Strive

SCHOOL

/ Numerical Encoding

Numerical Features

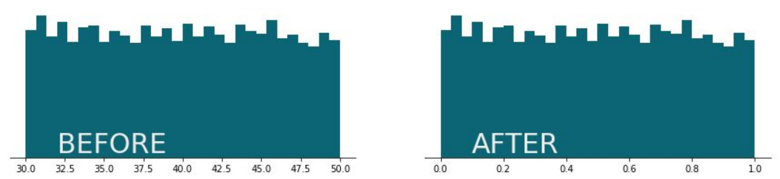
/ Numerical features are:

- Discrete numbers
  - Example: Age of the person.
- Continuous numbers
  - Example: Height of the person.
  - Example: Weight of the person.

Sklearn methods

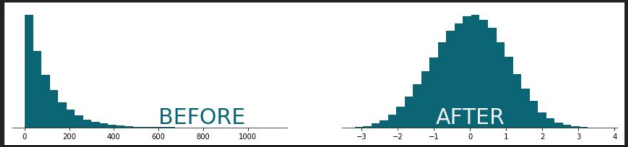
A) Scaling

- Min-max scaling `MinMaxScaler()`
- Max-abs scaling `MaxAbsScaler()`
- Standard scaling `StandardScaler()`
- Robust scaling `RobustScaler()`



B) Normalization

- Manually
  - Logarithm `np.log(1+x)`
  - Square root `np.sqrt(x+2/3)`
- PowerTransformer()
  - Box-Cox
  - Yeo-Johnson
- QuantileTransformer()
  - (aka GaussRank)



Other Sklearn methods

C) Create groups

- Binarize data `Binarizer()`
  - Set feature values to 0 or 1 according to a threshold.
- Create bins `KBinsDiscretizer()`
  - Bin continuous data into intervals.

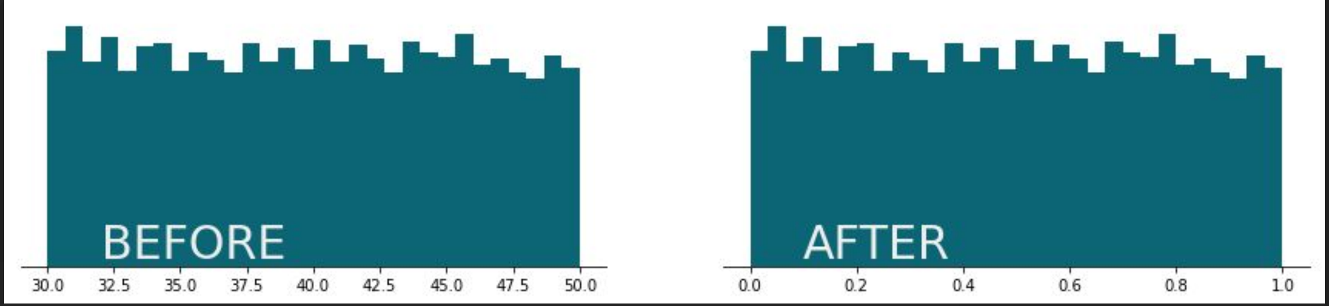
D) Create more features

- PolynomialFeatures()
  - Generate polynomial and interaction features.

This is useful for linear models only

/ A) Scaling

Transforming your data so that it fits within a specific scale, like 0-100 or 0-1.

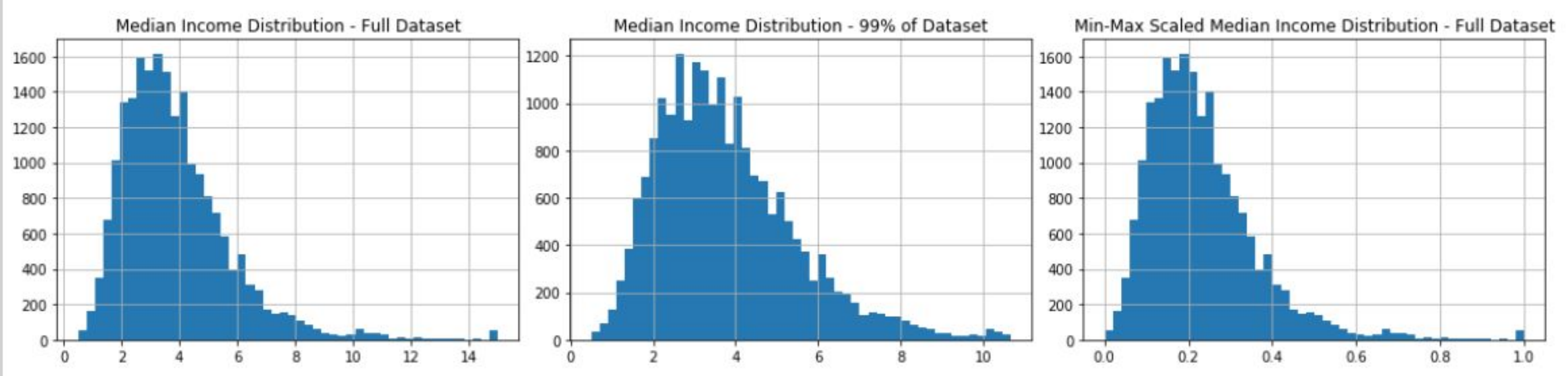


Min-max scaling

MinMaxScaler()

/ Rescales the data set such that all feature values are in the range [0, 1]

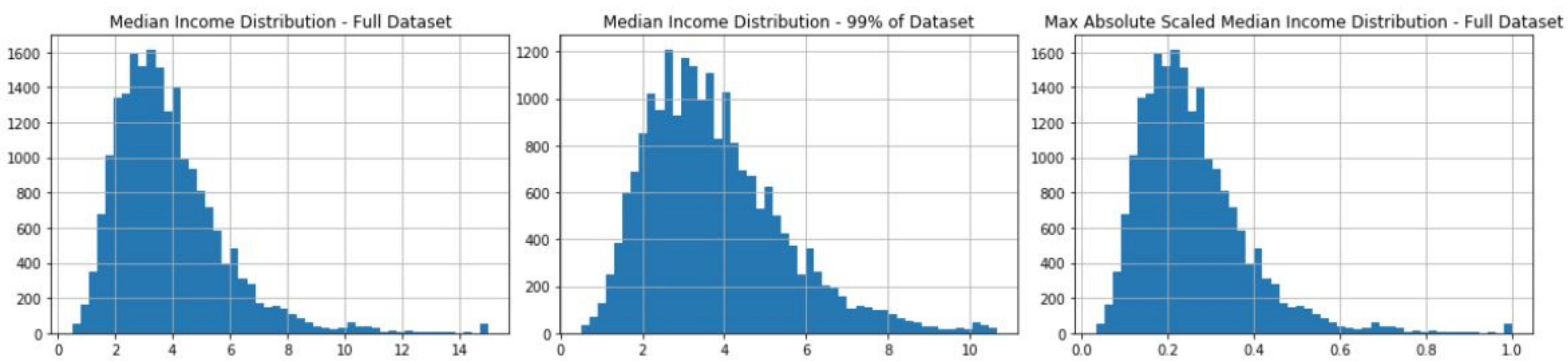
$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$



Max-abs scaling

MaxAbsScaler()

/ Scales and translates each feature individually such that the maximal absolute value will be 1.0

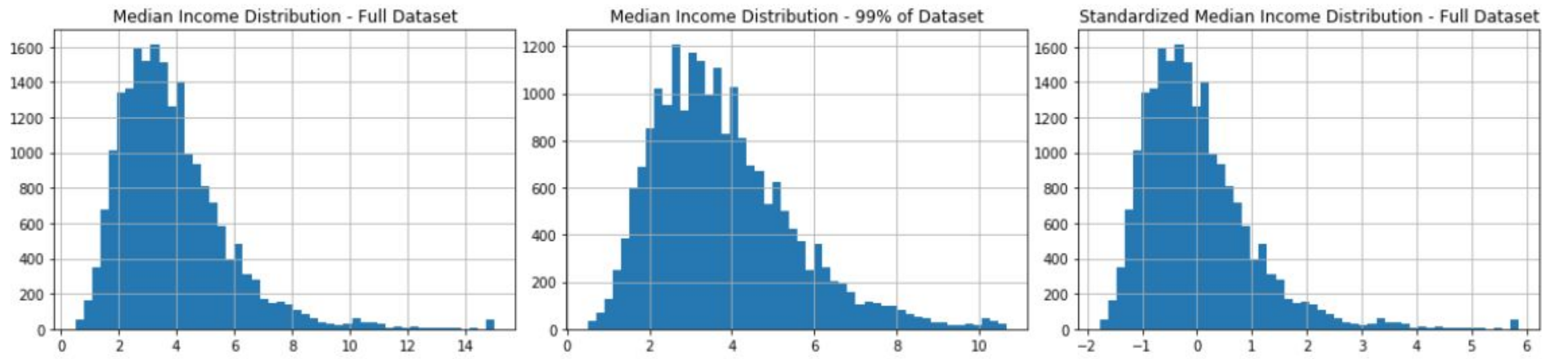


Standard scaling

StandardScaler()

/ Removes the mean and scaling to unit variance.

$$x' = \frac{x - \bar{x}}{\sigma}$$


Median Income Distribution - Full Dataset

Median Income Distribution - 99% of Dataset

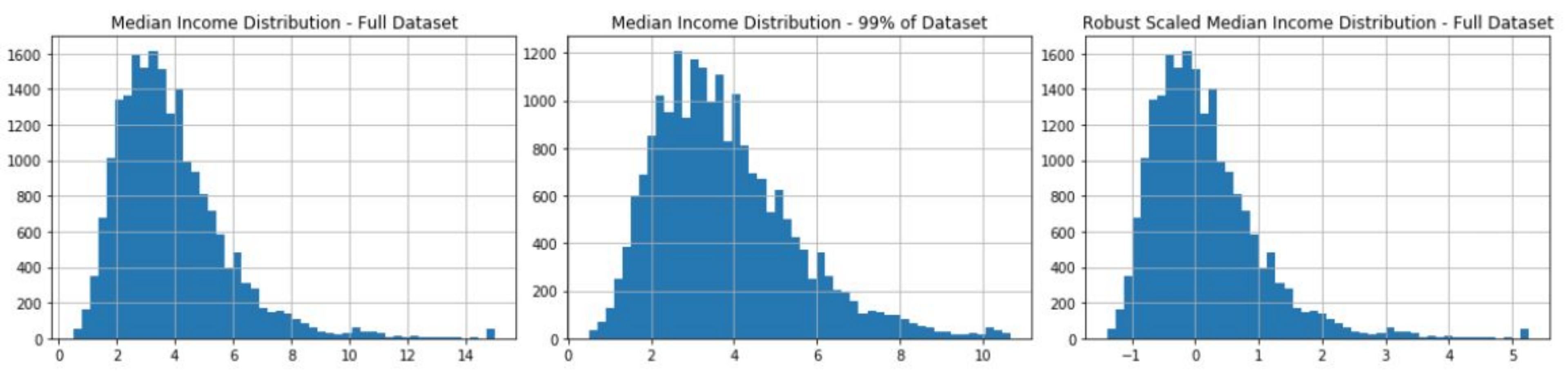
Standardized Median Income Distribution - Full Dataset

Robust scaling

RobustScaler()

/ Removes the median and scales the data according to a given quantile range. Defaults to the Interquartile Range (IQR). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$



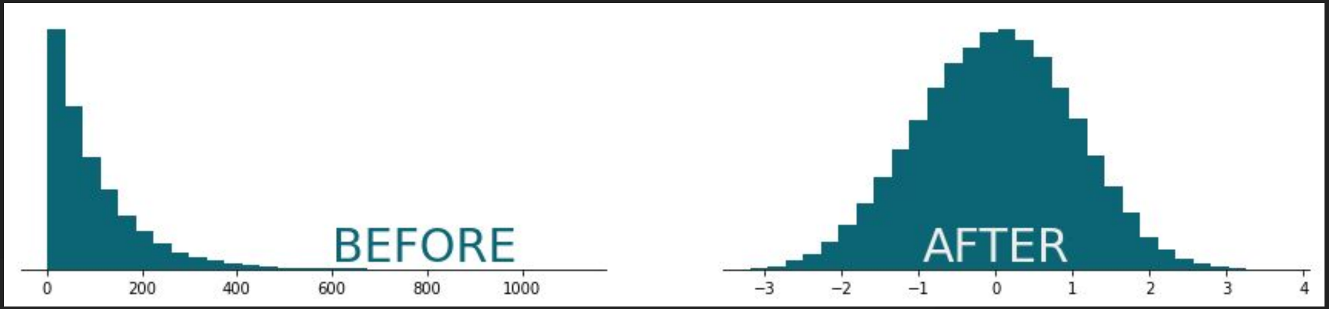
Median Income Distribution - Full Dataset

Median Income Distribution - 99% of Dataset

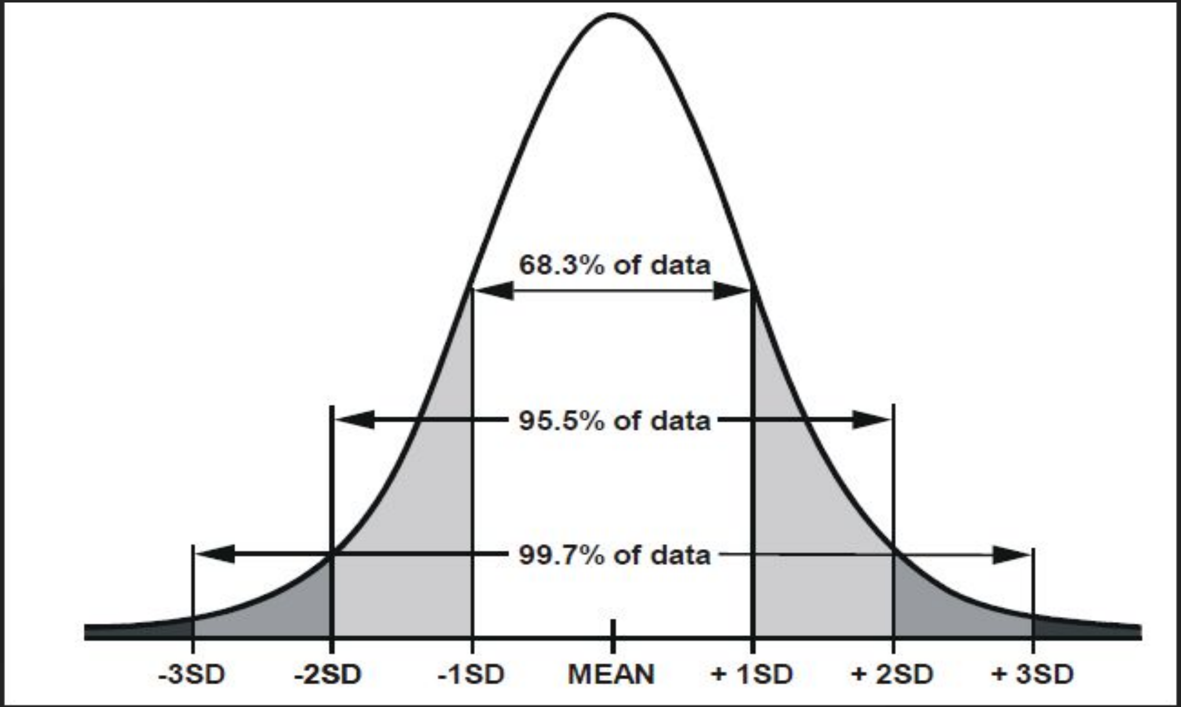
Robust Scaled Median Income Distribution - Full Dataset

/ B) Normalization

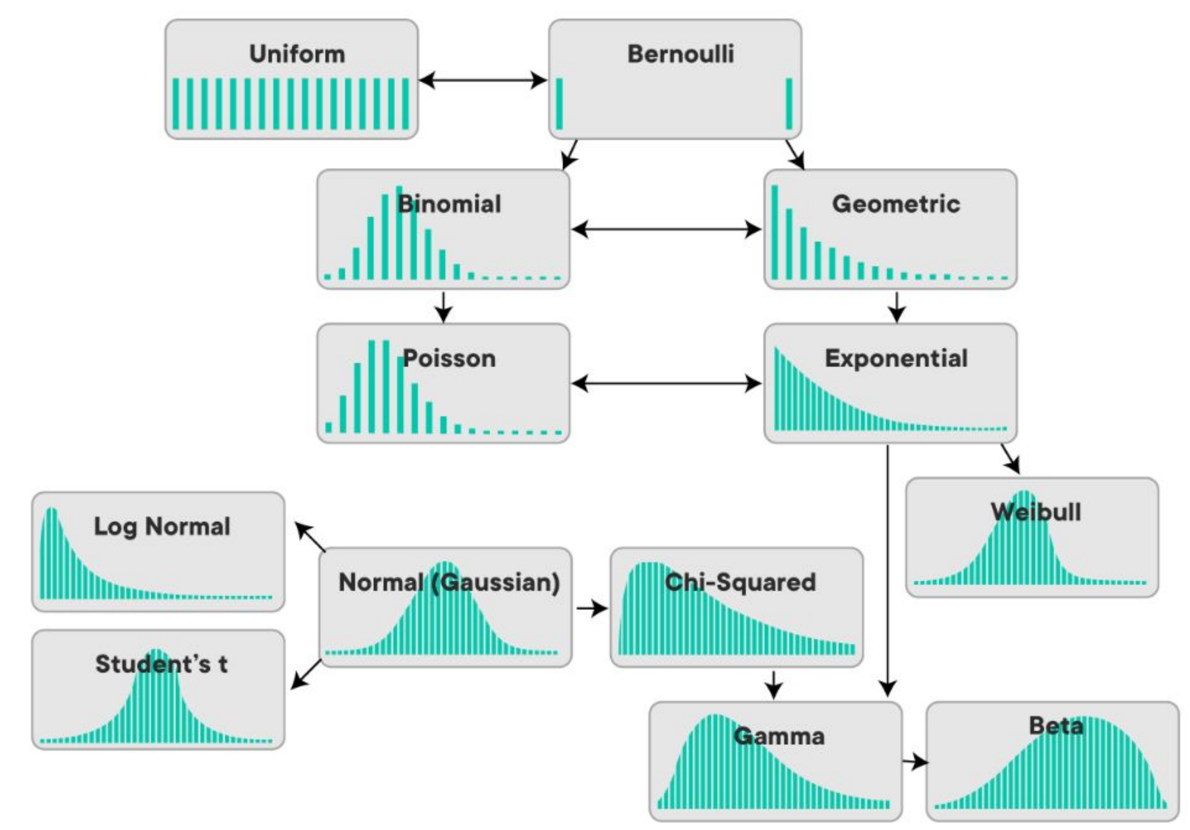
Changing the shape of the distribution to a Normal distribution ("bell curve")



Normal distribution (aka Gaussian distribution)



Other types of distributions

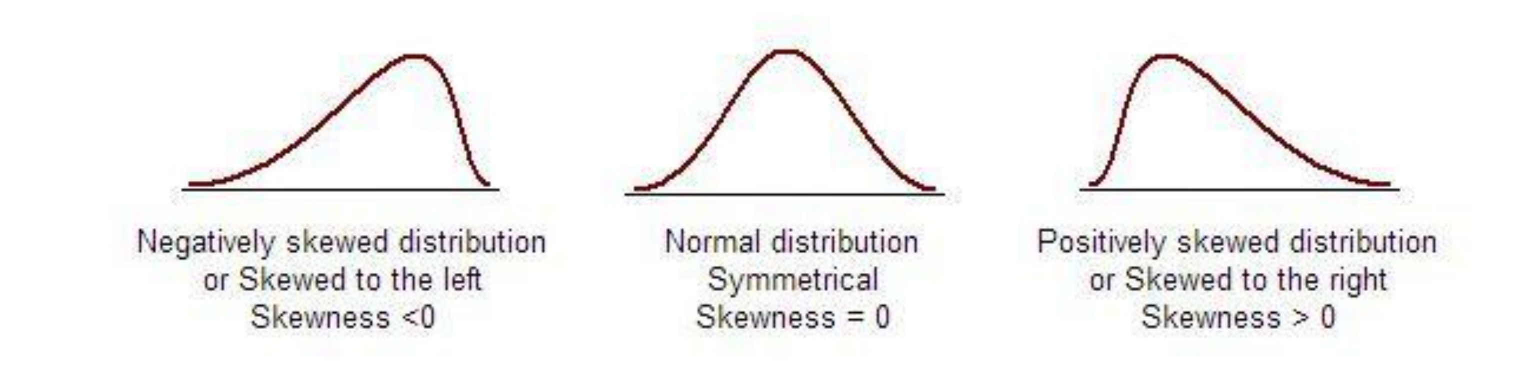


Skewness

`pandas.skew()`

/ A number to determine the asymmetry of the distribution.

/ Normal distribution have skewness = 0



Negatively skewed distribution
or Skewed to the left
Skewness <0

Normal distribution
Symmetrical
Skewness = 0

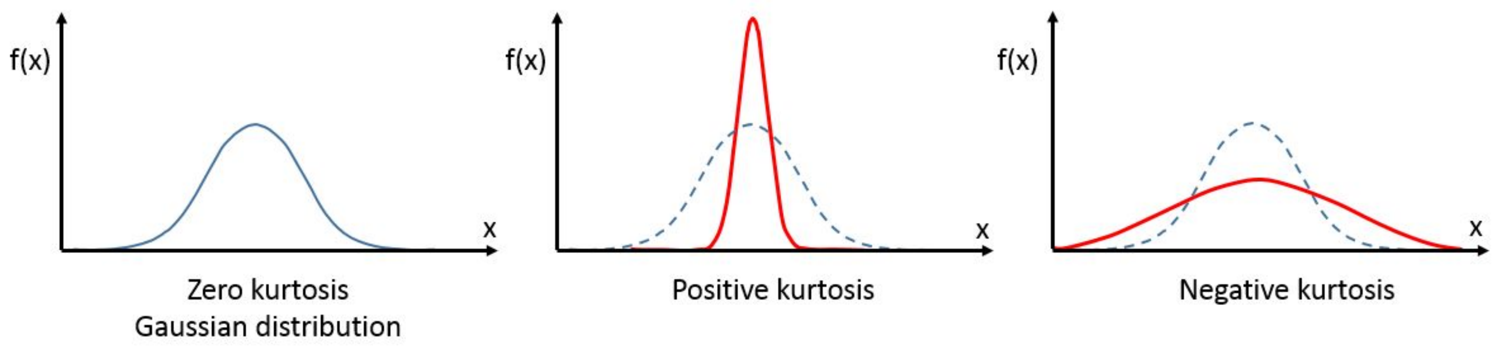
Positively skewed distribution
or Skewed to the right
Skewness > 0

Kurtosis

`pandas.kurt()`

/ from Greek: κυρτός meaning "curved, arching" is a measure of the "tailedness" of the distribution.

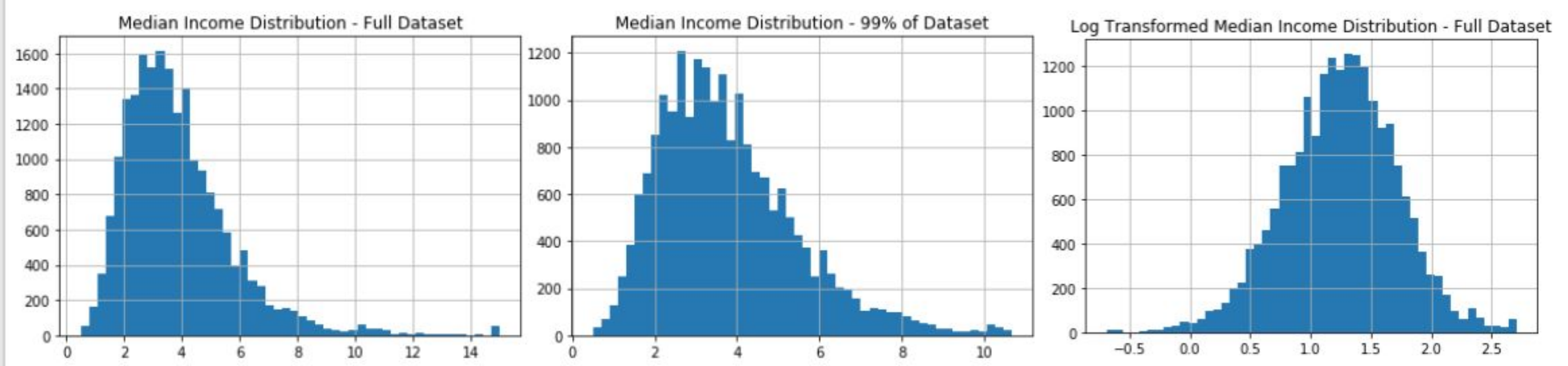
/ Normal distribution have kurtosis = 0



Logarithm

np.log()

- Common heuristic to normalize data. Usually combined with StandardScaler()



Median Income Distribution - Full Dataset

Median Income Distribution - 99% of Dataset

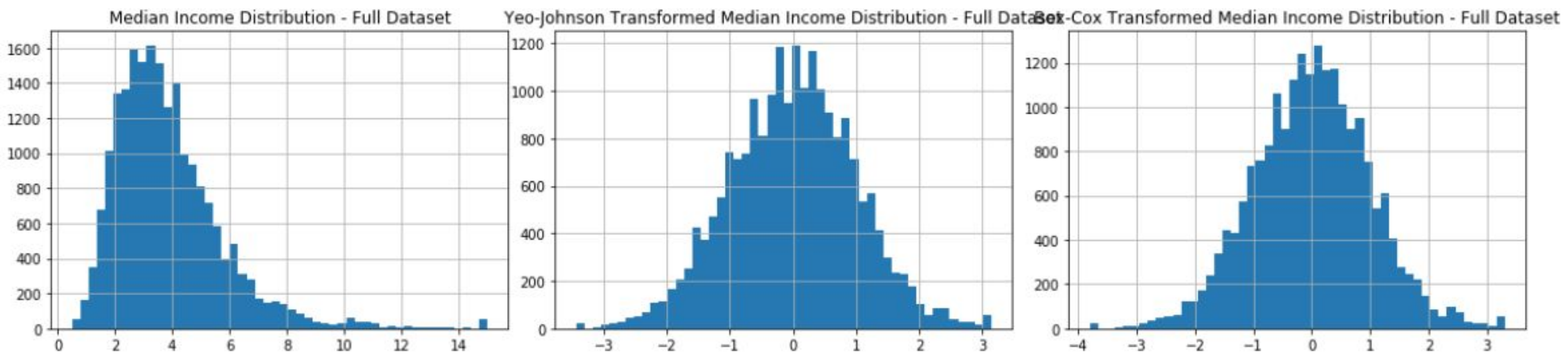
Log Transformed Median Income Distribution - Full Dataset

BoxCox and Yeo-Johnson

PowerTransformer()

/ Good methods:

- BoxCox: Can only be used for positive values
  - `PowerTransformer(method="box-cox")`
- Yeo-Johnson: Similar to Box-cox but can be used for negative values.
  - `PowerTransformer(method="yeo-johnson")`



Median Income Distribution - Full Dataset

Yeo-Johnson Transformed Median Income Distribution - Full Dataset

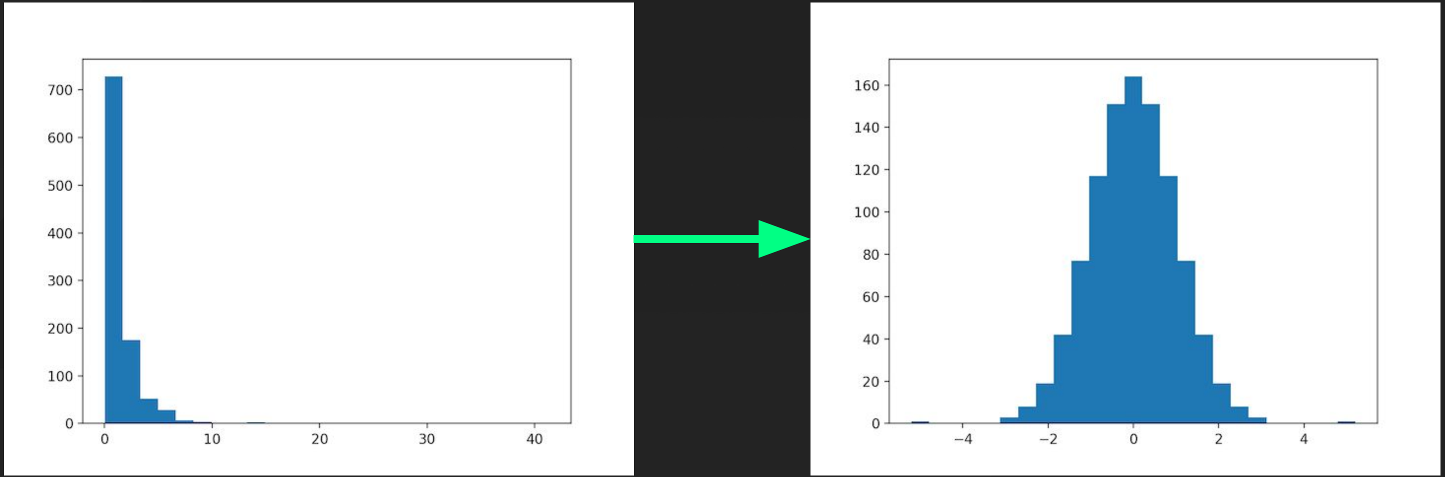
Box-Cox Transformed Median Income Distribution - Full Dataset

QuantileTransfomer

QuantileTransformer()

/ Probably the best normalizer. Makes a nearly perfect normal distribution!

```
QuantileTransformer( n_quantiles=100,    # 100 is a good hyperparameter
                     output_distribution="normal",
                     random_state=0)
```



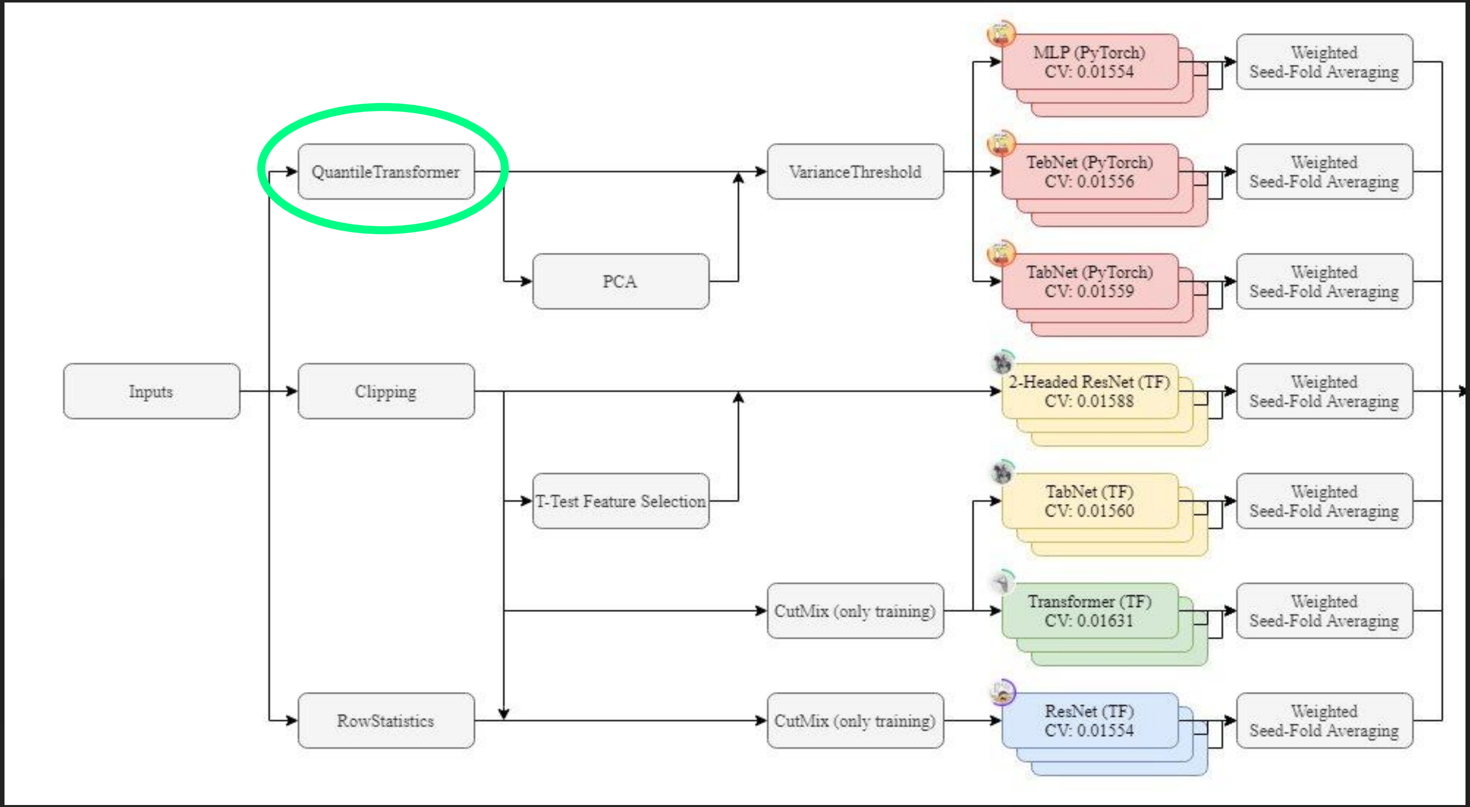
Rank Gauss

/ Probably the best normalizer. Makes a nearly perfect normal distribution!

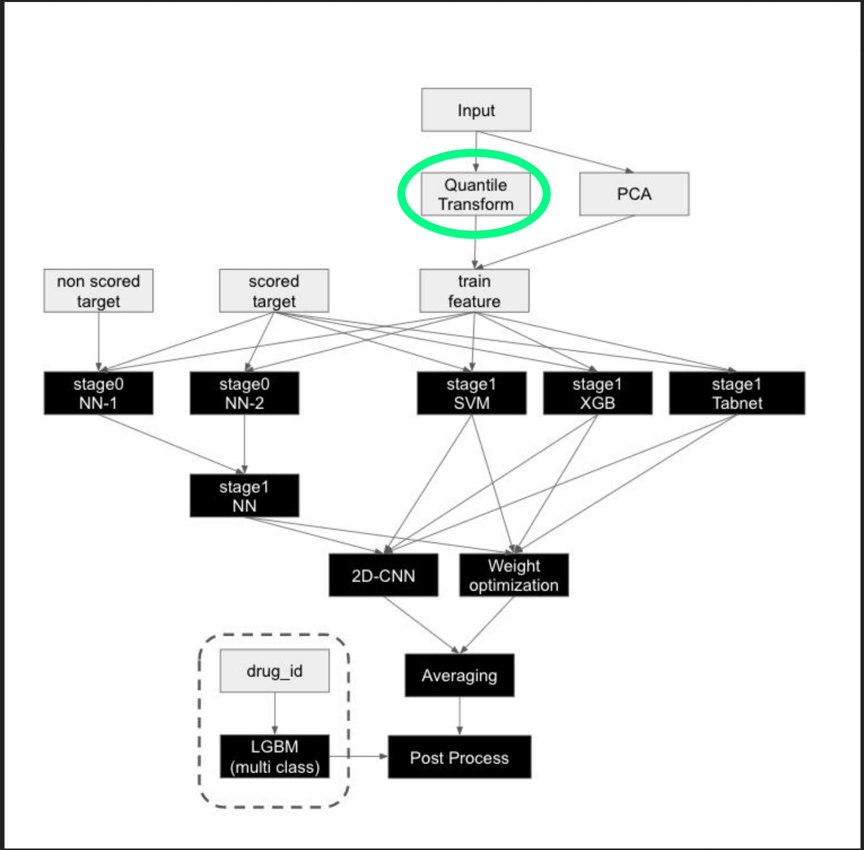
```python
from scipy.special import erfinv

def rank_gauss(x):
    n      = x.shape[0] # Number of samples (rows)
    temp   = x.argsort()
    rank_x = temp.argsort() / n
    rank_x = rank_x.mean()
    rank_x *= 2
    efi_x  = erfinv(rank_x)
    efi_x  -= efi_x.mean()
    return efi_x
```

Kaggle example


Inputs
QuantileTransformer
Clipping
RowStatistics
PCA
T-Test Feature Selection
VarianceThreshold
CutMix (only training)
CutMix (only training)
MLP (PyTorch) CV: 0.01554
TebNet (PyTorch) CV: 0.01556
TabNet (PyTorch) CV: 0.01559
2-Headed ResNet (TF) CV: 0.01588
TabNet (TF) CV: 0.01560
Transformer (TF) CV: 0.01631
ResNet (TF) CV: 0.01554
Weighted Seed-Fold Averaging
Weighted Seed-Fold Averaging
Weighted Seed-Fold Averaging
Weighted Seed-Fold Averaging
Weighted Seed-Fold Averaging
Weighted Seed-Fold Averaging
Weighted Seed-Fold Averaging

Kaggle example



/ Advice for Preprocessing numerical feats

The rule of thumb is:

- Tree models → Does not need anything
- Other models:
  - Simple approach → `StandardScaler()` (and `log()` when needed)
  - Advanced approach → `QuantileTransformer()`

Feature Generation

- Variable combination (division, multiplication, etc)
  - For example if we have the variable "square meters" and "house price" we can get the variable "price per square meter" for free.
- Variable transformation (root, log, square,)

/ Q&A

What are your doubts?

Strive
SCHOOL