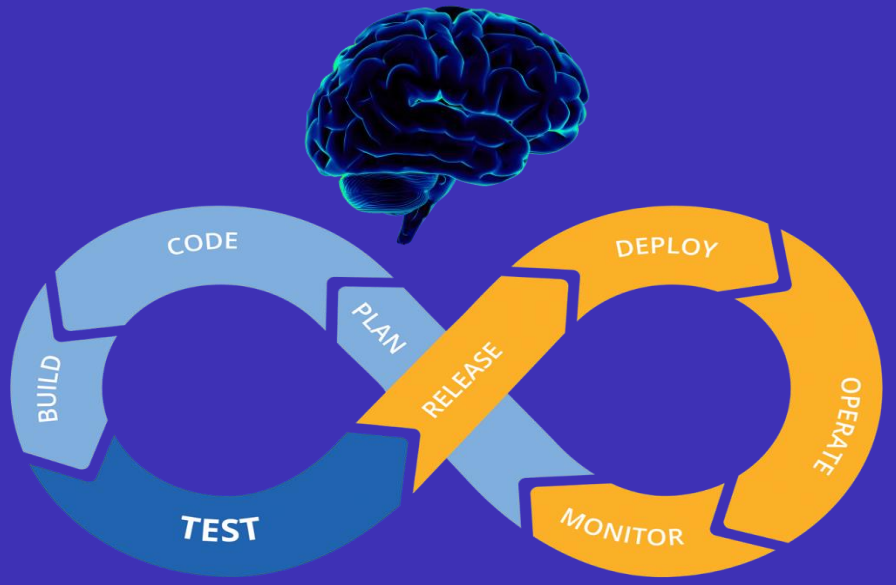
CSED!

"A Place to Invent and Learn"



Machine Learning Course

Module 1: Data Preprocessing for machine learning

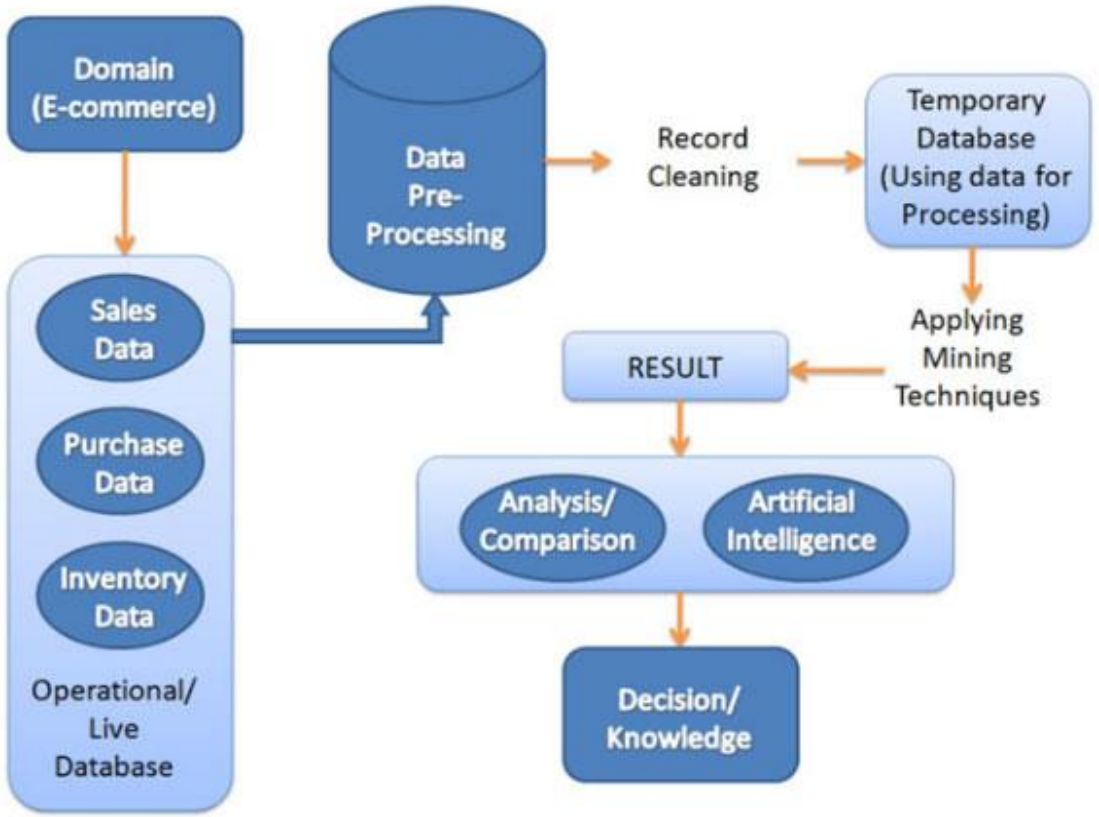
Topics

- What is Data Pre-processing in machine learning
- Data Imputation
- Data Transformation
- Data Splitting
- Dimensionality Reduction
- Data Augmentation

Ye mere sath hi kyu hota hai

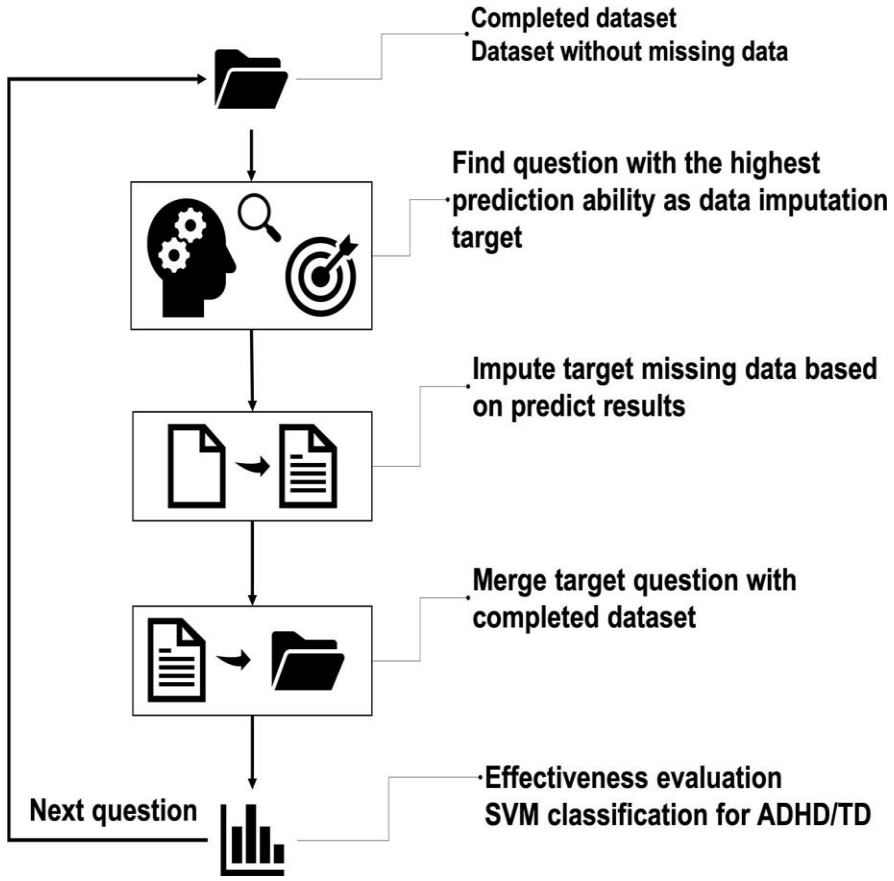
What is Data Pre-processing in machine learning

Data preprocessing in machine learning refers to the steps and techniques used to prepare raw data for modeling. It is a critical phase in the machine learning pipeline as the quality and suitability of the data directly impact the effectiveness of the models built upon it.



Data Imputation

Data imputation is a technique used to handle missing values in a dataset. Missing values can arise from various reasons such as errors in data collection, corruption, or simply the unavailability of information. Handling these missing values is crucial because many machine learning algorithms cannot work with datasets that contain missing values. Here, we'll cover various imputation techniques and demonstrate how to implement them using Python.



Identifying Missing Data

First, let's import the necessary libraries and create a sample dataset with missing values.

```
import numpy as np
import pandas as pd


# Sample dataset
data = {
   'A': [1, 2, np.nan, 4, 5],
   'B': [5, np.nan, np.nan, 8, 10],
   'C': ['cat', 'dog', 'cat', np.nan, 'dog']
}

df = pd.DataFrame(data)
print(df)
```

Output

```
   A    B    C
0  1.0  5.0  cat
1  2.0  NaN  dog
2  NaN  NaN  cat
3  4.0  8.0  NaN
4  5.0 10.0  dog
```

Simple Imputation Methods

Removing Missing Values

In some cases, it may be appropriate to remove rows or columns with missing values.

```
# Removing rows with missing values
df_dropped_rows = df.dropna()
print(df_dropped_rows)
```

```
# Removing columns with missing values
df_dropped_cols = df.dropna(axis=1)
print(df_dropped_cols)
```

Simple Imputation Methods

Imputing with a Constant Value

You can fill missing values with a constant value, such as 0 or the mean of the column.

```
# Fill with a constant value (e.g., 0)
df_filled_zero = df.fillna(0)
print(df_filled_zero)
```

Imputing with Mean/Median/Mode

Filling missing numerical values with the mean, median, or mode of the column is a common approach.

```
# Imputing with mean for numerical columns
df['A'] = df['A'].fillna(df['A'].mean())
df['B'] = df['B'].fillna(df['B'].mean())
print(df)
```

```
# Imputing with mode for categorical columns
df['C'] = df['C'].fillna(df['C'].mode()[0])
print(df)
```

Advanced Imputation Methods

Scikit-learn's SimpleImputer

SimpleImputer from scikit-learn provides various strategies for imputation.

```
from sklearn.impute import SimpleImputer

# Imputing numerical columns with mean
numerical_imputer = SimpleImputer(strategy='mean')
df[['A', 'B']] = numerical_imputer.fit_transform(df[['A', 'B']])
print(df)

# Imputing categorical columns with most frequent (mode)
categorical_imputer = SimpleImputer(strategy='most_frequent')
df[['C']] = categorical_imputer.fit_transform(df[['C']])
print(df)
```

Advanced Imputation Methods

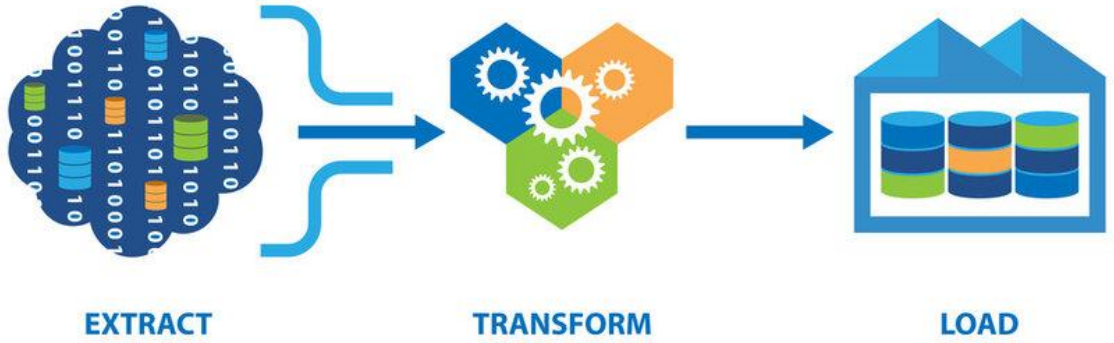
IterativeImputer

IterativeImputer models each feature with missing values as a function of other features and uses that estimate for imputation.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

iterative_imputer = IterativeImputer()
df[['A', 'B']] = iterative_imputer.fit_transform(df[['A', 'B']])
```

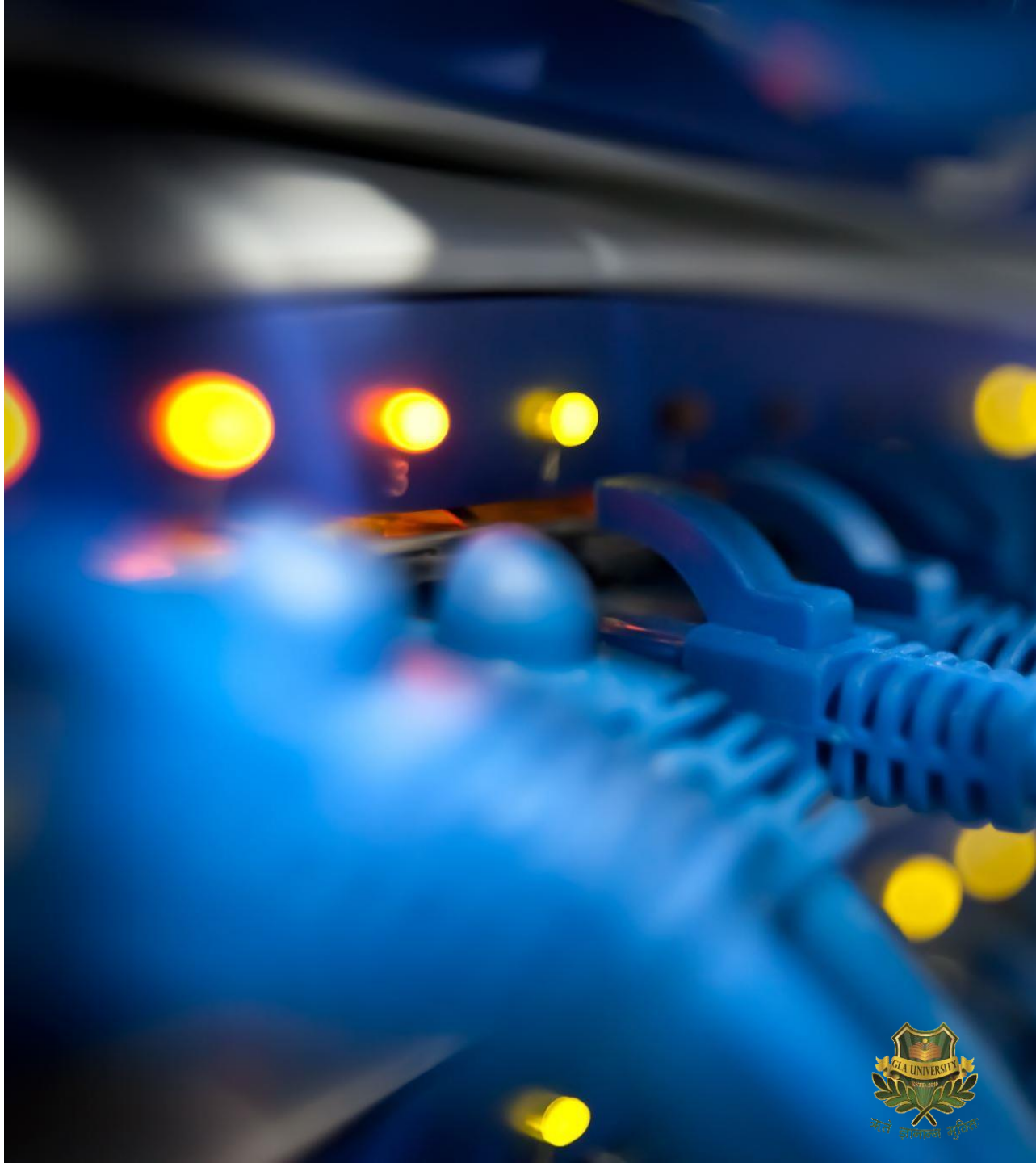
Data Transformation

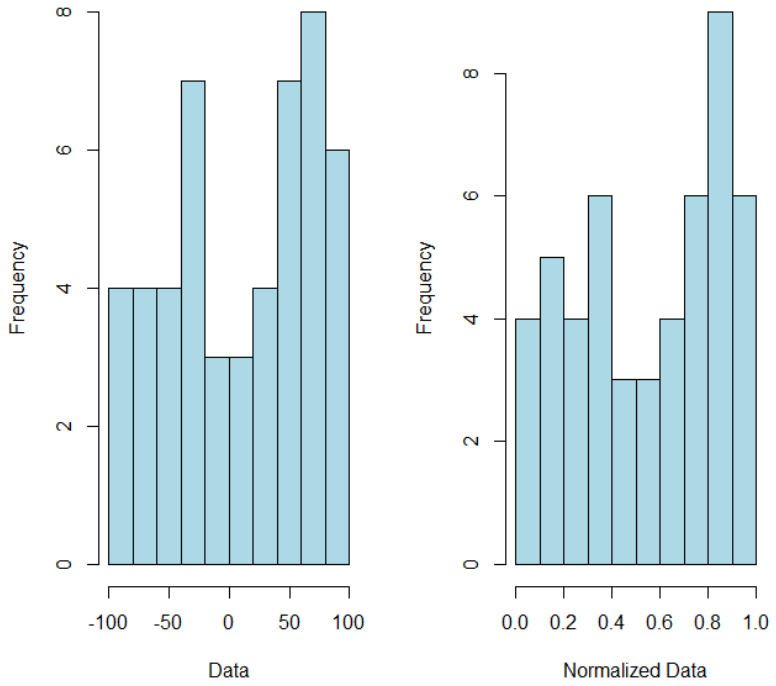
In machine learning, data transformation refers to the process of converting raw data into a format that is suitable and optimal for modeling. This process aims to improve the performance of machine learning algorithms by making the data more understandable, normalized, and aligned with the assumptions of the models being used.



Types of Data Transformation

- **Normalization**
- **Standardization**
- Log Transformation
- Power Transformation
- Box-Cox Transformation
- Quantile Transformation
- **Feature Scaling**

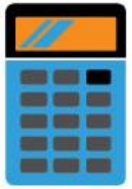




Normalization

Normalization in machine learning refers to the process of adjusting the values of numerical features in a dataset to a common scale, typically between 0 and 1, without distorting differences in the ranges of values. This is particularly important for algorithms that rely on the distance between data points, such as k-nearest neighbors (KNN) or support vector machines (SVM), and for those that use gradient descent for optimization, such as neural networks and logistic regression. This is useful when the features have different units or scales.

Normalization Mathematical Formula

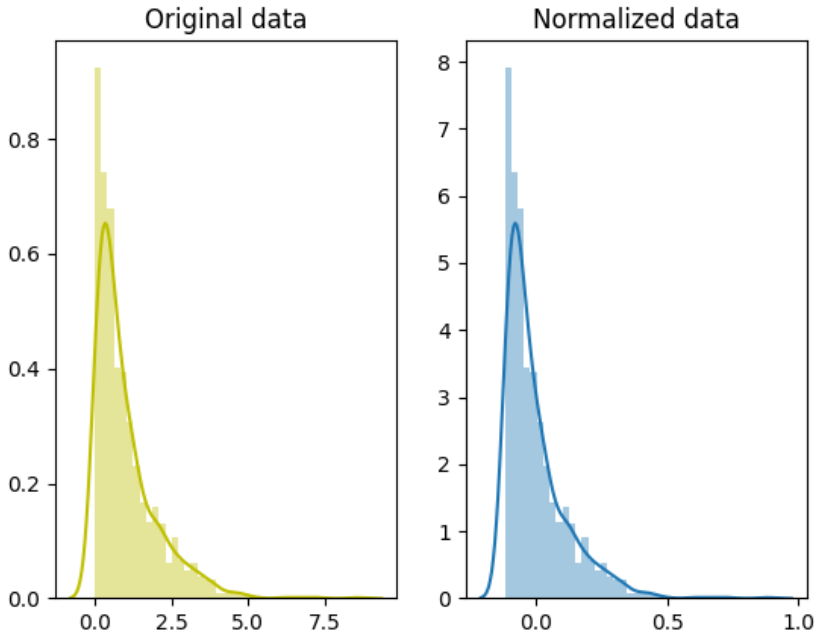


$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Normalization

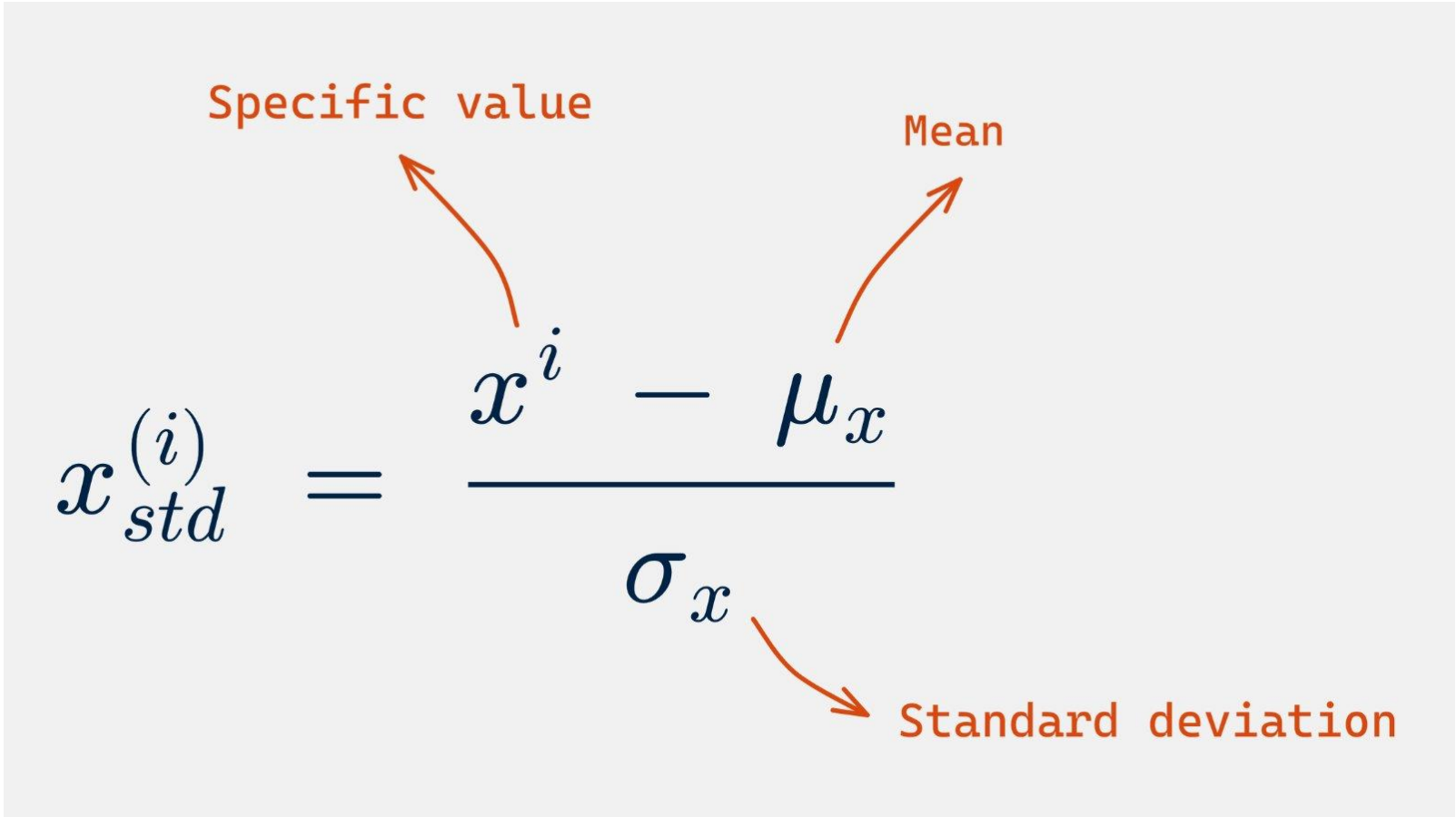
```
from sklearn.preprocessing import MinMaxScaler


# Sample dataset
data = {'Feature1': [10, 20, 30, 40, 50], 'Feature2': [100, 200, 300, 400, 500]}
df = pd.DataFrame(data)


# Normalization
scaler = MinMaxScaler()
df_normalized = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
print(df_normalized)
```

	Feature1	Feature2
0	0.0	0.0
1	0.25	0.25
2	0.5	0.5
3	0.75	0.75
4	1.0	1.0



Standardization

Standardization, also known as Z-score normalization, is a data preprocessing technique used in machine learning to transform the features of a dataset such that they have a mean of zero and a standard deviation of one. This is particularly useful for algorithms that assume the data is normally distributed, such as linear regression, logistic regression, and k-means clustering. This is useful when the features have different scales but are normally distributed.

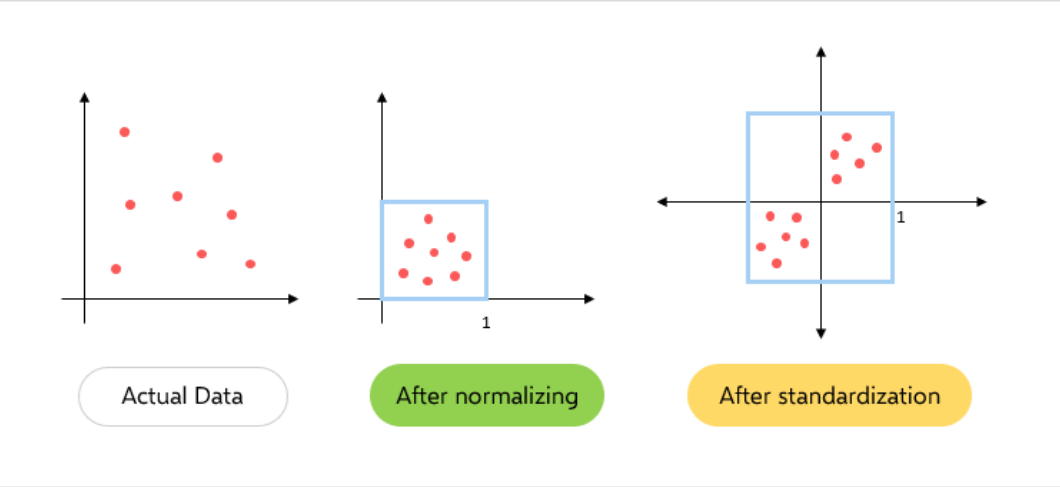


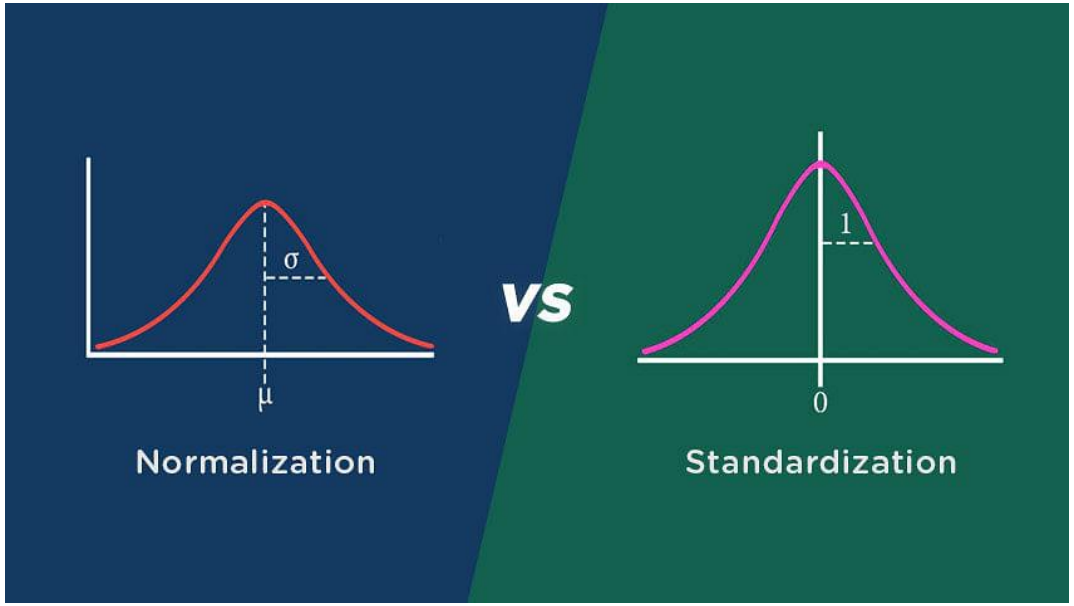
Standardization Mathematical Formula

Standardization

```
from sklearn.preprocessing import StandardScaler

# Sample dataset
data = {'Feature1': [10, 20, 30, 40, 50], 'Feature2': [100, 200, 300, 400, 500]}
df = pd.DataFrame(data)

# Standardization
scaler = StandardScaler()
df_standardized = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
print(df_standardized)
```

	Feature1	Feature2
0	-1.414214	-1.414214
1	-0.707107	-0.707107
2	0.000000	0.000000
3	0.707107	0.707107
4	1.414214	1.414214

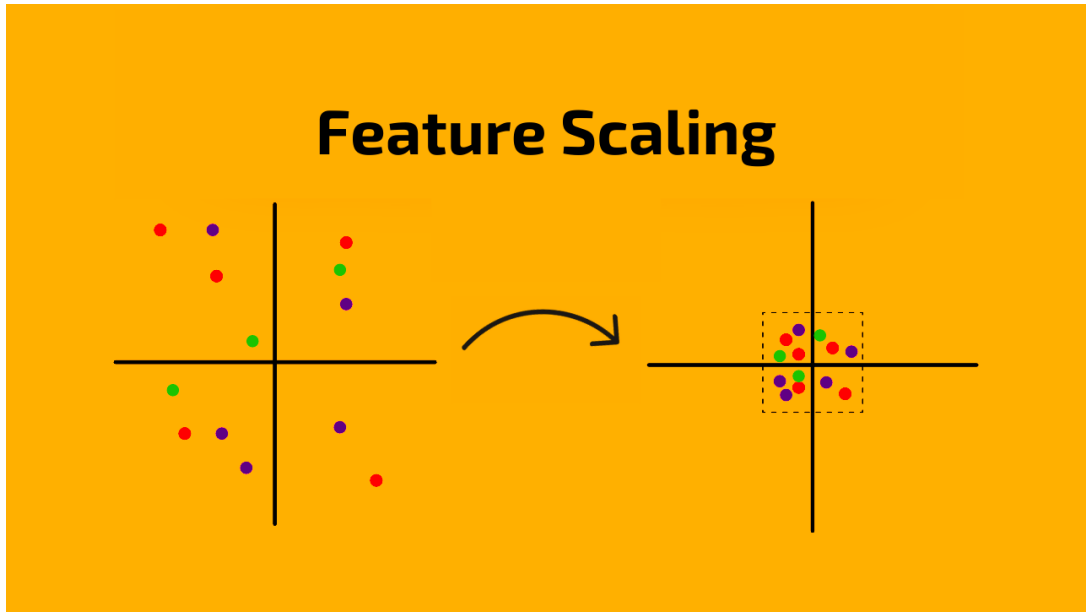




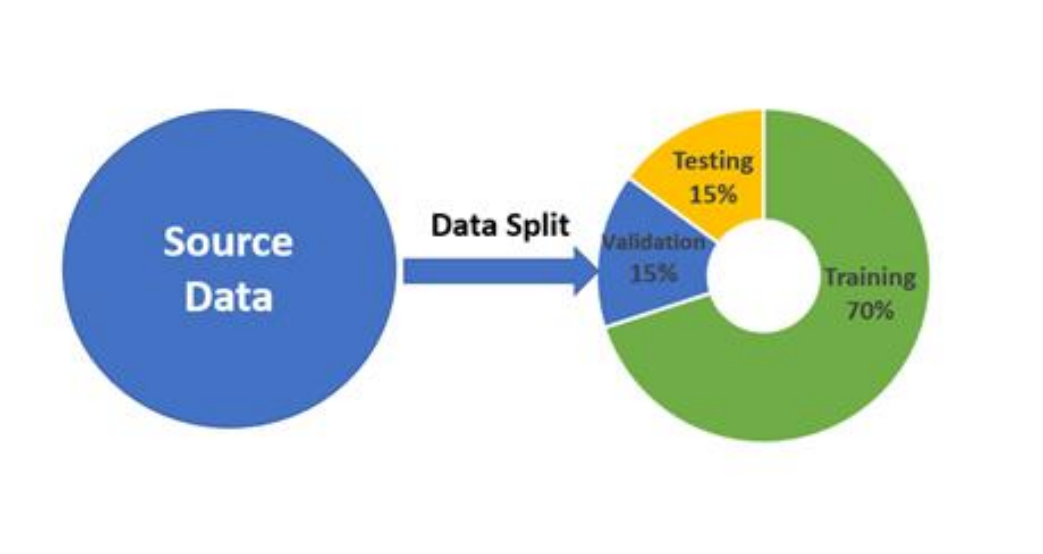
Normalization Vs Standardization

Feature Scaling

Feature scaling is a method used to normalize the range of independent variables or features in a dataset. In machine learning, it is essential to ensure that features contribute equally to the analysis, especially for algorithms that rely on distance calculations or gradient-based optimizations.



Data Splitting



Splitting the dataset into training and test sets is a fundamental step in machine learning. It ensures that we can evaluate our model's performance on unseen data. Feature scaling is crucial when features have different scales, as it helps in improving the performance and convergence speed of many machine learning algorithms.

Splitting the dataset typically involves dividing it into two or three sets:

- Training Set: Used to train the model.
- Test Set: Used to evaluate the model.
- Validation Set: (Optional) Used to tune the model parameters and validate the performance during training.

Data Splitting

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Sample dataset
data = {
   'Feature1': [10, 20, 30, 40, 50],
   'Feature2': [100, 200, 300, 400, 500],
   'Target': [1, 0, 1, 0, 1]
}
df = pd.DataFrame(data)
print("Original Dataset:")
print(df)
```

	Feature1	Feature2	Target
0	10	100	1
1	20	200	0
2	30	300	1
3	40	400	0
4	50	500	1

Data Splitting

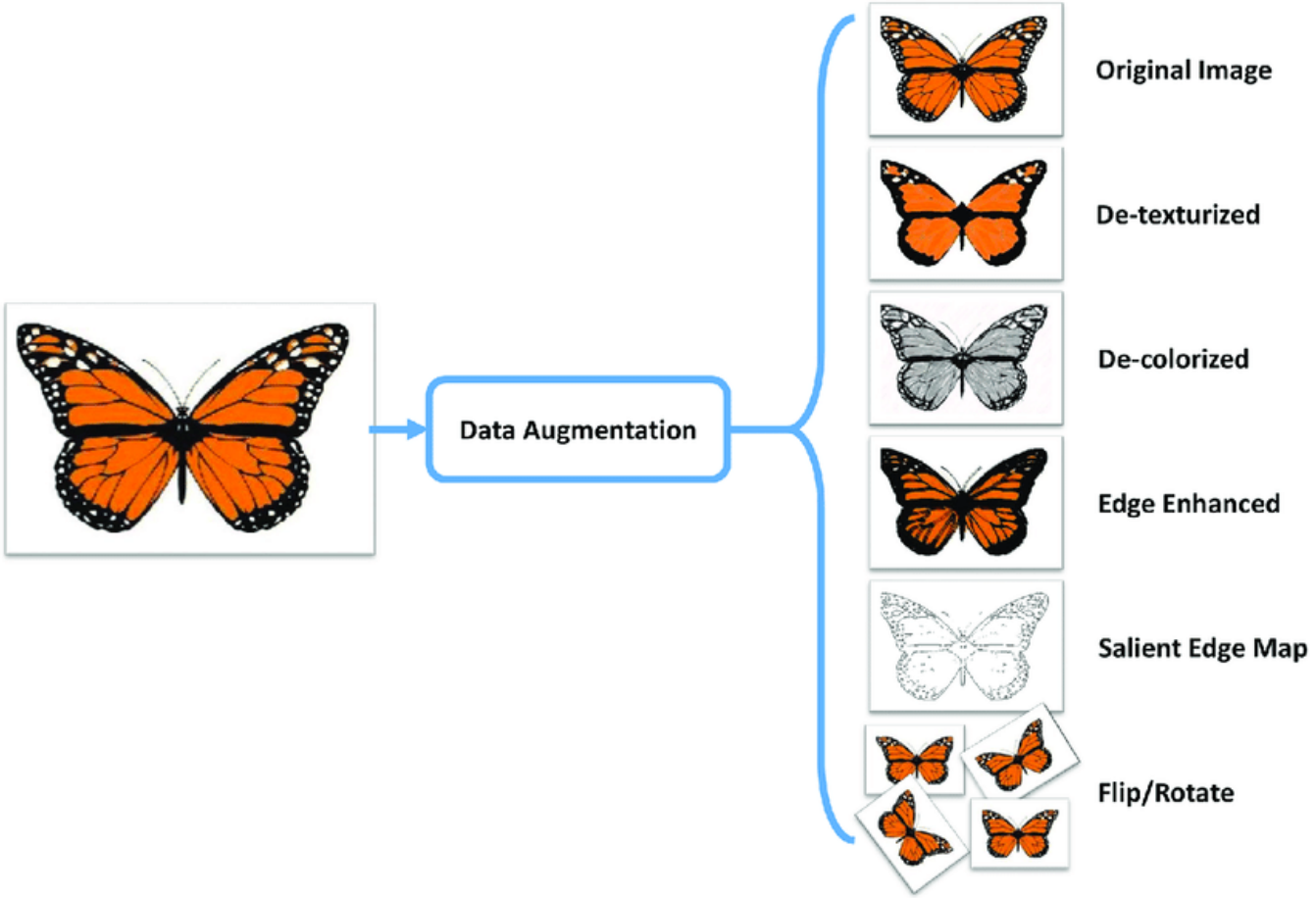
```
# Splitting the dataset
X = df.drop('Target', axis=1)  # Features
y = df['Target']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training Features:")
print(X_train)
print("Test Features:")
print(X_test)
```

Dimensionality Reduction

Dimensionality reduction is a crucial preprocessing step in machine learning that involves reducing the number of input variables (features) in a dataset. This technique helps in simplifying models, reducing overfitting, and improving the computational efficiency of algorithms.

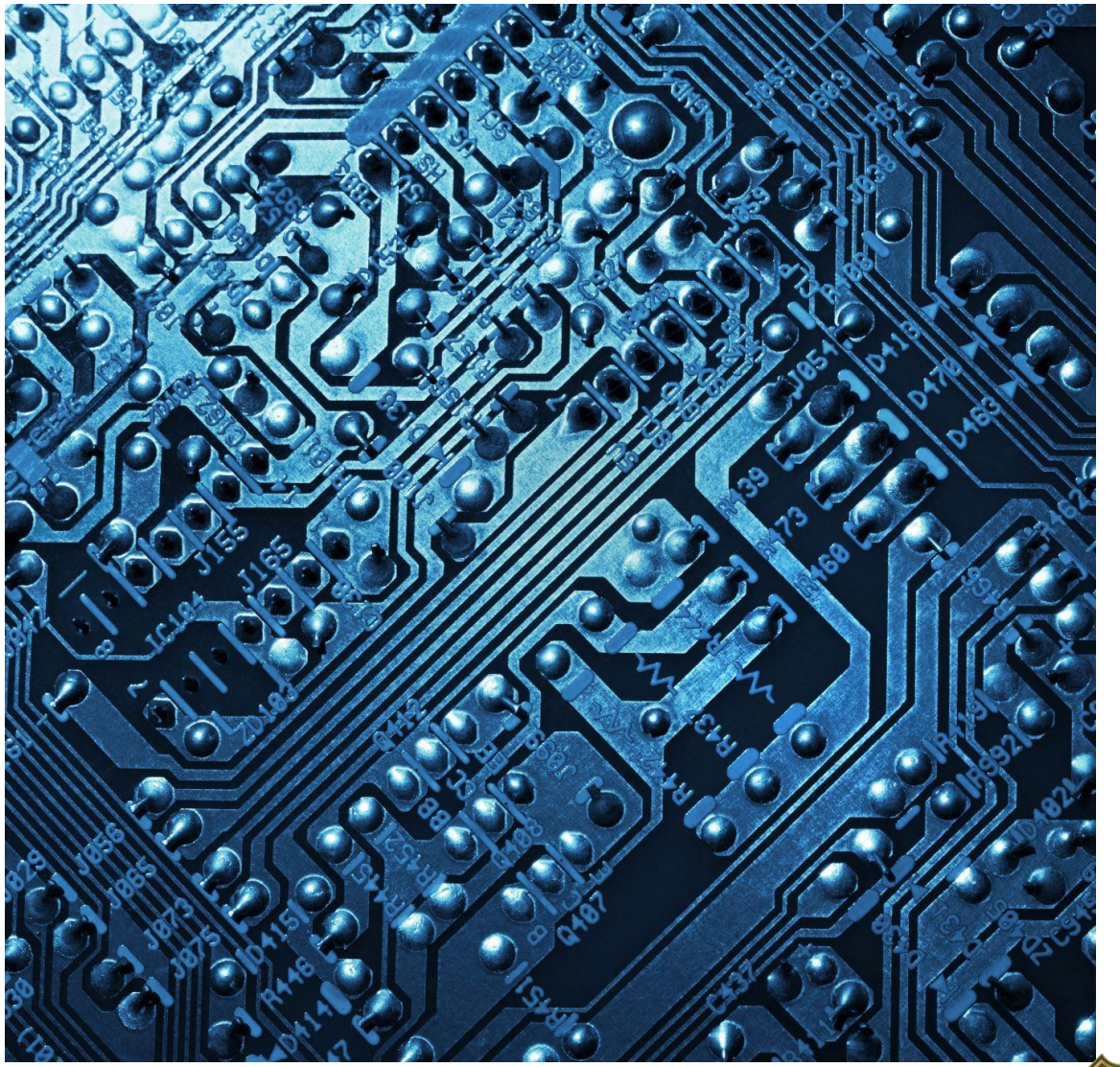
Data Augmentation

Data augmentation is a technique used to increase the amount and diversity of data available for training machine learning models without actually collecting new data. This is especially useful in situations where data is limited, as it can help improve the robustness and performance of models by providing more varied training examples.



Conclusion

Data preprocessing is a crucial step in the machine learning pipeline. It involves transforming raw data into a format that can be effectively and efficiently used by machine learning algorithms. The quality and relevance of the data greatly impact the performance of the models.




MY TOAST GOT BURNT
THIS MORNING
THANKS
OBAMA
quickmeme.com