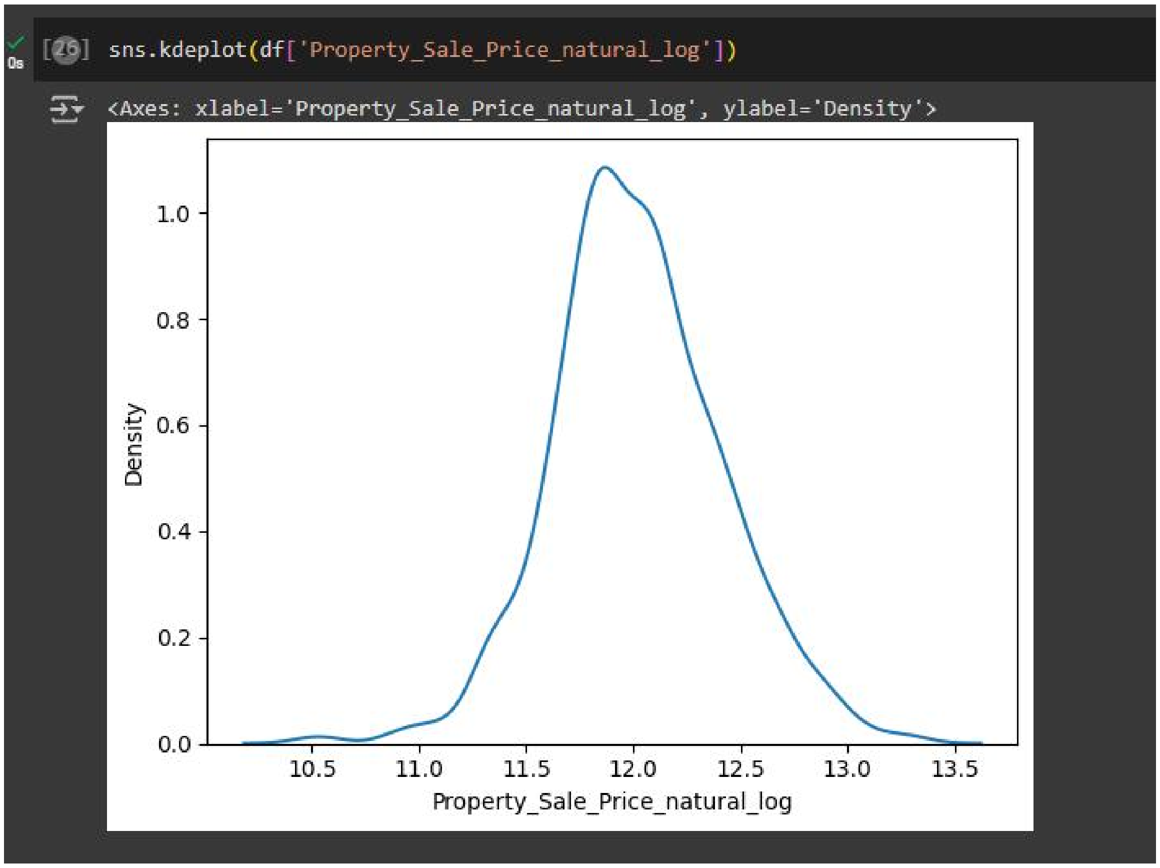
The target variable, "Property_Sale_Price," appears to be right-skewed, meaning that most of the property prices are concentrated towards the lower end, with a few higher-priced properties extending the distribution's tail. This skewness can affect the performance of regression models, particularly those that assume normally distributed residuals, like linear regression.

To handle this skewness, you might consider applying a transformation to the target variable, such as a log transformation (`log(Property_Sale_Price)`), to make the distribution more normal. This can help improve model performance and make the predictions more accurate for a wider range of property prices.

Trandsform Data

Use log Transform on Property_Sale_Price for better distribution

```
df['Property_Sale_Price_natural_log'] = np.log(df['Property_Sale_Price'])
```



```
[26] sns.kdeplot(df['Property_Sale_Price_natural_log'])
```

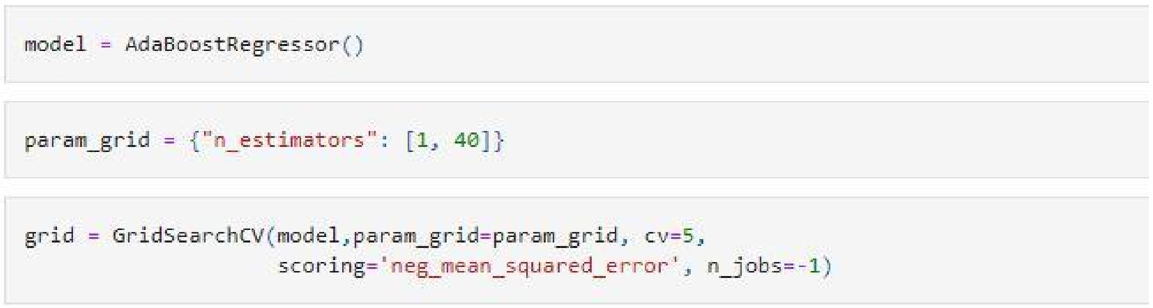
```
<Axes: xlabel='Property_Sale_Price_natural_log', ylabel='Density'>
```

We use **'Property_Sale_Price_natural_log':**

The log transformation is often used to reduce skewness of a measurement variable.

1- AdaBoost Regressor:

n_estimators: Controls the number of weak learners (decision trees by default). A higher number may improve performance but could also increase training time.(Impact on performance and trade-offs)



```
model = AdaBoostRegressor()
```

```
param_grid = {"n_estimators": [1, 40]}
```

```
grid = GridSearchCV(model,param_grid=param_grid, cv=5,
                    scoring='neg_mean_squared_error', n_jobs=-1)
```

Evaluation using original house price

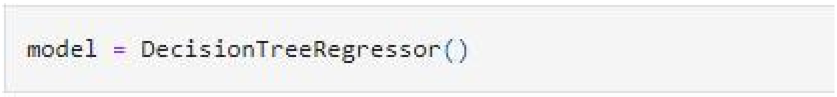
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.82	19.54	13.682

Evaluation using Log Transform of house price

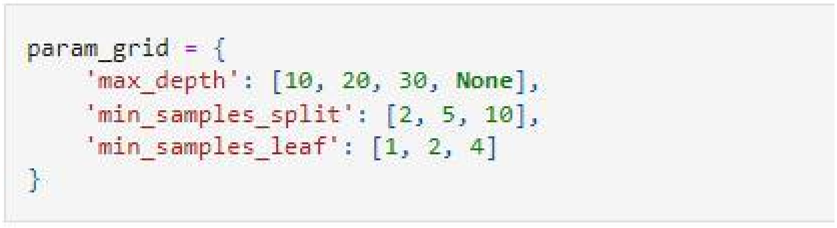
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.836	1.416	1.077

2- Decision Tree Regressor:

- `max_depth`: The maximum depth of the tree. Higher values increase model complexity.
- `min_samples_split`: The minimum number of samples required to split an internal node.
- `min_samples_leaf`: The minimum number of samples that must be present in a leaf node



```
model = DecisionTreeRegressor()
```



```
param_grid = {
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Evaluation using original house price

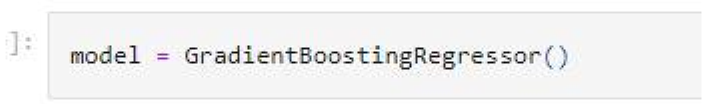
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.77	21.9	13.449

Evaluation using Log Transform of house price

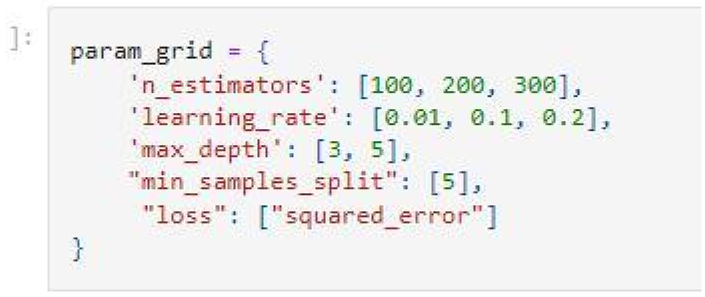
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.836	1.415	1.04

3- **Gradient Boosting Regressor:**

- `n_estimators`: Number of boosting stages.
- `learning_rate`: Shrinks the contribution of each tree. Smaller values make the model more robust to overfitting but require more iterations.
- `max_depth`: Limits the depth of the trees to control overfitting.
- `min_samples_split`: Controls minimum samples required to split.
- `loss`: Loss function to be minimized.



```
]:  model = GradientBoostingRegressor()
```



```
]:  param_grid = {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 5],
        "min_samples_split": [5],
         "loss": ["squared_error"]
    }
```

Evaluation using original house price

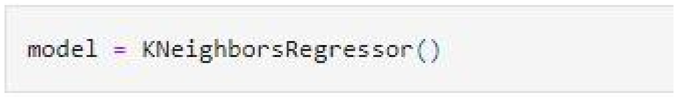
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.899	14.67	9.29

Evaluation using Log Transform of house price

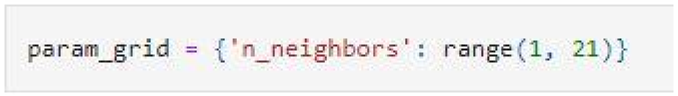
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.93	0.93	0.7

4- KNeighborsRegressor :

- `n_neighbors`: Number of neighbors to use for predictions. Smaller numbers can increase model variance, larger numbers reduce it.



```
model = KNeighborsRegressor()
```



```
param_grid = {'n_neighbors': range(1, 21)}
```

Evaluation using original house price

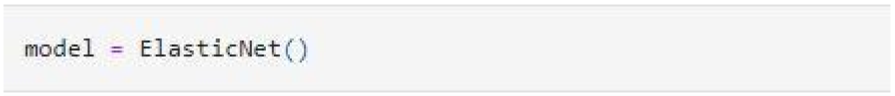
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.74	23.67	14.76

Evaluation using Log Transform of house price

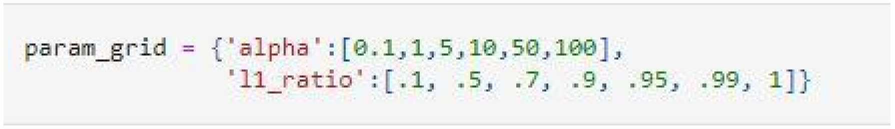
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.75	1.75	1.23

5- **ElasticNet :**

- `alpha`: Regularization strength.
- `l1_ratio`: Mix between L1 (Lasso) and L2 (Ridge) regularization. 1 means Lasso, 0 means Ridge.



```
model = ElasticNet()
```



```
param_grid = {'alpha':[0.1,1,5,10,50,100],
              'l1_ratio':[.1, .5, .7, .9, .95, .99, 1]}
```

Evaluation using original house price

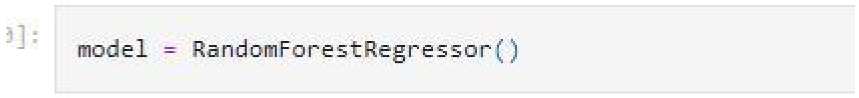
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.89	15.37	9.7

Evaluation using Log Transform of house price

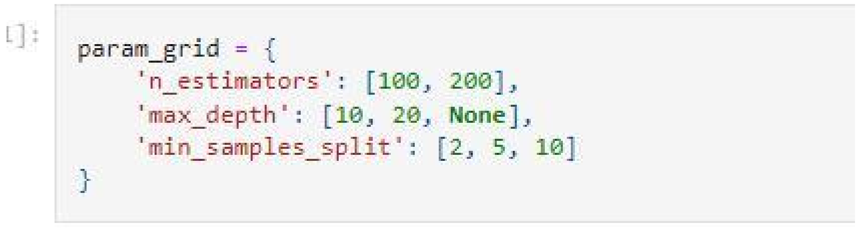
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.92	0.96	0.7

6- RandomForestRegressor :

- `n_estimators`: Number of trees in the forest.
- `max_depth`: Limits the depth of the trees.
- `min_samples_split`: Minimum samples to split a node.



```
model = RandomForestRegressor()
```



```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}
```

Evaluation using original house price

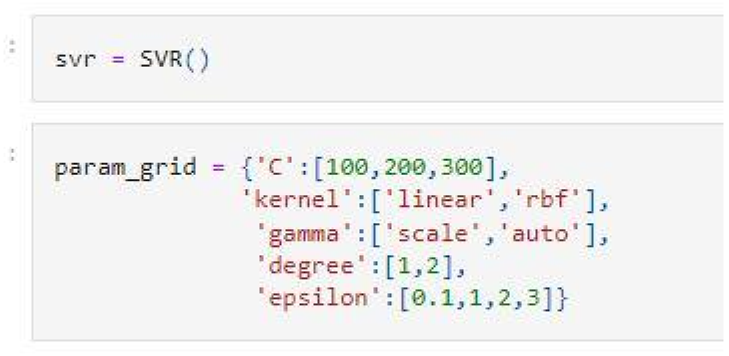
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.88	15.82	9.82

Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.9	1.07	0.75

7- **Support Vector Regressor (SVR) :**

- C: Regularization parameter. Larger values give higher accuracy but risk overfitting.
- kernel: Kernel type (e.g., 'linear', 'rbf').
- gamma: Kernel coefficient for non-linear models.
- degree: Degree of the polynomial kernel function (used for 'poly' kernel).
- epsilon: Epsilon in the epsilon-SVR model.



```
svr = SVR()
```

```
param_grid = {'C':[100,200,300],
              'kernel':['linear','rbf'],
               'gamma':['scale','auto'],
               'degree':[1,2],
               'epsilon':[0.1,1,2,3]}
```

Evaluation using original house price

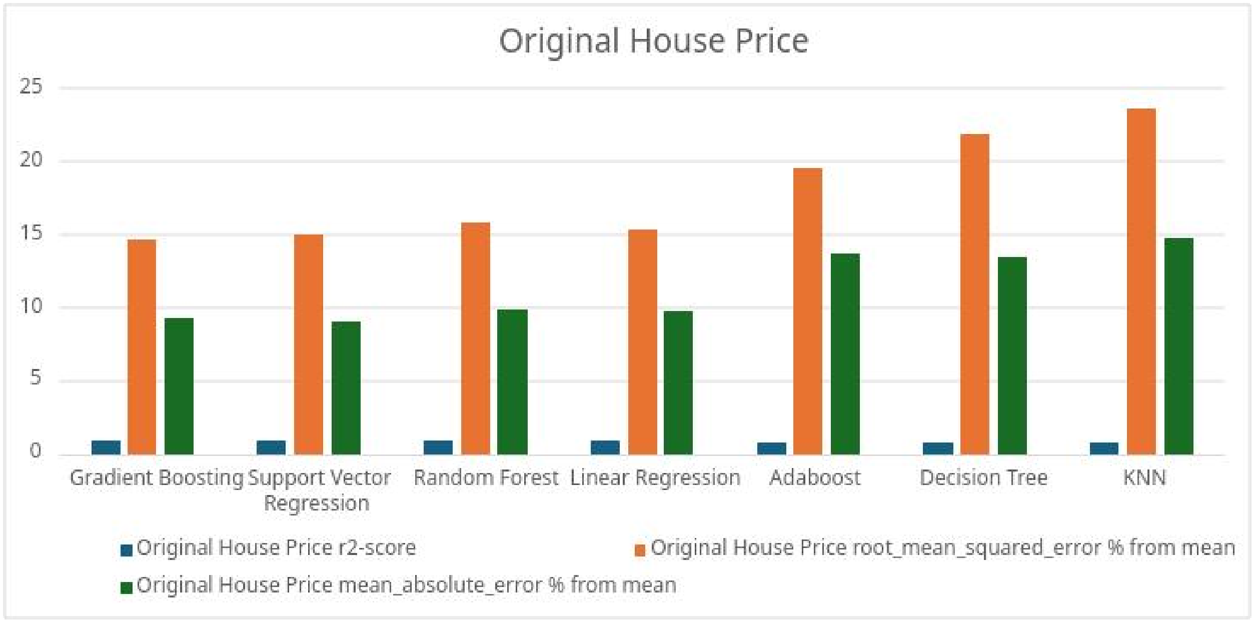
r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.89	15.06	9.097

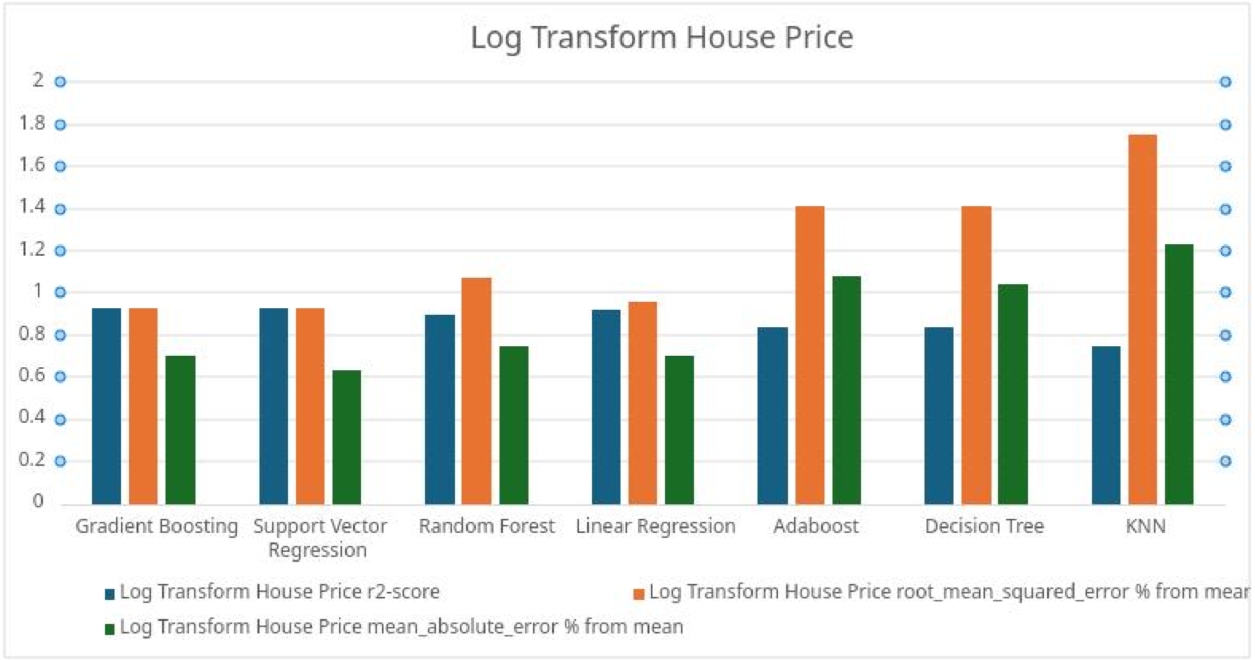
Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.93	0.93	0.63

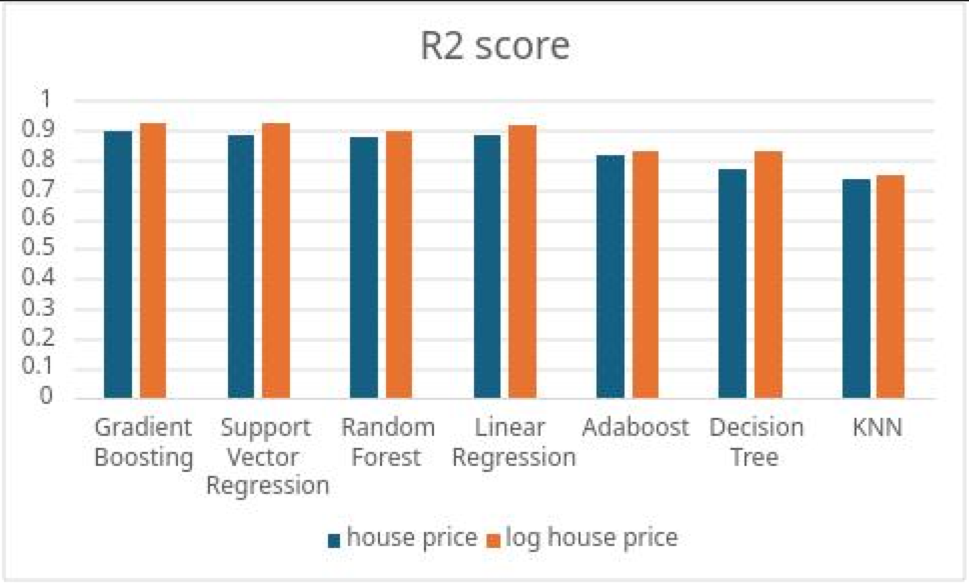
Conclusion:

- **Gradient Boosting** and **Support Vector Regression** stand out as the best-performing models, showing lower error percentages and higher R² scores, which indicate better predictive accuracy.
- **Random Forest** and **Linear Regression** are also strong performers, offering a balance between simplicity and effectiveness.
- **Adaboost**, **Decision Tree**, and **K-Nearest Neighbors (KNN)** perform worse, with higher errors and lower R² scores, suggesting they may need further tuning or could be excluded from further analysis.





algorithm	house price	Log house price
Gradient Boosting	0.899	0.93
Support Vector Regression	0.89	0.93
Random Forest	0.88	0.9
Linear Regression	0.89	0.92
Adaboost	0.82	0.836
Decision Tree	0.77	0.836
KNN	0.74	0.75


R2 score
Gradient Boosting
Support Vector Regression
Random Forest
Linear Regression
Adaboost
Decision Tree
KNN
house price
log house price