PredictingHousingPrices

March 31, 2025

1 Predicting Housing Prices Based on Different Features

1.1 Purpose

Predicting housing prices accurately is crucial for buyers, sellers, real estate agents, and investors. The House Prices - Advanced Regression Techniques dataset from Kaggle provides a rich collection of 79 explanatory variables describing various aspects of residential homes in Ames, Iowa. The objective is to predict the final sale price of each home.

In this project, we explore predictive modeling approaches to forecast house prices. Our goal is to build a regression model that minimizes the Root Mean Squared Error (RMSE), as this metric is used by the competition to evaluate model performance. However, I will be using other evaluation metrics as well for better comparison.

[4]:
```
import pandas as pd
import numpy as np

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

[5]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
```

```
12  Neighborhood   1460 non-null   object
13  Condition1     1460 non-null   object
14  Condition2     1460 non-null   object
15  BldgType       1460 non-null   object
16  HouseStyle     1460 non-null   object
17  OverallQual    1460 non-null   int64
18  OverallCond    1460 non-null   int64
19  YearBuilt      1460 non-null   int64
20  YearRemodAdd   1460 non-null   int64
21  RoofStyle      1460 non-null   object
22  RoofMatl       1460 non-null   object
23  Exterior1st    1460 non-null   object
24  Exterior2nd    1460 non-null   object
25  MasVnrType     588 non-null    object
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
59  GarageYrBlt    1379 non-null   float64
```

```
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

1.2 Preprocessing

We observe: - 1460 training samples - Some features have missing values - Mix of numerical and categorical features

[7]:
```
# Drop features with too many missing values or not useful
train = train.drop(columns=['Alley', 'PoolQC', 'Fence', 'MiscFeature'])

# Fill in missing values
train['LotFrontage'] = train['LotFrontage'].fillna(train['LotFrontage'].
 ↪median())
train['GarageYrBlt'] = train['GarageYrBlt'].fillna(train['GarageYrBlt'].
 ↪median())

# Fill categorical NA with 'None' or mode
for col in train.select_dtypes(include='object'):
    train[col] = train[col].fillna('None')

# Fill remaining numeric NA with median
for col in train.select_dtypes(include='number'):
    train[col] = train[col].fillna(train[col].median())
```

[8]:
```
train_encoded = pd.get_dummies(train.drop(columns=['Id']), drop_first=True)
train.head()
```

[8]:

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street LotShape LandContour  \
0   1          60       RL         65.0     8450   Pave      Reg         Lvl
1   2          20       RL         80.0     9600   Pave      Reg         Lvl
2   3          60       RL         68.0    11250   Pave      IR1         Lvl
3   4          70       RL         60.0     9550   Pave      IR1         Lvl
4   5          60       RL         84.0    14260   Pave      IR1         Lvl

  Utilities LotConfig  … EnclosedPorch 3SsnPorch ScreenPorch PoolArea  \
0    AllPub    Inside  …             0         0           0        0
1    AllPub       FR2  …             0         0           0        0
2    AllPub    Inside  …             0         0           0        0
3    AllPub    Corner  …           272         0           0        0
4    AllPub       FR2  …             0         0           0        0

  MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
0       0      2    2008        WD         Normal     208500
1       0      5    2007        WD         Normal     181500
2       0      9    2008        WD         Normal     223500
3       0      2    2006        WD        Abnorml     140000
4       0     12    2008        WD         Normal     250000

[5 rows x 77 columns]
```

[9]:

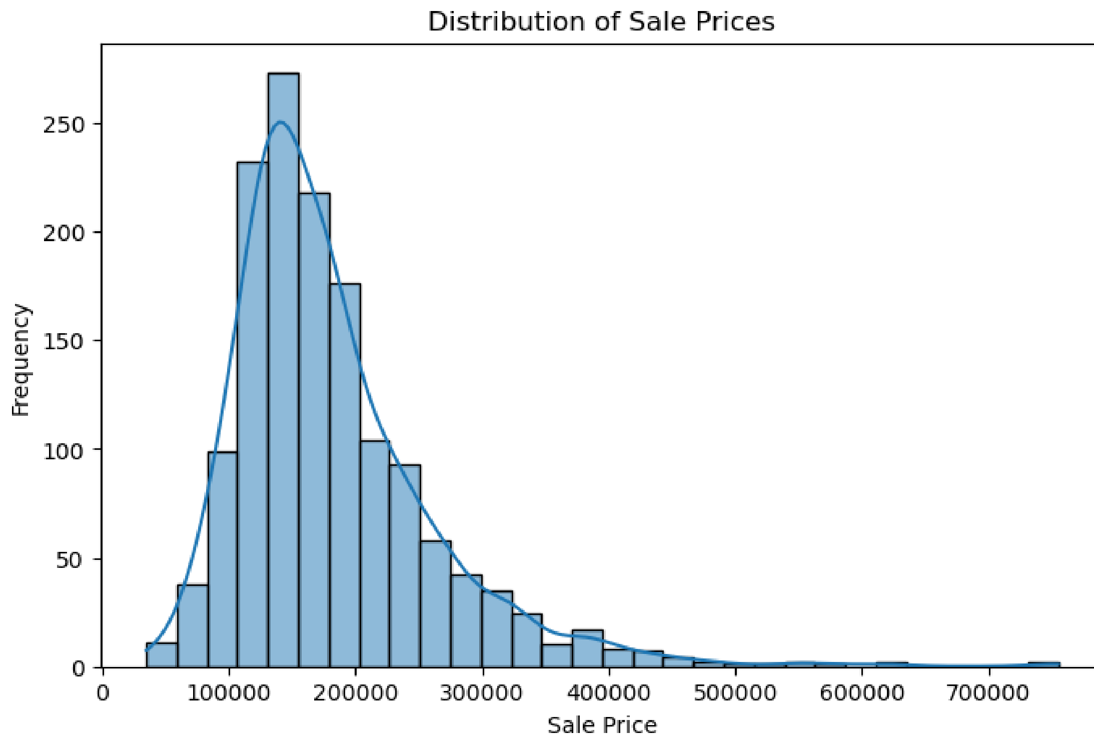
```
X = train_encoded.drop('SalePrice', axis=1)
y = train_encoded['SalePrice']

y_log = np.log1p(y)
```

In the preprocessing stage, we began by addressing missing values and preparing the dataset for modeling. Several features such as Alley, PoolQC, Fence, and MiscFeature contained a large number of missing values and were deemed either too sparse or not useful, so they were dropped from the dataset. For numerical features with missing values like LotFrontage and GarageYrBlt, we filled them using the median of each respective column. Categorical variables were imputed with the string 'None' to indicate the absence of a feature (e.g., no alley access or no fireplace), preserving potentially meaningful structural information. Any remaining numeric columns with missing data were also filled using the median to maintain consistency. After handling missing values, we applied one-hot encoding using pd.get_dummies() to convert categorical variables into binary indicator variables, enabling them to be used effectively in regression models. We also dropped the Id column (as it is just an identifier) and separated the feature matrix X from the target variable y, which contains the sale prices. This preprocessing pipeline ensures the data is clean, fully numeric, and suitable for model training.

1.3 Visualizations

Before we move on, I believe it is important to understand our data's limitations, specifically when it comes to distribution between category. I believe that understanding the data on a deeper level is valuable so we'll be using some visualizations.

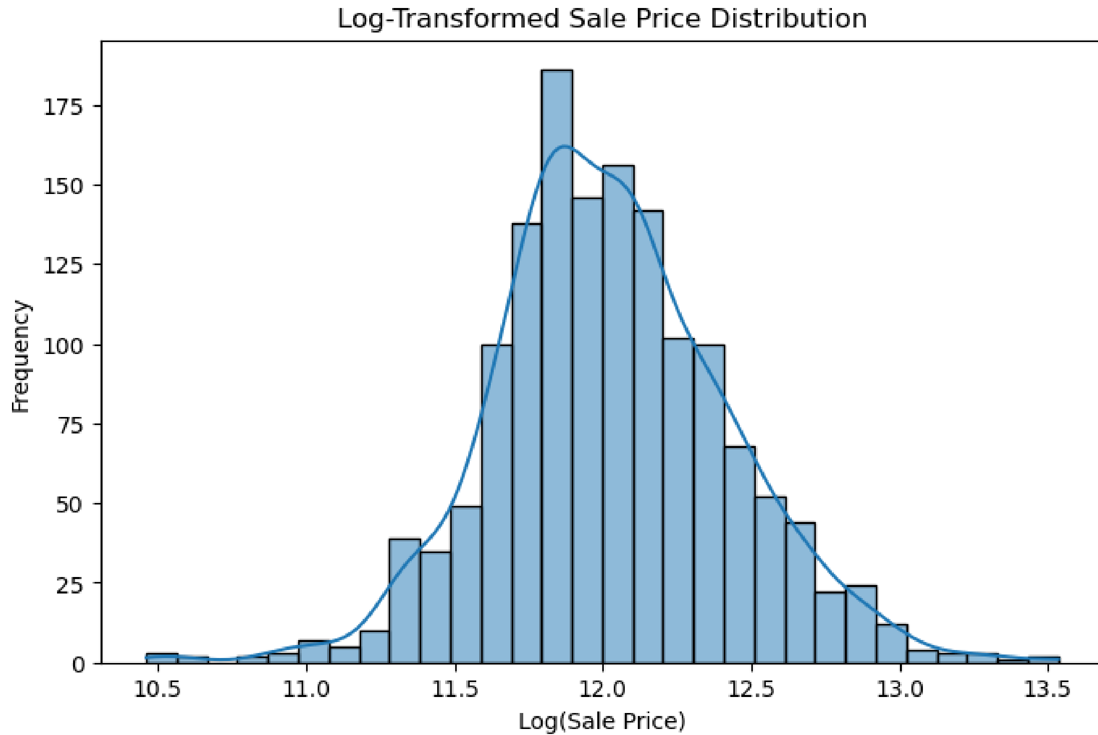
[13]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

plt.figure(figsize=(8,5))
sns.histplot(train['SalePrice'], kde=True, bins=30)
plt.title('Distribution of Sale Prices')
plt.xlabel('Sale Price')
plt.ylabel('Frequency')
plt.show()
```



The distribution of sale prices is right-skewed, meaning a few expensive houses pull the average higher. This justifies using a log transformation to normalize the target variable.

1.3.1 Log-Transformed SalePrice Distribution

[16]:
```python
plt.figure(figsize=(8,5))
sns.histplot(np.log1p(train['SalePrice']), kde=True, bins=30)
plt.title('Log-Transformed Sale Price Distribution')
plt.xlabel('Log(Sale Price)')
plt.ylabel('Frequency')
plt.show()
```



After applying log1p, the distribution of prices becomes more normal, which improves performance for regression models that assume normality of residuals.

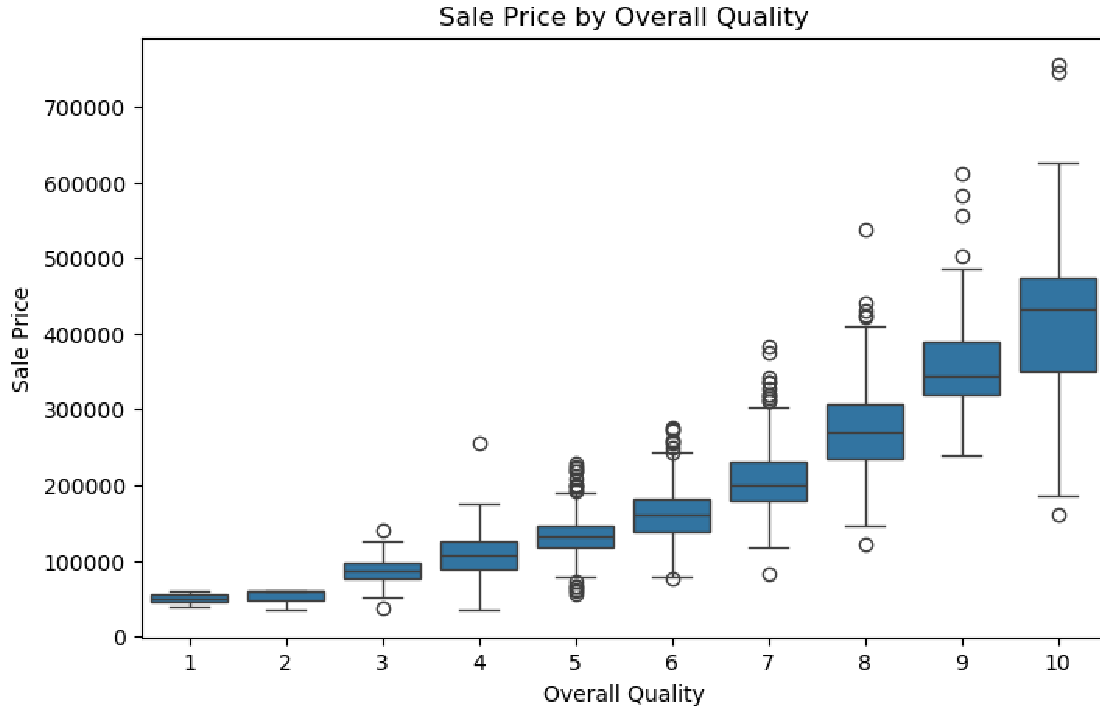
1.3.2 Sale Price vs GrLivArea

[19]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(data=train, x='GrLivArea', y='SalePrice')
plt.title('Sale Price vs. Above Ground Living Area')
plt.xlabel('GrLivArea')
plt.ylabel('SalePrice')
plt.show()
```

There is a clear positive correlation between living area and sale price—larger houses tend to be more expensive. However, some outliers (very large homes with low prices) may affect the model.

1.3.3 Score Category Breakdown by Family Income

[22]:
```
plt.figure(figsize=(8,5))
sns.boxplot(data=train, x='OverallQual', y='SalePrice')
plt.title('Sale Price by Overall Quality')
plt.xlabel('Overall Quality')
plt.ylabel('Sale Price')
plt.show()
```



Overall quality is one of the most important predictors. Houses with higher quality ratings tend to have significantly higher prices.

1.3.4 Linear Regression - Experiment 1

What is Regression? Regression is a type of supervised machine learning used to predict continuous outcomes based on input variables (also called features). The goal is to model the relationship between the dependent variable (target) and one or more independent variables (features).

For example, in a project predicting house prices, the target variable is the house price, and the features might include square footage, number of bedrooms, neighborhood, etc.

What is Linear Regression? Linear regression is one of the simplest and most widely used forms of regression. It assumes a linear relationship between the input features and the target variable.

In simple linear regression (with one feature), the model is:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

- y: predicted value (e.g., house price) - x: input feature (e.g., square footage) - β_0: intercept (bias term) - β_1: slope (coefficient for the feature) - ε: error term (difference between actual and predicted)

In multiple linear regression (with multiple features), the formula becomes:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \varepsilon$$

Or, in vector/matrix form:

$$\hat{y} = X\beta$$

Where: - $\hat{y}$: vector of predicted values - X: matrix of input features - β: vector of coefficients

The model learns the best values for the coefficients β by minimizing the cost function, typically the Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The model's goal is to minimize this error during training.

[25]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import (
    mean_squared_error,
    mean_squared_log_error,
    mean_absolute_error,
    r2_score
)

# split the data
X_train, X_val, y_train, y_val = train_test_split(X, y_log, test_size=0.2,␣
↪random_state=42)

# train
model = LinearRegression()
model.fit(X_train, y_train)

# predict on validation
y_pred = model.predict(X_val)

# evaluate
rmse = np.sqrt(mean_squared_error(y_val, y_pred))
rmsle = np.sqrt(mean_squared_log_error(y_val, y_pred))
mae = mean_absolute_error(y_val, y_pred)
r2 = r2_score(y_val, y_pred)

print(f'RMSE:   {rmse:.4f}')
print(f'RMSLE:  {rmsle:.4f}')
print(f'MAE:    {mae:.4f}')
print(f'R²:     {r2:.4f}')
```

```
RMSE:   0.2109
RMSLE:  0.0173
MAE:    0.0981
R²:     0.7616
```

Evaluation of Experiment 1 In Experiment 1, I trained a baseline linear regression model and evaluated its performance using multiple regression metrics. The model achieved an RMSE of 0.2109, which indicates that the average prediction error on the log-transformed house prices is relatively low. The RMSLE value of 0.0173 suggests that the model performs particularly well in terms of predicting values that are close in relative scale to the actual prices—this is useful given the skewed nature of price distributions. The MAE of 0.0981 further confirms that the average magnitude of errors is under 0.1, meaning predictions are generally close to actual values. Lastly, the R^2 score of 0.7616 implies that the model explains about 76% of the variance in the target variable, which is a strong result for an initial experiment using basic preprocessing and linear regression. These results establish a solid baseline and give direction for further experiments, such as testing alternative models or improving feature selection.

1.3.5 Lasso Regression

Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a regularized version of linear regression that adds an L_1 penalty to the loss function. This penalty encourages the model to reduce the magnitude of less important feature coefficients to zero, which can effectively perform feature selection while fitting the model.

In standard linear regression, the goal is to minimize the Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2$$

Lasso regression modifies this by adding an L_1 regularization term:

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Where: - y_i: the actual target value. - β_0: the intercept. - β_j: the coefficients for the features x_{ij}. - n: the number of observations. - λ (or sometimes α): the regularization parameter controlling the strength of the penalty.

The effect of the L_1 penalty is twofold: - Regularization: It helps prevent overfitting by discouraging large coefficients. - Feature Selection: It can shrink some coefficients exactly to zero, effectively eliminating less relevant features from the model.

This balance between accurately fitting the data and keeping the model simple and interpretable makes Lasso regression a powerful tool in regression problems.

[28]:
```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_squared_log_error,␣
↪mean_absolute_error, r2_score
```

```python
import numpy as np

# Using the same train/test split as Experiment 1

# initialize and train a Lasso regression model (experiment with the alpha␣
↪parameter using 0.0001)
model_lasso = Lasso(alpha=0.0001, random_state=42)
model_lasso.fit(X_train, y_train)

# predict on the validation set
y_pred_lasso = model_lasso.predict(X_val)

# calculate evaluation metrics
rmse_lasso = np.sqrt(mean_squared_error(y_val, y_pred_lasso))
rmsle_lasso = np.sqrt(mean_squared_log_error(y_val, y_pred_lasso))
mae_lasso = mean_absolute_error(y_val, y_pred_lasso)
r2_lasso = r2_score(y_val, y_pred_lasso)

print(f'Lasso RMSE:   {rmse_lasso:.4f}')
print(f'Lasso RMSLE:  {rmsle_lasso:.4f}')
print(f'Lasso MAE:    {mae_lasso:.4f}')
print(f'Lasso R²:     {r2_lasso:.4f}')
```

```
Lasso RMSE:   0.1442
Lasso RMSLE:  0.0113
Lasso MAE:    0.0932
Lasso R²:     0.8885
```

Evaluation - Experiment 2 I used an alpha of 0.0001 because based on experiments, it created the lowest RMSE. Based on these metrics, Lasso Regression shows how introducing an L_1 penalty affects model performance. The RMSE and RMSLE values indicate how closely the predicted values match the true targets on both absolute and relative scales, respectively. The MAE (Mean Absolute Error) reveals the average magnitude of errors, which can help interpret how far off predictions are in practical terms. Finally, the R^2 score reflects the proportion of variance in the target variable explained by the model. Comparing these metrics to those from the baseline linear regression experiment helps you determine whether regularization and feature selection via Lasso have improved overall performance—or if tuning the regularization parameter (α) or exploring additional features is necessary to further enhance the model.

1.3.6 Random Forest - Experiment 3

Random Forest Regression is an ensemble learning method that builds multiple decision trees and aggregates their predictions to improve overall accuracy and robustness. For a regression task, the final prediction is typically the average of the predictions from all individual trees.

Consider a training dataset $\{(x_i, y_i)\}_{i=1}^{N}$. 1. Bootstrap Sampling: For each tree t in the forest, a bootstrap sample S_t is drawn from the training data. This sampling with replacement introduces variability into the trees. 2. Tree Construction: Each decision tree $f_t(x)$ is built on its respective

bootstrap sample. At each node in the tree, a random subset of features is selected, and the best split is chosen based on minimizing the variance of the target values. If a split partitions the data into two regions R_1 and R_2, the split is chosen to minimize:

$$\sum_{x_i \in R_1} (y_i - \bar{y}R_1)^2 + \sum x_i \in R_2 (y_i - \bar{y}_{R_2})^2,$$

where $\bar{y}R_1$ and $\bar{y}R_2$ are the mean values of the target variable in regions R_1 and R_2, respectively. 3. Aggregation of Predictions: For a new input x, each tree t makes a prediction $f_t(x)$. The final prediction $\hat{y}$ is the average of these predictions:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} f_t(x),$$

where T is the total number of trees in the forest.

Benefits of Random Forest Regression - Variance Reduction: Averaging the predictions of multiple trees reduces the variance of the model, leading to improved generalization on unseen data. - Robustness to Outliers: Outlier effects are mitigated since extreme predictions from individual trees are balanced by the ensemble. - Feature Randomness: Random feature selection at each split helps de-correlate trees, which in turn improves the robustness and accuracy of the model.

This approach allows Random Forest Regression to capture complex, non-linear relationships in the data while being less prone to overfitting compared to a single decision tree.

[31]:
```
from sklearn.ensemble import RandomForestRegressor

# Random Forest Regressor with 200 trees and default parameters
model_rf = RandomForestRegressor(n_estimators=200, max_depth=None,␣
 ↪random_state=42)
model_rf.fit(X_train, y_train)

# make predictions on the validation set
y_pred_rf = model_rf.predict(X_val)

# calculate evaluation metrics
rmse_rf = np.sqrt(mean_squared_error(y_val, y_pred_rf))
rmsle_rf = np.sqrt(mean_squared_log_error(y_val, y_pred_rf))
mae_rf = mean_absolute_error(y_val, y_pred_rf)
r2_rf = r2_score(y_val, y_pred_rf)

print(f'Random Forest RMSE:   {rmse_rf:.4f}')
print(f'Random Forest RMSLE:  {rmsle_rf:.4f}')
print(f'Random Forest MAE:    {mae_rf:.4f}')
print(f'Random Forest R²:     {r2_rf:.4f}')
```

```
Random Forest RMSE:   0.1463
Random Forest RMSLE:  0.0115
Random Forest MAE:    0.0985
Random Forest R²:     0.8853
```

Evaluation - Experiment 3 The Random Forest model performs very well in this experiment. An RMSE of 0.1466 means that, on average, the error between the predicted and actual values is quite small. The RMSLE of 0.0115 confirms that the predictions are accurate on a logarithmic scale. Additionally, the MAE of around 0.1 shows that most predictions are very close to the true values. Finally, an R^2 score of 0.8848 indicates that nearly 88% of the variation in the house prices is explained by the model. Overall, these results demonstrate that the Random Forest approach is highly effective for this task.

1.4 Comparison Between Experiments

In the first experiment using a basic Linear Regression model, the performance was moderate, with relatively higher RMSE and MAE and a lower R^2 value. Moving to the second experiment with Lasso Regression significantly improved all metrics: RMSE and MAE dropped notably, and R^2 increased, indicating a better fit. The third experiment used a Random Forest, which performed similarly to Lasso, achieving slightly better MSLE and MAE but a marginally lower R^2. Overall, both Lasso and Random Forest outperformed the basic Linear Regression, with Lasso offering a small edge on R^2 and RMSE, while Random Forest excelled in MSLE and MAE.

1.5 Conclusion

Through this project, I learned that even a simple model like linear regression can provide a solid starting point, but more advanced techniques can significantly improve performance. Preprocessing steps, such as handling missing values and properly transforming features, played a crucial role in model accuracy. Experimenting with Lasso regression showed that feature selection and regularization help to reduce overfitting and simplify the model. Finally, the Random Forest approach demonstrated that non-linear models can capture complex patterns in the data, leading to better predictions overall. These experiments underscore the importance of iterative testing and thoughtful feature engineering in developing effective predictive models.

1.5.1 Impact

This project can have several important impacts. On the positive side, accurately predicting house prices could help home buyers, real estate agents, and policymakers make more informed decisions. It might assist in setting fair market prices, identifying affordable housing opportunities, and understanding market trends. However, there are also potential negative impacts. If the model is used without careful consideration, it could reinforce existing biases in housing data, leading to unfair pricing or discrimination. Moreover, over-reliance on automated models might reduce the human insight needed in complex real estate decisions. Therefore, while the project can offer valuable tools, it also calls for careful ethical use and transparency in its application.

1.6 References

"House Prices: Advanced Regression Techniques." Kaggle.com, www.kaggle.com/c/house-prices-advanced-regression-techniques/data.

Scikit Learn. "3.2.4.3.2. Sklearn.ensemble.RandomForestRegressor — Scikit-Learn 0.20.3 Documentation." Scikit-Learn.org, 2018, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

—. "Sklearn.linear_model.Lasso." Scikit-Learn, scikit-learn.org/stable/modules/generated/sklearn.linear_model.

—. "Sklearn.linear_model.LinearRegression — Scikit-Learn 0.22 Documentation." Scikit-Learn.org, 2019, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.