Feature scaling is a vital pre processing step in machine learning that involves transforming numerical features to a common scale. It plays a major role in ensuring accurate and efficient model training and performance. Scaling techniques aim to normalize the range, distribution, and magnitude of features, reducing potential biases and inconsistencies that may arise from variations in their values.

Well, Feature scaling, in the context of machine learning, refers to the process of transforming the numerical features of a dataset into a standardized range.

It involves bringing all the features to a similar scale, so that no single feature dominates the learning algorithm. By scaling the features, we can ensure that they contribute equally to the model’s performance.

**Role of Feature Scaling in Machine Learning:**

Feature scaling plays a crucial role in machine learning for a variety of reasons:

· Many machine learning algorithms use distance-based calculations to make predictions. If the features are not scaled, those with larger values can have a disproportionate impact on the results.

· Feature scaling can help improve the convergence speed and performance of some optimization algorithms.

· This helps in handling skewed data and outliers, which can influence the model’s behavior.

**Importance of Feature Scaling in Machine Learning**

**Enhancing Model Performance:**

[Feature scaling](https://www.pickl.ai/blog/feature-scaling-in-machine-learning/) can significantly enhance the performance of machine learning models. Scaling the features makes it easier for algorithms to find the optimal solution, as the different scales of the features do not influence them.

It can lead to faster convergence and more accurate predictions, especially when using algorithms like k-nearest neighbors, support vector machines, and neural networks.

**Addressing Skewed Data and Outliers:**

Skewed data and outliers can negatively impact the performance of machine learning models. Scaling the features can help in handling such cases. By transforming the data to a standardized range, it reduces the impact of extreme values and makes the model more robust.

This is particularly beneficial for algorithms that assume a normal distribution and are sensitive to outliers, such as linear regression.

**Faster Convergence During Training:**

For gradient descent-based algorithms, feature scaling can speed up the convergence by helping the optimization algorithm reach the minima faster.

Since gradient descent updates the model parameters in steps proportional to the gradient of the error with respect to the parameter, having features on the same scale allows the algorithm to take more uniform steps towards the optimum and reduces the number of iterations needed.

**Balanced Feature Influence:**

When features are on different scales, there is a risk that larger-scale features will dominate the model’s decisions, while smaller-scale features are neglected. Feature scaling ensures that each feature has the opportunity to influence the model without being overshadowed by other features simply because of their scale.

**Improved Algorithm Behavior:**

Certain[machine learning algorithms](https://www.pickl.ai/blog/top-deep-learning-algorithms-in-machine-learning/), particularly those that use distance metrics like Euclidean or Manhattan distance, assume that all features are centered around zero and have variance in the same order.

Without feature scaling, the distance calculations could be skewed, leading to biases in the model and potentially misleading results. Feature scaling normalizes the range of features so that each one contributes equally to the distance calculations.

**Different Types of Feature Scaling Techniques:**

There are different types of feature scaling techniques as well that you will learn during machine learning free course. These are:

**Normalization:**

Normalization, also known as min-max scaling, transforms the features to a range between 0 and 1. It subtracts the minimum value of the feature and divides it by the range (maximum value minus minimum value). This technique is suitable when the distribution of the data does not follow a Gaussian distribution.

**Standardization:**

Standardization transforms the features to have a mean of 0 and a standard deviation of 1. It subtracts the mean of the feature and divides it by the standard deviation.

This technique is preferable when the data is normally distributed or when we don’t know the distribution in advance. Standardization maintains the shape of the distribution and does not bound the features to a specific range.

**Practical Examples and Best Practices for Feature Scaling:**

Understanding feature scaling in theory is one thing, but applying it correctly in practice is equally important. Consider the following best practices:

**Feature Scaling Workflow:**

When applying feature scaling, it’s crucial to fit the scaler on the training data and then use the same scaler to transform the test data. This ensures consistency and prevents data leakage.

Furthermore, it’s beneficial to evaluate the impact of feature scaling on your specific machine learning task to determine which method works best.

**Handling Categorical Variables:**

Feature scaling is not typically applied to categorical variables, as their values represent different categories and do not have a numerical scale.

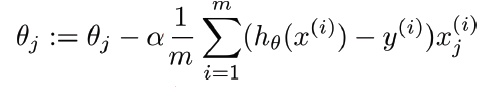
Categorical variables may require additional preprocessing techniques such as one-hot encoding before being used in machine learning algorithms.

**Dealing with Time-Series Data:**

For time-series data, feature scaling can be applied to each individual time series or across all time series, depending on the context. Consider the characteristics of your data and the requirements of your machine learning model to make an informed decision.

Gradient Descent Based Algorithms

Machine learning algorithms like [**linear regression**](https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/), [logistic regression](https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/), [neural network](https://www.analyticsvidhya.com/blog/2022/01/introduction-to-neural-networks/), PCA (principal component analysis), etc., that use gradient descent as an optimization technique require data to be scaled. Take a look at the formula for gradient descent below:

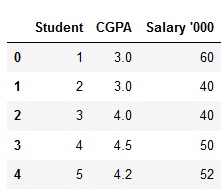
[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/gradient-descent.png)

The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in the ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model.

Distance-Based Algorithms

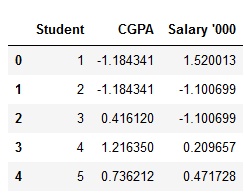
Distance algorithms like [KNN](https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), [K-means clustering](https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), and [SVM](https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/)(support vector machines) are most affected by the range of features. This is because, behind the scenes, **they are using distances between data points to determine their similarity.**

For example, let’s say we have data containing high school CGPA scores of students (ranging from 0 to 5) and their future incomes (in thousands Rupees):

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/knn_ex.png)

Since both the features have different scales, there is a chance that higher weightage is given to features with higher magnitudes. This will impact the performance of the machine learning algorithm; obviously, we do not want our algorithm to be biased towards one feature.

Therefore, we scale our data before employing a distance based algorithm so that all the features contribute equally to the result.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/knn_ex_scaled.png)

The effect of scaling is conspicuous when we compare the Euclidean distance between data points for students A and B, and between B and C, before and after scaling, as shown below:

* Distance AB before scaling =>Euclidean distance
* Distance BC before scaling =>Euclidean distance
* Distance AB after scaling =>Euclidean distance
* Distance BC after scaling =>Euclidean distance

Tree-Based Algorithms

[Tree-based algorithms](https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), on the other hand, are fairly insensitive to the scale of the features. Think about it, a decision tree only splits a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. Other features do not influence this split on a feature.

So, the remaining features have virtually no effect on the split. This is what makes them invariant to the scale of the features!

What is Normalization?

Normalization, a vital aspect of Feature Scaling, is a data preprocessing technique employed to standardize the values of features in a dataset, bringing them to a common scale. This process enhances data analysis and modeling accuracy by mitigating the influence of varying scales on machine learning models.

**Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.**

Here’s the formula for normalization:

[Normalization equation](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Norm_eq.gif)

Here, Xmax and Xmin are the maximum and the minimum values of the feature, respectively.

* When the value of X is the minimum value in the column, the numerator will be 0, and hence X’ is 0
* On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator, and thus the value of X’ is 1
* If the value of X is between the minimum and the maximum value, then the value of X’ is between 0 and 1

What is Standardization?

Standardization is another Feature scaling method where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.

Here’s the formula for standardization:

[Standardization equation](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Stand_eq.gif)  
Feature scaling: Muis the mean of the feature values andFeature scaling: Sigmais the standard deviation of the feature values. Note that, in this case, the values are not restricted to a particular range.

Now, the big question in your mind must be when should we use normalization and when should we use standardization? Let’s find out!

The Big Question – Normalize or Standardize?

| **Normalization** | **Standardization** |
| --- | --- |
| Rescales values to a range between 0 and 1 | Centers data around the mean and scales to a standard deviation of 1 |
| Useful when the distribution of the data is unknown or not Gaussian | Useful when the distribution of the data is Gaussian or unknown |
| Sensitive to outliers | Less sensitive to outliers |
| Retains the shape of the original distribution | Changes the shape of the original distribution |
| May not preserve the relationships between the data points | Preserves the relationships between the data points |
| Equation: (x – min)/(max – min) | Equation: (x – mean)/standard deviation |

However, at the end of the day, the choice of using normalization or standardization will depend on your problem and the machine learning algorithm you are using. There is no hard and fast rule to tell you when to normalize or standardize your data. **You can always start by fitting your model to raw, normalized, and standardized data and comparing the performance for the best results.**

*It is a good practice to fit the scaler on the training data and then use it to transform the testing data. This would avoid any data leakage during the model testing process. Also, the scaling of target values is generally not required.*

Implementing Feature Scaling in Python

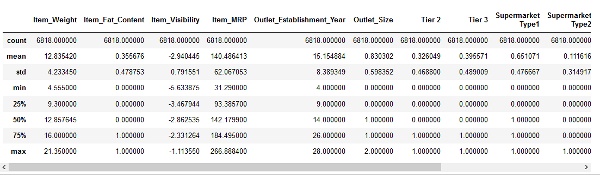
Now comes the fun part – putting what we have learned into practice. I will be applying feature scaling to a few machine-learning algorithms on the Big Mart dataset. I’ve taken on the DataHack platform.

I will skip the preprocessing steps since they are out of the scope of this tutorial. But you can find them neatly explained in this [article](https://www.analyticsvidhya.com/blog/2016/02/bigmart-sales-solution-top-20/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization). Those steps will enable you to reach the top 20 percentile on the hackathon leaderboard, so that’s worth checking out!

So, let’s first split our data into training and testing sets:

**Python Code:**

Before moving to the feature scaling part, let’s glance at the details of our data using the **pd.describe()** method:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/NormVsStand_1.png)

We can see that there is a huge difference in the range of values present in our numerical features: **Item\_Visibility**, **Item\_Weight, Item\_MRP,**and **Outlet\_Establishment\_Year**. Let’s try and fix that using feature scaling!

*Note: You will notice negative values in the Item\_Visibility feature because I have taken log-transformation to deal with the skewness in the feature.*

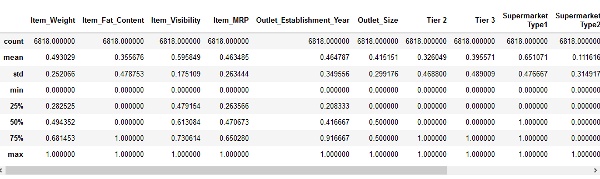
Normalization Using sklearn (scikit-learn)

To normalize your data, you need to import the MinMaxScaler from the [sklearn](https://courses.analyticsvidhya.com/courses/get-started-with-scikit-learn-sklearn?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization" \t "_blank) library and apply it to our dataset. So, let’s do that!

|  |  |
| --- | --- |
|  | # data normalization with sklearn |
|  | from sklearn.preprocessing import MinMaxScaler |
|  |  |
|  | # fit scaler on training data |
|  | norm = MinMaxScaler().fit(X\_train) |
|  |  |
|  | # transform training data |
|  | X\_train\_norm = norm.transform(X\_train) |
|  |  |
|  | # transform testing dataabs |
|  | X\_test\_norm = norm.transform(X\_test) |

[view raw](https://gist.github.com/aniruddha27/a41a35725ec02006bb3156e5483cb184/raw/5a15a3fbae1a6967171185127c458674cd021f22/NormalizationVsStandarization_2.py)[NormalizationVsStandarization\_2.py](https://gist.github.com/aniruddha27/a41a35725ec02006bb3156e5483cb184#file-normalizationvsstandarization_2-py)hosted with ❤ by [GitHub](https://github.com/)

Let’s see how normalization has affected our dataset:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/NormVsStand_2.png)

All the features now have a minimum value of 0 and a maximum value of 1. Perfect!

Try out the above code in the live coding window below!!

Next, let’s try to standardize our data.

Standardization Using sklearn

To standardize your data, you need to import the StandardScaler from the sklearn library and apply it to our dataset. Here’s how you can do it:

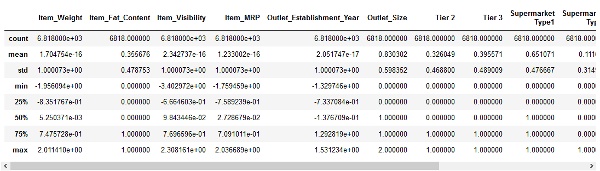
|  |  |
| --- | --- |
|  | # data standardization with sklearn |
|  | from sklearn.preprocessing import StandardScaler |
|  |  |
|  | # copy of datasets |
|  | X\_train\_stand = X\_train.copy() |
|  | X\_test\_stand = X\_test.copy() |
|  |  |
|  | # numerical features |
|  | num\_cols = ['Item\_Weight','Item\_Visibility','Item\_MRP','Outlet\_Establishment\_Year'] |
|  |  |
|  | # apply standardization on numerical features |
|  | for i in num\_cols: |
|  |  |
|  | # fit on training data column |
|  | scale = StandardScaler().fit(X\_train\_stand[[i]]) |
|  |  |
|  | # transform the training data column |
|  | X\_train\_stand[i] = scale.transform(X\_train\_stand[[i]]) |
|  |  |
|  | # transform the testing data column |
|  | X\_test\_stand[i] = scale.transform(X\_test\_stand[[i]]) |

[view raw](https://gist.github.com/aniruddha27/965ff8b01e19de1cffdb5cbe703d5495/raw/948e4d05a4e65291ff54f6986f552ea398af89de/NormalizationVsStandarization_3.py)[NormalizationVsStandarization\_3.py](https://gist.github.com/aniruddha27/965ff8b01e19de1cffdb5cbe703d5495#file-normalizationvsstandarization_3-py)hosted with ❤ by [GitHub](https://github.com/)

You would have noticed that I only applied standardization to my numerical columns, not the other [One-Hot Encoded](https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization) features. Standardizing the One-Hot encoded features would mean assigning a distribution to categorical features. You don’t want to do that!

But why did I not do the same while normalizing the data? Because One-Hot encoded features are already in the range between 0 to 1. So, normalization would not affect their value.

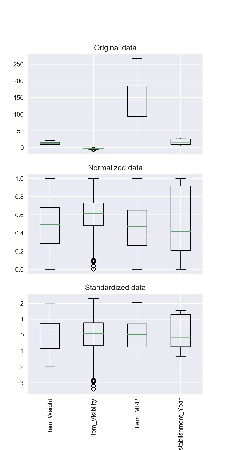
Right, let’s have a look at how standardization has transformed our data:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/NormVsStand_3.png)

The numerical features are now centered on the mean with a unit standard deviation. Awesome!

Comparing Unscaled, Normalized, and Standardized Data

It is always great to visualize your data to understand the distribution present. We can see the comparison between our unscaled and scaled data using boxplots.



You can notice how scaling the features brings everything into perspective. The features are now more comparable and will have a similar effect on the learning models.

Applying Scaling to Machine Learning Algorithms

It’s now time to train some machine learning algorithms on our data to compare the effects of different Feature scaling techniques on the algorithm’s performance. I want to see the effect of scaling on three algorithms in particular: K-Nearest Neighbors, Support Vector Regressor, and Decision Tree.

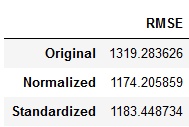
Now, let’s delve into training machine learning algorithms on our dataset to assess the impact of various scaling techniques on their performance. Specifically, I aim to observe the effects of scaling on three key algorithms: K-Nearest Neighbors, Support Vector Regressor, and Decision Tree. This analysis will provide valuable insights into the significance of feature scaling in machine learning and how it influences the outcomes of these algorithms.

K-Nearest Neighbors

As we saw before, KNN is a distance-based algorithm that is affected by the range of features. Let’s see how it performs on our data before and after scaling:

|  |  |
| --- | --- |
|  | # training a KNN model |
|  | from sklearn.neighbors import KNeighborsRegressor |
|  | # measuring RMSE score |
|  | from sklearn.metrics import mean\_squared\_error |
|  |  |
|  | # knn |
|  | knn = KNeighborsRegressor(n\_neighbors=7) |
|  |  |
|  | rmse = [] |
|  |  |
|  | # raw, normalized and standardized training and testing data |
|  | trainX = [X\_train, X\_train\_norm, X\_train\_stand] |
|  | testX = [X\_test, X\_test\_norm, X\_test\_stand] |
|  |  |
|  | # model fitting and measuring RMSE |
|  | for i in range(len(trainX)): |
|  |  |
|  | # fit |
|  | knn.fit(trainX[i],y\_train) |
|  | # predict |
|  | pred = knn.predict(testX[i]) |
|  | # RMSE |
|  | rmse.append(np.sqrt(mean\_squared\_error(y\_test,pred))) |
|  |  |
|  | # visualizing the result |
|  | df\_knn = pd.DataFrame({'RMSE':rmse},index=['Original','Normalized','Standardized']) |
|  | df\_knn |

[view raw](https://gist.github.com/aniruddha27/66119a2050fc808d2bdb7d4544ae75b6/raw/f7d5d7854dfc734b910aefc9513ab7e3140c175e/NormalizationVsStandarization_4.py)[NormalizationVsStandarization\_4.py](https://gist.github.com/aniruddha27/66119a2050fc808d2bdb7d4544ae75b6#file-normalizationvsstandarization_4-py)hosted with ❤ by [GitHub](https://github.com/)

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/NormVsStand_knn.png)

You can see that scaling the features has brought down the RMSE score of our KNN model. Specifically, the normalized data performs a tad bit better than the standardized data.

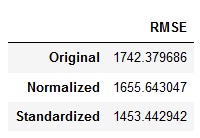
*Note: I am measuring the RMSE here because this competition evaluates the RMSE.*

Support Vector Regressor

[SVR](https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization) is another distance-based algorithm. So let’s check out whether it works better with normalization or standardization:

|  |  |
| --- | --- |
|  | # training an SVR model |
|  | from sklearn.svm import SVR |
|  | # measuring RMSE score |
|  | from sklearn.metrics import mean\_squared\_error |
|  |  |
|  | # SVR |
|  | svr = SVR(kernel='rbf',C=5) |
|  |  |
|  | rmse = [] |
|  |  |
|  | # raw, normalized and standardized training and testing data |
|  | trainX = [X\_train, X\_train\_norm, X\_train\_stand] |
|  | testX = [X\_test, X\_test\_norm, X\_test\_stand] |
|  |  |
|  | # model fitting and measuring RMSE |
|  | for i in range(len(trainX)): |
|  |  |
|  | # fit |
|  | svr.fit(trainX[i],y\_train) |
|  | # predict |
|  | pred = svr.predict(testX[i]) |
|  | # RMSE |
|  | rmse.append(np.sqrt(mean\_squared\_error(y\_test,pred))) |
|  |  |
|  | # visualizing the result |
|  | df\_svr = pd.DataFrame({'RMSE':rmse},index=['Original','Normalized','Standardized']) |
|  | df\_svr |

[view raw](https://gist.github.com/aniruddha27/a49f58527ef006d7b58cae03ba59e9df/raw/00949b28470a33a67eca94629e52f586cacab13d/NormalizationVsStandarization_5.py)[NormalizationVsStandarization\_5.py](https://gist.github.com/aniruddha27/a49f58527ef006d7b58cae03ba59e9df#file-normalizationvsstandarization_5-py)hosted with ❤ by [GitHub](https://github.com/)

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/NormVsStand_svr.png)

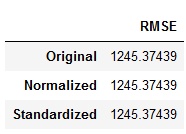
We can see that scaling the features does bring down the RMSE score. And the standardized data has performed better than the normalized data. Why do you think that’s the case?

The [sklearn documentation](https://scikit-learn.org/stable/modules/preprocessing.html" \l "standardization-or-mean-removal-and-variance-scaling" \t "_blank) states that SVM, with RBF kernel,  assumes that all the features are centered around zero and variance is of the same order. This is because a feature with a variance greater than that of others prevents the estimator from learning from all the features. Great!

Decision Tree

We already know that a [Decision tree](https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/) is invariant to feature scaling. But I wanted to show a practical example of how it performs on the data:

|  |  |
| --- | --- |
|  | # training a Decision Tree model |
|  | from sklearn.tree import DecisionTreeRegressor |
|  | # measuring RMSE score |
|  | from sklearn.metrics import mean\_squared\_error |
|  |  |
|  | # Decision tree |
|  | dt = DecisionTreeRegressor(max\_depth=10,random\_state=27) |
|  |  |
|  | rmse = [] |
|  |  |
|  | # raw, normalized and standardized training and testing data |
|  | trainX = [X\_train,X\_train\_norm,X\_train\_stand] |
|  | testX = [X\_test,X\_test\_norm,X\_test\_stand] |
|  |  |
|  | # model fitting and measuring RMSE |
|  | for i in range(len(trainX)): |
|  |  |
|  | # fit |
|  | dt.fit(trainX[i],y\_train) |
|  | # predict |
|  | pred = dt.predict(testX[i]) |
|  | # RMSE |
|  | rmse.append(np.sqrt(mean\_squared\_error(y\_test,pred))) |
|  |  |
|  | # visualizing the result |
|  | df\_dt = pd.DataFrame({'RMSE':rmse},index=['Original','Normalized','Standardized']) |
|  | df\_dt |

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/NormVsStand_dt.png)

**Methods of Performing Feature Scaling**

Two common methods of feature scaling are

* Normalisation (Min-Max Scaling)
* Standardization (Standard Scaling)

**Normalization**

Normalization, also known as min-max scaling is a data preprocessing technique used to adjust the values of features in a dataset to a common scale.

When normalizing features, values are shited and rescaled so that they end up ranging from 0 to 1.

When the relationship between the range of your features are meaningful and you want to preserve this information normalization is a better choice

# import MinMaxScaler  
from sklearn.preprocessing import MinMaxScaler  
  
# fit scaler on training data  
norm = MinMaxScaler().fit(numerical\_df\_train)  
  
# transfrom training data  
df\_train\_norm = norm.transform(numerical\_df\_train)  
  
# transform testing data  
df\_test\_norm = norm.transform(numerical\_df\_test)

**Standardization**

Standardization is another scaling method where the values are centred around the mean with a unit standard deviation. This means that the mean of your feature becomes zero and the resultant distribution has a unit standard deviation.

Standardization is a good choice for features that have a gaussian distribution.

# import StandardScaler  
from sklearn.preprocessing import StandardScaler  
  
# fit scaler on training data  
scale = StandardScaler().fit(df\_train\_stand)  
  
# transform training data  
train\_stand = scale.transform(df\_train\_stand)  
  
# transform testing data  
test\_stand = scale.transform(df\_test\_stand)

It is good practice to fit the scales on the training data and then use it to transform testing data. This would avoid any leakage during the model testing process.

Here are some differences between Normalization and Standerdization

Normalization | Standardization  
 -------------------------- -----------------------------  
Rescales values to range | Centres data around the mean   
between 0 and 1 | and scales to a standard deviation of one  
  
Less sensitive to Outliers | Sensitive to outliers  
  
Preserves the relationship | May not preserve relationship between  
between data points | data points

***Note:****Scaling target values are generally not required.*

**Considerations:**

* Always consider the requirements and assumptions of the specific algorithm you are using. Some algorithms may perform well with either standardizationon or normalization, while others may be sensitive to the choice.
* Experiment with both standardization and normalization, and observe the impact on your model’s performance through techniques like cross-validation.
* Pay attention to outliers, as standardization can be sensitive to extreme values. In the presence of outliers, robust scalers or other data preprocessing techniques may be considered.

Some features in your dataset can be normalized while some standardized, it depends on the nature of the feature which you will have to identify and choose what scaling method to apply.

**Feature Scaling unsing sklearn (scikit-learn)**

Feature Scaling is a technique used to standardize the range of features in a dataset, transforming your data to make it more optimized for modelling.

Real-world datasets often contain features that vary in degrees of magnitude, range and units, ie: weight - 197 pounds, distance\_ran - 4 miles, without scaling, your model will think the feature “weight” is generally larger than “distance\_ran” and would give more emphasis to “weight” even tho “weight” and “distance\_ran” are just separate unit of measurements with different magnitudes.

For machine learning models to interpret these features on the same scale we need to perform feature scaling, the purpose is to ensure that all features contribute equally to the model and to avoid the dominance of features with larger values.

And the variations in feature values can lead to biased model performance or difficulties during the learning process.

It is good practice to fit the scales on the training data and then use it to transform testing data. This would avoid any leakage during the model testing process. Scaling target values are generally not required.

In [1]:

**import** pandas **as** pd

**from** sklearn.preprocessing **import** MinMaxScaler

In [2]:

df\_test **=** pd**.**read\_csv(r"C:/Users/user/Documents/datasets/big\_mart\_test.csv")

df\_train **=** pd**.**read\_csv(r"C:/Users/user/Documents/datasets/big\_mart\_train.csv")

In [12]:

df\_train**.**columns

Out[12]:

Index(['Item\_Identifier', 'Item\_Weight', 'Item\_Fat\_Content', 'Item\_Visibility',

'Item\_Type', 'Item\_MRP', 'Outlet\_Identifier',

'Outlet\_Establishment\_Year', 'Outlet\_Size', 'Outlet\_Location\_Type',

'Outlet\_Type', 'Item\_Outlet\_Sales'],

dtype='object')

In [15]:

*# seprating numerical columns to scale*

numerical\_df\_train **=** df\_train[['Item\_Weight', 'Item\_Visibility', 'Item\_MRP', 'Outlet\_Establishment\_Year']]

numerical\_df\_test **=** df\_test[['Item\_Weight', 'Item\_Visibility', 'Item\_MRP', 'Outlet\_Establishment\_Year']]

**Unscaled Data**

In [67]:

*# Set up the matplotlib figure with a 2x2 grid*

fig, axes **=** plt**.**subplots(nrows**=**2, ncols**=**2, figsize**=**(10, 8))

*# Flatten the axes array for easier iteration*

axes **=** axes**.**flatten()

*# Iterate over each column and plot a histogram*

**for** i, column **in** enumerate(numerical\_df\_train):

sns**.**histplot(numerical\_df\_train[column], ax**=**axes[i], bins**=**20, kde**=False**)

axes[i]**.**set\_title(f'Histogram of {column}')

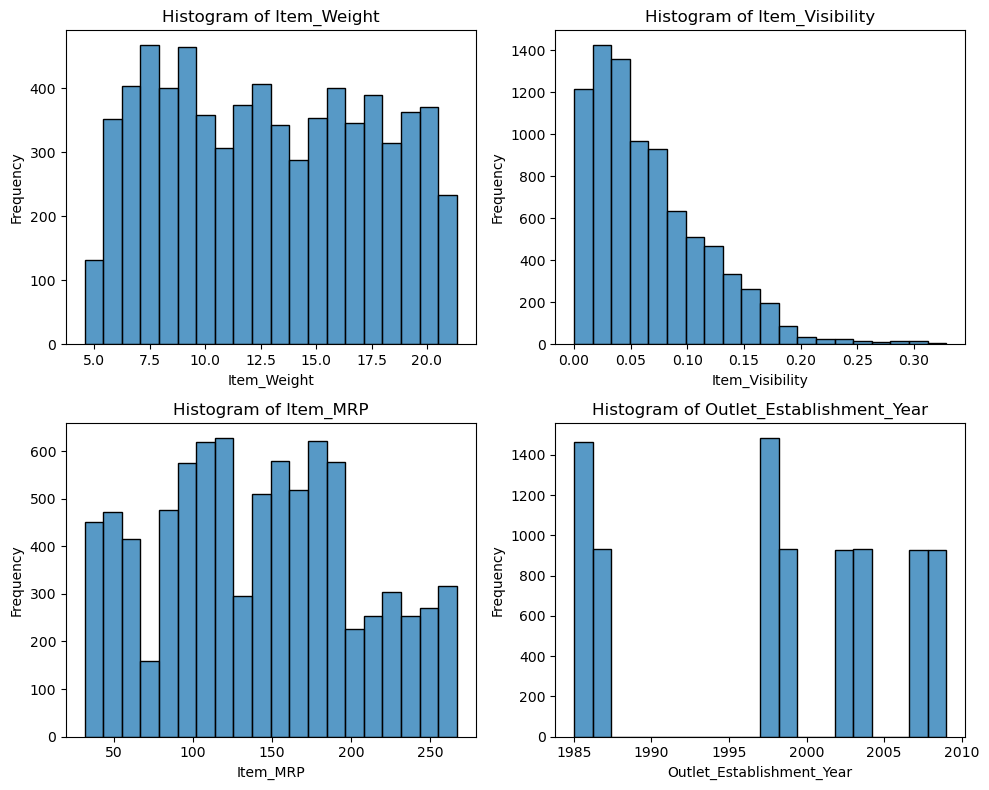
axes[i]**.**set\_xlabel(column)

axes[i]**.**set\_ylabel('Frequency')

*# Adjust layout and show the plot*

plt**.**tight\_layout()

plt**.**show()



In [ ]:

*# Loop through each numerical column and create a separate histogram plot*

**for** column **in** numerical\_df\_train**.**select\_dtypes(include**=**'number'):

plt**.**figure(figsize**=**(6, 4)) *# Adjust the figure size as needed*

sns**.**histplot(numerical\_df\_train[column], bins**=**30, kde**=False**)

*# save plt as img*

plt**.**savefig(f"{column}\_without\_scaling.png")

plt**.**show()

**Normalization using MinMaxScaler**

Normalization is a data preprocessing technique used to adjust the values of features in a dataset to a common scale. When normalizing features, values are shited and rescaled so tht they end up ranging from 0 to 1. Normalization is also known as min-max scaling.

In [16]:

*# fit scaler on training data*

norm **=** MinMaxScaler()**.**fit(numerical\_df\_train)

*# transfrom training data*

df\_train\_norm **=** norm**.**transform(numerical\_df\_train)

*# transform testing data*

df\_test\_norm **=** norm**.**transform(numerical\_df\_test)

In [18]:

df\_train\_norm **=** pd**.**DataFrame(df\_train\_norm)

df\_test\_norm **=** pd**.**DataFrame(df\_test\_norm)

In [25]:

df\_test\_norm**.**head()

Out[25]:

|  | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| **0** | 0.964275 | 0.023036 | 0.325012 | 0.583333 |
| **1** | 0.222983 | 0.117018 | 0.237819 | 0.916667 |
| **2** | 0.598095 | 0.303221 | 0.893316 | 0.541667 |
| **3** | 0.164335 | 0.046860 | 0.525233 | 0.916667 |
| **4** | NaN | 0.361153 | 0.861381 | 0.000000 |

In [35]:

*# Rename the columns*

new\_column\_names **=** {0: 'Item\_Weight',

1: 'Item\_Visibility',

2: 'Item\_MRP',

3: 'Outlet\_Establishment\_Year'}

df\_train\_norm**.**rename(columns**=**new\_column\_names, inplace**=True**)

df\_test\_norm**.**rename(columns**=**new\_column\_names, inplace**=True**)

In [21]:

numerical\_df\_train**.**head()

Out[21]:

|  | **Item\_Weight** | **Item\_Visibility** | **Item\_MRP** | **Outlet\_Establishment\_Year** |
| --- | --- | --- | --- | --- |
| **0** | 9.30 | 0.016047 | 249.8092 | 1999 |
| **1** | 5.92 | 0.019278 | 48.2692 | 2009 |
| **2** | 17.50 | 0.016760 | 141.6180 | 1999 |
| **3** | 19.20 | 0.000000 | 182.0950 | 1998 |
| **4** | 8.93 | 0.000000 | 53.8614 | 1987 |

In [36]:

df\_train\_norm**.**columns

Out[36]:

Index(['Item\_Weight', 'Item\_Visibility', 'Item\_MRP',

'Outlet\_Establishment\_Year'],

dtype='object')

In [38]:

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

In [59]:

*# Set up the matplotlib figure with a 2x2 grid*

fig, axes **=** plt**.**subplots(nrows**=**2, ncols**=**2, figsize**=**(10, 8))

*# Flatten the axes array for easier iteration*

axes **=** axes**.**flatten()

*# Iterate over each column and plot a histogram*

**for** i, column **in** enumerate(df\_train\_norm):

sns**.**histplot(df\_train\_norm[column], ax**=**axes[i], bins**=**20, kde**=False**)

axes[i]**.**set\_title(f'Histogram of {column}')

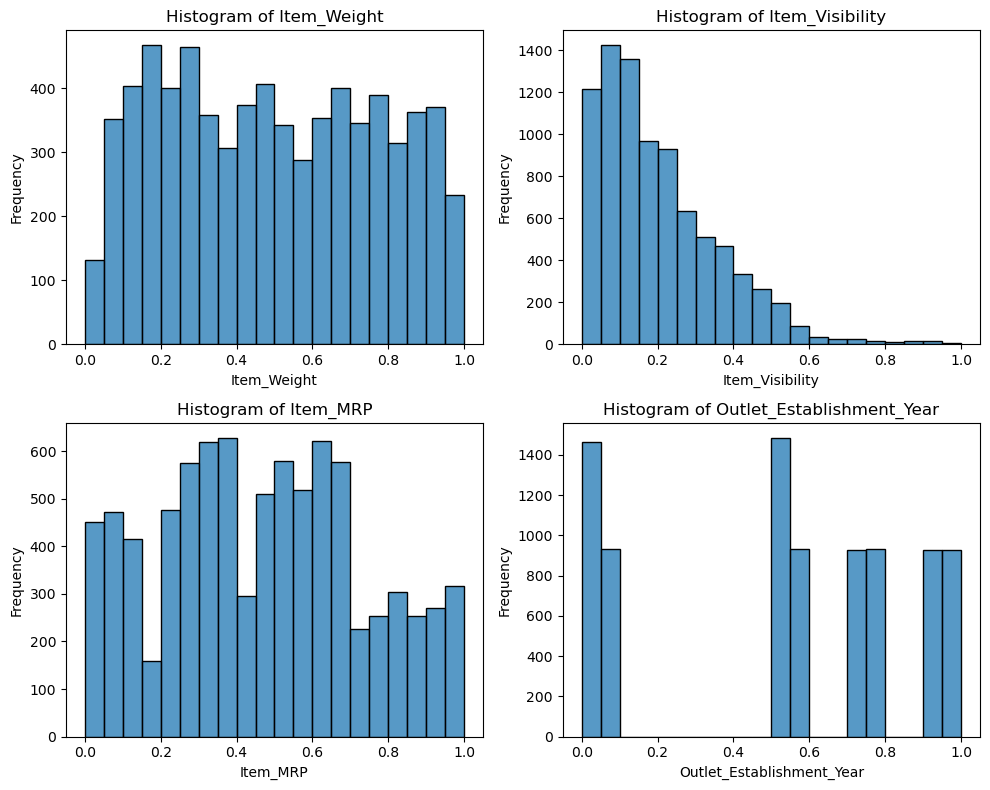
axes[i]**.**set\_xlabel(column)

axes[i]**.**set\_ylabel('Frequency')

*# Adjust layout and show the plot*

plt**.**tight\_layout()

plt**.**show()



In [ ]:

*# Loop through each numerical column and create a separate histogram plot*

**for** column **in** df\_train\_norm**.**select\_dtypes(include**=**'number'):

plt**.**figure(figsize**=**(6, 4)) *# Adjust the figure size as needed*

sns**.**histplot(df\_train\_norm[column], kde**=False**)

*# Save the plot as an image file (e.g., PNG)*

plt**.**savefig(f"{column}\_normalization.png")

plt**.**show()

**Standardization**

Standardization is another scaling method where the values are centred around the mean with a unit standard deviation. This means that the mean of your feature becomes zero and the resultant distribution has a unit standard deviation

In [44]:

**from** sklearn.preprocessing **import** StandardScaler

In [45]:

df\_train\_stand **=** numerical\_df\_train**.**copy()

df\_test\_stand **=** numerical\_df\_test**.**copy()

In [46]:

*# fit scaler on training data*

scale **=** StandardScaler()**.**fit(df\_train\_stand)

*# transform training data*

train\_stand **=** scale**.**transform(df\_train\_stand)

*# transform testing data*

test\_stand **=** scale**.**transform(df\_test\_stand)

In [48]:

*# convert to Dataframe*

train\_stand **=** pd**.**DataFrame(train\_stand)

test\_stand **=** pd**.**DataFrame(test\_stand)

In [50]:

*# Rename the columns*

new\_column\_names **=** {0: 'Item\_Weight',

1: 'Item\_Visibility',

2: 'Item\_MRP',

3: 'Outlet\_Establishment\_Year'}

train\_stand**.**rename(columns**=**new\_column\_names, inplace**=True**)

test\_stand**.**rename(columns**=**new\_column\_names, inplace**=True**)

In [51]:

train\_stand**.**head()

Out[51]:

|  | **Item\_Weight** | **Item\_Visibility** | **Item\_MRP** | **Outlet\_Establishment\_Year** |
| --- | --- | --- | --- | --- |
| **0** | -0.766217 | -0.970732 | 1.747454 | 0.139541 |
| **1** | -1.494175 | -0.908111 | -1.489023 | 1.334103 |
| **2** | 0.999834 | -0.956917 | 0.010040 | 0.139541 |
| **3** | 1.365966 | -1.281758 | 0.660050 | 0.020085 |
| **4** | -0.845905 | -1.281758 | -1.399220 | -1.293934 |

In [61]:

*# Set up the matplotlib figure with a 2x2 grid*

fig, axes **=** plt**.**subplots(nrows**=**2, ncols**=**2, figsize**=**(10, 8))

*# Flatten the axes array for easier iteration*

axes **=** axes**.**flatten()

*# Iterate over each column and plot a histogram*

**for** i, column **in** enumerate(train\_stand):

sns**.**histplot(train\_stand[column], ax**=**axes[i], bins**=**20, kde**=False**)

axes[i]**.**set\_title(f'Histogram of {column}')

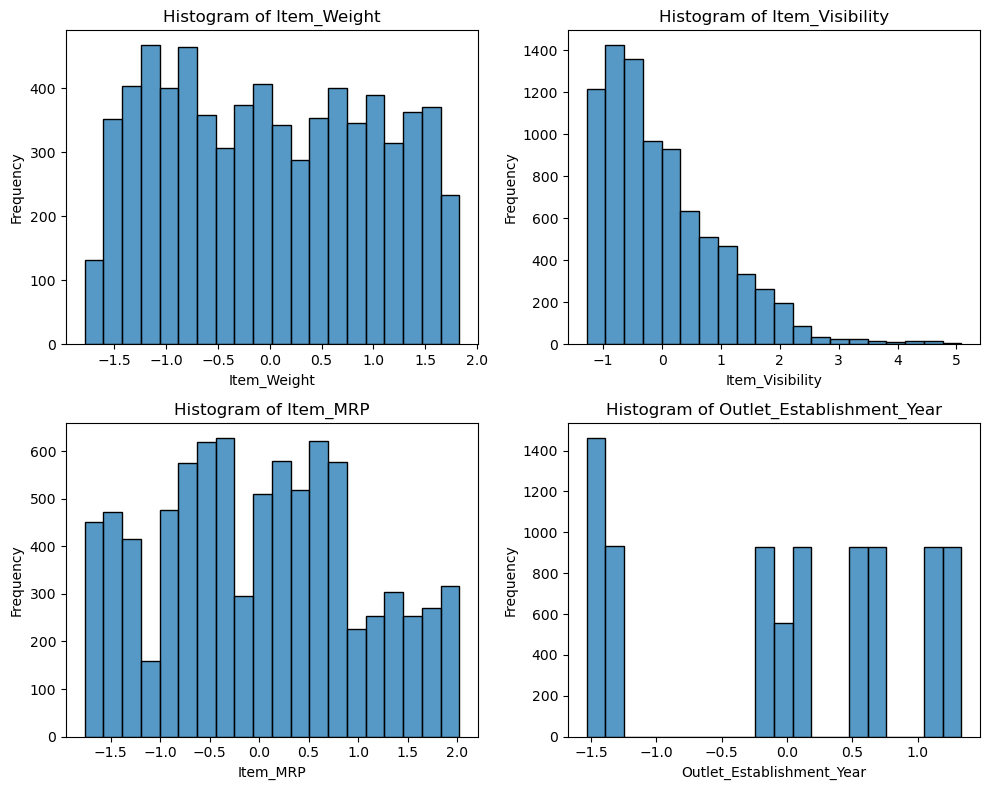
axes[i]**.**set\_xlabel(column)

axes[i]**.**set\_ylabel('Frequency')

*# Adjust layout and show the plot*

plt**.**tight\_layout()

plt**.**show()



In [ ]:

*# Loop through each numerical column and create a separate histogram plot*

**for** column **in** train\_stand**.**select\_dtypes(include**=**'number'):

plt**.**figure(figsize**=**(6, 4)) *# Adjust the figure size as needed*

sns**.**histplot(train\_stand[column], kde**=False**)

*# save plot as img*

plt**.**savefig(f"{column}\_standerdized.png")

**Conclusion**

Some features in your dataset can be normalized while some standardized, it depends on the nature of the feature which you will have to identify and choose what scaling method to apply.