Preprocessing and Scaling

What is Feature Scaling

- Feature scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale.

- The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values.

- Feature scaling becomes necessary when dealing with datasets containing features that have different ranges, units of measurement, or orders of magnitude. In such cases, the variation in feature values can lead to biased model performance or difficulties during the learning process.

- There are several common techniques for feature scaling, including **standardization, normalization, and min-max scaling.** These methods adjust the feature values while preserving their relative relationships and distributions.

Feature Scaling

- Feature Scaling is a critical step in building accurate and effective ML models.
- One key aspect of feature engineering is scaling, normalization, and standardization, which involves transforming the data to make it more suitable for modeling.
- These techniques can help to improve model performance, reduce the impact of outliers, and ensure that the data is on the same scale.

Feature Scaling

- By applying feature scaling, the dataset's features can be transformed to a more consistent scale, making it easier to build accurate and effective machine learning models.

- Scaling facilitates meaningful comparisons between features, improves model convergence, and prevents certain features from overshadowing others based solely on their magnitude.

Why Should we Use Feature Scaling?

Some machine learning algorithms are sensitive to feature scaling, while others are virtually invariant.

1. Gradient Descent Based Algorithms

Machine learning algorithms like linear regression, logistic regression, neural network, PCA (principal component analysis), etc., that use gradient descent as an optimization technique require data to be scaled.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Why Should we Use Feature Scaling?

- 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.
- Having features on a similar scale can help the gradient descent converge more quickly towards the minima.

Why Should we Use Feature Scaling?

2. **Distance-Based Algorithms**

Distance algorithms like KNN, K-means clustering, and SVM(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, a dataset containing high school CGPA scores of students (ranging from 0 to 5) and their future incomes (in thousands Rupees):

Why Should we Use Feature Scaling?

	Student	CGPA	Salary '000
0	1	3.0	60
1	2	3.0	40
2	3	4.0	40
3	4	4.5	50
4	5	4.2	52

- 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;
- We do not want our algorithm to be biased towards one feature.
- Therefore, scale data before employing a distance based algorithm so that all the features contribute equally to the result.

Why Should we Use Feature Scaling?

	Student	CGPA	Salary '000
0	1	-1.184341	1.520013
1	2	-1.184341	-1.100699
2	3	0.416120	-1.100699
3	4	1.216350	0.209657

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 $=> \sqrt{(40-60)^2+(3-3)^2} = 20$
- Distance BC before scaling $=> \sqrt{(40-40)^2+(4-3)^2} = 1$
- Distance AB after scaling $=> \sqrt{(1.1+1.5)^2+(1.18-1.18)^2} = 2.6$
- Distance BC after scaling $=> \sqrt{(1.1-1.1)^2+(0.41+1.18)^2} = 1.59$

Why Should we Use Feature Scaling?

3. Tree-Based Algorithms

- Tree-based algorithms, on the other hand, are fairly insensitive to the scale of the features.

- 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**.

What is Normalization?

The formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

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:

$$X' = \frac{X - \mu}{\sigma}$$

μ is the mean of the feature values and σ is the standard deviation of the feature values. Note that, in this case, the values are not restricted to a particular range.

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

When should we use normalization and when should we use standardization?

- The choice of using normalization or standardization will depend on our problem and the machine learning algorithm we are using.
- There is no hard and fast rule to tell when to normalize or standardize the data.
- We 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.

```
import pandas as pd
# spliting training and testing data
from sklearn.model_selection import train_test_split

X = df
y = target


X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.2,random_state=27)
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermarket Type2
count	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000
mean	12.835420	0.355676	-2.940445	140.486413	15.154884	0.830302	0.326049	0.395571	0.651071	0.111616
std	4.233450	0.478753	0.791551	62.067053	8.389349	0.598352	0.468800	0.489009	0.476667	0.314917
min	4.555000	0.000000	-5.633875	31.290000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	9.300000	0.000000	-3.467944	93.385700	9.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	12.857645	0.000000	-2.862535	142.179900	14.000000	1.000000	0.000000	0.000000	1.000000	0.000000
75%	16.000000	1.000000	-2.331264	184.495000	26.000000	1.000000	1.000000	1.000000	1.000000	0.000000
max	21.350000	1.000000	-1.113550	266.888400	28.000000	2.000000	1.000000	1.000000	1.000000	1.000000

We can see that there is a huge difference in the range of values present in numerical features:
Item_Visibility, **Item_Weight, Item_MRP,** and **Outlet_Establishment_Year**

Fix it using feature scaling

Normalization Using sklearn (scikit-learn)

To normalize the data, import the MinMaxScaler from the sklearn library and apply it to the dataset.

```
# 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)
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermarket Type2
count	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000
mean	0.493029	0.355676	0.595849	0.463485	0.464787	0.415151	0.326049	0.395571	0.651071	0.111616
std	0.252066	0.478753	0.175109	0.263444	0.349556	0.299176	0.468800	0.489009	0.476667	0.314917
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.282525	0.000000	0.479154	0.263566	0.208333	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.494352	0.000000	0.613084	0.470673	0.416667	0.500000	0.000000	0.000000	1.000000	0.000000
75%	0.681453	1.000000	0.730614	0.650280	0.916667	0.500000	1.000000	1.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

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

Standardization Using sklearn

To standardize the data,
import the StandardScaler from the sklearn library and apply it to the dataset.

```
# 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]])
```

Observation

- Apply standardization to numerical columns only, not the other One-Hot Encoded features.

- Standardizing the One-Hot encoded features would mean assigning a distribution to categorical features. We don't want to do that.

Why did we 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.

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermar Typ
count	6.818000e+03	6818.000000	6.818000e+03	6.818000e+03	6.818000e+03	6818.000000	6818.000000	6818.000000	6818.000000	6818.000
mean	1.704754e-16	0.355676	2.342737e-16	1.233002e-16	2.051747e-17	0.830302	0.326049	0.395571	0.651071	0.111
std	1.000073e+00	0.478753	1.000073e+00	1.000073e+00	1.000073e+00	0.598352	0.468800	0.489009	0.476667	0.314
min	-1.956094e+00	0.000000	-3.402972e+00	-1.759459e+00	-1.329746e+00	0.000000	0.000000	0.000000	0.000000	0.000
25%	-8.351767e-01	0.000000	-6.664603e-01	-7.589239e-01	-7.337084e-01	0.000000	0.000000	0.000000	0.000000	0.000
50%	5.250371e-03	0.000000	9.843446e-02	2.728679e-02	-1.376709e-01	1.000000	0.000000	0.000000	1.000000	0.000
75%	7.475728e-01	1.000000	7.696596e-01	7.091011e-01	1.292819e+00	1.000000	1.000000	1.000000	1.000000	0.000
max	2.011410e+00	1.000000	2.308161e+00	2.036689e+00	1.531234e+00	2.000000	1.000000	1.000000	1.000000	1.000

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