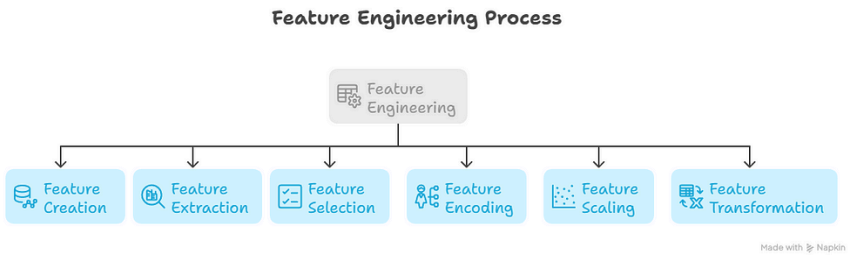
Feature Transformation in Machine Learning: A Key Component of Feature Engineering

Introduction to Feature Engineering

Feature engineering is the process of transforming raw data into meaningful features that improve the performance of machine learning models. The goal is to create features that help algorithms learn better and make more accurate predictions.

Feature engineering includes several tasks, such as:

1. **Feature Creation**: Generating new features from existing data.
2. **Feature Extraction**: Extracting useful information from complex data (like images, text, or time series).
3. **Feature Selection**: Identifying the most important features and removing irrelevant or redundant ones.
4. **Feature Encoding**: Converting categorical variables (like “Male”, “Female”) into numerical formats.
5. **Feature Scaling**: Normalizing all features to the same scale for better comparison.
6. **Feature Transformation**: Applying mathematical techniques to adjust the data distribution, helping the model perform better.



1. Feature Creation

Feature creation involves generating new features from the existing ones. These new features can give your machine learning model better insights.

Example:

Imagine you have a dataset with two features: **height** and **weight**. You can create a new feature **BMI** (Body Mass Index) to make your data more informative.

Formula for BMI:

$$BMI = \frac{\text{weight}}{\text{height}^2}$$

- If the height is 1.75 meters and weight is 70 kg, BMI = 70 / (1.75 * 1.75) = **22.86**

```
Before Feature Creation (Original Data):
   Height  Weight
0    1.75      70
1    1.82      80
2    1.60      50
3    1.68      65
4    1.90      95

After Feature Creation (With BMI):
   Height  Weight        BMI
0    1.75      70  22.857143
1    1.82      80  24.151673
2    1.60      50  19.531250
3    1.68      65  23.030045
4    1.90      95  26.315789
```

2. Feature Extraction

Feature extraction is the process of transforming raw, complex data (like images or text) into simpler, more usable features.

Example:

- **Text Data:** Imagine you have a paragraph of text. You can extract important keywords or phrases to make the text easier to analyze.
- **Text:** "Machine learning helps computers learn from data."
  Keywords: "Machine learning", "computers", "data".

```
Before Feature Extraction (Raw Text):
['Machine learning helps computers learn from data.']

After Feature Extraction (Extracted Keywords):
['computers' 'data' 'helps' 'learn' 'learning' 'machine']
```

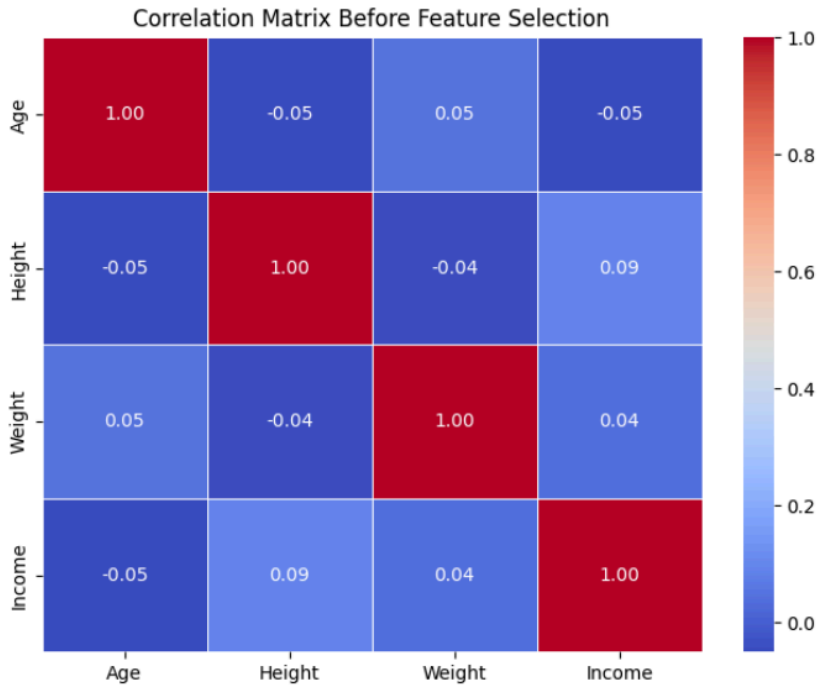
- **Image Data:** For an image, you can extract edges, shapes, or patterns to identify important features.

3. Feature Selection

Feature selection is about choosing only the most important features and removing the irrelevant ones. This helps make your model more efficient and less prone to overfitting.

Example:

You have a dataset with many features, such as **age**, **height**, **weight**, and **income**. By using techniques like **correlation matrices**, you can find that **age** and **income** are highly correlated. You might choose to keep only **age** to reduce redundancy.




```
Dataframe After Feature Selection (Removed 'Income'):
   Age  Height  Weight
0   56     158      67
1   46     157      93
2   32     161      62
3   25     183      76
4   38     182      71
```

4. Feature Encoding

Feature encoding is used to convert categorical data (like names or colors) into numerical format so that machine learning models can understand them.

Example:

- You have a column with colors: **Red**, **Green**, **Blue**.
- You can assign numbers to these colors using **Label Encoding**:
- Red = 0
- Green = 1
- Blue = 2

Now, these colors are converted into numbers that models can work with.

```
Original and Encoded Data:
   Color  Color_encoded
0    Red              2
1  Green              1
2   Blue              0
3  Green              1
4   Blue              0
5    Red              2
6  Green              1
```

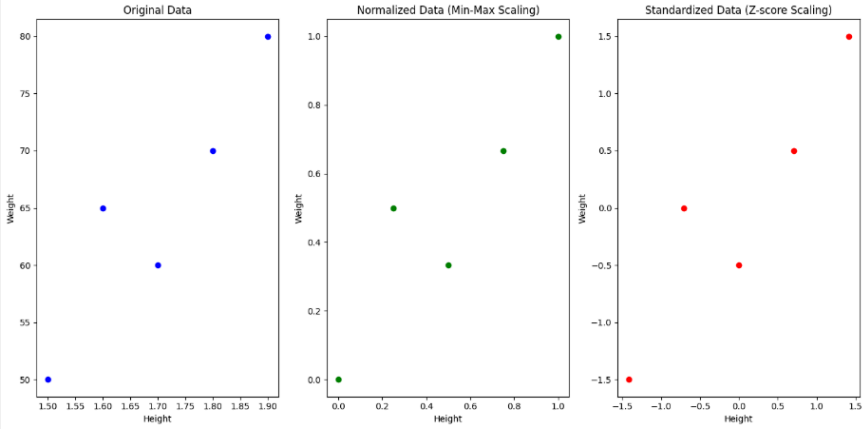
5. Feature Scaling

Feature scaling helps standardize or normalize numerical data, ensuring that all features are on the same scale. This prevents any one feature from dominating the model.

Example:

- **Normalization**: If you have features like **height (1.5, 1.8, 1.7)** and **weight (50, 70, 60)**, you can normalize them so all values are between 0 and 1:
- Height: 1.5 -> 0, 1.8 -> 1, 1.7 -> 0.67
- Weight: 50 -> 0, 70 -> 1, 60 -> 0.33
- **Standardization**: You can also scale data so it has a mean of 0 and a standard deviation of 1.

```
Original Data:
   Height  Weight
0     1.5      50
1     1.8      70
2     1.7      60
3     1.9      80
4     1.6      65

Normalized Data (Min-Max Scaling):
   Height    Weight
0    0.00  0.000000
1    0.75  0.666667
2    0.50  0.333333
3    1.00  1.000000
4    0.25  0.500000

Standardized Data (Z-score Scaling):
     Height  Weight
0 -1.414214    -1.5
1  0.707107     0.5
2  0.000000    -0.5
3  1.414214     1.5
4 -0.707107     0.0
```

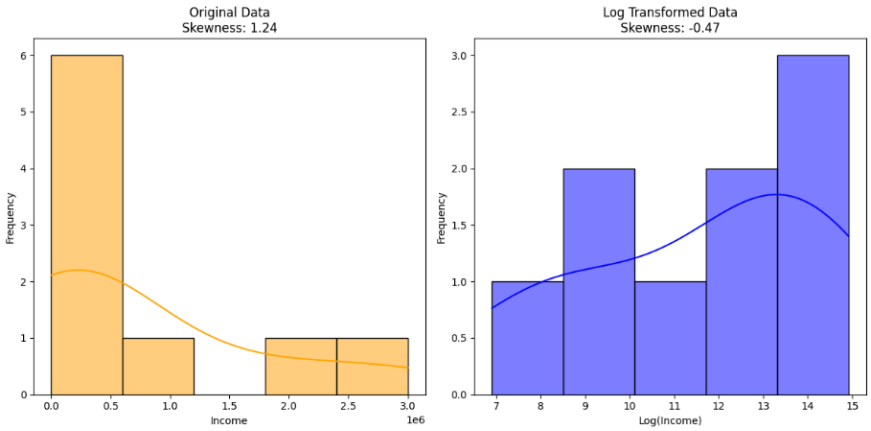


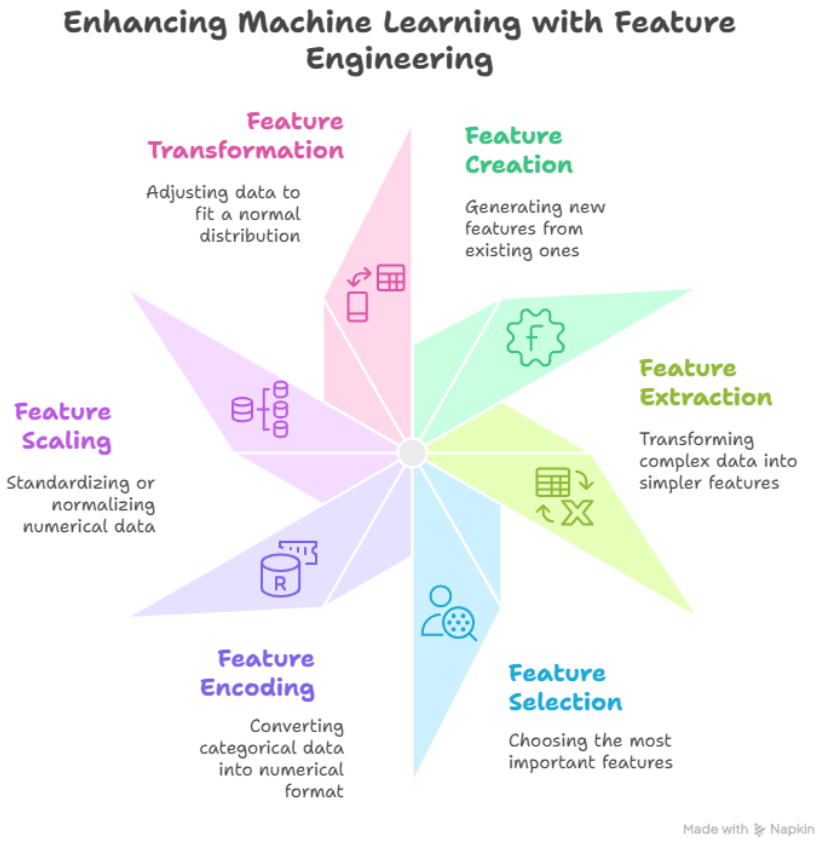
6. Feature Transformation

Feature transformation involves applying mathematical techniques to adjust or modify data, often to make it closer to a normal distribution.

Example:

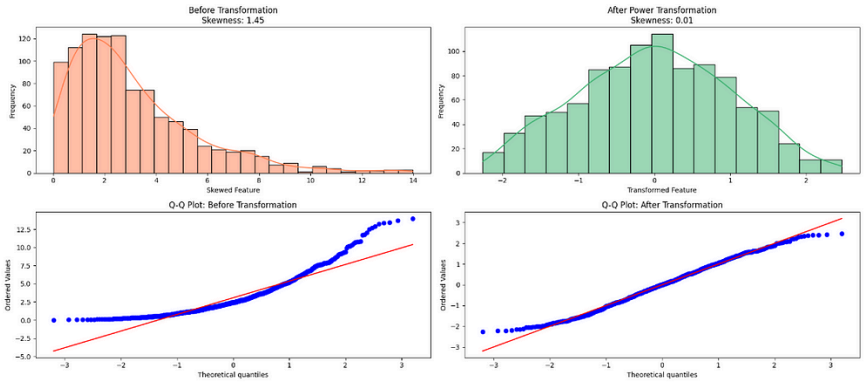
- Imagine you have a **skewed income dataset: 1,000, 5,000, 10,000, 100,000.**
- A **log transformation** can be applied to reduce the skew and bring the values closer to a normal distribution:
- Log(1000) = 3.00, Log(5000) = 3.70, Log(10000) = 4.00, Log(100000) = 5.00.



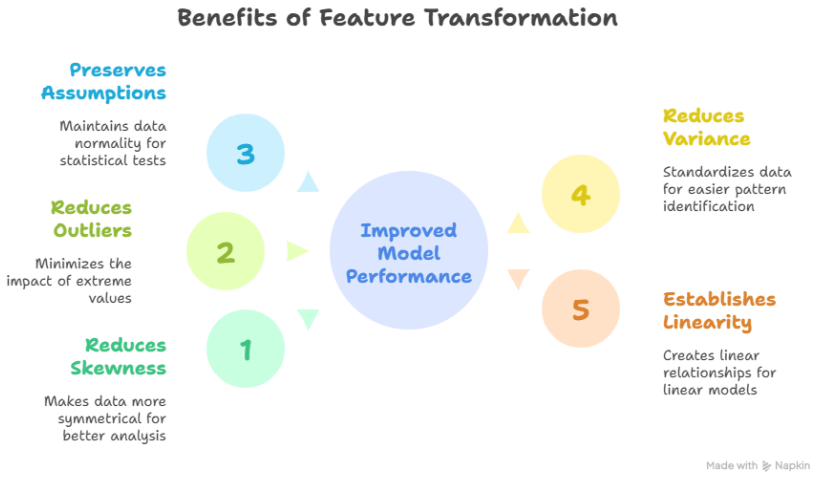


What is Feature Transformation?

Feature transformation is a mathematical technique used to adjust or modify the distribution of data in a way that it resembles a **normal distribution.** This process helps machine learning algorithms perform better and meet certain statistical assumptions.



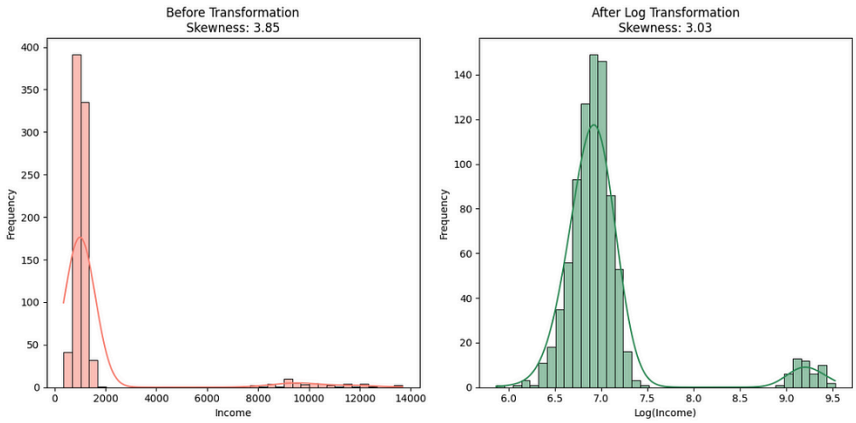
Why is Feature Transformation Important?



1. **Reduces Skewness:**

Feature transformation helps in making data more symmetrical, reducing skewness. For instance, if the data has a long tail (right or left), applying transformations like the **logarithm** or **Box-Cox** can make it more balanced.

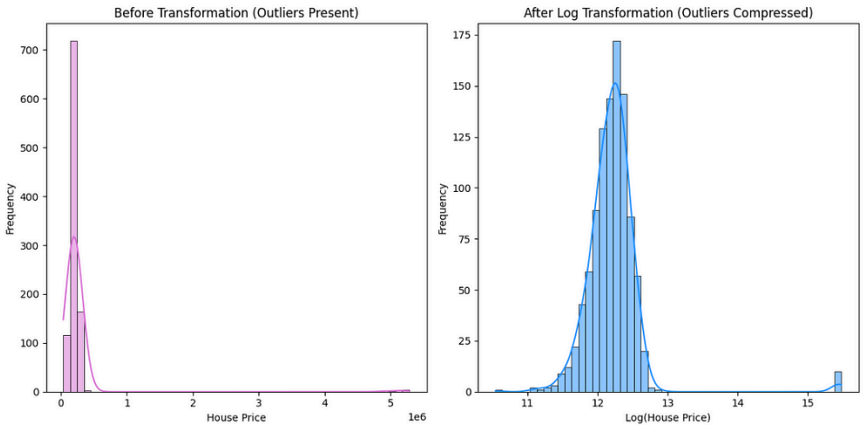
Example:
If you have income data where most values are clustered around the lower range but a few high incomes (outliers) skew the data, a log transformation can reduce the impact of these extreme values.



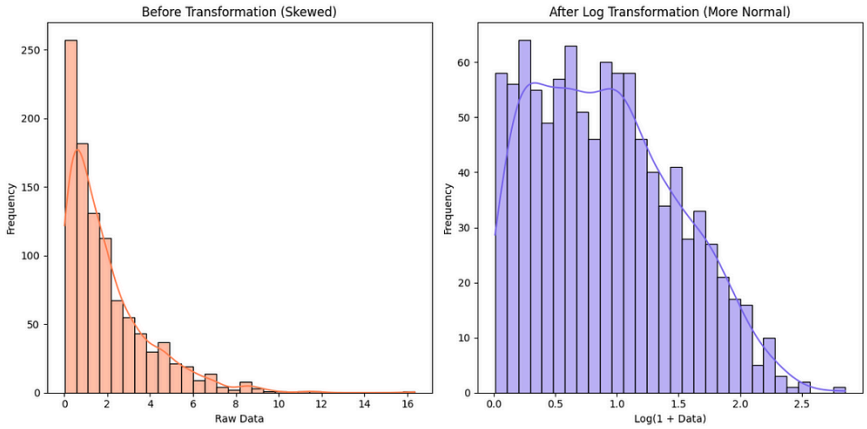
2. Reduces the Impact of Outliers:

Feature transformation can compress the effect of extreme outliers, which can disproportionately affect model performance, especially in linear models.

Example:
For a dataset with house prices, extreme values (like multi-million dollar houses) can be transformed using a square root or log transformation to make the data more manageable.

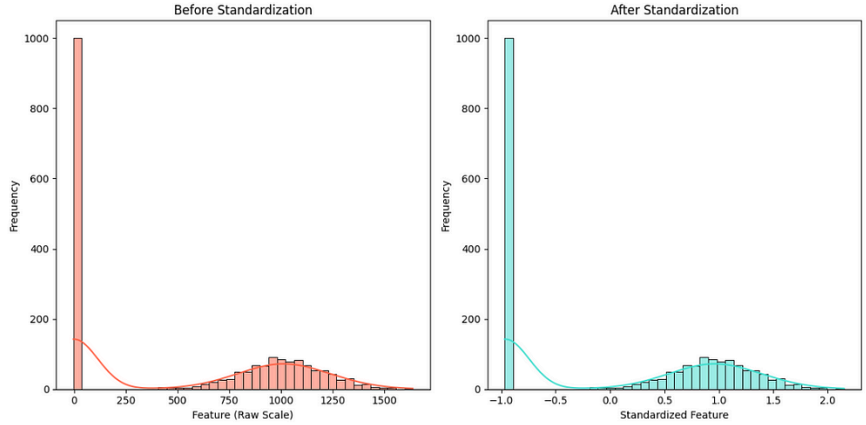


3. Preserves Assumptions in Statistical Testing:
Many machine learning models assume that the data is normally distributed. Feature transformation helps in maintaining assumptions, especially when performing hypothesis testing, **Z-tests**, **T-tests**, **ANOVA**, or working with algorithms that require normality, such as **Gaussian Naive Bayes**.



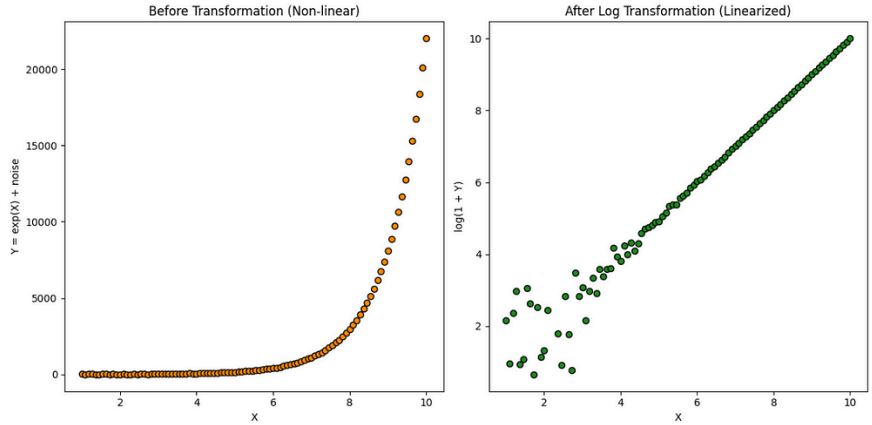
4. Reduces Variance:

In some cases, data can be spread out across a wide range, creating high variance. Transformation can help standardize the data, making it easier for models to identify patterns.



5. Establishes Linear Relationships:

Feature transformation helps create a more linear relationship between the input data and the target variable. This is particularly important for algorithms like **linear regression** or **logistic regression**, where the model works best with linear relationships.



Which Algorithms Benefit from Feature Transformation?

Feature transformation can significantly improve the performance of several machine learning algorithms, particularly those that assume a linear relationship or normal distribution:

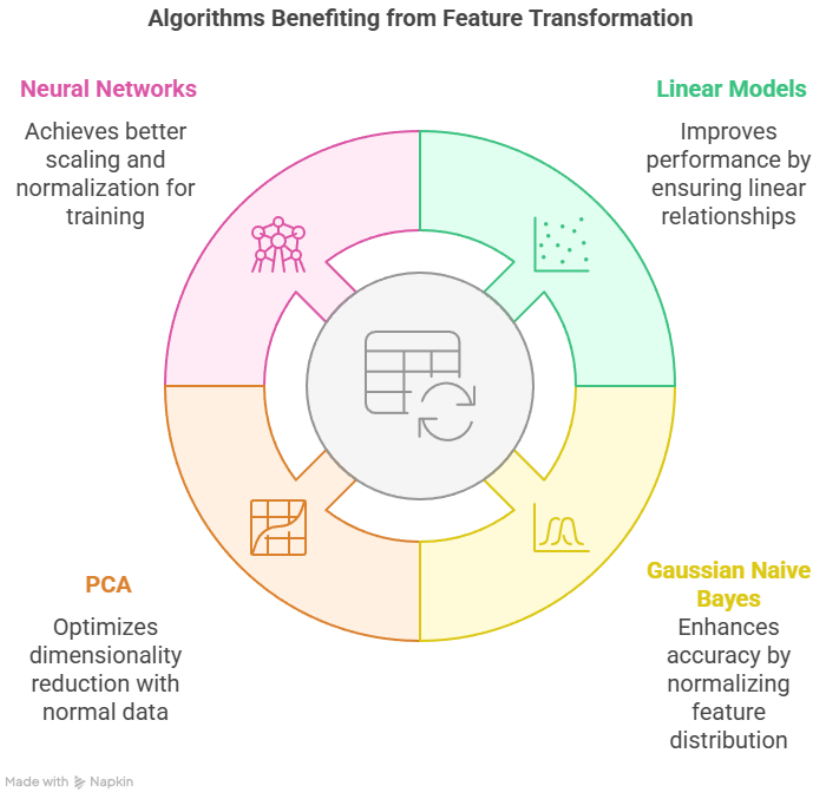
1. **Linear Models:**

- **Linear Regression**
- **Logistic Regression**
- **Support Vector Machines (SVM)**

2. Gaussian Naive Bayes:
This algorithm assumes that features are normally distributed. Applying feature transformation to the data can improve its accuracy.

3. PCA (Principal Component Analysis):
PCA is used for dimensionality reduction in unsupervised learning. It assumes that the data is normally distributed, and applying transformations can improve the results.

4. Neural Networks (ANN—Artificial Neural Networks):
Neural networks benefit from data that is well-scaled and normalized. Feature transformation can help achieve that, especially when the data contains large variances or skewness.



Which Algorithms Do Not Support Feature Transformation?

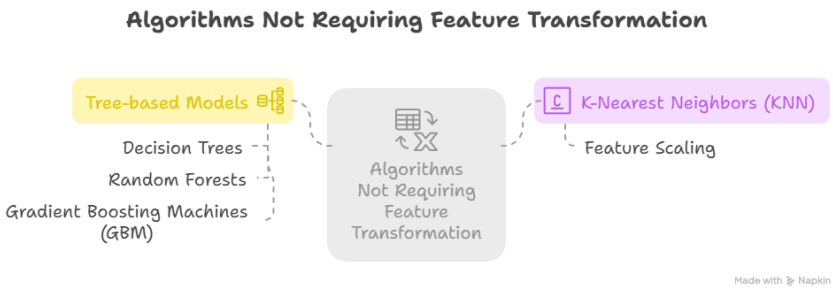
Not all machine learning algorithms benefit from feature transformation. Some algorithms are more robust to variations in data distribution and do not rely on assumptions like normality:

1. **Tree-based Models:**

- **Decision Trees**
- **Random Forests**
- **Gradient Boosting Machines (GBM)**

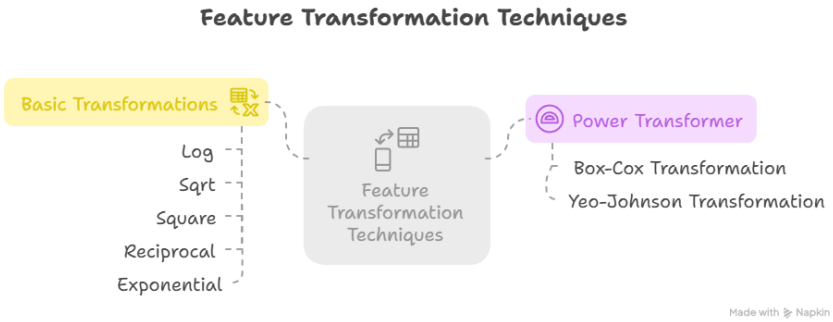
These models do not rely on feature distributions and can handle skewed or non-normal data without requiring transformation.

2. K-Nearest Neighbors (KNN):
KNN is distance-based, and while feature scaling might be helpful, it doesn't strictly require feature transformation. However, **feature scaling** (not transformation) might still be needed to avoid features with larger scales dominating the distance metric.

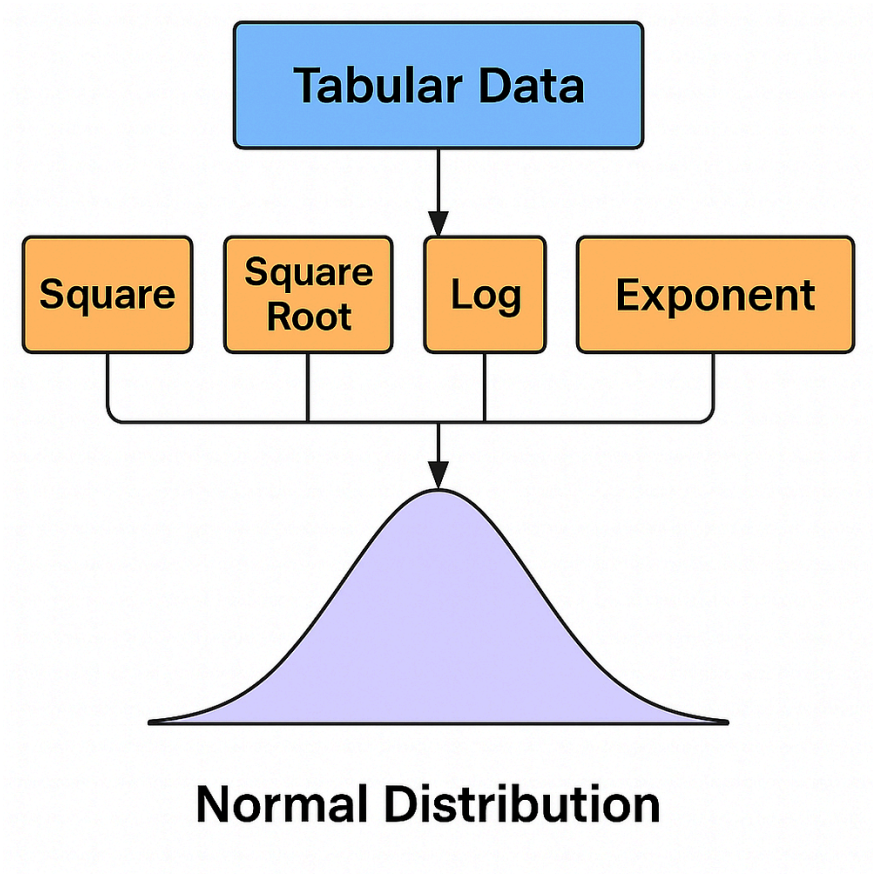


Types of Feature Transformation:

1. **Basic Transformations**
2. **Power Transformer**



Basic Transformations:

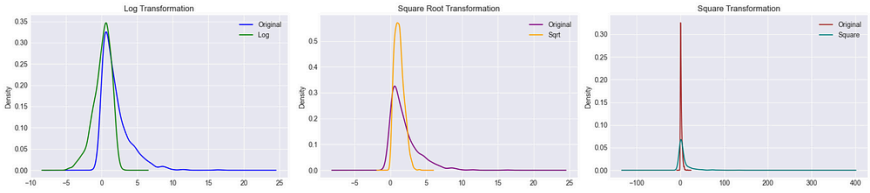


These are direct mathematical operations applied to numerical features.

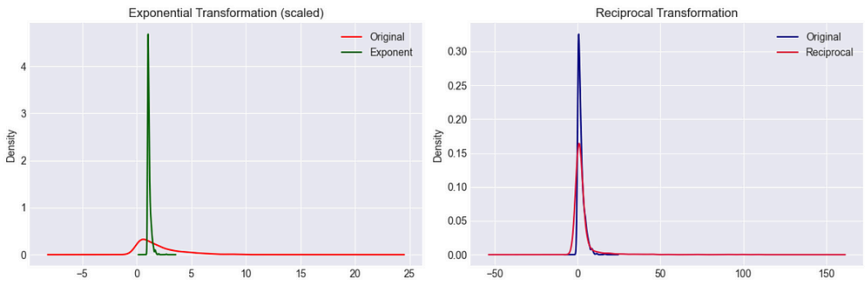
Technique	Formula	Manual Example	Python Code
Logarithmic	$y = \log(x)$	x = 100 → log(100) = 2	`np.log(x)`
Square Root	$y = \sqrt{x}$	x = 25 → √25 = 5	`np.sqrt(x)`
Square	$y = x^2$	x = 3 → 3^2 = 9	`np.power(x, 2)`
Exponential	$y = e^{\wedge}x$	x = 1 → $e^1 \approx 2.71$	`np.exp(x)`
Reciprocal	$y = 1/x$	x = 5 → 1/5 = 0.2	`1 / x`

Example:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Set style
plt.style.use('seaborn-v0_8-darkgrid')

# Generate sample skewed data
np.random.seed(42)
df = pd.DataFrame({
    'Feature 1': np.random.exponential(scale=2, size=1000)
})

plt.figure(figsize=(18, 4))

# Log
plt.subplot(1, 3, 1)
df['Feature 1'].plot(kind='kde', color='blue', label='Original')
np.log(df['Feature 1']).plot(kind='kde', color='green', label='Log')
plt.title("Log Transformation")
plt.legend()

# Square Root
plt.subplot(1, 3, 2)
df['Feature 1'].plot(kind='kde', color='purple', label='Original')
np.sqrt(df['Feature 1']).plot(kind='kde', color='orange', label='Sqrt')
plt.title("Square Root Transformation")
plt.legend()

# Square
plt.subplot(1, 3, 3)
df['Feature 1'].plot(kind='kde', color='brown', label='Original')
np.square(df['Feature 1']).plot(kind='kde', color='teal', label='Square')
plt.title("Square Transformation")
plt.legend()

plt.tight_layout()
plt.show()
```




```python
plt.figure(figsize=(12, 4))

# Exponent
plt.subplot(1, 2, 1)
df['Feature 1'].plot(kind='kde', color='red', label='Original')
np.exp(df['Feature 1'] / df['Feature 1'].max()).plot(kind='kde', color='darkgreen', label='Exponent')
plt.title("Exponential Transformation (scaled)")
plt.legend()

# Reciprocal
plt.subplot(1, 2, 2)
df['Feature 1'].plot(kind='kde', color='navy', label='Original')
(1 / df['Feature 1']).plot(kind='kde', color='crimson', label='Reciprocal')
plt.title("Reciprocal Transformation")
plt.legend()

plt.tight_layout()
plt.show()
```



These are useful when:

- Data is skewed
- You want to compress wide-ranging values
- You need to reduce the impact of outliers

Power Transformer Techniques

These transformations aim to **normalize the data,** and are more sophisticated than basic transformations.

i) Box-Cox Transformation

- **Used when:** All values in the feature are **positive.**
- **Goal:** Make non-normal data approximate a **normal distribution.**
- **Formula:**

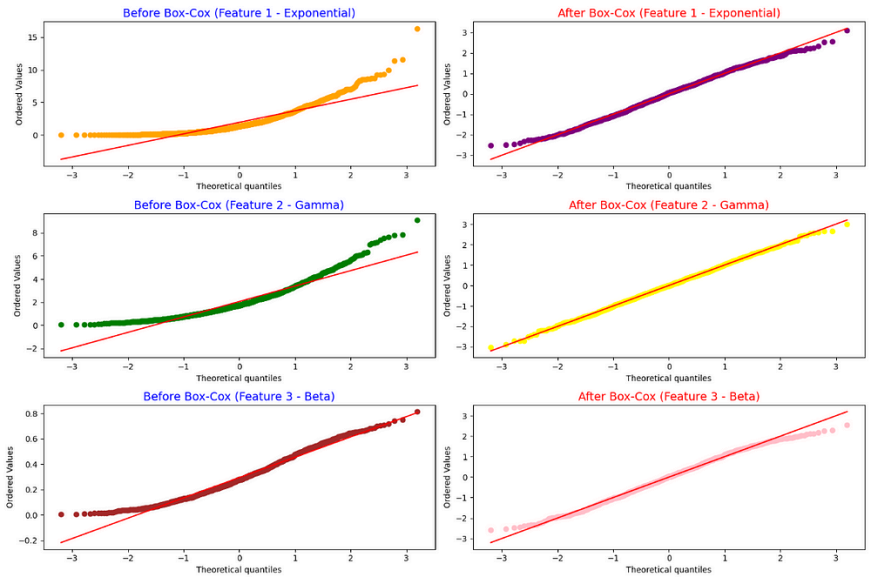
$$y = \frac{x^\lambda - 1}{\lambda}, \text{ if } \lambda \neq 0$$
$$y = \log(x), \text{ if } \lambda = 0$$

Manual Example:
Suppose `x = 10`, and λ = 0.5

$$y = \frac{10^{0.5} - 1}{0.5} = \frac{3.16 - 1}{0.5} = 4.32$$

Example:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.preprocessing import PowerTransformer

# Create a new DataFrame with mixed distributions for Box-Cox
np.random.seed(42)
df_boxcox = pd.DataFrame({
    'Feature 1': np.random.exponential(scale=2, size=1000),  # Exponential distribution (positively skewed)
    'Feature 2': np.random.gamma(shape=2, scale=1, size=1000),  # Gamma distribution (positively skewed)
    'Feature 3': np.random.beta(a=2, b=5, size=1000)  # Beta distribution (bounded between 0 and 1)
})

# Apply Box-Cox Transformation (works only for positive data)
pt = PowerTransformer(method='box-cox')

# Create subplots for Before and After transformation for each feature
plt.figure(figsize=(15, 10))

# Feature 1 (Exponential Distribution)
plt.subplot(3, 2, 1)
stats.probplot(df_boxcox['Feature 1'], plot=plt)
plt.title("Before Box-Cox (Feature 1 - Exponential)", fontsize=14, color='blue')
plt.gca().get_lines()[0].set_color('orange')

plt.subplot(3, 2, 2)
d1 = pt.fit_transform(df_boxcox[['Feature 1']])
stats.probplot(d1.flatten(), plot=plt)
plt.title("After Box-Cox (Feature 1 - Exponential)", fontsize=14, color='red')
plt.gca().get_lines()[0].set_color('purple')

# Feature 2 (Gamma Distribution)
plt.subplot(3, 2, 3)
stats.probplot(df_boxcox['Feature 2'], plot=plt)
plt.title("Before Box-Cox (Feature 2 - Gamma)", fontsize=14, color='blue')
plt.gca().get_lines()[0].set_color('green')

plt.subplot(3, 2, 4)
d2 = pt.fit_transform(df_boxcox[['Feature 2']])
stats.probplot(d2.flatten(), plot=plt)
plt.title("After Box-Cox (Feature 2 - Gamma)", fontsize=14, color='red')
plt.gca().get_lines()[0].set_color('yellow')

# Feature 3 (Beta Distribution)
plt.subplot(3, 2, 5)
stats.probplot(df_boxcox['Feature 3'], plot=plt)
plt.title("Before Box-Cox (Feature 3 - Beta)", fontsize=14, color='blue')
plt.gca().get_lines()[0].set_color('brown')

plt.subplot(3, 2, 6)
d3 = pt.fit_transform(df_boxcox[['Feature 3']])
stats.probplot(d3.flatten(), plot=plt)
plt.title("After Box-Cox (Feature 3 - Beta)", fontsize=14, color='red')
plt.gca().get_lines()[0].set_color('pink')

plt.tight_layout()
plt.show()
```



ii) Yeo-Johnson Transformation

- **Used when:** Feature has **both positive and negative** values.
- **Formula** (simplified conditions):

For x ≥ 0:

$$y = \frac{(x+1)^\lambda - 1}{\lambda}, \text{ if } \lambda \neq 0$$
$$y = \log(x+1), \text{ if } \lambda = 0$$

For x < 0:

$$y = -\frac{(1-x)^{(2-\lambda)} - 1}{2-\lambda}, \text{ if } \lambda \neq 2$$
$$y = -\log(-x+1), \text{ if } \lambda = 2$$

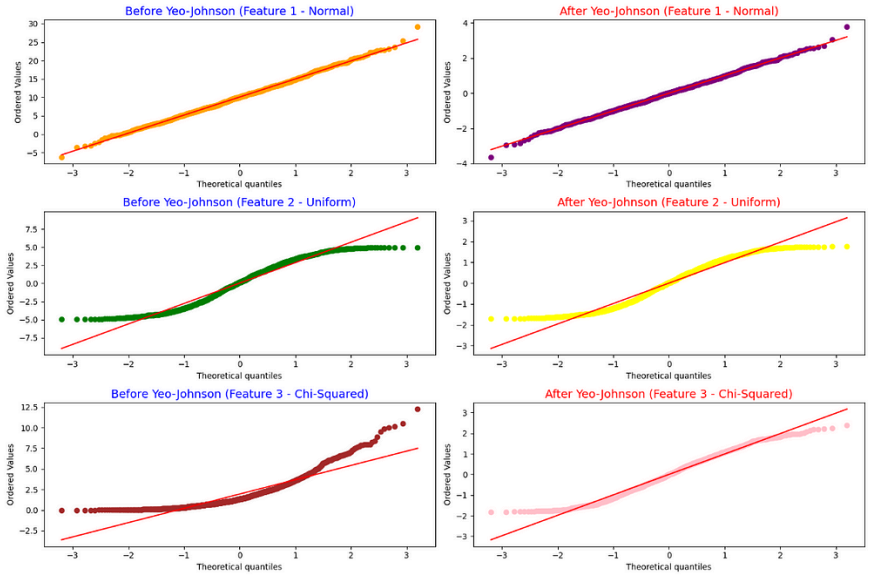
Manual Example:

Let `x = -2`, λ = 2

$$y = -\log(-(-2)+1) = -\log(3) \approx -1.09$$

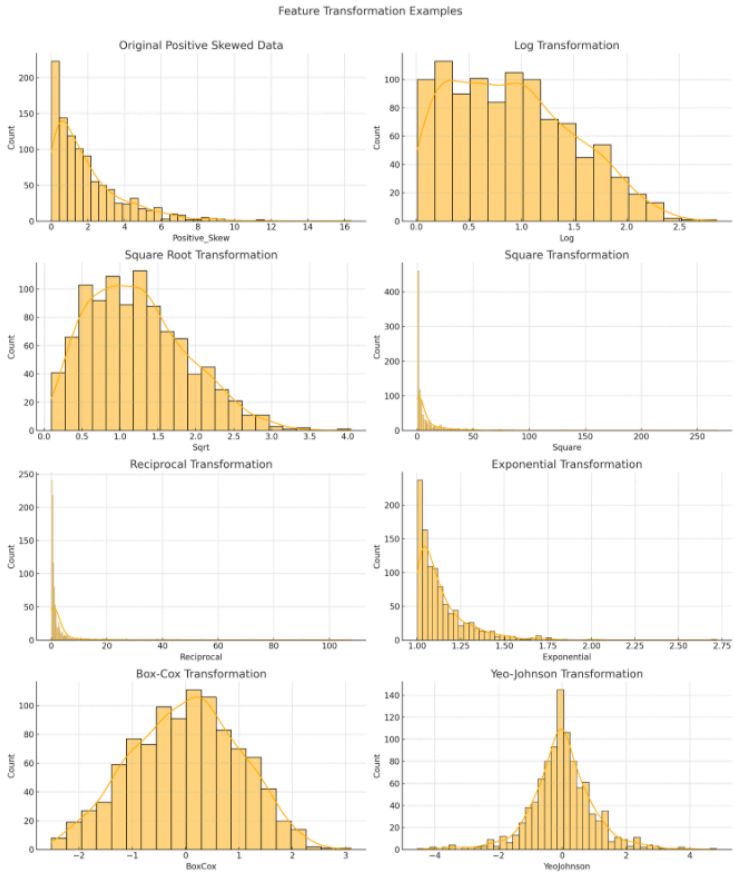
Example:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.preprocessing import PowerTransformer

# Create a new DataFrame with mixed distributions
np.random.seed(42)
df_yeo = pd.DataFrame({
    'Feature 1': np.random.normal(loc=10, scale=5, size=1000),  # Normal distribution
    'Feature 2': np.random.uniform(low=-5, high=5, size=1000),   # Uniform distribution (can be negative)
    'Feature 3': np.random.chisquare(df=2, size=1000)  # Chi-squared distribution
})

# Apply Yeo-Johnson Transformation
pt_y = PowerTransformer(method='yeo-johnson')

# Create subplots for Before and After transformation for each feature
plt.figure(figsize=(15, 10))

# Feature 1 (Normal Distribution)
plt.subplot(3, 2, 1)
stats.probplot(df_yeo['Feature 1'], plot=plt)
plt.title("Before Yeo-Johnson (Feature 1 - Normal)", fontsize=14, color='blue')
plt.gca().get_lines()[0].set_color('orange')

plt.subplot(3, 2, 2)
d1 = pt_y.fit_transform(df_yeo[['Feature 1']])
stats.probplot(d1.flatten(), plot=plt)
plt.title("After Yeo-Johnson (Feature 1 - Normal)", fontsize=14, color='red')
plt.gca().get_lines()[0].set_color('purple')

# Feature 2 (Uniform Distribution)
plt.subplot(3, 2, 3)
stats.probplot(df_yeo['Feature 2'], plot=plt)
plt.title("Before Yeo-Johnson (Feature 2 - Uniform)", fontsize=14, color='blue')
plt.gca().get_lines()[0].set_color('green')

plt.subplot(3, 2, 4)
d2 = pt_y.fit_transform(df_yeo[['Feature 2']])
stats.probplot(d2.flatten(), plot=plt)
plt.title("After Yeo-Johnson (Feature 2 - Uniform)", fontsize=14, color='red')
plt.gca().get_lines()[0].set_color('yellow')

# Feature 3 (Chi-squared Distribution)
plt.subplot(3, 2, 5)
stats.probplot(df_yeo['Feature 3'], plot=plt)
plt.title("Before Yeo-Johnson (Feature 3 - Chi-Squared)", fontsize=14, color='blue')
plt.gca().get_lines()[0].set_color('brown')

plt.subplot(3, 2, 6)
d3 = pt_y.fit_transform(df_yeo[['Feature 3']])
stats.probplot(d3.flatten(), plot=plt)
plt.title("After Yeo-Johnson (Feature 3 - Chi-Squared)", fontsize=14, color='red')
plt.gca().get_lines()[0].set_color('pink')

plt.tight_layout()
plt.show()
```



Manual Interpretation with Simple Examples:

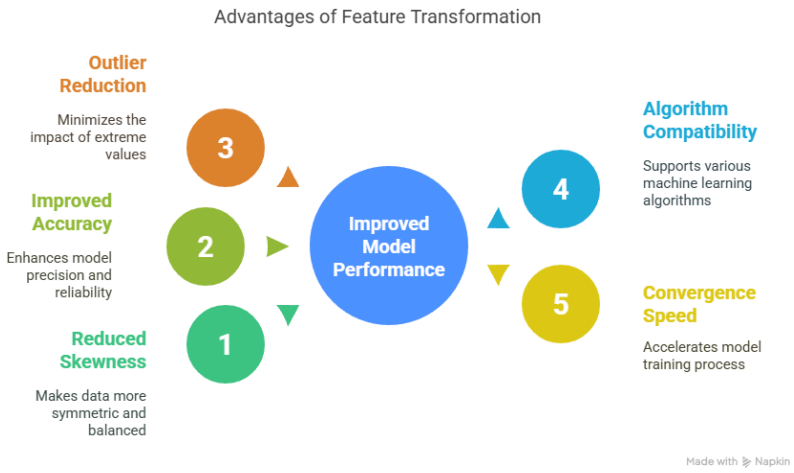
Transformation Type	Example Value (x = 4)	Transformed Value (y)	Use Case
Log	`log(4)`	≈ 1.39	Reduce right skew
Sqrt	`√4`	= 2.0	Moderate skew
Square	`4²`	= 16	Amplify larger values
Reciprocal	`1/4`	= 0.25	Reduce impact of large values
Exponential	`e^4`	≈ 54.6	Spread small values wider
Box-Cox	Only for positive x	Automatically finds best λ	Normalize positive data
Yeo-Johnson	Works for positive & negative x	Automatically finds best λ	Normalize any real data



Advantages of Feature Transformation:

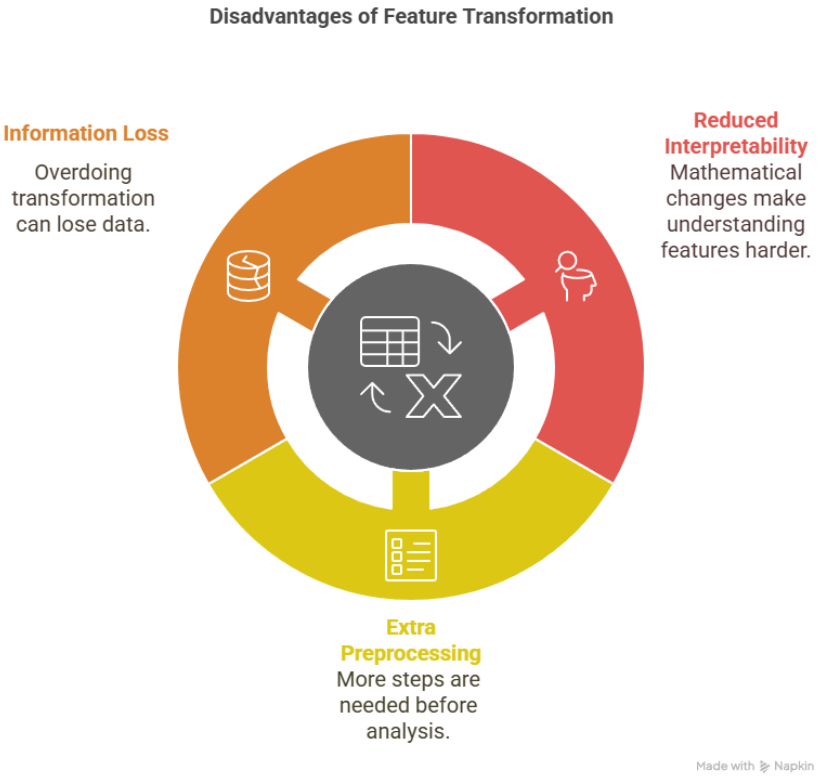
- **Reduces skewness** and makes data more symmetric
- **Improves model accuracy** for algorithms sensitive to data distribution

- **Reduces the impact of outliers**
- Helps algorithms like **Linear Regression**, **SVM**, **Gaussian Naive Bayes**, **PCA**, and **Neural Networks**
- Can improve convergence speed during training



Disadvantages of Feature Transformation:

- **Interpretability** may reduce due to mathematical changes
- Requires **extra preprocessing steps**
- Risk of **information loss** if transformation is overdone


Disadvantages of Feature Transformation
Information Loss
Overdoing transformation can lose data.
Reduced Interpretability
Mathematical changes make understanding features harder.
Extra Preprocessing
More steps are needed before analysis.
Made with Napkin