Predicting Housing Prices

March 24, 2025

1 Role of Family on Student Performance

1.1 Purpose and Questions

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 Logarithmic Error (RMSLE), as this metric is used by the competition to evaluate model performance.

1.1.1 Questions to Answer

- Does family income play a significant role in the performance of students?
- Does tutoring, typically only afforded by the well-off families, increase performance?
- What role does the family of a student have in their performance on exams?

I will be using a dataset from Kaggle with close to 20 features. You can access the dataset with this link https://www.kaggle.com/datasets/lainguyn123/student-performance-factors/data. I will be focusing on the role of family in student performance.

[4]:
```
import pandas as pd

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

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

```
<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
```

[5]:

```
                Id   MSSubClass  LotFrontage        LotArea  OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000  1460.000000
mean    730.500000    56.897260    70.049958   10516.828082     6.099315
std     421.610009    42.300571    24.284752    9981.264932     1.382997
min       1.000000    20.000000    21.000000    1300.000000     1.000000
25%     365.750000    20.000000    59.000000    7553.500000     5.000000
50%     730.500000    50.000000    69.000000    9478.500000     6.000000
75%    1095.250000    70.000000    80.000000   11601.500000     7.000000
max    1460.000000   190.000000   313.000000  215245.000000    10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1  …  \
count  1460.000000  1460.000000   1460.000000  1452.000000  1460.000000  …
mean      5.575342  1971.267808   1984.865753   103.685262   443.639726  …
std       1.112799    30.202904     20.645407   181.066207   456.098091  …
min       1.000000  1872.000000   1950.000000     0.000000     0.000000  …
```

```
25%         5.000000  1954.000000   1967.000000     0.000000     0.000000  …
50%         5.000000  1973.000000   1994.000000     0.000000   383.500000  …
75%         6.000000  2000.000000   2004.000000   166.000000   712.250000  …
max         9.000000  2010.000000   2010.000000  1600.000000  5644.000000  …

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch  ScreenPorch  \
count  1460.000000  1460.000000    1460.000000  1460.000000  1460.000000
mean     94.244521    46.660274      21.954110     3.409589    15.060959
std     125.338794    66.256028      61.119149    29.317331    55.757415
min       0.000000     0.000000       0.000000     0.000000     0.000000
25%       0.000000     0.000000       0.000000     0.000000     0.000000
50%       0.000000    25.000000       0.000000     0.000000     0.000000
75%     168.000000    68.000000       0.000000     0.000000     0.000000
max     857.000000   547.000000     552.000000   508.000000   480.000000

          PoolArea       MiscVal       MoSold       YrSold      SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000    1460.000000
mean      2.758904     43.489041     6.321918  2007.815753  180921.195890
std      40.177307    496.123024     2.703626     1.328095   79442.502883
min       0.000000      0.000000     1.000000  2006.000000   34900.000000
25%       0.000000      0.000000     5.000000  2007.000000  129975.000000
50%       0.000000      0.000000     6.000000  2008.000000  163000.000000
75%       0.000000      0.000000     8.000000  2009.000000  214000.000000
max     738.000000  15500.000000    12.000000  2010.000000  755000.000000

[8 rows x 38 columns]
```

1.2 Preprocessing

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

[7]:
```python
# 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)
```

[9]:
```
X = train_encoded.drop('SalePrice', axis=1)
y = train_encoded['SalePrice']


y_log = np.log1p(y)  # for RMSE
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 5
      1 X = train_encoded.drop('SalePrice', axis=1)
      2 y = train_encoded['SalePrice']
----> 5 y_log = np.log1p(y)

NameError: name 'np' is not defined
```

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.

[ ]:
```
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

[ ]:
```
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

[ ]:
```
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

[ ]:
```
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

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_log_error
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y_log, test_size=0.2,␣
↪random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_val)
rmsle = np.sqrt(mean_squared_log_error(y_val, y_pred))
print(f'RMSLE: {rmsle:.4f}')
```

[ ]:

```

```

1.4 Summary

1.4.1 Bias and Limitations

1.5 References

[ ]:

```

```