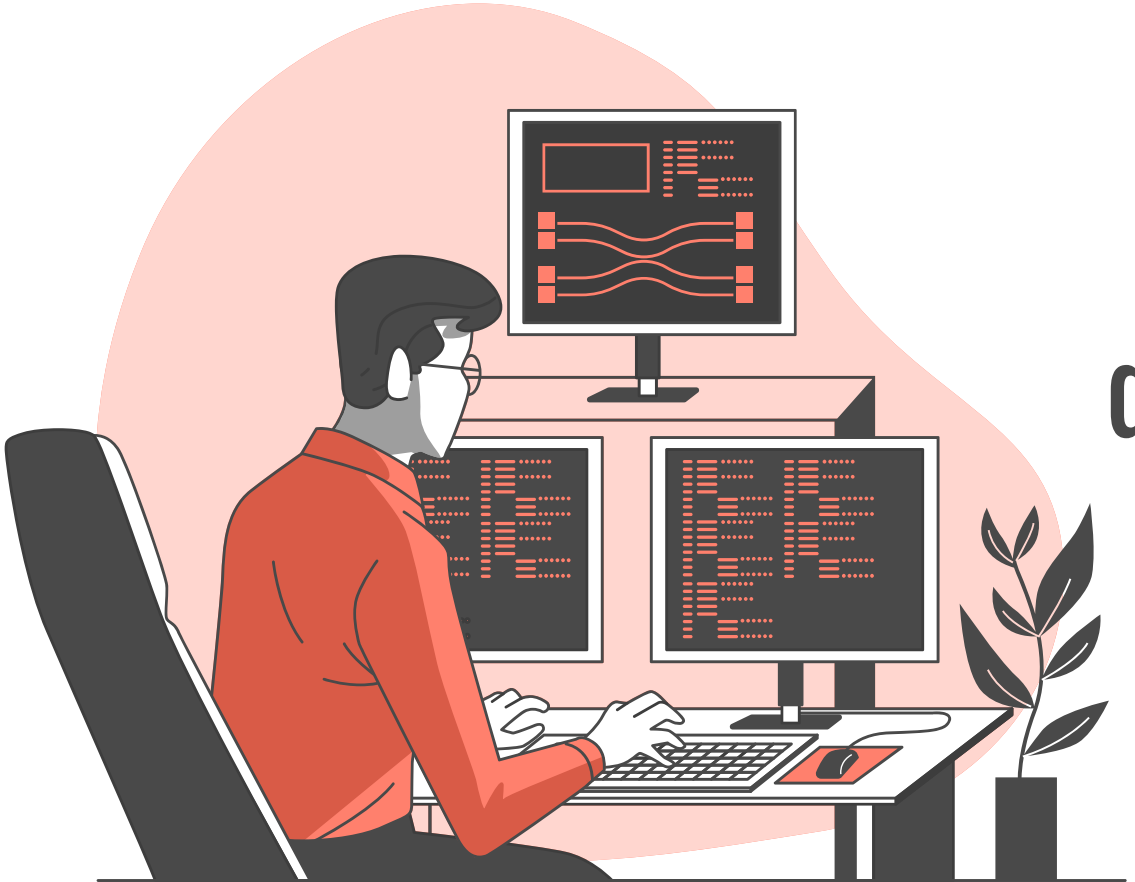


# Preprocessing data (scikit learn)

Renan Carlos Loewenstein

# Introduction

- **What is preprocessing data?**
  - Preprocessing data is a crucial step in machine learning as it involves transforming raw data into a format that can be easily understood and analyzed by algorithms
- **Why it is important?**
  - It helps to prepare data for analysis and modeling, improving the accuracy and efficiency of the dataset

# Preprocessing techniques

- **Scaling**
  - Adjust the values of features in a dataset to a standard range, often between 0 to 1. Scaling helps compare features with different units or widely varying ranges.

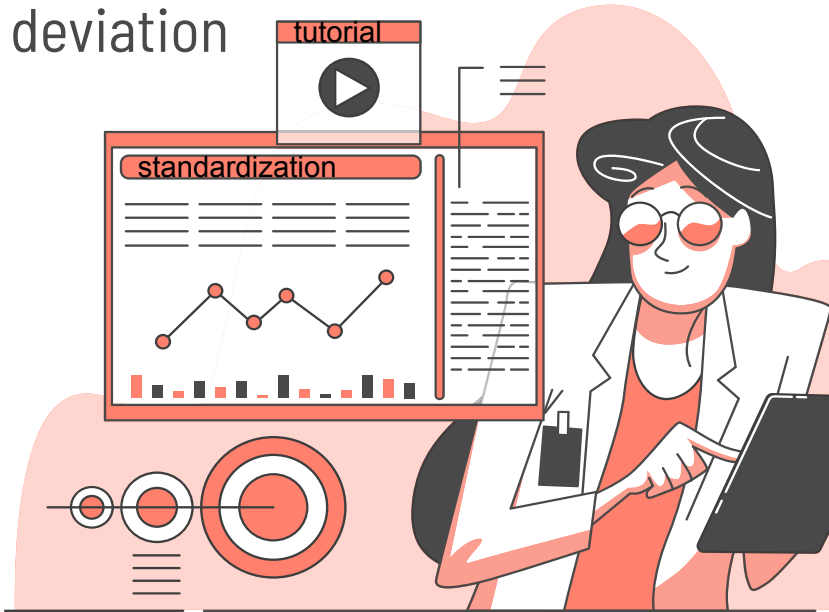
# Scaling Example

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
...
>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[0.5       , 0.        , 1.        ],
       [1.        , 0.5       , 0.33333333],
       [0.        , 1.        , 0.        ]])
```

MinMaxScaler: transforms all the input variables, so they're all on the same scale between zero and one

# Preprocessing techniques

- **Standardization**
  - Transform the variables of a dataset so that they follow a normal distribution with a mean of zero and a standard deviation of one.



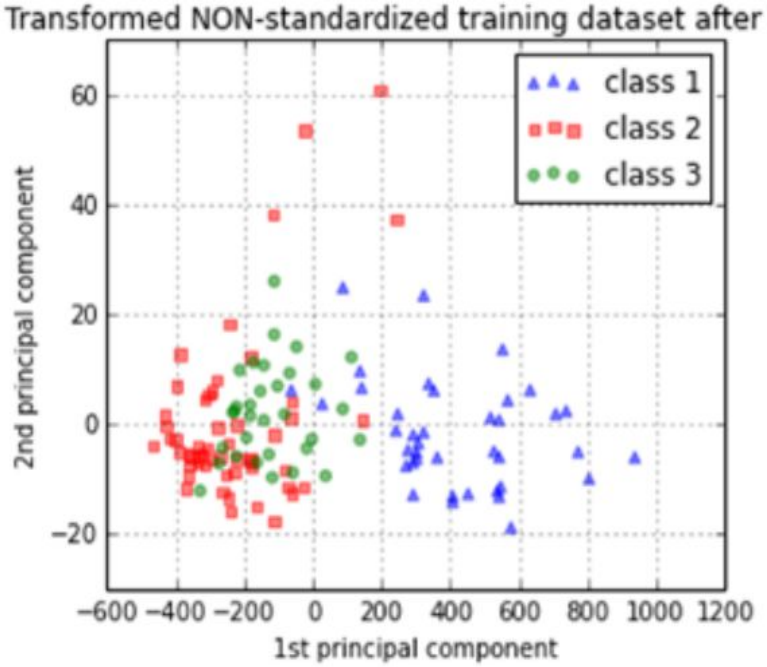
# Standardization Example

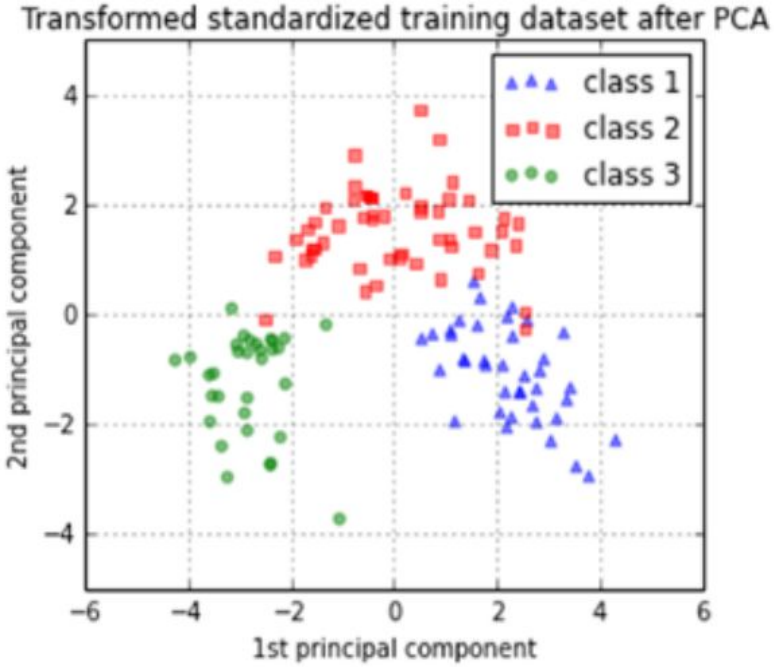
```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> scaler
StandardScaler()

>>> scaler.mean_
array([1. ..., 0. ..., 0.33...])

>>> scaler.scale_
array([0.81..., 0.81..., 1.24...])

>>> X_scaled = scaler.transform(X_train)
>>> X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

- **Why it is important?**
  - It scales the data to a common range, preventing some variables from dominating others in the analysis

StandardScaler: brings all features to the same magnitude (mean is 0 and variance is 1)

# Standardization Example



# Preprocessing techniques

- **Normalization**
  - Rescale numeric data in a dataset to a common scale, typically between 0 and 1, without changing their distribution.

```
>>> X = [[ 1., -1.,  2.],
...      [ 2.,  0.,  0.],
...      [ 0.,  1., -1.]]
>>> X_normalized = preprocessing.normalize(X, norm='l2')

>>> X_normalized
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])
```

Normalizer: scales each data point such that the feature vector has a Euclidian length of 1. Every data point is scaled by a different number (by the inverse of its length)

# Normalization vs Standardization

## Normalization

Rescales values to a range between 0 and 1

Useful when the distribution of the data is unknown or not Gaussian

Sensitive to outliers

Retains the shape of the original distribution

May not preserve the relationship between the data points

## Standardization

Centers data around the mean and scales to a standard deviation of 1

Useful when the distribution of the data is Gaussian or unknown

Less sensitive to outliers

Changes the shape of the original distribution

´Preserves the relationships between the data points

# Preprocessing techniques

- **Encoding**
  - Is the process of converting data from one representation to another
  - Is commonly used to convert categorical data, which represents qualitative variables, into numerical data that can be used by machine learning algorithms

# Encoding Example

```
>>> enc = preprocessing.OrdinalEncoder(encoded_missing_value=-1)
>>> X = [['male'], ['female'], [np.nan], ['female']]
>>> enc.fit_transform(X)
array([[ 1.],
       [ 0.],
       [-1.],
       [ 0.]])
```

- There are different encoding techniques such as one-hot encoding, label encoding, and binary encoding

# Preprocessing techniques

- **Discretization**
  - Is a technique that can group continuous data into categories. This can be useful for datasets with continuous data, since it can transform them into datasets with only nominal attributes.

# Discretization Example

```
>>> X = [[ 1., -1.,  2.],
...      [ 2.,  0.,  0.],
...      [ 0.,  1., -1.]]

>>> binarizer = preprocessing.Binarizer().fit(X)  # fit does nothing
>>> binarizer
Binarizer()

>>> binarizer.transform(X)
array([[1., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.]])
```

# Preprocessing techniques

- **Imputation of missing values**
  - The SimpleImputer class provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located. This class also allows for different missing values encodings.

# Preprocessing techniques

- **Imputation of missing values**

```
>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
SimpleImputer()
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> print(imp.transform(X))
[[4.          2.        ]
 [6.          3.666...]
 [7.          6.        ]]
```

# Preprocessing techniques

- **Imputation of missing values**

```
>>> import pandas as pd
>>> df = pd.DataFrame([["a", "x"],
...                    [np.nan, "y"],
...                    ["a", np.nan],
...                    ["b", "y"]], dtype="category")
...
>>> imp = SimpleImputer(strategy="most_frequent")
>>> print(imp.fit_transform(df))
[['a' 'x']
 ['a' 'y']
 ['a' 'y']
 ['b' 'y']]
```