# House Prices - Advanced Regression Techniques

Luca Corsetti

University of Bologna

2025

## Project Objective & Approach

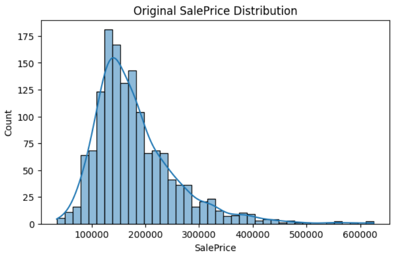
To build a regression model that predicts the sale price of houses in Ames, Iowa, using advanced regression techniques. The workflow followed these steps:

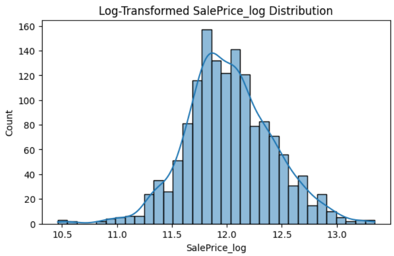
- Exploratory Data Analysis (EDA)
- Data cleaning and preprocessing
- Modeling & Evaluation
- Bonus - Trying to improve the Kaggle score

## The Dataset

1. **1460** training samples with **79** features describing various aspects of residential homes
2. Mixed data types: **numerical** (e.g GrLivArea), **categorical** (e.g Neighborhood), **ordinal** (e.g OverallQual)
3. Target variable in the train.csv dataset: **SalePrice**

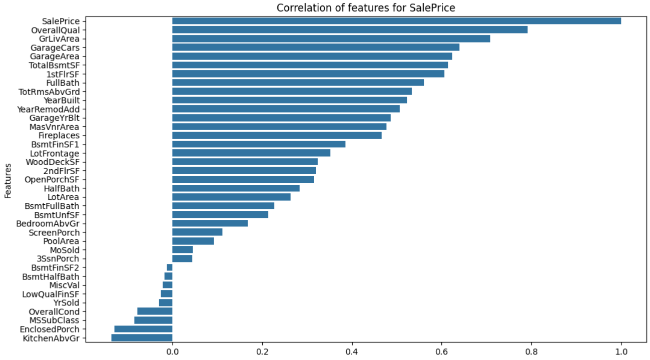
## Dataset Exploration: the target variable **SalePrice**





- Right-skewed distribution
- Solution: Apply log transformation to normalize the distribution

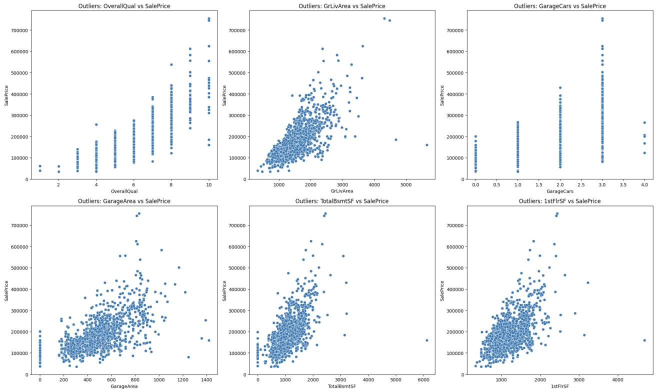
## Dataset Exploration: most influent features



Most influential features based on correlation with SalePrice ($> 0.6$):

- OverallQual (0.79) - ordinal
- GrLivArea (0.71) - numerical
- GarageCars (0.64) - numerical
- GarageArea (0.62) - numerical
- TotalBsmtSF (0.61) - numerical

## Dataset Exploration: outliers detection



Solution: Remove outliers based on domain knowledge as they were likely data entry errors.

## Data Preparation: handling missing values

- Some features (e.g PoolQC, Alley, Fence) were read with **NaN** values because Pandas interpreted **None** as **NaN**. These features actually represent the absence of a feature, so they **should not be** removed.
- Whereas other features (e.g LotFrontage) have real missing values that need to be imputed. For this, the **median** of the feature was used.

## Data Preparation: feature encoding

- Ordinal features (e.g ExternalQual: Ex, Gd, TA, Fa and Po) need to be mapped to numerical values **based on their order**.
- Categorical features (e.g Neighborhood) are **one-hot encoded** to **create binary features** since there is no ordinal relationship between categories.

```
# Ordinal encoding
from sklearn.preprocessing import OrdinalEncoder

qual_cols = ['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'HeatingQC', 'KitchenQual',
↪  'FireplaceQu', 'GarageQual', 'GarageCond']
# other ordinal columns here ..

qual_encoder = OrdinalEncoder(categories=[qual_cats] * len(qual_cols),
↪  handle_unknown='use_encoded_value', unknown_value=-1)
# other ordinal encoders here ..

train_dataset_cleaned[qual_cols] = qual_encoder.fit_transform(train_dataset_cleaned[qual_cols])
# same with others

# Nominal encoding
from sklearn.preprocessing import OneHotEncoder

df_final['MSSubClass'] = df_final['MSSubClass'].astype(str)
df_test_final['MSSubClass'] = df_test_final['MSSubClass'].astype(str)

nominal_cols = df_final.select_dtypes(include=['object']).columns
ohe = OneHotEncoder(drop='first', sparse_output=False, handle_unknown='ignore')

# Fit on the training data and transform it
encoded_train = ohe.fit_transform(df_final[nominal_cols])
```

## Modeling: strategy and metrics used

- Trained various regression models: Ridge, Random Forset, XGBoost
- Evaluated using Root Mean Squared Error (RMSE) and R-squared ($R^2$) metrics
- Fine-tuned hyperparameters using **GridSearchCV** with cross-validation (or **RandomSearchCV** to save time)

## Modeling: strategy and metrics used

$$\text{error} = \text{actual} - \text{predicted}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\text{actual} - \text{predicted})^2$$

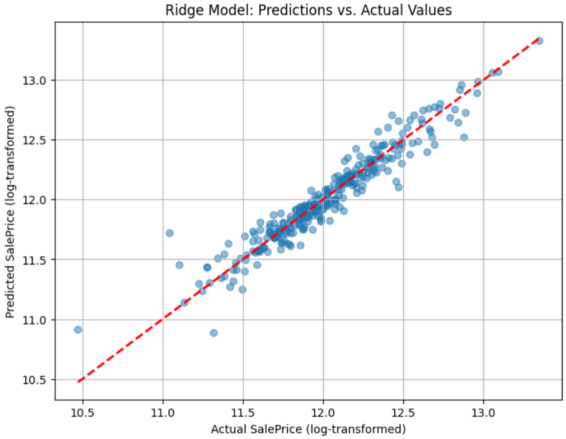
$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{actual} - \text{predicted})^2}$$

$$\text{SS}_{\text{res}}(\text{Sum of squared residuals}) = \sum i = 1^n(\text{actual} - \text{prediction})^2$$

$$\text{SS}_{\text{tot}}(\text{Total sum of squares}) = \sum i = 1^n(\text{actual} - \text{mean})^2$$

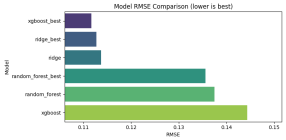
$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

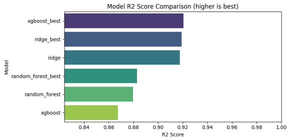
## Modeling: Ridge Regression



- Why not **Linear Regression**? More robust to **multicollinearity** and **overfitting**
- Best hyperparameter *alpha* $= 26.366508987303583$
- $RMSE = 0.1127$; $R^2 = 0.9192$ - not bad!

## Advanced Models: Random Forest & XGBoost





- **Tree based ensemble models** that can capture non-linear relationships and interactions between features
- Fine tuned hyperparameters of both, finding the **best model to be XGBoost**

## Advanced Models: Random Forest & XGBoost

```python
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb

param_grid = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [1.0, 'sqrt']
}

rf_random_search = RandomizedSearchCV(
    estimator=RandomForestRegressor(random_state=random_state),
    param_distributions=param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    random_state=random_state
)

# ..

param_grid = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.05, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 1.0]
}

xgb_grid_search = GridSearchCV(
    estimator=xgb.XGBRegressor(random_state=random_state),
    param_grid=param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)
```

## Submitting to Kaggle

- For this competition, Kaggle uses **RMSE** to evaluate the predictions on the test set
- Submitting the predictions of the XGBoost best model to Kaggle placed me in $1172^{th}$ position with a score of 0.12810
- **Can we do better?**

## Submitting to Kaggle

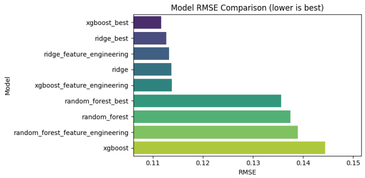
```
def create_submission(model, X_train_full, y_train_full, X_test_full,
↪   output_file_name='submission'):
    params = model.get_params()
    model_type_name = type(model).__name__

    model = None

    match model_type_name:
        case 'Ridge':
            model = Ridge(**params)
        case 'RandomForestRegressor':
            model = RandomForestRegressor(**params)
        case 'XGBRegressor':
            model = xgb.XGBRegressor(**params)
        case _:
            print(f"Error: Unhandled model type '{model_type_name}'")
            return

    model.fit(X_train_full, y_train_full)
    print("Retraining complete.")

    X_test_prepared = X_test_full.copy()

    print("Making predictions on the prepared test set...")
    final_predictions_log = model.predict(X_test_prepared)

    final_predictions = np.expm1(final_predictions_log)

    submission = pd.DataFrame({'Id': X_test_prepared.index, 'SalePrice': final_predictions})
    submission.to_csv(f'{output_file_name}.csv', index=False)
```

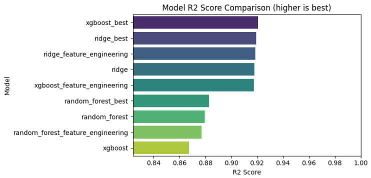
## Improving: Feature Engineering

- Noticed that I could've created more meaningful features from the existing ones
- Created new features like TotalSF (Total Square Footage), TotalBathrooms, Age, etc. that aggregate existing features into one.

```
def add_features(df):
    df['TotalSF'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']
    df['TotalBath'] = df['FullBath'] + 0.5 * df['HalfBath'] + df['BsmtFullBath'] + 0.5 *
    ↪  df['BsmtHalfBath']
    df['TotalPorchSF'] = df['OpenPorchSF'] + df['EnclosedPorch'] + df['3SsnPorch'] +
    ↪  df['ScreenPorch']

    df['HouseAge'] = df['YrSold'] - df['YearBuilt']
    df['RemodelAge'] = df['YrSold'] - df['YearRemodAdd']

    df['IsNew'] = (df['YrSold'] == df['YearBuilt']).astype(int)
    df['WasRemodeled'] = (df['YearRemodAdd'] != df['YearBuilt']).astype(int)

    df['OverallQual_x_TotalSF'] = df['OverallQual'] * df['TotalSF']
    df['OverallQual_x_HouseAge'] = df['OverallQual'] * df['HouseAge']

    return df
```

## Improving: Feature Engineering



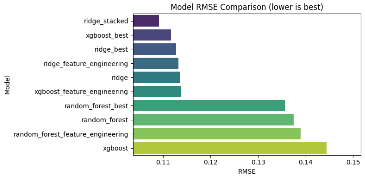


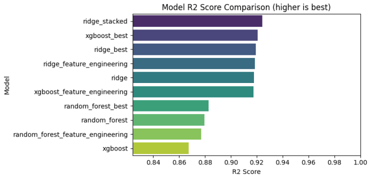
- **Retrained the models with the new features**, surprisingly none of them improved the score.

## Improving: Stacking

- Stacking is an ensemble learning technique that **combines multiple models** to improve predictive performance **by training a meta-model on their outputs**.
- Used Ridge, Random Forest and XGBoost as base models and trained a **Ridge regression as meta-model**.

## Improving: Stacking





- Retrained the models using Cross-Validation to generate out-of-fold predictions for the training set.
- Trained the meta-model on these predictions.
- Final Kaggle score after stacking: 0.12624, which placed me in the $829^{th}$ position!

## Improving: Stacking

```
base_models = [models['ridge_best']['model'], models['xgboost_best']['model'],
↪   models['random_forest_best']['model']]
base_model_names = ['ridge_best', 'xgboost_best', 'random_forest_best']

models['ridge_stacked'] = {
    'model': None,
    'prediction': None,
    'metrics': {}
}

models['ridge_stacked']['model'] = RidgeCV()

kf = KFold(n_splits=5, shuffle=True, random_state=random_state)

meta_features_train = np.zeros((X_train.shape[0], len(base_models)))
meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

for i, model in enumerate(base_models):
    print(f"Processing base model {i+1}/{len(base_models)}: {base_model_names[i]}...")

    # Create out-of-fold predictions for training data
    for train_idx, val_idx in kf.split(X_train):
        model.fit(X_train.values[train_idx], y_train.values[train_idx])
        meta_features_train[val_idx, i] = model.predict(X_train.values[val_idx])

    # Create predictions for test data (by fitting on full training data)
    model.fit(X_train, y_train)
    meta_features_test[:, i] = model.predict(X_test)

models['ridge_stacked']['model'].fit(meta_features_train, y_train)
```

## Conclusion

- **Dove deep** into data exploration and preprocessing to prepare the dataset for modeling.
- **Learned** about **XGBoost** and **ensemble methods** like stacking to improve model performance.
- Achieved a **final RMSE of** 0.12624 on Kaggle, placing me in the 829th **position** out of **4925**.