<https://www.analyticsvidhya.com/blog/2020/07/types-of-feature-transformation-and-scaling/>

Feature Scaling and Transformation:

* MinMax Scaler\*\*\*

|  |
| --- |
| from sklearn.preprocessing import MinMaxScaler  scaler = MinMaxScaler(feature\_range=(5, 10))  df\_scaled[col]=scaler.fit\_transform(features.values) |

* Standard Scaler\*\*\*

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  df\_scaled[col]=scaler.fit\_transform(features.values) |

* MaxAbsScaler
* Robust Scaler
* Quantile Transformer Scaler
* Log Transformation\*\*\*

|  |
| --- |
| df['log\_income'] = np.log(df['Income']) |

* Power Transformer Scaler
* Unit Vector Scaler/Normalizer

WHY NEEDED

To be sure that the model treats both these variables equally? When we feed these features to the model as is, there is every chance that the income feature will influence the result more due to its larger value. But this doesn’t necessarily mean it is more important as a predictor. So, to give importance to both Age, and Income, we need feature scaling.

|  |
| --- |
| df\_scaled = df.copy()  col\_names = ['Income', 'Age']  features = df\_scaled[col\_names] |

MinMax Scaler

The MinMax scaler is one of the simplest scalers to understand. It just scales all the data between 0 and 1.

x\_scaled = (x – x\_min)/(x\_max – x\_min)

|  |
| --- |
| from sklearn.preprocessing import MinMaxScaler  scaler = MinMaxScaler()  df\_scaled[col\_names] = scaler.fit\_transform(features.values)  #We can also set the range  scaler = MinMaxScaler(feature\_range=(5, 10)) |

Standard Scaler

For each feature, the Standard Scaler scales the values such that the mean is 0 and the standard deviation is 1(or the variance).

x\_scaled = x – mean/std\_dev

However, Standard Scaler assumes that the distribution of the variable is normal. Thus, in case, the variables are not normally distributed, we

* either choose a different scaler or
* first, convert the variables to a normal distribution and then apply this scaler.

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  df\_scaled[col\_names] = scaler.fit\_transform(features.values)  df\_scaled |

Log Transform

It is primarily used to convert a skewed distribution to a normal distribution/less-skewed distribution. In this transform, we take the log of the values in a column and use these values as the column instead. It is because the log function is equipped to deal with large numbers. Here is an example- log(10) = 1, log(100) = 2, and log(10000) = 4.

Thus, the log operation had a dual role:

* Reducing the impact of too-low values
* Reducing the impact of too-high values.

|  |
| --- |
| df['log\_income'] = np.log(df['Income'])  # We created a new column to store the log values |

If our data has negative values or values ranging from 0 to 1, we cannot apply log transform directly – since the log of negative numbers and numbers between 0 and 1 is undefined, we would get error or NaN values in our data. In such cases, we can add a number to these values to make them all greater than 1. Then, we can apply the log transform.