Multiple Linear Regressions

Imagine you're moving out of town and need to sell your house. How do you determine a fair asking price? You might compare it to your neighbor's house, which has the same number of bedrooms—but yours has a bigger backyard, so you think it should be worth more. Another friend's house has the same square footage and number of bedrooms as yours but is located in a more preferable neighborhood, which could increase its value.

If you had a dataset with details on all the houses in town - features like square footage, number of bedrooms, location, and their final sale prices - you could use it to estimate your home's value. One way to do this is through **linear regression**.

What is Linear Regression?

A linear regression is a model of the relationship between one dependent variable and one or more independent variables. In our example, the price of a house is *dependent* on several factors - the number of bedrooms and bathrooms in the house, its location, and so on. If your house has 3 bedrooms, you can't change that in the immediate future, so it is what it is. Formally, the 'bedrooms' feature is *independent*.

In this post, we will use a multiple linear regression model to predict the price of a house. A multiple linear regression model is a linear regression that extends simple linear regression by using more than one predictor variable to predict the target variable. Instead of just one feature (like square footage), we use several to improve prediction accuracy.

Our dataset initially contains 13 predictors, but after some preprocessing, we end up with 14 variables. If you're interested in the details of data cleaning and feature selection, check out the 'Feature Engineering' and 'Correlation Matrix' code file sections.

When Should You Use Multiple Linear Regression?

You've learned what multiple linear regression is, but how can you be sure it works for your data? Before diving into the code, check if your data meets key assumptions. These assumptions help ensure that your model is reliable and produces meaningful insights.

P.S. A more detailed explanation of the first four assumptions can be found in my previous post.

1. **Linearity**

The relationship between the independent and dependent variables is linear.

This means that the effect of each independent variable on the dependent variable can be modeled with a straight-line equation.

2. **Independence of observations**

Each data point should be independent of the others, meaning that no observations are overly correlated. If observations are highly correlated, the model may struggle to correctly estimate the impact of each variable. This can lead to misleading feature importance, unstable coefficients, and ultimately, unreliable predictions.

3. **Homoscedasticity**

Homoscedasticity means that the variance of residuals remains consistent across the range of the independent variables. In a homoscedastic dataset, residuals - the green lines indicated in the below plot - do not vary much.

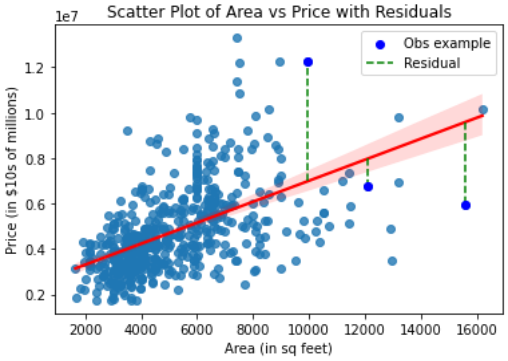

Figure 1

But that's not the case with our original housing dataset visualized above. The dataset creates a cone-shaped pattern, a classic sign of **heteroscedasticity**—the opposite of what we

want. Heteroscedasticity can be dealt with in many ways, but with this dataset, I will simply remove the *anomalies* to allow for handling inconsistencies more accurately. Anomalies, or outliers, are data points that deviate significantly from the trend seen in the rest of the dataset.

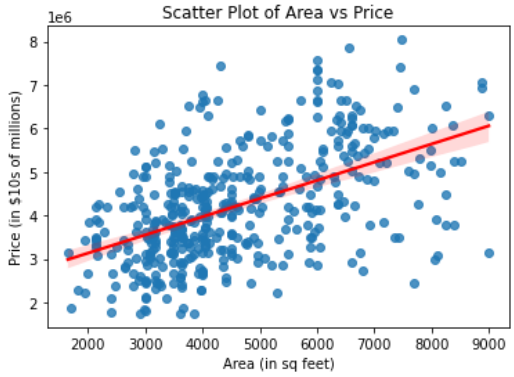

Figure 2

In the scatterplot above, you'll see the dataset's distribution after removing anomalies using the Interquartile Range (IQR)[1] Method, a.k.a. the Fence Rule. You'll notice that the distribution of residuals has improved. This method detects outliers by analyzing the spread of data between the first (Q1) and third (Q3) quartiles, flagging values that fall too far outside this range.

- **Initial dataset size:** 545 observations
- **After outlier removal:** 438 observations

While this reduction may seem significant, it ensures that our model generalizes well rather than being skewed by extreme values, conforming to the assumption of homoscedasticity.

4. Normality of error distribution

The error term (ε) represents the difference between the observed value and the true, unobservable regression function prediction. Because the true regression is unobservable, we can't know the exact difference between an ideal prediction and the actual value. That is why we use residuals to estimate the error terms. In other words, this assumption states that the distribution of residuals should be normal. This ensures that our model doesn't systematically over- or under-predict in certain scenarios.

[1] To learn more about this technique and how it is coded, refer to the 'MLR_anomoly_removal' code file

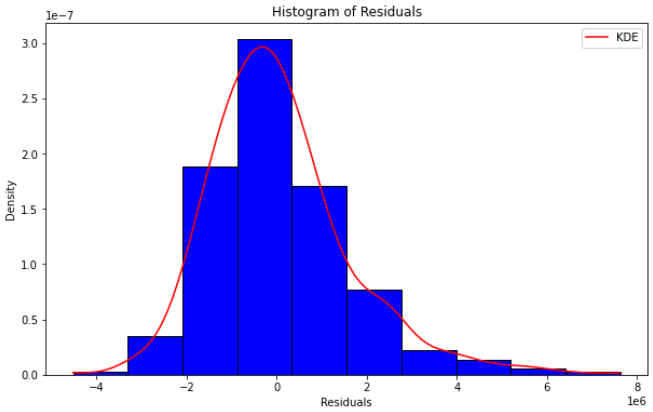

Figure 3

In the original dataset, the residuals generally follow a normal distribution, though there's a slight skew to the right. This suggests that there may be room for improvement in the model. After removing anomalies from the dataset - mentioned in the previous assumption - the spread of residuals looks like this:

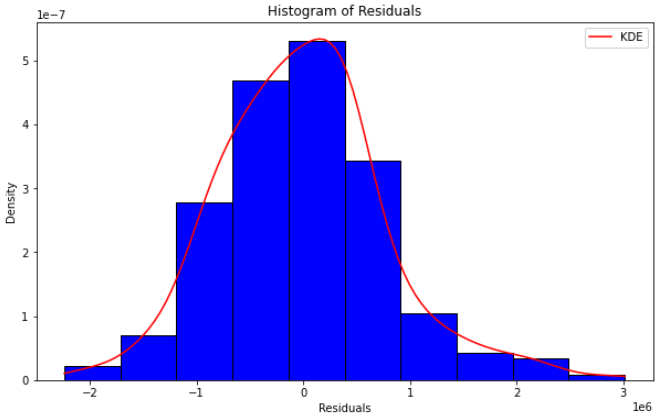

Figure 4

The bar graph above displays a marked improvement in the skew of the distribution as compared to Figure 3. While not perfectly normal, the skew has been noticeably reduced, which strengthens the model's predictive reliability.

5. No Multicollinearity

Multicollinearity occurs when two or more independent model variables are highly correlated[2], making it difficult to determine their individual effects on each variable on the target variable.

[2] For a better understanding of correlation, refer to this post.

For example, in a house price prediction model, both lot size and total square footage might be used as predictors. Since larger lots typically accommodate larger houses, these features could be highly correlated, making it difficult for the model to distinguish their separate impacts. This makes it difficult for the model to determine whether lot size or total square footage is driving changes in house prices, resulting in unstable coefficients and less reliable insights. To check for multicollinearity, let's build a *correlation matrix* - a plot that shows us the degree of correlation between each variable:

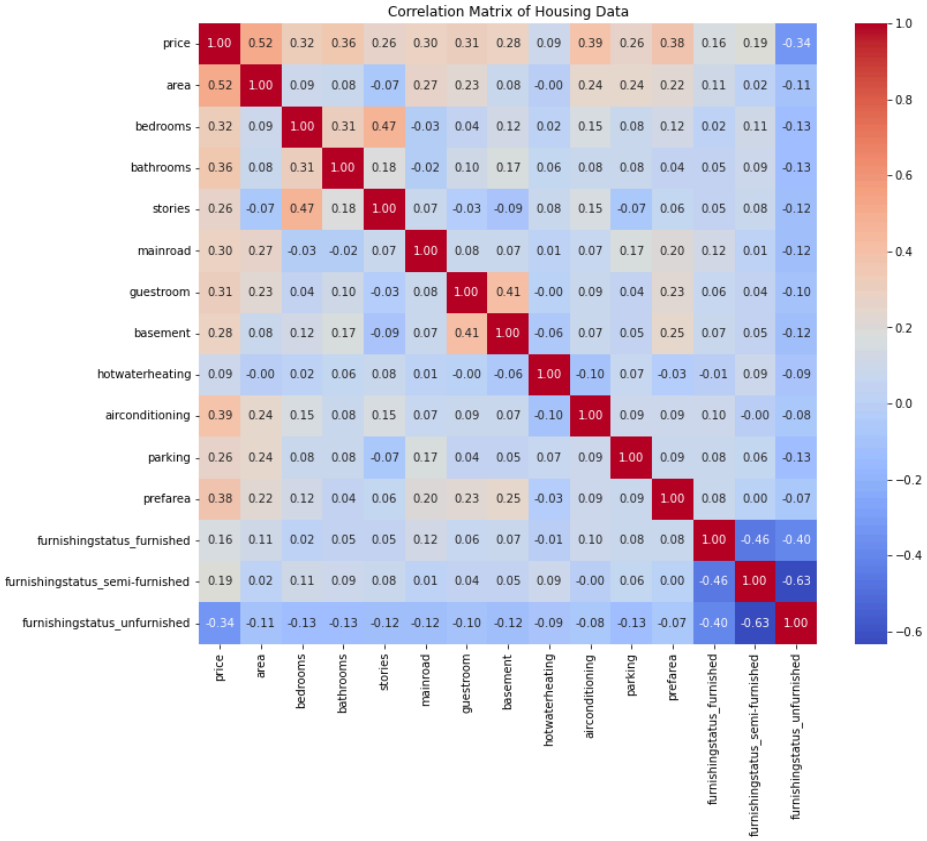

Figure 5

Each square displays a number that reflects the strength and direction of correlation between the variable in the column and that in the row; 1 denotes that the variables are perfectly positively correlated, 0 denotes that they are not correlated at all, and -1 denotes that the variables are perfectly negatively correlated. An absolute correlation coefficient that is greater than 0.5 is considered a strong correlation[3], and this is the threshold we'll use to avoid multicollinearity. Looking at the bottom right corner of the matrix, you'll notice variables that are shaded in with darker colors, signifying moderate to high correlation.

[3] https://www.scribbr.com/statistics/pearson-correlation-coefficient/

‘Furnishingstatus_semi-furnished’ and ‘Furnishingstatus_unfurnished’ variables have a high correlation coefficient of - 0.63. Intuitively, this makes sense—if a house is unfurnished, it cannot be semi-furnished, and vice versa.

Now that we’ve ensured our data meets these key assumptions, it’s time to build our multiple linear regression model and estimate house prices. Let's dive in!

Can We Build It? Yes, We Can!

With Python, initializing and training a linear regression model is remarkably straightforward—it takes just two lines of code:

```
model = LinearRegression()
model.fit(X1+X2+...+Xp, Y)
```

The first line sets up the model, and the second line uses the fit() function to learn the relationship between all the independent variables and the price. Python’s Scikit-learn library makes it that easy to initialize a multiple linear regression model, but what’s going on inside the ‘black box’? In machine learning, when we call something a ‘black box’, we mean that the internal process is hidden from us. Though scikit-learn makes it easy to use this function, we still want to understand what’s going on inside.

Math in the Black Box

The goal of multiple linear regression is to estimate the best-fit relationship between multiple predictors (X_1, X_2, ..., X_p) and a target variable (Y) so we can predict future values of Y given new inputs. We achieve this by using the following equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$$

Where

1. $\beta 0$ is the y-intercept. It is the predicted value of Y when all Xs are 0.

2. $\beta_1, \beta_2, \ldots, \beta_p$ are the regression coefficients, representing how much Y changes per unit increase in each predictor while keeping others constant.
   Given that p is the number of independent variables in the model, the equation implies that we add the rest of the coefficients to the equation. Since our model has 13 predictors, we continue adding $\beta_3, \beta_4, \beta_5$, and so on, all the way till β_{13} to its respective predictor.

3. ε is the error term - the unexplained variation in Y, covered in our normality of error distribution assumption.

Rewritten in the context of house prices, the equation would look like this:

Price = $+\beta_1 \times$ (Square Footage) $+ \beta_2 \times$ (Bedrooms) $+ \ldots + \beta_{14} \times$ (Furnishing) $+ \varepsilon$

But how do we actually compute these coefficients? That's where **Ordinary Least Squares (OLS)** comes in.

Linear regressions are synonymous with OLS regressions. The ordinary Least Squares method is a technique for estimating linear regression coefficients by minimizing the sum of square residuals (SSR):

$$\text{SSR} = \sum_{i=1}^{n} (y_i - \widehat{y}_i)^2$$

To minimize the SSR, we find the partial derivatives of β_0 and β_1. Solving for these derivatives gives us the following formulas for the regression coefficients:

$$\beta_1 = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} (x_i - x_1)^2} \quad \text{and} \quad \beta_0 = \overline{y} - \beta_1 \overline{x}$$

We could continue to use the same equations for β_0 and every β_i. While this continues to be a mathematically sound solution, calculations can get tedious, especially if you are using a

massive dataset with more than 13 predictors. Hence, for multiple linear regressions, we use matrix operations instead.

Regression in Matrix Form

For this next part, you'll need to know some basic linear algebra. You'll remember that a matrix is a rectangular arrangement of numbers into rows and columns[4] (a quick refresher can also be found on the same website). To represent the regression equation in matrix form, we define three key matrices:

- Design Matrix (X) - also known as a regressor matrix. The first column consists of 1s to represent a constant term, each column thereafter represents a regressor, and each row represents an observation. Our design matrix X will have $p+1$ columns and n rows. Dimensions:

$$X = n \times (p+1)$$

- Target vector (y) - populated by all the observed Y values. Dimensions:

$$y = n \times 1$$

- Coefficient vector (β) - A matrix of predicted regression coefficients (including β_0). Dimensions:

$$\beta = (p+1) \times 1$$

Now that we have built design and target matrices, the first order of business is to build an equation where the squared sum of residuals is minimized. In a scenario where the residuals are the lowest possible value - 0 - the equation would be:

$$Y - (\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon) = 0$$

Re-arranging this equation, the predicted y-value will equal the y-value observed.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$$

If we were to write this out in matrix form, where each row represents the residual of the nth observation, the matrix would look like this:

[4] https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:matrices/x9e81a4f98389efdf:mat-intro/a/intro-to-matrices

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} \beta 0 + \beta 1 x_{1,1} + \beta 2 x_{1,2} + \dots + \beta p x_{1,p} + \varepsilon_1 \\ \beta 0 + \beta 1 x_{2,1} + \beta 2 x_{2,2} + \dots + \beta p x_{2,p} + \varepsilon_2 \\ \dots \\ \beta 0 + \beta 1 x_{n,1} + \beta 2 x_{n,2} + \dots + \beta p x_{n,p} + \varepsilon_n \end{bmatrix}$$

The little numbers next to the xs denote the row and column of the element in the matrix we are using. For example, $x_{1,2}$ would be the value in the first row and second column, or the element in the first observation of the second predictor.

If you look at this matrix carefully, you'll notice that on the right-hand side, the design matrix X is multiplied by the coefficient vector, and the result is added to a vector of error terms. Broken down, the operation would look like this:

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ \dots & \dots & \dots & \dots \\ x_{n,1} & x_{n,2} & \dots & x_{n,p} \end{bmatrix} \times \begin{bmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \\ \varepsilon_n \end{bmatrix}$$

Generalizing the variables in the matrix, the operation would look like this:

$$y = X\beta + \varepsilon$$

Re-arranging the formula, the error term would be:

$$\varepsilon = y - X\beta$$

Recall from the normality of error distribution assumption that we learned that residuals are an approximation for error terms. Therefore, residuals are estimated to be

$$y - X\beta$$

Cool, we have calculated a theoretical residual value by creating an equation, where y is the actual observed value and $X\beta$ is the predicted value of y. To proceed to the next step in the SSR calculation, the residual must be squared. To find the residual square, we calculate the dot product between the above expression and the transposed expression, which can be written as:

$$(y - X\beta)^T \times (y - X\beta)$$

Just like in simple linear regressions, to find the coefficients that minimize the SSR, calculus must be used. From the equation above, we must solve for the partial derivative of the predicted β vector. I won't indulge you in the specific calculations, though here is a great resource to understand the calculus methods that bring us to the result:

$$\beta = (X^T X)^{-1} X^T y$$

The β vector containing all predicted values of $\beta 0$ through βp can now be calculated using this formula. Smooth, isn't it?

Given these equations, here's how I would calculate the β vector in Python manually. The code includes two methods, one building a linear regression model: one uses the Scikit-learn library, and the other is manually coded. The code file is documented step-by-step and also gives a better understanding of how to compute the math explained above:

```
def beta_vector(X,y):
    design_m = np.c_[np.ones(X.shape[0]), X]
    target_v = y

    XtX = design_m.T @ design_m
    XtX_inv = np.linalg.pinv(XtX)

    coeff = XtX_inv @ (design_m.T @ target_v)
    return coeff
```

By understanding the math behind multiple linear regression, you gain deeper insight into how models interpret relationships between variables. With Python, implementing regression is simple, but knowing what happens inside the "black box" allows you to fine-tune models for better accuracy.

Predictions

Now that we have our coefficients, it's time to put them to use! The predicted values follow the multiple linear regression equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$$

Since OLS minimizes the error term ε as much as possible, we can estimate predictions using $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$ to predict the prices of houses. We know $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$ can also be expressed as $X\beta$. This means that to compute predictions, we take the dot product of our design matrix X and the coefficient vector β. In Python, this is done in a single line:

```
Y_pred = X@β
```

And there you have it - we have predicted the prices of houses using multiple linear regression.

Evaluating the Model

Now that our model is built and we're predicting prices of houses, how do we know it performs well? We use key evaluation metrics[5] to measure accuracy:

- Mean Absolute Error: This calculates the average absolute residual error. In simple terms, MAE tells you how far the model's predictions are from the actual values on average.

- Mean Squared Error: calculates the average squared residual error. Squaring the errors penalizes larger discrepancies more than smaller ones, which helps MSE highlight significant mistakes in predictions.

- Root Mean Squared Error: is the square root of the MSE metric. Because the MSE is a squared value, it is not in the same units as the predicted variable, like the MAE is. Thus, we take the root of MSE or RMSE to get the average magnitude of the error.

[5] For more information on this metrics, refer to my previous post

- R^2 score: or the coefficient of determination, tells us how well the model can explain variance in the predicted variable. The R^2 score is calculated using the formula: $1 - \frac{SSR}{SST}$. Where:

- SSR (Sum of Squared Residuals) measures the total squared error in predictions.
- SST (Total Sum of Squares) represents the total variance in the actual values. It is calculated using the formula:

$$\sum_{i=1}^{n} (y_i - \bar{y})^2$$

By comparing SSR to SST, the R^2 score tells us how well the model explains the variation in the dependent variable. The closer an R^2 value is to 1, the better the model, while a value closer to 0 suggests that the model doesn't explain much of the variance in the data.

In my code file, I've demonstrated how these 3 model evaluation metrics can be coded manually in Python.

Interpreting the Model

You're so close! You've built your model and evaluated it, and now it's time to interpret what these metrics mean in the context of our dataset.

1. By taking the absolute differences between predicted and actual values, the MAE is not concerned with the *direction* of the residuals. In our case, with house prices ranging from $1.75 million to $8 million, an MAE of $568,421.76 means that, on average, the model's predictions are off by almost $0.5 million. Given that the mean house price is $4.22 million, this accounts for a 12.5% deviation—a reasonable starting point but with room for improvement.

2. MSE squares residual errors, and it's particularly useful for highlighting large errors. While this helps identify significant mistakes, it also makes the metric sensitive to outliers. Our MSE score is 556,516,442,272, or more than half a trillion — yikes, this is contrary to the decent start we were off to! While large errors could be the main suspect for this high MSE score, I will venture a guess and say that our dataset still has many

extreme outliers. It's possible that certain houses have unique, unrecorded features affecting their prices, making them difficult to predict accurately.

3. The RMSE is just the square root of the MSE score. This makes the error easier to interpret because it's in the same units as the original data (dollars). With an RMSE of $746,000, our model's predictions could be off by $0.75 million on average. This deviation accounts for 18.5% of the mean house price, which is slightly worse than MAE, reinforcing the impact of large errors.

4. Lastly, the R^2 score explains how well the model can predict the variance in the data. With an R^2 of 0.73 (or 73%), our model explains a significant portion of the variation in house prices. Typically, an R^2 value over 60% is considered good, so our multiple linear regression did well on this metric! The model can explain a majority of the variance seen in the dataset, though the remaining 27% variance suggests that some predictive features might be missing.

Conclusion

Multiple linear regressions are a powerful mathematical tool for beginner statisticians and machine learning practitioners. If you've followed along from the simple linear regression post, you'll see major improvements in our model's performance. By incorporating more features, handling outliers, and fine-tuning our model, we were able to make more accurate predictions.

While our model shows significant progress, there's still room for refinement. We addressed heteroscedasticity and multicollinearity, but the residuals remain non-normally distributed. A common fix is applying a log or square root transformation to the target variable—something we'll explore in later posts as we move into more advanced techniques. Additionally, the high MSE suggests missing predictive features. Some houses with similar independent variables still show vast price differences, hinting at unrecorded factors influencing price. Thinking like a data scientist, I'm beginning to wonder if crucial features are not captured in the data that could help us improve the prediction of a house. If this were a corporate or

research project, a deeper feature analysis would be crucial—examining neighborhood effects, renovations, or other key attributes could improve accuracy.

Despite these challenges, this multiple linear regression model outperforms our simple linear regression. The R^2 score has significantly improved, and on average, residuals have dropped by a million dollars—a clear sign that adding complexity can enhance predictive power.

In this post, we've learned:

- What multiple linear regression is
- Assumptions of multiple linear regressions and how to assess your dataset for them
- How to identify and remove outliers using the IQR method
- Two mathematical approaches to multiple linear regressions -
    - the linear algebra/ matrix operations method
    - the extension of simple linear regressions
- How to build a multiple linear regression using the scikit-learn library in Python
- How to build a multiple linear regression manually in Python
- How to predict values using scikit and manually in Python
- How to evaluate the model using MAE, MSE, RMSE, and R^2 metrics with scikit and manually in Python
- How to interpret model evaluations
- As a bonus: How to create the boxplots and violin plots for data visualization

Whew, that's a lot to take in! We're done covering linear regressions for now. Next, we'll look at non-linear regressions. Stay tuned for a deep dive into the black box of logistic regressions, where we'll look at how to classify data as opposed to making predictions.