SUPERVISED MACHINE LEARNING:

LINEAR REGRESSION

COURSE PROJECT

Predicting House Prices with Linear Regression

IBM / COURSERA

IDDI ABDUL AZIZ

Introduction

Machine learning (ML) helps in finding complex and potentially useful patterns in data. These patterns are fed to a Machine Learning model that can then be used on new data points — a process called making predictions or performing inference.

Building a Machine Learning model is a multistep process. Each step presents its own technical and conceptual challenges. In this project, I will focus on the process of selecting, transforming, and augmenting the source data to create powerful predictive signals to the target variable using Supervised learning: Linear Regression.

Simple Linear Regression is a statistical technique that allows us to summarize and study relationships between two continuous i.e. numeric variables:

- The variable we are predicting is called the *criterion* or *response* or *dependent* variable, and
- The variable we are basing our predictions on is called the *predictor* or *explanatory* or independent

Simple linear regression gets its adjective 'simple', because it concerns the study of only one predictor variable.

Linear Regression attempts to model the linear relationship between *predictor variables* (X) and a *numeric target variable* (y). For each observation that is evaluated with the model, the actual value of the target (y) is compared to the predicted value of the target (ŷ), and the difference in these values are known as r*esiduals.* The goal of a linear regression model is to minimize the sum of the squared residuals. The equation for a multiple linear regression is shown below.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots \beta_p x_p + \varepsilon$$

Data Description

For this project, the Ames Housing Data Set was used. This data set includes information for houses sold in Ames, Iowa from 2006 to 2010. There are 2930 observations and over 80 features included in the data set.

Below, I will describe the steps that I took to build a linear regression model to predict house prices. Before I could begin exploring the data, I imported the libraries shown below. I did also clean the missing values and ensured that there were no abnormal values and that each feature was of the appropriate data type.

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
```

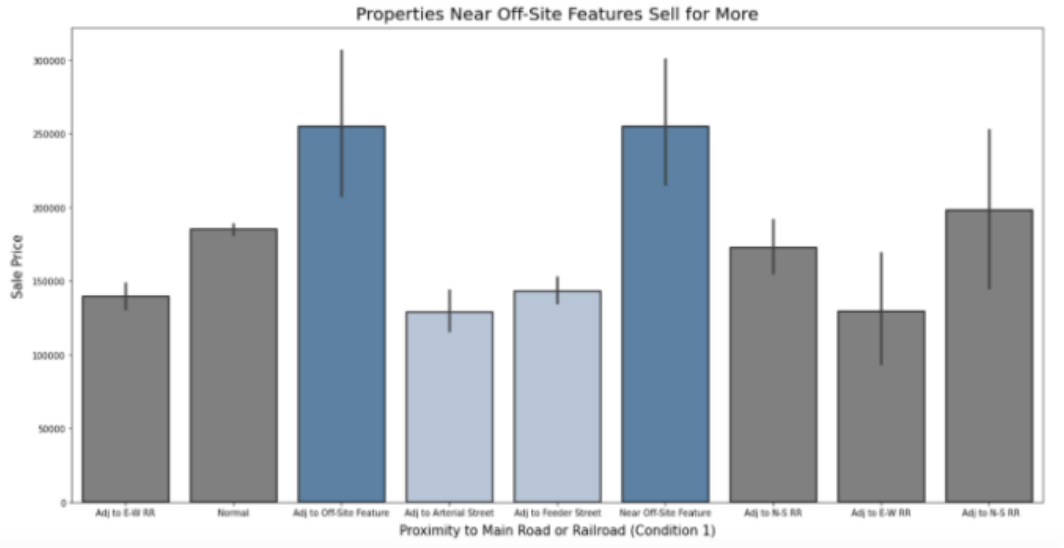
Summary of Exploratory Data Analysis (EDA)

EDA is very important for exploring the relationship between the different features and target *'SalePrice'*. For numeric features, I created a correlation matrix and specifically looked for any features that had a correlation coefficient with an absolute value of 0.5 or higher. This really helped in identifying several features that were strongly correlated with the sale price, including the overall quality of the house, the living area, and the size of the garage.

```
corr_target = abs(corr_matrix)
salient_num_features = corr_target.loc[corr_target['Sale Price'] > 0.5, :]
salient_num_features
```

For categorical variables, understanding the relationship between the different categories and sale price was a little more complicated. In total, this data set contained over 20 categorical features, and some features had over 10 different categories. In order to reduce dimensionality of the model, I knew that simply *one-hot-encoding* the categories was not an option. To solve this issue, I created bar plots for each categorical feature to explore the differences in mean sale price between the different categories. Interestingly, as shown in the plot below, I began to notice that some of the categories of various features began to cluster together. By recognizing these clusters, I was able to engineer features to represent the categorical variables.

'SalePrice' in this barplot created with Matplotlib and Seaborn, it is evident that both the light blue bars and the dark blue bars are similar to each other in sale price and category description. Recognizing these similarities, this categorical feature with nine categories can be represented by just two binary features.



Model Building

I ought to understand if this work is ready to go or if it needs review. Modeling focuses on developing models that are either descriptive or predictive. So here, I perform Predictive modeling, which is a process that uses data mining and probability to forecast outcomes. For predictive modeling, data scientists use a training set that is a set of historical data in which the outcomes are already known. This step can be repeated more times until the model understands the question and answer to it.

If we have categorical data, then we need to create dummy variables, I also split the data into train and test sets with a test size of 20%. I tried three different models and evaluated them using R2 score.

```
def display_R2_scores(model, X_train, y_train, X_test, y_test):
    print(f'The mean cross validation score for this model is {round(cross_val_score(model, X_train, y_train).mean(),
    print(f'The training score for this model is {round(model.score(X_train, y_train), 4)}.')
    print(f'The testing score for this model is {round(model.score(X_test, y_test),4)}.')
```

First Model 1 – The baseline/null model created using ***DummyRegressor ()***. It was well conducted to explore the R2 score when the mean was predicted for each observation.

Second Model 2 – This model was a linear regression model using features identified to be important during EDA.

Third Model 3 – After seeing that the linear regression model had room for improvement, I pull all the possible features into a model to create an overfit model that could then be regularised.

Fourth Model 4 – The first type of regularization that I tried was Ridge Regression, which uses L2 penalty.

```
# Scale the data
ss = StandardScaler()
Z_train = ss.fit_transform(X_train_1)
Z_test = ss.transform(X_test_1)
scaled_holdout = ss.transform(X_holdout_1)

# Model
ridge = Ridge(alpha = 1025)
ridge = ridge.fit(Z_train, y_train_1)
```

Fifth Model 5 – The second type of regularisation that I tried was LASSO Regression.

```
lasso = Lasso(alpha = 5000, max_iter = 4000)
lasso.fit(Z_train, y_train_1);
```

Sixth Model 6 – The final model was a linear regression with features identified to be important during EDA and the LASSO regression. Because that target was not normally distributed, for this model, it was log-transformed.

```
lr = LinearRegression()
lr.fit(X_train_4, y_train_4);
display_R2_scores(lr1, X_train_4, y_train_4, X_test_4, y_test_4)
```

Model Selection

As shown in the table below, of each of the models that I built, the Linear Regression that was built performed the best in terms of R2 score, and it was also not overfit.

Model Name	Training Score	Testing Score
Baseline/Null Model	0%	0%
Linear Regression (Overfit)	94.6%	74.9%
Ridge	89.8%	81.9%
LASSO	84.6%	80.8%
Linear Regression (Features from EDA and LASSO)	87.5%	87.8%

Model Evaluation

I further evaluate the Linear Regression model by calculating the Root Mean Squared Error and examine the residuals. In order to increase interpretability, the logged y-values was exponentiated for model evaluation.

After selecting my best model, I further evaluated its performance by calculating the Root Mean Squared Error (RMSE) and by comparing the actual values to the predicted values.

```
y_test = np.exp(y_test_4) # Convert back to actual (non-log) values
y_pred = np.exp(lr1.predict(X_test_4)) # Convert back to actual (non-log) values
resids = y_test - y_pred

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f'The RMSE is {round(rmse,2)}.')
```

According to the value of the RMSE, the predictions for Sale Price made by the Linear Regression model are likely off by about $23,625.19. This is less than 1/3 of the standard deviation of the Sale Price in the training data. Although it would be nice to improve this number, this is an okay starting point to improve from.

Prediction of *'SalesPrice'* compared to the actual values.

From the below plot, we are able to see that this model does not perform equally well at predicting all prices. Specifically, the model seems to be good at predicting the price of properties within the range of $90,000 to $225,000, but it does not do a good job at either extreme. In the original testing data set, 75% of the houses were priced at or under $214,000. The model might not perform well at extreme values due to insufficient training data for these cases.

Conclusions

Although the linear regression model created in this post is not perfect, it is able to account for approximately 87.8% of the variation in *'SalePrice'* of a property, and it also is able to predict the *'SalePrice'* within $23,625.19. Further exploration of the residuals of this model revealed a specific weakness in this model at predicting extreme values. Specifically, the ideal range for predictions appeared to be within $90,000 to $225,000. In order to better predict prices outside of this range, some changes could be made to the model. Ideas for improvement include log transforming features and coding some categorical variables as ordinal.