House Price Prediction Project Report

(M.Junaid 109(4B))

Table of Contents

Introduction

This report explains a machine learning project that predicts house prices. The code was written to preprocess data and create predictions that can be submitted to a Kaggle competition.

Data Loading and Exploration

First, we load the house price dataset and look at its basic information. This helps us understand what data we're working with, including the types of columns and how many records we have.

```python
import pandas as pd
df=pd.read_csv("train.csv")
df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	1	60	RL	65.0	8450	Pave	NaN	Reg
1	2	20	RL	80.0	9600	Pave	NaN	Reg
2	3	60	RL	68.0	11250	Pave	NaN	IR1
3	4	70	RL	60.0	9550	Pave	NaN	IR1

Handling Missing Values

We check which columns have missing values. We need to fill these missing values because machine learning models can't work with incomplete data.

```python
null_columns = df.isnull().sum()
print(f'Total number of columns with null values: {null_columns[null_columns > 0].count()}')
print(null_columns[null_columns > 0])
```

```
otal number of columns with null values: 19
otFrontage      259
lley           1369
asVnrType       872
asVnrArea         8
smtQual          37
smtCond          37
smtExposure      38
```

Filling Missing Values in Text Columns

For text columns, we fill missing values with the most common value in each column. This is a simple approach that works well for categorical data.

```python
# Object columns filled
for column in null_columns[null_columns > 0].index:
    if df[column].dtype == 'object':
        mode_value = df[column].mode()[0]
        df[column].fillna(mode_value, inplace=True)
```

```
C:\Users\M.Junaid\AppData\Local\Temp\ipykernel_6916\1486561903.py:5: FutureWarning: A value is try
The behavior will change in pandas 3.0. This inplace method will never work because the intermedia
```

Filling Missing Values in Number Columns

For number columns, we use a smarter approach. We first check for outliers (unusual values) using the IQR method. Then we fill missing values with the average of normal values. This helps prevent outliers from affecting our estimates.

```
for column in null_columns[null_columns > 0].index:
    if df[column].dtype in ['int64', 'float64']:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        if df[(df[column] < lower_bound) | (df[column] > upper_bound)].empty:
            mean_value = df[column].mean()
            df[column].fillna(mean_value, inplace=True)
        else:
            values = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)][column]
            df[column].fillna(values.mean(), inplace=True)
            df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = values.mean()
```

Converting Text to Numbers

Machine learning models need numbers, not text. We convert all text columns to numbers using Label Encoding, which assigns a unique number to each text value.

```
from sklearn.preprocessing import LabelEncoder
object_columns = df.select_dtypes(include=['object']).columns.tolist()
label_encoder = LabelEncoder()
for column in object_columns:
    df[column] = label_encoder.fit_transform(df[column].astype(str))

df.dtypes
```

```
Id                 int64
MSSubClass         int64
MSZoning           int64
LotFrontage      float64
LotArea            int64
                  ...
MoSold             int64
YrSold             int64
SaleType           int64
SaleCondition      int64
SalePrice          int64
```

Preparing Data for Training

- Separate the features (X) from what we want to predict (Y)
- Remove the ID column since it's not useful for prediction
- Scale the data so all features have similar ranges, which helps the model learn better

```python
from sklearn.preprocessing import StandardScaler
X=df.drop(['Id', 'SalePrice'], axis=1)
Y=df['SalePrice']
scaler = StandardScaler()
train_data=scaler.fit_transform(X)
```

Training the Model

We use a Random Forest model because it works well for many prediction problems. The model learns patterns from our training data to predict house prices. We save the trained model to a file so we can use it later without retraining.

```python
from sklearn.ensemble import RandomForestRegressor
import joblib

model = RandomForestRegressor()

model.fit(train_data, Y)

joblib.dump(model, 'Mymodel.pkl')
```

```
['Mymodel.pkl']
```

+ Code

Testing Process

Loading and Preprocessing Test Data

We load the test data and apply the same preprocessing steps as we did with the training data.

```python
import pandas as pd
df=pd.read_csv("test.csv")
df.head()
```

Python

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	...	120	0	NaN
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	...	144	0	NaN

5 rows × 80 columns

Handling Missing Values in Test Data

The test data also has missing values that need to be filled.

```python
null_columns = df.isnull().sum()
print(f'Total number of columns with null values: {null_columns[null_columns > 0].count()}')
print(null_columns[null_columns > 0])
```

```
Total number of columns with null values: 33
MSZoning           4
LotFrontage      227
Alley           1352
Utilities          2
Exterior1st        1
Exterior2nd        1
MasVnrType       894
MasVnrArea        15
BsmtQual          44
BsmtCond          45
BsmtExposure      44
BsmtFinType1      42
BsmtFinSF1         1
BsmtFinType2      42
BsmtFinSF2         1
BsmtUnfSF          1
TotalBsmtSF        1
BsmtFullBath       2
BsmtHalfBath       2
```

Filling Missing Values and Encoding

We apply the same methods to fill missing values and encode categorical variables as we did with the training data.

```python
# Object columns filled
for column in null_columns[null_columns > 0].index:
    if df[column].dtype == 'object':
        mode_value = df[column].mode()[0]
        df[column].fillna(mode_value, inplace=True)
```

```python
for column in null_columns[null_columns > 0].index:
    if df[column].dtype in ['int64', 'float64']:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        if df[(df[column] < lower_bound) | (df[column] > upper_bound)].empty:
            mean_value = df[column].mean()
            df[column].fillna(mean_value, inplace=True)
        else:
            values = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)][column]
            df[column].fillna(values.mean(), inplace=True)
            df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = values.mean()
```

```python
from sklearn.preprocessing import LabelEncoder
object_columns = df.select_dtypes(include=['object']).columns.tolist()
label_encoder = LabelEncoder()
for column in object_columns:
    df[column] = label_encoder.fit_transform(df[column].astype(str))
```

Making Predictions on Test Data

- Loads our saved model
- Uses it to predict prices for the test data
- Creates a submission file with house IDs and predicted prices
- Saves this file for uploading to Kaggle

```python
from sklearn.preprocessing import StandardScaler
X=df.drop('Id', axis=1)
scaler = StandardScaler()
Test_data=scaler.fit_transform(X)
```

```python
import joblib

model = joblib.load('Mymodel.pkl')
```

```python
predictions = model.predict(Test_data)
submission_df = pd.DataFrame({'Id': df['Id'], 'SalePrice': predictions})


submission_df.to_csv('submission_file.csv', index=False)
```

Model Accuracy

The model's performance was evaluated using metrics like Root Mean Squared Error (RMSE) and R-squared.