# Day49_Project_House_Price_Prediction_Using_All_Regression_Models

July 18, 2025

**Housing Price Prediction Using Multiple Regression Algorithms and Pickle**

**Introduction**

Today, we are moving a step ahead from individual regression algorithms and learning how to evaluate and compare multiple regression models in one go using a unified structure.

So far, we have explored:

- Regression models like Linear Regression, Ridge, Lasso, Random Forest, SVM, etc.

**In this notebook, we'll:**

- Train and evaluate multiple regression algorithms
- Save each trained model as a .pkl file using pickle
- Compare their performance using MAE, MSE, and $R^2$
- Prepare these models to be used with any frontend like Streamlit

# 1 Import Required Libraries

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import (LinearRegression, Ridge, Lasso, ElasticNet,␣
 ↪SGDRegressor, HuberRegressor)
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
import lightgbm as lgb
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import pickle
```

## 2 Load Dataset

[2]:
```
data = pd.read_csv(r"C:\Users\Lenovo\OneDrive\Desktop\Python Everyday␣
↪work\Github work\ML_Project\USA_Housing.csv")
data.head()
```

[2]:
```
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046             7.188236                   5.586729
4      59982.197226             5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
2                          5.13     36882.159400  1.058988e+06
3                          3.26     34310.242831  1.260617e+06
4                          4.23     26354.109472  6.309435e+05

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701…
1  188 Johnson Views Suite 079\nLake Kathleen, CA…
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482…
3                          USS Barnett\nFPO AP 44820
4                         USNS Raymond\nFPO AE 09386
```

## 3 Preprocessing

We drop columns that won't help with prediction (Address is non-numeric), and separate the target (Price).

[3]:
```
X = data.drop(['Price', 'Address'], axis=1)
y = data['Price']
```

## 4 Train-Test Split

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

## 5 Scale data for models that require it

[6]:
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

## 6 Define All Regressor Models

[7]:
```python
models = {
    # Models that need scaled data
    'LinearRegression': LinearRegression(),
    'RobustRegression': HuberRegressor(),
    'RidgeRegression': Ridge(),
    'LassoRegression': Lasso(),
    'ElasticNet': ElasticNet(),
    'PolynomialRegression': Pipeline([
        ('poly', PolynomialFeatures(degree=4)),
        ('linear', LinearRegression())
    ]),
    'SGDRegressor': SGDRegressor(),
    'ANN': MLPRegressor(hidden_layer_sizes=(100,), max_iter=1000),
    'SVM': SVR(),
    'KNN': KNeighborsRegressor(),

    # Models that don’t need scaling
    'RandomForest': RandomForestRegressor(),
    'LGBM': lgb.LGBMRegressor(),
    'XGBoost': xgb.XGBRegressor()
}
```

## 7 Train Models, Evaluate, and Save as .pkl

[8]:
```python
results = []

for name, model in models.items():
    # Check if model requires scaling
    if name in ['LinearRegression', 'RobustRegression', 'RidgeRegression',␣
↪'LassoRegression',
                'ElasticNet', 'PolynomialRegression', 'SGDRegressor', 'ANN',␣
↪'SVM', 'KNN']:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
```

```
    # Evaluation
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    results.append({
        'Model': name,
        'MAE': round(mae, 2),
        'MSE': round(mse, 2),
        'R2': round(r2, 4)
    })

    # Save model
    with open(f'{name}.pkl', 'wb') as f:
        pickle.dump(model, f)

print("All models trained, evaluated, and saved as .pkl files.")
```

```
C:\Users\Lenovo\anaconda3\Lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:780:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.000492 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1256
[LightGBM] [Info] Number of data points in the train set: 4000, number of used
features: 5
[LightGBM] [Info] Start training from score 1231911.452183
All models trained, evaluated, and saved as .pkl files.
```

## 8 Show Model Comparison Results

[9]:
```
results_df = pd.DataFrame(results)
results_df.sort_values(by='R2', ascending=False, inplace=True)
results_df.reset_index(drop=True, inplace=True)
print(results_df)
```

```
                  Model       MAE           MSE      R2
0      RobustRegression  82659.92  1.054623e+10  0.9147
1       RidgeRegression  82658.16  1.054893e+10  0.9147
2          SGDRegressor  82567.49  1.054623e+10  0.9147
3       LassoRegression  82657.87  1.054970e+10  0.9146
4      LinearRegression  82657.95  1.054972e+10  0.9146
5  PolynomialRegression  84013.48  1.073798e+10  0.9131
```

```
6                 LGBM    92133.99  1.309771e+10   0.8940
7         RandomForest    97709.90  1.473850e+10   0.8808
8              XGBoost   101565.19  1.613868e+10   0.8694
9                  KNN   105521.78  1.710311e+10   0.8616
10          ElasticNet   121396.83  2.288246e+10   0.8149
11                 SVM   282858.36  1.234840e+11   0.0009
12                 ANN  1175960.23  1.483829e+12 -11.0052
```

# 9 Save Results to CSV

[10]:
```python
results_df.to_csv('model_evaluation_results.csv', index=False)
print("Model evaluation saved to model_evaluation_results.csv")
```

```
Model evaluation saved to model_evaluation_results.csv
```

# 10 Final Notes

- All models are saved and can be loaded back in Streamlit or any app using pickle.load(open('model.pkl', 'rb')).
- This helps you pick the best model for production use based on accuracy ($R^2$), error (MAE), etc.
- You can now build a Streamlit app that allows you to upload input and choose a model to predict!

# 11 Conclusion

In this project, we developed and compared multiple regression models to predict **housing prices** based on various area-specific features such as average income, house age, number of rooms, number of bedrooms, and area population.

After training and evaluating 13 different models, we found the following insights:

- **Top Performing Models** (based on low MAE, MSE and high $R^2$ Score):
  - **SGD Regressor**: Lowest MAE (82,567) and high $R^2$ (0.9147)
  - **Ridge Regression** and **Robust Regression** closely followed with similar performance and strong generalization.
- **Baseline Models** like **Linear Regression** and **Lasso Regression** performed decently with an $R^2$ of ~**0.9146**, suggesting the problem is well-suited for linear approaches.
- **Polynomial Regression** slightly improved performance but at the cost of increased model complexity.
- **Ensemble Models** such as **LGBM** and **Random Forest** performed well but not significantly better than simpler models.
- **XGBoost** and **KNN** underperformed compared to others, with higher error values.

- **SVM** and **ANN** (Artificial Neural Network) showed **very poor performance**, especially the ANN which had an **$R^2$ of -11.0052**, indicating severe overfitting or improper tuning.

### 11.1 Final Recommendation

Considering performance and interpretability, **SGD Regressor**, **Ridge Regression**, and **Robust Regression** are the **best choices** for deployment in this use case. These models offer:

- High accuracy
- Low error margins
- Better generalization
- Simpler implementation and tuning

### 11.2 Future Scope

- Hyperparameter tuning for all models
- More advanced feature engineering or scaling methods
- Experimenting with stacked models or deep learning (with proper normalization and regularization)