

**Online Student Training for “Artificial Intelligence & Machine Learning”**  
**(4<sup>th</sup> Feb, 2021 – 17<sup>th</sup> Mar, 2021)**

# Data Preprocessing in Machine Learning


*Faculty Trainer*

*Naw Varsha Pipada*

*Asst. Prof.*

*Department of Computer Science &  
Engineering*

# Contents

- ▶ What is Data Preprocessing
  - ▶ What's the need
  - ▶ Steps for Data Preprocessing
  - ▶ Data Preprocessing Techniques
- 


# Data Preprocessing

- ▶ <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing>


The screenshot shows the scikit-learn website's documentation for the `sklearn.preprocessing` module. The page title is "sklearn.preprocessing: Preprocessing and Normalization". It includes a sidebar with navigation links (Prev, Up, Next), the scikit-learn logo, version information (0.24.1), and an API Reference section listing various modules like `sklearn.base`, `sklearn.calibration`, `sklearn.cluster`, `sklearn.compose`, `sklearn.covariance`, `sklearn.cross_decomposition`, `sklearn.datasets`, `sklearn.decomposition`, and `sklearn.discriminant_analysis`. The main content area describes the module's purpose (scaling, centering, normalization, binarization) and provides a user guide. Below this, a table lists various preprocessing classes and their functions.

<code>preprocessing.Binarizer(*[, threshold, copy])</code>	Binarize data (set feature values to 0 or 1) according to a threshold.
<code>preprocessing.FunctionTransformer([func, ...])</code>	Constructs a transformer from an arbitrary callable.
<code>preprocessing.KBinsDiscretizer([n_bins, ...])</code>	Bin continuous data into intervals.
<code>preprocessing.KernelCenterer()</code>	Center a kernel matrix.
<code>preprocessing.LabelBinarizer(*[, neg_label, ...])</code>	Binarize labels in a one-vs-all fashion.
<code>preprocessing.LabelEncoder()</code>	Encode target labels with value between 0 and <code>n_classes-1</code> .
<code>preprocessing.MultiLabelBinarizer(*[, ...])</code>	Transform between iterable of iterables and a multilabel format.
<code>preprocessing.MaxAbsScaler(*[, copy])</code>	Scale each feature by its maximum absolute value.
<code>preprocessing.MinMaxScaler([feature_range, ...])</code>	Transform features by scaling each feature to a given range.
<code>preprocessing.Normalizer([norm, copy])</code>	Normalize samples individually to unit norm.
<code>preprocessing.OneHotEncoder(*[, categories, ...])</code>	Encode categorical features as a one-hot numeric array.
<code>preprocessing.OrdinalEncoder(*[, ...])</code>	Encode categorical features as an integer array.
<code>preprocessing.PolynomialFeatures([degree, ...])</code>	Generate polynomial and interaction features.
<code>preprocessing.PowerTransformer([method, ...])</code>	Apply a power transform featurewise to make data more Gaussian-like.
<code>preprocessing.QuantileTransformer(*[, ...])</code>	Transform features using quantiles information.
<code>preprocessing.RobustScaler(*[, ...])</code>	Scale features using statistics that are robust to outliers.
<code>preprocessing.StandardScaler(*[, copy, ...])</code>	Standardize features by removing the mean and scaling to unit variance.


# Data Preprocessing

- ▶ It involves transforming raw data into an understandable format.
  - ▶ Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends and is likely to contain many errors.
  - ▶ Data Preprocessing is a proven method of resolving such issues.
  - ▶ We should assess the quality of data
    - Mismatching in data types
    - Different dimensions of data arrays
    - Mixture of data values
    - Outliers in the dataset
    - Missing Values
- 


# Why We Need Data Preprocessing?

- ▶ Real-world data tend to be incomplete, noisy, and inconsistent.
  - ▶ This can lead to a poor quality of collected data and further to a low quality of models built on such data.
  - ▶ In order to address these issues, data preprocessing provides operations which can organise the data into a proper form for better understanding in machine process.
- 

# Why We Need Data Preprocessing?

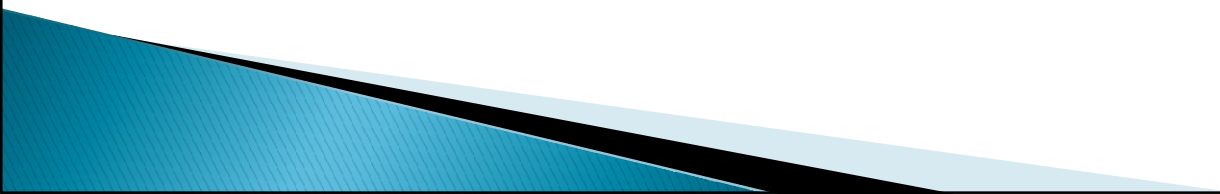
- ▶ Make our database more accurate
    - We eliminate the incorrect or missing values that are there as a result of the human factor or bugs.
  - ▶ Boost consistency
    - When there are inconsistencies in data or duplicates, it affects the accuracy of the results.
  - ▶ Make the database more complete
    - We can fill in the attributes that are missing if needed.
  - ▶ Smooth the data
    - This way we make it easier to use and interpret.
- 

# Steps in Data Preprocessing

- ▶ 1. Import libraries
  - ▶ 2. Read data
  - ▶ 3. Checking for missing values
  - ▶ 4. Checking for categorical data
  - ▶ 5. Standardize the data
  - ▶ 6. PCA transformation
  - ▶ 7. Data splitting
- 

# Data Preprocessing Techniques

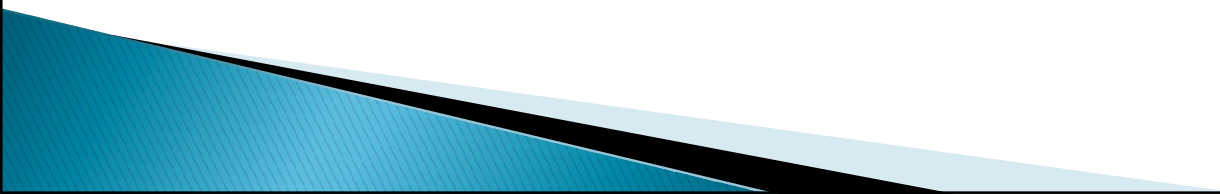
- ▶ Data Cleaning/Cleansing
- ▶ Data Transformation
- ▶ Data Reduction





# Data Cleaning

- ▶ Handling Missing data
- ▶ Handling Noisy Data



# Missing Data: Handling Null Values

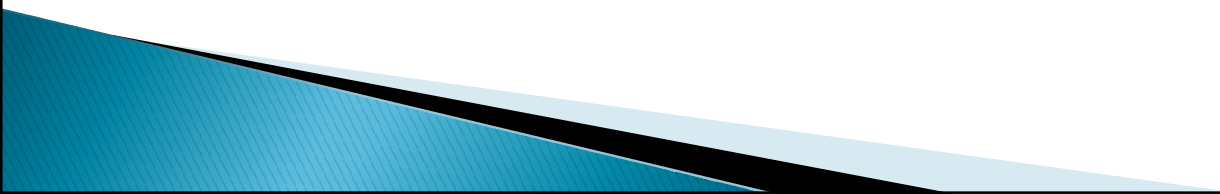
- ▶ In any real-world dataset there are always few null values.
- ▶ No model can handle these NULL or NaN values on its own so we need to intervene.
- ▶ To check whether we have null values in our dataset or not, `isnull()` method can be used

```
df.isnull()  
# Returns a boolean matrix, if the value is NaN then True otherwise  
False
```

```
df.isnull().sum()  
# Returns the column names along with the number of NaN values in  
that particular column
```

# Handling Null Values

- ▶ Dropping Rows and Columns with Null Values
- ▶ Imputation



# Handling Null Values

- ▶ 1. By dropping the rows or columns containing null values, dropna() method can be used

```
df.dropna()
```

- ▶ Various parameters of this function are:

- axis
  - We can specify axis=0 if we want to remove the rows and axis=1 if we want to remove the columns.
- how
  - If we specify how = 'all' then the rows and columns will only be dropped if all the values are NaN. By default how is set to 'any'.
- thresh
  - It determines the threshold value so if we specify thresh=5 then the rows having less than 5 real values will be dropped.
- subset
  - If we have 4 columns A, B, C and D then if we specify subset=['C'] then only the rows that have their C value as NaN will be removed.
- inplace
  - By default, no changes will be made to your dataframe. So if you want these changes to reflect onto your dataframe then you need to use inplace = True.

# Handling Null Values

## ▶ 2. Imputation

- Imputation is simply the process of substituting the missing values of our dataset.
- We can do this by defining our own customised function or we can simply perform imputation by using the **SimpleImputer** class provided by sklearn.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(df[['Weight']])
df['Weight'] = imputer.transform(df[['Weight']])
```

# Handling Null Values

## ▶ Imputer class takes in a few parameters

- `missing_values`
  - This is the value which has to be replaced in the dataset. This could either be an integer, or NaN. NaN will be the default value.
- `strategy`
  - This is the strategy we'll be using to calculate the value which has to replace the NaN occurrences in the dataset. There are three different strategies we can use mean, median and most\_frequent. "mean" is the default value here.
- `axis`
  - This can take one of two values — 0 and 1. This will decide if the Imputer will apply the strategy along the rows or along the columns. 0 for columns, and 1 for rows.
- `verbose`
  - This will just decide the verbosity of the Imputer. By default, it's set to 0.
- `copy`
  - This will decide if a copy of the original object has to be made, or if the Imputer should change the dataset in-place. By default, it is set to True.

# Handling Null Values

- ▶ Some other methods can also be used to impute like fillna() method

```
df.fillna(df.mean())
```

# Noisy Data

- ▶ A large amount of additional meaningless data is called *noise*.
  - duplicates or semi-duplicates of the data records;
  - data segments, which have no value for a particular research;
  - unnecessary information fields for each of the variables.
- ▶ Possible solution methods
  - Binning
  - Regression
  - Clustering



# Data Transformation

- ▶ Data transformation consists of the methods of turning the data into an appropriate format for the computer to learn from.
  - Encoding Categorical Variables
  - Feature Scaling
  - Discretization
  - Generalization

# Encoding Categorical Variables

- ▶ Sometimes our data is in qualitative form, that is we have texts as our data. We can find categories in text form
- ▶ To encode such categorical variables, *LabelEncoder* library can be used.

```
from sklearn.preprocessing import LabelEncoder

labelencoder_X = LabelEncoder()

X[:,0] = labelencoder_X.fit_transform(X[:,0])
```

Difficulty saving money	Counts	Frequencies
Very	231	46%
Somewhat	196	39%
Not very	58	12%
Not at all	14	3%
Not sure	1	~0%
Total	500	100%

# Example: map() method

```
1 import pandas as pd
2 df_cat = pd.DataFrame(data =
3     [['green', 'M', 10.1, 'class1'],
4     ['blue', 'L', 20.1, 'class2'],
5     ['white', 'M', 30.1, 'class1']])
6 df_cat.columns = ['color', 'size', 'price', 'classlabel']
7 print(df_cat)
8 size_mapping = {'M':1, 'L':2}
9 df_cat['size'] = df_cat['size'].map(size_mapping)
10 print(df_cat)
```

	color	size	price	classlabel
0	green	M	10.1	class1
1	blue	L	20.1	class2
2	white	M	30.1	class1

	color	size	price	classlabel
0	green	1	10.1	class1
1	blue	2	20.1	class2
2	white	1	30.1	class1

## Example – LabelEncoder()

```
1 import pandas as pd
2 df_cat = pd.DataFrame(data =
3     [['green', 'M', 10.1, 'class1'],
4     ['blue', 'L', 20.1, 'class2'],
5     ['white', 'M', 30.1, 'class1']])
6 df_cat.columns = ['color', 'size', 'price', 'classlabel']
7 print(df_cat)
8 size_mapping = {'M':1, 'L':2}
9 df_cat['size'] = df_cat['size'].map(size_mapping)
10 print(df_cat)
11 from sklearn.preprocessing import LabelEncoder
12 class_le = LabelEncoder()
13 df_cat['classlabel'] = class_le.fit_transform(df_cat['classlabel'].values)
14 print(df_cat)
```

	color	size	price	classlabel
0	green	M	10.1	class1
1	blue	L	20.1	class2
2	white	M	30.1	class1

	color	size	price	classlabel
0	green	1	10.1	class1
1	blue	2	20.1	class2
2	white	1	30.1	class1

	color	size	price	classlabel
0	green	1	10.1	0
1	blue	2	20.1	1
2	white	1	30.1	0

# Feature Scaling

## ► Normalization

- 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.

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

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

# Feature Scaling : Example

```
# 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 data
X_test_norm = norm.transform(X_test)
```

# Feature Scaling

## ▶ Standardization

- Standardization is another scaling technique 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.

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

- ▶  $\mu$  is the mean of the feature values and  $\sigma$  is the standard deviation of the feature values.

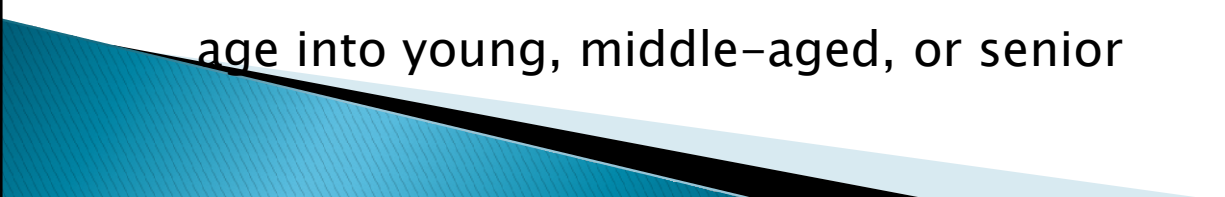
# Feature Scaling : Example

```
from sklearn.preprocessing import StandardScaler  
sc_X = StandardScaler()
```

```
X_train = sc_X.fit_transform(X_train)  
X_test = sc_X.transform(X_test)
```



# Discretization & Generalization

- ▶ Data discretization is defined as a process of converting continuous data attribute values into a finite set of intervals with minimal loss of information
  - ▶ Data Generalization is to convert low-level data features to high-level data features. For example, house addresses can be generalized to higher-level definitions, such as town or country. Similarly, the values for numeric attributes may be mapped to higher level concepts like, age into young, middle-aged, or senior
- 

# Data Reduction

## ▶ Feature Selection

- Feature Selection is the process of selecting a subset of relevant features for use in model construction
- Feature selection methods can be used to identify and remove unneeded, irrelevant and redundant attributes from data that do not contribute to the accuracy of a predictive model or may in fact decrease the accuracy of the model.

## ▶ Dimensionality Reduction

- Dimensionality reduction, or dimension reduction, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension

## ▶ Numerosity reduction

- Numerosity Reduction is a data reduction technique which replaces the original data by smaller form of data representation.

Thank You!!!

