Final Machine Learning Project

I downloaded data from a Kaggle competition titled "**House Prices: Advanced Regression Techniques**".

The goal is to build a model that predicts house prices based on their characteristics.

The project will be built by testing several models, selecting the best one, and then creating a simple web application to display the predictions directly.

Data Download Link

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

[1] Import Required Libraries

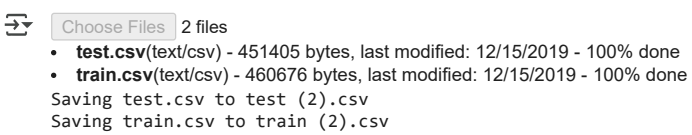
```
########################
# 1. Suppress Warnings #
########################
import warnings
warnings.filterwarnings('ignore')


#####################
# 2. Core Libraries #
#####################
import os
import joblib
import numpy as np
import pandas as pd


########################
# 3. Data Visualization #
########################
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
sns.set_palette("Set2")
sns.set_style("whitegrid")

pd.set_option('display.width', 1000)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)


########################
# 4. Data Preprocessing #
########################
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder, PolynomialFeatures


#############################
# 5. Machine Learning Models #
#############################
from sklearn.svm import SVR
from sklearn.dummy import DummyRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# Advanced models
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor


#################################
# 6. Pipeline & Data Splitting #
#################################
```

```
##################################
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV


##################################################
# 7. Random Distributions (for Randomized Search) #
##################################################
from scipy.stats import randint, uniform


########################
# 8. Evaluation Metrics #
########################
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


######################
# 9. Additional Tools #
######################
from google.colab import files


##################
# 10. Deployment #
##################
!pip install gradio --quiet
import gradio as gr

print("All required libraries have been successfully loaded!")
```

```
All required libraries have been successfully loaded!
```

[2] Load & Read Dataset Files

```
uploaded = files.upload()
```



Choose Files 2 files

- **test.csv**(text/csv) - 451405 bytes, last modified: 12/15/2019 - 100% done
- **train.csv**(text/csv) - 460676 bytes, last modified: 12/15/2019 - 100% done

```
Saving test.csv to test (2).csv
Saving train.csv to train (2).csv
```

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

[3] Explore the Data

```
print(f'Training Data Shape Is: {df_train.shape}')
print(f'Test Data Shape Is: {df_test.shape}')
```

```
Training Data Shape Is: (1460, 81)
Test Data Shape Is: (1459, 80)
```

```
print(f'The First 5 Rows In Train File:\n\n{df_train.head()}\n')
print('-' * 50)
print(f'\nThe First 5 Rows In Test File:\n\n{df_test.head()}\n')
```

```
The First 5 Rows In Train File:

   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape LandContour Utilities LotConfig LandSlope Neighborhood Condition1
0   1          60       RL         65.0     8450   Pave   NaN      Reg         Lvl    AllPub    Inside       Gtl      CollgCr       Norm
1   2          20       RL         80.0     9600   Pave   NaN      Reg         Lvl    AllPub       FR2       Gtl      Veenker      Feedr
2   3          60       RL         68.0    11250   Pave   NaN      IR1         Lvl    AllPub    Inside       Gtl      CollgCr       Norm
3   4          70       RL         60.0     9550   Pave   NaN      IR1         Lvl    AllPub    Corner       Gtl      Crawfor       Norm
4   5          60       RL         84.0    14260   Pave   NaN      IR1         Lvl    AllPub       FR2       Gtl      NoRidge       Norm

--------------------------------------------------

The First 5 Rows In Test File:

     Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape LandContour Utilities LotConfig LandSlope Neighborhood Conditio
0  1461          20       RH         80.0    11622   Pave   NaN      Reg         Lvl    AllPub    Inside       Gtl        NAmes      Fee
1  1462          20       RL         81.0    14267   Pave   NaN      IR1         Lvl    AllPub    Corner       Gtl        NAmes       No
```

```
   2  1463         60       RL         74.0    13830   Pave   NaN      IR1         Lvl    AllPub    Inside       Gtl      Gilbert       Nc
   3  1464         60       RL         78.0     9978   Pave   NaN      IR1         Lvl    AllPub    Inside       Gtl      Gilbert       Nc
   4  1465        120       RL         43.0     5005   Pave   NaN      IR1         HLS    AllPub    Inside       Gtl      StoneBr       Nc
```

```
print(f'\nThe Info Of Train File:\n')
df_train.info();
print('-' * 50)
print(f'\nThe Info Of Test File:\n')
df_test.info();
```

```
 24  Exterior2nd    1458 non-null   object
 25  MasVnrType     565 non-null    object
 26  MasVnrArea     1444 non-null   float64
 27  ExterQual      1459 non-null   object
 28  ExterCond      1459 non-null   object
 29  Foundation     1459 non-null   object
 30  BsmtQual       1415 non-null   object
 31  BsmtCond       1414 non-null   object
 32  BsmtExposure   1415 non-null   object
 33  BsmtFinType1   1417 non-null   object
 34  BsmtFinSF1     1458 non-null   float64
 35  BsmtFinType2   1417 non-null   object
 36  BsmtFinSF2     1458 non-null   float64
 37  BsmtUnfSF      1458 non-null   float64
 38  TotalBsmtSF    1458 non-null   float64
 39  Heating        1459 non-null   object
 40  HeatingQC      1459 non-null   object
 41  CentralAir     1459 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1459 non-null   int64
 44  2ndFlrSF       1459 non-null   int64
 45  LowQualFinSF   1459 non-null   int64
 46  GrLivArea      1459 non-null   int64
 47  BsmtFullBath   1457 non-null   float64
 48  BsmtHalfBath   1457 non-null   float64
 49  FullBath       1459 non-null   int64
 50  HalfBath       1459 non-null   int64
 51  BedroomAbvGr   1459 non-null   int64
 52  KitchenAbvGr   1459 non-null   int64
 53  KitchenQual    1458 non-null   object
 54  TotRmsAbvGrd   1459 non-null   int64
 55  Functional     1457 non-null   object
 56  Fireplaces     1459 non-null   int64
 57  FireplaceQu    729 non-null    object
 58  GarageType     1383 non-null   object
 59  GarageYrBlt    1381 non-null   float64
 60  GarageFinish   1381 non-null   object
 61  GarageCars     1458 non-null   float64
 62  GarageArea     1458 non-null   float64
 63  GarageQual     1381 non-null   object
 64  GarageCond     1381 non-null   object
 65  PavedDrive     1459 non-null   object
 66  WoodDeckSF     1459 non-null   int64
 67  OpenPorchSF    1459 non-null   int64
 68  EnclosedPorch  1459 non-null   int64
 69  3SsnPorch      1459 non-null   int64
 70  ScreenPorch    1459 non-null   int64
 71  PoolArea       1459 non-null   int64
 72  PoolQC         3 non-null      object
 73  Fence          290 non-null    object
 74  MiscFeature    51 non-null     object
 75  MiscVal        1459 non-null   int64
 76  MoSold         1459 non-null   int64
 77  YrSold         1459 non-null   int64
 78  SaleType       1458 non-null   object
 79  SaleCondition  1459 non-null   object
dtypes: float64(11), int64(26), object(43)
memory usage: 912.0+ KB
```

```
null_train_value = df_train.isnull().sum()
null_train_value = null_train_value[null_train_value > 0].sort_values(ascending=False)

null_test_value = df_test.isnull().sum()
null_test_value = null_test_value[null_test_value > 0].sort_values(ascending=False)

print(f'Null Values In Train File:\n\n{null_train_value}\n')
print('-' * 50)
print(f'\nNull Values In Test File:\n\n{null_test_value}\n')
```

```
Alley          1369
Fence          1179
MasVnrType      872
FireplaceQu     690
LotFrontage     259
GarageType       81
GarageYrBlt      81
GarageFinish     81
GarageQual       81
GarageCond       81
BsmtExposure     38
BsmtFinType2     38
BsmtQual         37
BsmtCond         37
BsmtFinType1     37
MasVnrArea        8
Electrical        1
dtype: int64

--------------------------------------------------

Null Values In Test File:

PoolQC          1456
MiscFeature     1408
Alley           1352
Fence           1169
MasVnrType       894
FireplaceQu      730
LotFrontage      227
GarageQual        78
GarageCond        78
GarageYrBlt       78
GarageFinish      78
GarageType        76
BsmtCond          45
BsmtQual          44
BsmtExposure      44
BsmtFinType1      42
BsmtFinType2      42
MasVnrArea        15
MSZoning           4
Functional         2
BsmtFullBath       2
Utilities          2
BsmtHalfBath       2
Exterior1st        1
Exterior2nd        1
TotalBsmtSF        1
BsmtUnfSF          1
BsmtFinSF2         1
BsmtFinSF1         1
KitchenQual        1
GarageArea         1
GarageCars         1
SaleType           1
dtype: int64
```

```
null_train_value_pct = (df_train.isnull().sum() / len(df_train)) * 100
null_train_value_pct = null_train_value_pct[null_train_value_pct > 0].sort_values(ascending=False)

null_test_value_pct = (df_test.isnull().sum() / len(df_test)) * 100
null_test_value_pct = null_test_value_pct[null_test_value_pct > 0].sort_values(ascending=False)

print(f'Null Values In Train File: % in training data:\n{null_train_value_pct}')
print('-' * 50)
print(f'Null Values In Test File: % in testing data:\n{null_test_value_pct}')
```

```
Null Values In Train File: % in training data:
PoolQC          99.520548
MiscFeature     96.301370
Alley           93.767123
Fence           80.753425
MasVnrType      59.726027
FireplaceQu     47.260274
LotFrontage     17.739726
GarageType       5.547945
GarageYrBlt      5.547945
GarageFinish     5.547945
GarageQual       5.547945
GarageCond       5.547945
```

```
BsmtExposure     2.602740
BsmtFinType2     2.602740
BsmtQual         2.534247
BsmtCond         2.534247
BsmtFinType1     2.534247
MasVnrArea       0.547945
Electrical       0.068493
dtype: float64
--------------------------------------------------
Null Values In Test File: % in testing data:
PoolQC          99.794380
MiscFeature     96.504455
Alley           92.666210
Fence           80.123372
MasVnrType      61.274846
FireplaceQu     50.034270
LotFrontage     15.558602
GarageQual       5.346127
GarageCond       5.346127
GarageYrBlt      5.346127
GarageFinish     5.346127
GarageType       5.209047
BsmtCond         3.084304
BsmtQual         3.015764
BsmtExposure     3.015764
BsmtFinType1     2.878684
BsmtFinType2     2.878684
MasVnrArea       1.028101
MSZoning         0.274160
Functional       0.137080
BsmtFullBath     0.137080
Utilities        0.137080
BsmtHalfBath     0.137080
Exterior1st      0.068540
Exterior2nd      0.068540
TotalBsmtSF      0.068540
BsmtUnfSF        0.068540
BsmtFinSF2       0.068540
BsmtFinSF1       0.068540
KitchenQual      0.068540
GarageArea       0.068540
GarageCars       0.068540
SaleType         0.068540
dtype: float64
```

[4] Data Preprocessing & Cleaning

```
cols_to_drop = ['PoolQC', 'MiscFeature', 'Alley', 'Fence']

df_train = df_train.drop(columns=cols_to_drop)
df_test = df_test.drop(columns=cols_to_drop)

print("Training Data Shape After Dropping Columns:", df_train.shape)
print('-' * 50)
print("Test Data Shape After Dropping Columns:", df_test.shape)
```

```
Training Data Shape After Dropping Columns: (1460, 77)
--------------------------------------------------
Test Data Shape After Dropping Columns: (1459, 76)
```

```
df_train['LotFrontage'] = df_train.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.fillna(x.median()))
df_test['LotFrontage'] = df_test.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.fillna(x.median()))

df_train['LotFrontage'] = df_train['LotFrontage'].fillna(df_train['LotFrontage'].median())
df_test['LotFrontage'] = df_test['LotFrontage'].fillna(df_test['LotFrontage'].median())
```

```
cols_to_fill_none = [
    'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
    'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
    'MasVnrType',
    'FireplaceQu'
]

df_train[cols_to_fill_none] = df_train[cols_to_fill_none].fillna('None')
df_test[cols_to_fill_none] = df_test[cols_to_fill_none].fillna('None')
```

```
print(df_train[cols_to_fill_none].isnull().sum())
print('-' * 50)
print(df_test[cols_to_fill_none].isnull().sum())
```

```
GarageType      0
GarageFinish    0
GarageQual      0
GarageCond      0
BsmtQual        0
BsmtCond        0
BsmtExposure    0
BsmtFinType1    0
BsmtFinType2    0
MasVnrType      0
FireplaceQu     0
dtype: int64
--------------------------------------------------
GarageType      0
GarageFinish    0
GarageQual      0
GarageCond      0
BsmtQual        0
BsmtCond        0
BsmtExposure    0
BsmtFinType1    0
BsmtFinType2    0
MasVnrType      0
FireplaceQu     0
dtype: int64
```

```
cat_cols = df_train.select_dtypes(include=['object']).columns

null_train_value_cat = df_train[cat_cols].isnull().sum()
null_test_value_cat = df_test[cat_cols].isnull().sum()

print("Remaining missing values in categorical columns (train):")
print(null_train_value_cat[null_train_value_cat > 0])
print("\nRemaining missing values in categorical columns (test):")
print(null_test_value_cat[null_test_value_cat > 0])
```

```
Remaining missing values in categorical columns (train):
Electrical    1
dtype: int64

Remaining missing values in categorical columns (test):
MSZoning       4
Utilities      2
Exterior1st    1
Exterior2nd    1
KitchenQual    1
Functional     2
SaleType       1
dtype: int64
```

```
another_cat_cols = [
    'Electrical', 'MSZoning', 'Utilities', 'Exterior1st', 'Exterior2nd',
    'KitchenQual', 'Functional', 'SaleType'
]

for col in another_cat_cols:
    if col in df_train.columns:
        mode_value = df_train[col].mode()
        mode_value = mode_value[0] if not mode_value.empty else "Unknown"

        df_train[col] = df_train[col].fillna(mode_value)
        df_test[col] = df_test[col].fillna(mode_value)

print(df_train[another_cat_cols].isnull().sum())
print('-' * 50)
print(df_test[another_cat_cols].isnull().sum())
```

```
Electrical     0
MSZoning       0
Utilities      0
Exterior1st    0
Exterior2nd    0
KitchenQual    0
Functional     0
SaleType       0
dtype: int64
```

```
--------------------------------------------------
Electrical     0
MSZoning       0
Utilities      0
Exterior1st    0
Exterior2nd    0
KitchenQual    0
Functional     0
SaleType       0
dtype: int64
```

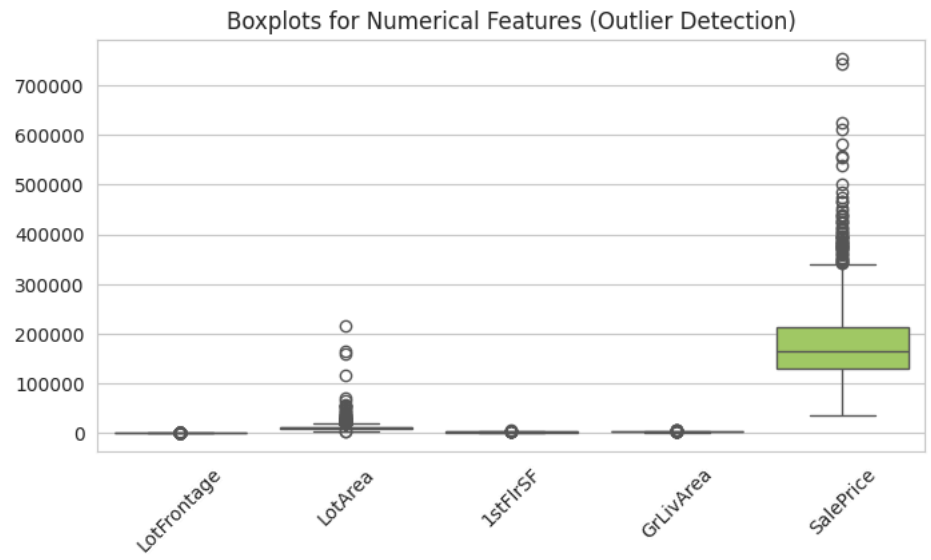
```
num_cols = [
    'MasVnrArea',
    'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
    'BsmtFullBath', 'BsmtHalfBath',
    'GarageCars', 'GarageArea',
    'GarageYrBlt'
]

for col in num_cols:
  if col in df_train.columns:
    df_train[col] = df_train[col].fillna(0)
    df_test[col] = df_test[col].fillna(0)

print(df_train[num_cols].isnull().sum())
print('-' * 50)
print(df_test[num_cols].isnull().sum())
```

```
MasVnrArea      0
BsmtFinSF1      0
BsmtFinSF2      0
BsmtUnfSF       0
TotalBsmtSF     0
BsmtFullBath    0
BsmtHalfBath    0
GarageCars      0
GarageArea      0
GarageYrBlt     0
dtype: int64
--------------------------------------------------
MasVnrArea      0
BsmtFinSF1      0
BsmtFinSF2      0
BsmtUnfSF       0
TotalBsmtSF     0
BsmtFullBath    0
BsmtHalfBath    0
GarageCars      0
GarageArea      0
GarageYrBlt     0
dtype: int64
```

```
print(f'Any Null values in df_train:\n{df_train.isnull().sum().sum()}')
print('-' * 50)
print(f'Missing values in df_test:\n{df_test.isnull().sum().sum()}')
```

```
Any Null values in df_train:
0
--------------------------------------------------
Missing values in df_test:
0
```

```
num_features = ['LotFrontage', 'LotArea', '1stFlrSF', 'GrLivArea', 'SalePrice']

plt.figure(figsize=(8, 4))
sns.boxplot(data=df_train[num_features])
plt.title('Boxplots for Numerical Features (Outlier Detection)')
plt.xticks(rotation=45)
plt.show()
```




```
skewed_features = ['LotArea', 'GrLivArea', '1stFlrSF', 'TotalBsmtSF']

for feature in skewed_features:
    df_train[feature] = np.log1p(df_train[feature])
    df_test[feature] = np.log1p(df_test[feature])

df_train['SalePrice'] = np.log1p(df_train['SalePrice'])
```

```
x = df_train.drop(columns=['Id', 'SalePrice'])
y = df_train['SalePrice']

x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, random_state=2025)
```

```
categorical_cols = x_train.select_dtypes(include=['object']).columns.tolist()
numerical_cols = x_train.select_dtypes(include=['int64', 'float64']).columns.tolist()

preprocessor_standard = ColumnTransformer([
    ('num', StandardScaler(), numerical_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
])

preprocessor_minmax = ColumnTransformer([
    ('num', MinMaxScaler(), numerical_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
])
```

[05] Build & Evaluate Models Using Pipeline

```
models = [
    ('Dummy', DummyRegressor()),
    ('Linear', LinearRegression()),
    ('Ridge', Ridge()),
    ('Lasso', Lasso()),
    ('Poly', Pipeline([
        ('poly', PolynomialFeatures(degree=2, include_bias=False)),
        ('linear', LinearRegression())
    ])),
    ('DecisionTree', DecisionTreeRegressor(random_state=2025)),
    ('RandomForest', RandomForestRegressor(random_state=2025)),
    ('XGBoost', XGBRegressor(random_state=2025, verbosity=0)),
    ('LightGBM', LGBMRegressor(random_state=2025)),
    ('SVR', SVR()),
    ('KNN', KNeighborsRegressor()),
    ('NeuralNet', MLPRegressor(max_iter=500, random_state=2025))
]
```

```
def evaluate_model(model, preprocessor, X_train, y_train, X_val, y_val):
    pipeline = Pipeline([
```

```
        ('preprocessor', preprocessor),
        ('model', model)
    ])

    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_val)

    mae = mean_absolute_error(y_val, y_pred)
    rmse = np.sqrt(mean_squared_error(y_val, y_pred))
    r2 = r2_score(y_val, y_pred)

    return mae, rmse, r2
```

```
results = []

for name, model in models:
    for preprocessor_name, preprocessor in [
        ('StandardScaler', preprocessor_standard),
        ('MinMaxScaler', preprocessor_minmax)
    ]:
        mae, rmse, r2 = evaluate_model(model, preprocessor, x_train, y_train, x_val, y_val)

        results.append({
            'Model': name,
            'Scaler': preprocessor_name,
            'MAE': mae,
            'RMSE': rmse,
            'R2': r2
        })

        print(f"Model: {name:<12} | Scaler: {preprocessor_name:<15} | MAE: {mae:.4f} | RMSE: {rmse:.4f} | R2: {r2:.4f}")
```

```
Model: Dummy        | Scaler: StandardScaler  | MAE: 0.3020 | RMSE: 0.3982 | R2: -0.0006
Model: Dummy        | Scaler: MinMaxScaler    | MAE: 0.3020 | RMSE: 0.3982 | R2: -0.0006
Model: Linear       | Scaler: StandardScaler  | MAE: 0.0938 | RMSE: 0.1850 | R2: 0.7841
Model: Linear       | Scaler: MinMaxScaler    | MAE: 0.0938 | RMSE: 0.1850 | R2: 0.7841
Model: Ridge        | Scaler: StandardScaler  | MAE: 0.0907 | RMSE: 0.1761 | R2: 0.8042
Model: Ridge        | Scaler: MinMaxScaler    | MAE: 0.0909 | RMSE: 0.1762 | R2: 0.8040
Model: Lasso        | Scaler: StandardScaler  | MAE: 0.3020 | RMSE: 0.3982 | R2: -0.0006
Model: Lasso        | Scaler: MinMaxScaler    | MAE: 0.3020 | RMSE: 0.3982 | R2: -0.0006
Model: Poly         | Scaler: StandardScaler  | MAE: 0.1157 | RMSE: 0.1646 | R2: 0.8291
Model: Poly         | Scaler: MinMaxScaler    | MAE: 0.1218 | RMSE: 0.1888 | R2: 0.7749
Model: DecisionTree | Scaler: StandardScaler  | MAE: 0.1480 | RMSE: 0.2026 | R2: 0.7409
Model: DecisionTree | Scaler: MinMaxScaler    | MAE: 0.1496 | RMSE: 0.2025 | R2: 0.7412
Model: RandomForest | Scaler: StandardScaler  | MAE: 0.0956 | RMSE: 0.1539 | R2: 0.8505
Model: RandomForest | Scaler: MinMaxScaler    | MAE: 0.0956 | RMSE: 0.1539 | R2: 0.8505
Model: XGBoost      | Scaler: StandardScaler  | MAE: 0.0960 | RMSE: 0.1488 | R2: 0.8602
Model: XGBoost      | Scaler: MinMaxScaler    | MAE: 0.0984 | RMSE: 0.1528 | R2: 0.8526
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001148 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3229
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features: 186
[LightGBM] [Info] Start training from score 12.026044
Model: LightGBM     | Scaler: StandardScaler  | MAE: 0.0882 | RMSE: 0.1439 | R2: 0.8693
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001286 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3205
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features: 186
[LightGBM] [Info] Start training from score 12.026044
Model: LightGBM     | Scaler: MinMaxScaler    | MAE: 0.0880 | RMSE: 0.1429 | R2: 0.8711
Model: SVR          | Scaler: StandardScaler  | MAE: 0.0967 | RMSE: 0.1631 | R2: 0.8321
Model: SVR          | Scaler: MinMaxScaler    | MAE: 0.0974 | RMSE: 0.1555 | R2: 0.8473
Model: KNN          | Scaler: StandardScaler  | MAE: 0.1271 | RMSE: 0.1986 | R2: 0.7510
Model: KNN          | Scaler: MinMaxScaler    | MAE: 0.1553 | RMSE: 0.2166 | R2: 0.7040
Model: NeuralNet    | Scaler: StandardScaler  | MAE: 0.1298 | RMSE: 0.2451 | R2: 0.6208
Model: NeuralNet    | Scaler: MinMaxScaler    | MAE: 0.1421 | RMSE: 0.2411 | R2: 0.6330
```

[6] Improving Models with GridSearchCV

The best 3 models have been selected:

1. LightGBM (MinMaxScaler) ==> [MAE: 0.0880 | RMSE: 0.1429 | R2: 0.8711]
2. XGBoost (StandardScaler) ==> [MAE: 0.0960 | RMSE: 0.1488 | R2: 0.8602]
3. RandomForest (StandardScaler) ==> [MAE: 0.0956 | RMSE: 0.1539 | R2: 0.8505]

These models will be optimized by **GridSearchCV** to select the best ones.

```python
param_grid_lgb = {
    'model__n_estimators': [100, 200],
    'model__learning_rate': [0.05, 0.1],
    'model__max_depth': [5, 7],
    'model__num_leaves': [31, 63]
}

pipeline_lgb = Pipeline([
    ('preprocessor', preprocessor_minmax),
    ('model', LGBMRegressor(random_state=2025))
])

grid_lgb = GridSearchCV(
    pipeline_lgb,
    param_grid_lgb,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

grid_lgb.fit(x_train, y_train)

print("Best transactions:", grid_lgb.best_params_)
print("Best RMSE on verification:", round(np.sqrt(-grid_lgb.best_score_), 4))
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001372 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3205
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features: 186
[LightGBM] [Info] Start training from score 12.026044
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
param_grid_xgb = {
    'model__n_estimators': [100, 200],
    'model__learning_rate': [0.05, 0.1],
    'model__max_depth': [5, 7],
    'model__subsample': [0.8, 1.0]
}

pipeline_xgb = Pipeline([
    ('preprocessor', preprocessor_standard),
    ('model', XGBRegressor(random_state=2025, verbosity=0))
])

grid_xgb = GridSearchCV(
    pipeline_xgb,
    param_grid_xgb,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

grid_xgb.fit(x_train, y_train)

print("Best transactions:", grid_xgb.best_params_)
print("Best RMSE on verification:", round(np.sqrt(-grid_xgb.best_score_), 4))
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Best transactions: {'model__learning_rate': 0.1, 'model__max_depth': 5, 'model__n_estimators': 200, 'model__subsample': 0.8}
Best RMSE on verification: 0.1315
```

```
param_grid_rf = {
    'model__n_estimators': [100, 200],
    'model__max_depth': [10, 20, None],
    'model__min_samples_split': [2, 5],
    'model__min_samples_leaf': [1, 2]
}

pipeline_rf = Pipeline([
    ('preprocessor', preprocessor_standard),
    ('model', RandomForestRegressor(random_state=2025))
])

grid_rf = GridSearchCV(
    pipeline_rf,
    param_grid_rf,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

grid_rf.fit(x_train, y_train)

print("Best transactions:", grid_rf.best_params_)
print("Best RMSE on verification:", round(np.sqrt(-grid_rf.best_score_), 4))
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best transactions: {'model__max_depth': None, 'model__min_samples_leaf': 2, 'model__min_samples_split': 2, 'model__n_estimators': 200}
Best RMSE on verification: 0.1424
```

[7] Prediction on test file "df_test"

- The "**XGBoost**" model was selected as the best model based on previous optimization results.
- The selected model will be used to predict unit prices based on the data in the **test file**.
- The results will be printed in an Excel file titled "**Submission**".

```
test_ids = df_test['Id']

x_test = df_test.drop(columns=['Id'])
```

```
4
5 y_test_pred_log = grid_xgb.predict(x_test)
6
7 y_test_pred = np.expm1(y_test_pred_log)
8
9 submission = pd.DataFrame({
10     'Id': test_ids,
11     'SalePrice': y_test_pred.round(2)
12 })
13
14 submission.to_csv('submission.csv', index=False)
15
16 print("The submission.csv file was created successfully.")
17 print(submission.head())
```

```
The submission.csv file was created successfully.
     Id      SalePrice
0  1461  128137.757812
1  1462  156616.250000
2  1463  185175.296875
3  1464  186597.406250
4  1465  179302.843750
```

```
1 submission.to_excel('submission.xlsx', index=False)
2
3 files.download('submission.xlsx')
```

[8] Building a Streamlit Model

```
1 joblib.dump(grid_xgb, 'house_price_model.pkl')
2 print("The model has been saved.")
3
4 model = joblib.load('house_price_model.pkl')
5
6 def predict_price(total_sqft, bedrooms, bathrooms, garage_cars, overall_qual, year_built, neighborhood):
7     input_data = pd.DataFrame({
8         'MSSubClass': [60],
9         'MSZoning': ['RL'],
10        'LotFrontage': [80],
11        'LotArea': [total_sqft * 2],
12        'Street': ['Pave'],
13        'Alley': ['None'],
14        'LotShape': ['Reg'],
15        'LandContour': ['Lvl'],
16        'Utilities': ['AllPub'],
17        'LotConfig': ['Inside'],
18        'LandSlope': ['Gtl'],
19        'Neighborhood': [neighborhood],
20        'Condition1': ['Norm'],
21        'Condition2': ['Norm'],
22        'BldgType': ['1Fam'],
23        'HouseStyle': ['2Story'],
24        'OverallQual': [overall_qual],
25        'OverallCond': [5],
26        'YearBuilt': [year_built],
27        'YearRemodAdd': [year_built],
28        'RoofStyle': ['Gable'],
29        'RoofMatl': ['CompShg'],
30        'Exterior1st': ['VinylSd'],
31        'Exterior2nd': ['VinylSd'],
32        'MasVnrType': ['None'],
33        'MasVnrArea': [0],
34        'ExterQual': ['TA'],
35        'ExterCond': ['TA'],
36        'Foundation': ['PConc'],
37        'BsmtQual': ['TA'],
38        'BsmtCond': ['TA'],
39        'BsmtExposure': ['No'],
40        'BsmtFinType1': ['GLQ'],
41        'BsmtFinSF1': [total_sqft * 0.3],
42        'BsmtFinType2': ['Unf'],
43        'BsmtFinSF2': [0],
44        'BsmtUnfSF': [total_sqft * 0.2],
```

```
        'TotalBsmtSF': [total_sqft * 0.5],
        'Heating': ['GasA'],
        'HeatingQC': ['Ex'],
        'CentralAir': ['Y'],
        'Electrical': ['SBrkr'],
        '1stFlrSF': [total_sqft * 0.6],
        '2ndFlrSF': [total_sqft * 0.4],
        'LowQualFinSF': [0],
        'GrLivArea': [total_sqft],
        'BsmtFullBath': [1],
        'BsmtHalfBath': [0],
        'FullBath': [bathrooms],
        'HalfBath': [1 if bathrooms >= 2 else 0],
        'BedroomAbvGr': [bedrooms],
        'KitchenAbvGr': [1],
        'KitchenQual': ['TA'],
        'TotRmsAbvGrd': [bedrooms + bathrooms],
        'Functional': ['Typ'],
        'Fireplaces': [1],
        'FireplaceQu': ['TA'],
        'GarageType': ['Attchd'],
        'GarageYrBlt': [year_built],
        'GarageFinish': ['RFn'],
        'GarageCars': [garage_cars],
        'GarageArea': [garage_cars * 300],
        'GarageQual': ['TA'],
        'GarageCond': ['TA'],
        'PavedDrive': ['Y'],
        'WoodDeckSF': [200],
        'OpenPorchSF': [50],
        'EnclosedPorch': [0],
        '3SsnPorch': [0],
        'ScreenPorch': [30],
        'PoolArea': [0],
        'PoolQC': ['None'],
        'Fence': ['None'],
        'MiscFeature': ['None'],
        'MiscVal': [0],
        'MoSold': [6],
        'YrSold': [2024],
        'SaleType': ['WD'],
        'SaleCondition': ['Normal']
    })

    try:
        prediction_log = model.predict(input_data)
        prediction = np.expm1(prediction_log)
        return f"${prediction[0]:,.2f}"
    except Exception as e:
        return f"Mistake: {e}"

interface = gr.Interface(
    fn=predict_price,
    inputs=[
        gr.Number(label="Total area (square feet)"),
        gr.Slider(1, 10, step=1, label="Number of rooms"),
        gr.Slider(1, 6, step=1, label="Number of bathrooms"),
        gr.Slider(0, 4, step=1, label="Garage capacity (number of cars)"),
        gr.Slider(1, 10, step=1, label="Build Quality (1 to 10)"),
        gr.Number(1800, 2024, step=1, label="Year of construction"),
        gr.Dropdown(['CollgCr', 'Veenker', 'Crawfor', 'NoRidge', 'Mitchel', 'Somerst', 'OldTown', 'BrkSide', 'NAmes'], label="The neigh
    ],
    outputs=gr.Textbox(label="Expected price"),
    title="Home price forecast",
    description="Enter the home's specifications to get the estimated price."
)

interface.launch()
```

```
The model has been saved.
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True` (you

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://1751d2e3db5ff482da.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working dir
```