

Disciplina:

MODELAGEM E PREPARAÇÃO DE DADOS PARA APRENDIZADO DE MÁQUINA

Professor: Rafael Barroso



PARTE 1

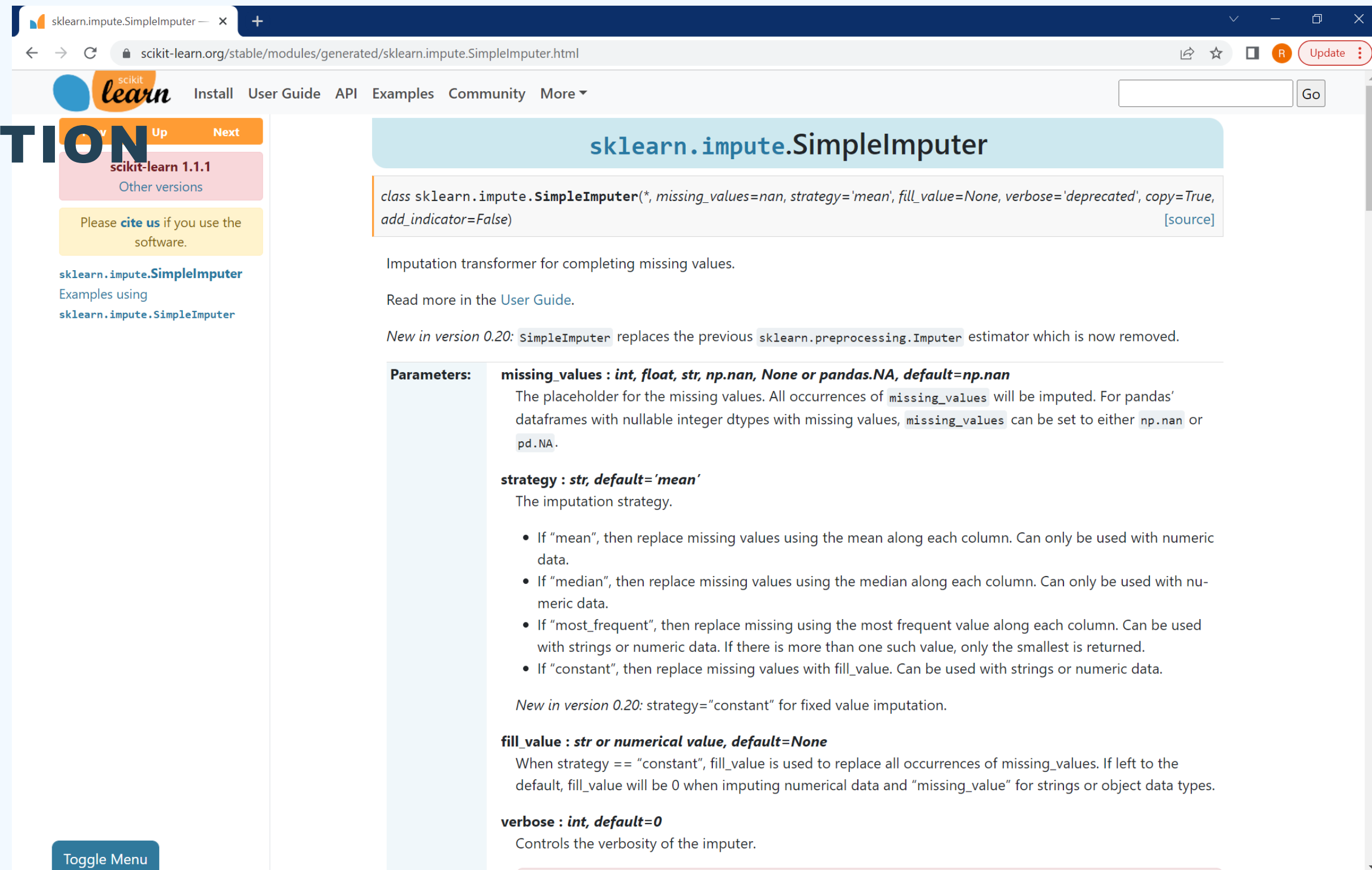
TRATAMENTO DE DADOS NULOS

TRATAMENTO DE DADOS NULOS

STATISTICAL IMPUTATION

- Faz uso de medidas de tendência para preencher dados ausentes;
- São de fácil entendimento, rápido cálculo e costumam gerar boa performance de algoritmos de aprendizado de máquina;
- Para efetivo preenchimento, pode fazer uso de:
 - Média,
 - Moda,
 - Mediana,
 - Constante.

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS STATISTICAL IMPUTATION



The screenshot shows the documentation for `sklearn.impute.SimpleImputer` on the scikit-learn website. The page includes a sidebar with navigation links, a main header for the class, a code block for the class signature, a description of the imputation transformer, and a detailed list of parameters with their descriptions and default values.

sklearn.impute.SimpleImputer

```
class sklearn.impute.SimpleImputer(*, missing_values=nan, strategy='mean', fill_value=None, verbose='deprecated', copy=True, add_indicator=False)
```

Imputation transformer for completing missing values.

Read more in the [User Guide](#).

New in version 0.20: `SimpleImputer` replaces the previous `sklearn.preprocessing.Imputer` estimator which is now removed.

Parameters:

- missing_values :** *int, float, str, np.nan, None or pandas.NA, default=np.nan*
The placeholder for the missing values. All occurrences of `missing_values` will be imputed. For pandas' dataframes with nullable integer dtypes with missing values, `missing_values` can be set to either `np.nan` or `pd.NA`.
- strategy :** *str, default='mean'*
The imputation strategy.
 - If "mean", then replace missing values using the mean along each column. Can only be used with numeric data.
 - If "median", then replace missing values using the median along each column. Can only be used with numeric data.
 - If "most_frequent", then replace missing using the most frequent value along each column. Can be used with strings or numeric data. If there is more than one such value, only the smallest is returned.
 - If "constant", then replace missing values with `fill_value`. Can be used with strings or numeric data.

New in version 0.20: `strategy="constant"` for fixed value imputation.
- fill_value :** *str or numerical value, default=None*
When `strategy == "constant"`, `fill_value` is used to replace all occurrences of `missing_values`. If left to the default, `fill_value` will be 0 when imputing numerical data and "missing_value" for strings or object data types.
- verbose :** *int, default=0*
Controls the verbosity of the imputer.

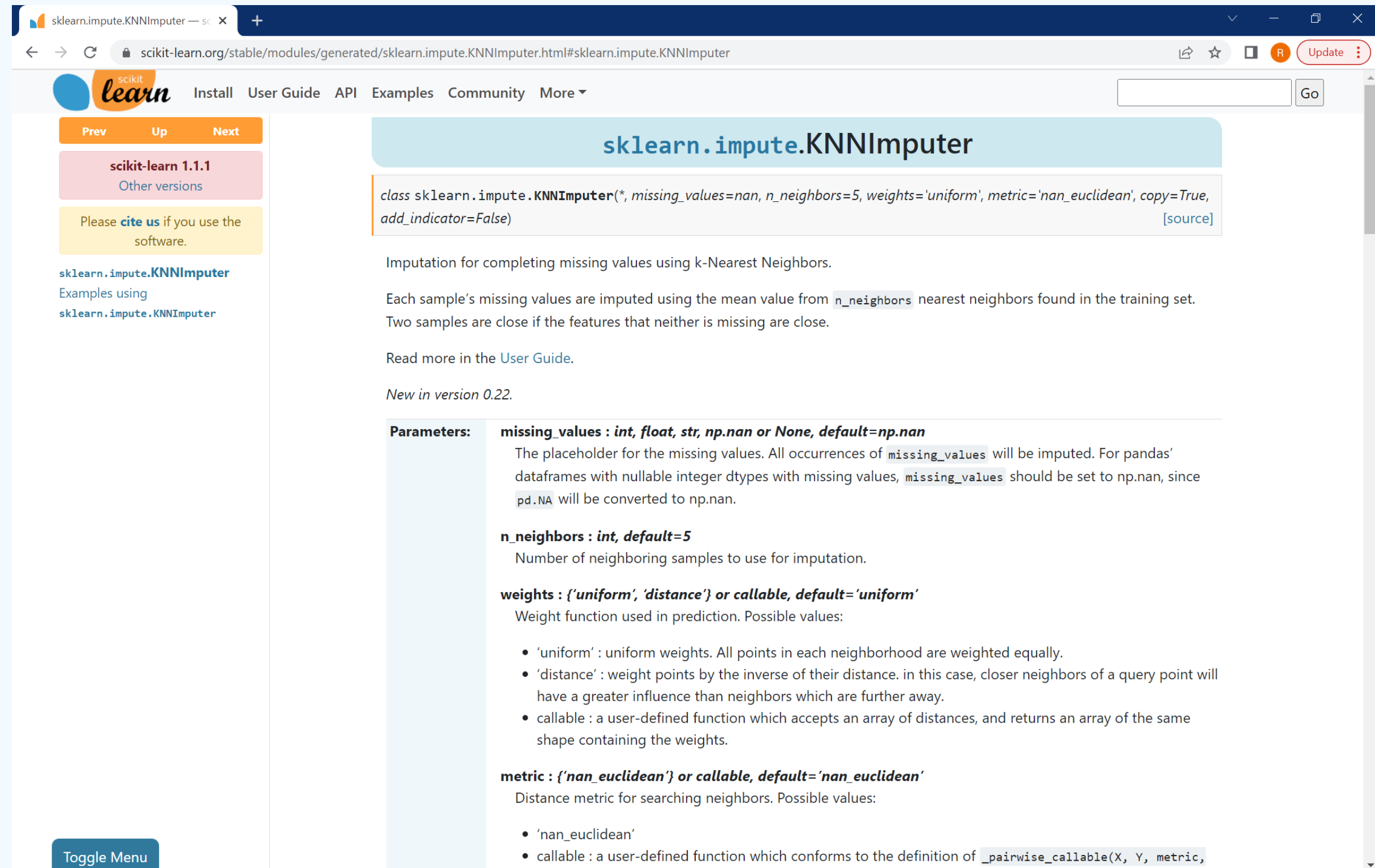
TRATAMENTO DE DADOS NULOS

KNN IMPUTATION

- Faz uso do conceito de criação de um modelo para prever o valor ausente;
- Para domínios numéricos, pode-se usar um modelo de regressão, mas algo tão simples como KNN também pode funcionar muito bem;
- KNN baseia-se em usar uma medida de distância e um número k de vizinhos para prever o valor.

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

KNN IMPUTATION



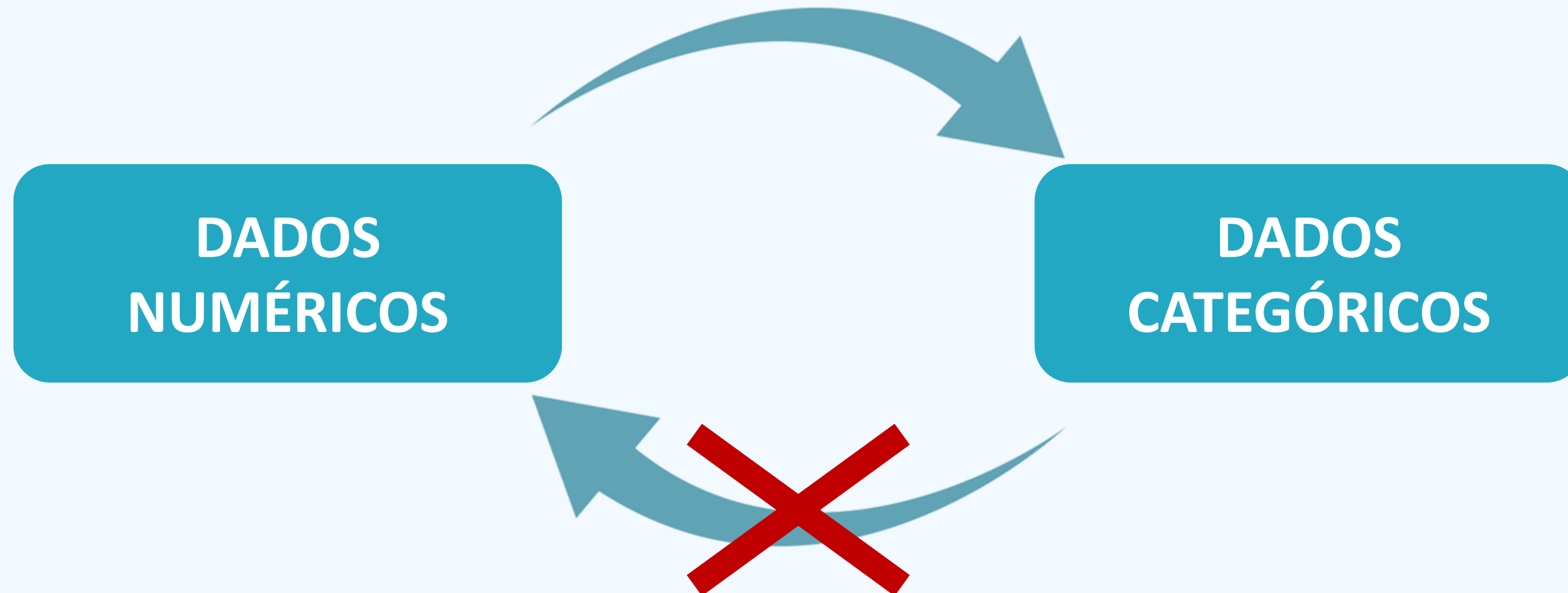
The screenshot shows the documentation for `sklearn.impute.KNNImputer` on the scikit-learn website. The page includes a navigation bar with links to 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. A sidebar on the left contains links for 'Prev', 'Up', 'Next', 'scikit-learn 1.1.1', 'Other versions', and a 'cite us' notice. The main content area displays the class name `sklearn.impute.KNNImputer` and its signature: `class sklearn.impute.KNNImputer(*, missing_values=nan, n_neighbors=5, weights='uniform', metric='nan_euclidean', copy=True, add_indicator=False)`. Below the signature, a brief description states: 'Imputation for completing missing values using k-Nearest Neighbors.' It further explains that missing values are imputed using the mean value from `n_neighbors` nearest neighbors found in the training set. A 'Parameters' section lists the following:

- missing_values**: `int, float, str, np.nan or None, default=np.nan`. The placeholder for the missing values. All occurrences of `missing_values` will be imputed. For pandas' dataframes with nullable integer dtypes with missing values, `missing_values` should be set to `np.nan`, since `pd.NA` will be converted to `np.nan`.
- n_neighbors**: `int, default=5`. Number of neighboring samples to use for imputation.
- weights**: `{'uniform', 'distance'} or callable, default='uniform'`. Weight function used in prediction. Possible values:
 - 'uniform': uniform weights. All points in each neighborhood are weighted equally.
 - 'distance': weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
 - callable: a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.
- metric**: `{'nan_euclidean'} or callable, default='nan_euclidean'`. Distance metric for searching neighbors. Possible values:
 - 'nan_euclidean'
 - callable: a user-defined function which conforms to the definition of `_pairwise_callable(X, Y, metric,`

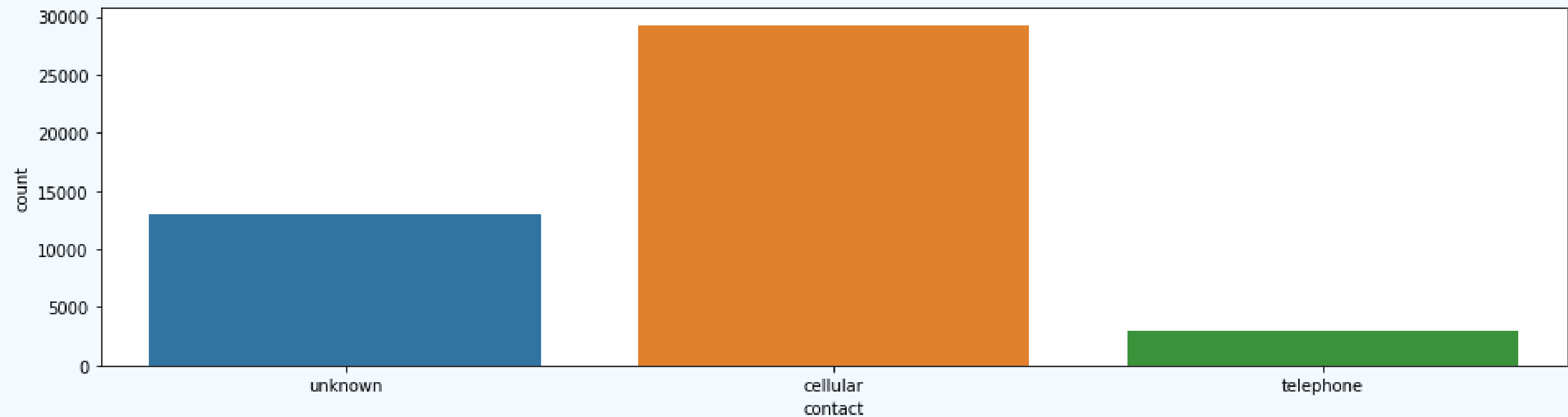
PARTE 2

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS



CONVERSÃO DE VARIÁVEIS CATEGÓRICAS



CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

ORDINAL ENCODING

- Também conhecida como *integer/label encoding*;
- Cada categoria recebe um valor inteiro;
- Ex: {'Green':1, 'Yellow':2, 'Red': 3};
- Funciona devido a relação de ordem natural existente em valores numéricos (inteiros);
- A ideia é repassar aos algoritmos essa relação intrínseca de “ordem”;
- Quando esta relação não está presente nos dados, ela será criada de forma artificial pelo *encoder*.

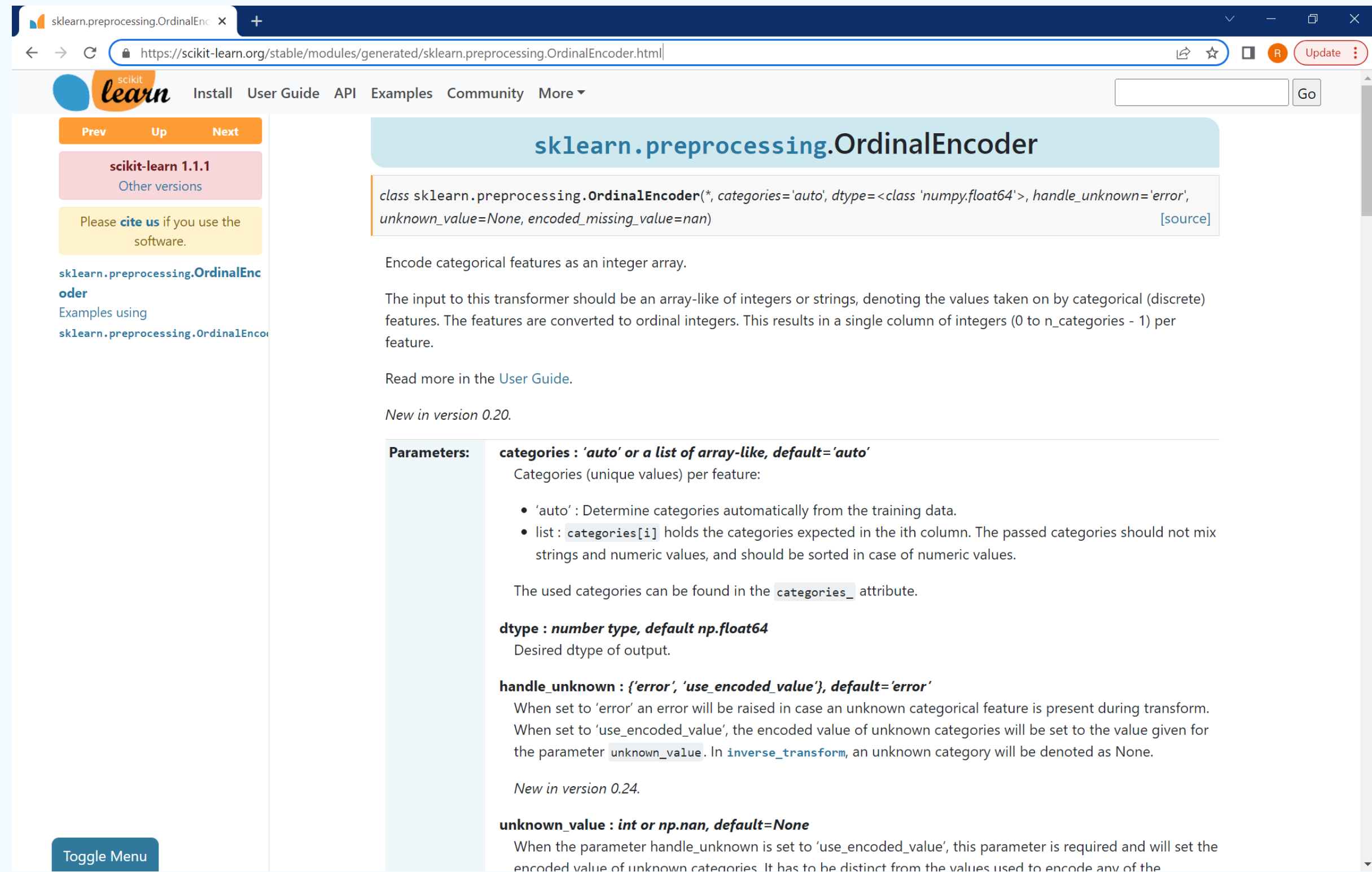
CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

Ordinal Encoding

Breakfast		Breakfast
Every day	➔	3
Never		0
Rarely		1
Most days		2
Never		0

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

ORDINAL ENCODING



The screenshot shows the official documentation for `sklearn.preprocessing.OrdinalEncoder` on the scikit-learn website. The page is titled "sklearn.preprocessing.OrdinalEncoder" and provides a detailed overview of the class. It includes the class signature, a brief description, and a list of parameters with their default values and descriptions.

sklearn.preprocessing.OrdinalEncoder

```
class sklearn.preprocessing.OrdinalEncoder(*, categories='auto', dtype=<class 'numpy.float64'>, handle_unknown='error', unknown_value=None, encoded_missing_value=nan)
```

Encode categorical features as an integer array.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are converted to ordinal integers. This results in a single column of integers (0 to `n_categories - 1`) per feature.

Read more in the [User Guide](#).

New in version 0.20.

Parameters:

- categories :** *'auto' or a list of array-like, default='auto'*
Categories (unique values) per feature:
 - 'auto' : Determine categories automatically from the training data.
 - list : `categories[i]` holds the categories expected in the `i`th column. The passed categories should not mix strings and numeric values, and should be sorted in case of numeric values.The used categories can be found in the `categories_` attribute.
- dtype :** *number type, default np.float64*
Desired dtype of output.
- handle_unknown :** *{'error', 'use_encoded_value'}, default='error'*
When set to 'error' an error will be raised in case an unknown categorical feature is present during transform. When set to 'use_encoded_value', the encoded value of unknown categories will be set to the value given for the parameter `unknown_value`. In `inverse_transform`, an unknown category will be denoted as `None`.
New in version 0.24.
- unknown_value :** *int or np.nan, default=None*
When the parameter `handle_unknown` is set to 'use_encoded_value', this parameter is required and will set the encoded value of unknown categories. It has to be distinct from the values used to encode any of the

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

ONE HOT ENCODING

- Usado em variáveis categóricas onde não há relação de ordem estabelecida;
- Para cada valor categórico é criado um código booleano/binário, gerando uma nova coluna que apresentará a ocorrência ou não dessa categoria para a observação;
- Dado que que a mesma observação não pode pertencer a duas ou mais categorias simultaneamente, então só um valor pode ser verdadeiro (daí o nome: *one hot*);
- Pode eliminar redundâncias, dado que se temos 3 possíveis categorias, apenas 2 demandam representação. Ou seja, para n categorias precisamos de $n-1$ representações.

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

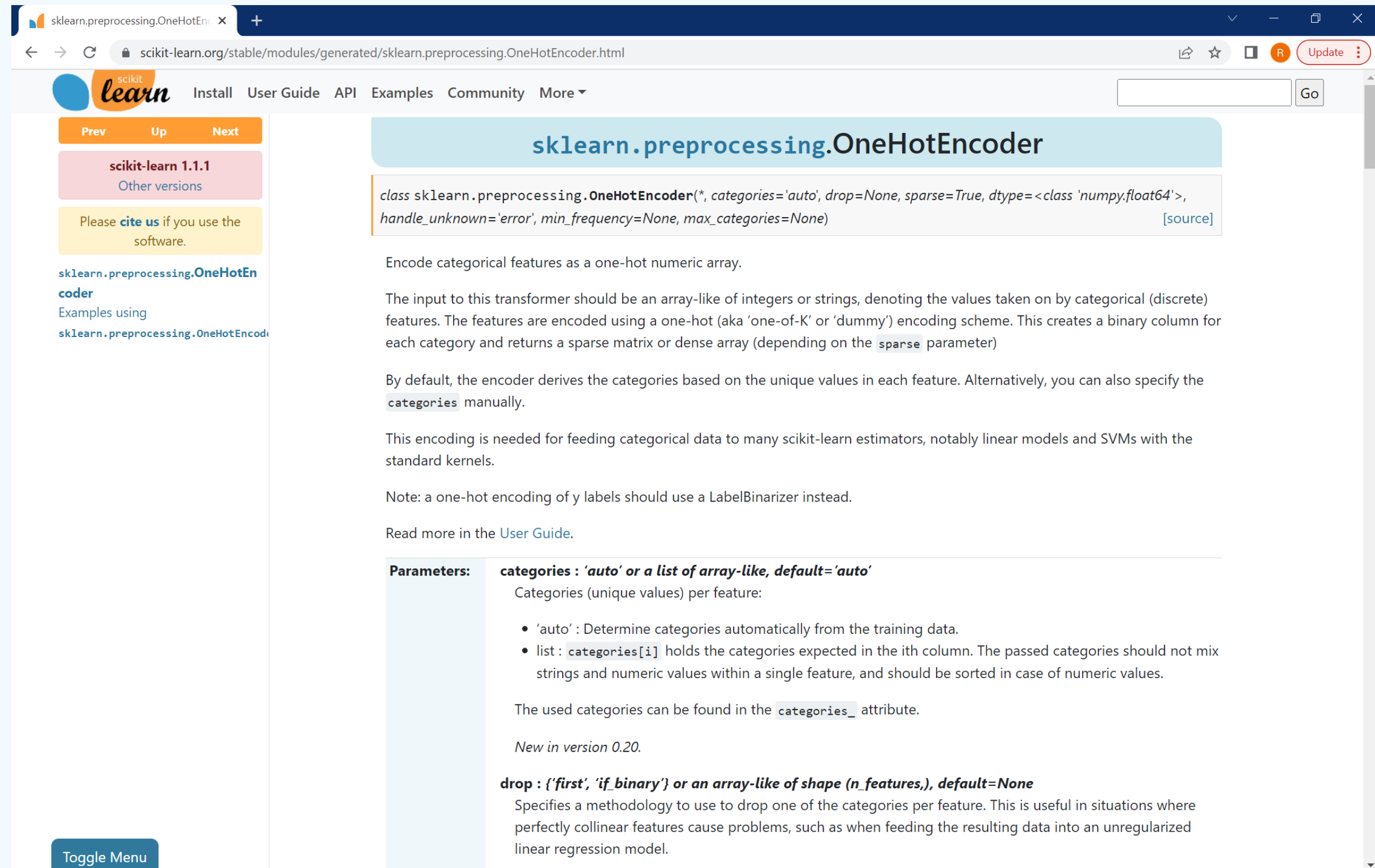
One Hot Encoding

OUT [4]:

	SibSp	Parch	Sex_male	Pclass_2	Pclass_3	Age_[20, 30]	Age_[30, 40]	Age_[40, 50]	Age_[50, 60]	Age_[60, 70]	Age_[70, 80]
0	1	0	1	0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	1	1	0	0	0	0	0
3	1	0	0	0	0	0	1	0	0	0	0
4	0	0	1	0	1	0	1	0	0	0	0

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

ONE HOT ENCODING



The screenshot shows the official documentation for `sklearn.preprocessing.OneHotEncoder` on the scikit-learn website. The page includes a navigation sidebar with links for 'Prev', 'Up', 'Next', 'scikit-learn 1.1.1', 'Other versions', and a 'cite us' notice. The main content area features the class name in a blue header, followed by the class signature: `class sklearn.preprocessing.OneHotEncoder(*, categories='auto', drop=None, sparse=True, dtype=<class 'numpy.float64'>, handle_unknown='error', min_frequency=None, max_categories=None)`. Below this, the documentation explains that the transformer encodes categorical features as a one-hot numeric array. It details the input requirements (array-like of integers or strings) and the output (sparse matrix or dense array). A note specifies that by default, categories are derived from unique values, but can be manually specified. Another note mentions that this encoding is needed for feeding categorical data to many estimators. A final note states that a one-hot encoding of y labels should use a `LabelBinarizer` instead. The 'Parameters' section lists `categories` (default 'auto') and `drop` (default None), with detailed explanations for each. The `categories` parameter is described as 'auto' (determine categories automatically) or a list of categories expected in the ith column. The `drop` parameter is described as 'first', 'if_binary', or an array-like of shape (n_features,). The page also includes a 'Toggle Menu' button at the bottom left.

sklearn.preprocessing.OneHotEncoder

```
class sklearn.preprocessing.OneHotEncoder(*, categories='auto', drop=None, sparse=True, dtype=<class 'numpy.float64'>, handle_unknown='error', min_frequency=None, max_categories=None)
```

Encode categorical features as a one-hot numeric array.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array (depending on the `sparse` parameter)

By default, the encoder derives the categories based on the unique values in each feature. Alternatively, you can also specify the `categories` manually.

This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs with the standard kernels.

Note: a one-hot encoding of y labels should use a `LabelBinarizer` instead.

Read more in the [User Guide](#).

Parameters:

categories : 'auto' or a list of array-like, default='auto'
Categories (unique values) per feature:

- 'auto': Determine categories automatically from the training data.
- list : `categories[i]` holds the categories expected in the ith column. The passed categories should not mix strings and numeric values within a single feature, and should be sorted in case of numeric values.

The used categories can be found in the `categories_` attribute.

New in version 0.20.

drop : {'first', 'if_binary'} or an array-like of shape (n_features,), default=None
Specifies a methodology to use to drop one of the categories per feature. This is useful in situations where perfectly collinear features cause problems, such as when feeding the resulting data into an unregularized linear regression model.

PARTE 3

CONVERSÃO DE VARIÁVEIS NUMÉRICAS

CONVERSÃO DE VARIÁVEIS NUMÉRICAS

NORMALIZATION

- Reescala os dados para garantir que assumam valores entre 0 e 1;
- Dessa forma, demanda que os valores mínimos e máximos sejam conhecidos, podendo ser utilizados a partir daqueles disponíveis na amostra;
- Após a normalização, se um dado fora do intervalo entre mínimo e máximo for apresentado, a normalização retornará um valor fora do intervalo $[0, 1]$.

$$y = \frac{x - x_{min}}{x_{max} - x_{min}}$$

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

NORMALIZATION

sklearn.preprocessing.MinMaxScaler

scikit-learn 1.1.1

Please cite us if you use the software.

sklearn.preprocessing.MinMaxScaler

Examples using sklearn.preprocessing.MinMaxScaler

sklearn.preprocessing.MinMaxScaler

```
class sklearn.preprocessing.MinMaxScaler(feature_range=(0, 1), *, copy=True, clip=False)
```

Transform features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

where min, max = feature_range.

This transformation is often used as an alternative to zero mean, unit variance scaling.

Read more in the [User Guide](#).

Parameters:

feature_range : tuple (min, max), default=(0, 1)

Desired range of transformed data.

copy : bool, default=True

Set to False to perform inplace row normalization and avoid a copy (if the input is already a numpy array).

clip : bool, default=False

Set to True to clip transformed values of held-out data to provided feature range.

New in version 0.24.

Attributes:

min_ : ndarray of shape (n_features,)

Per feature adjustment for minimum. Equivalent to `min - X.min(axis=0) * self.scale_`

scale_ : ndarray of shape (n_features,)

Per feature relative scaling of the data. Equivalent to `(max - min) / (X.max(axis=0) - X.min(axis=0))`

New in version 0.17: scale_ attribute

CONVERSÃO DE VARIÁVEIS NUMÉRICAS

STANDARDIZATION

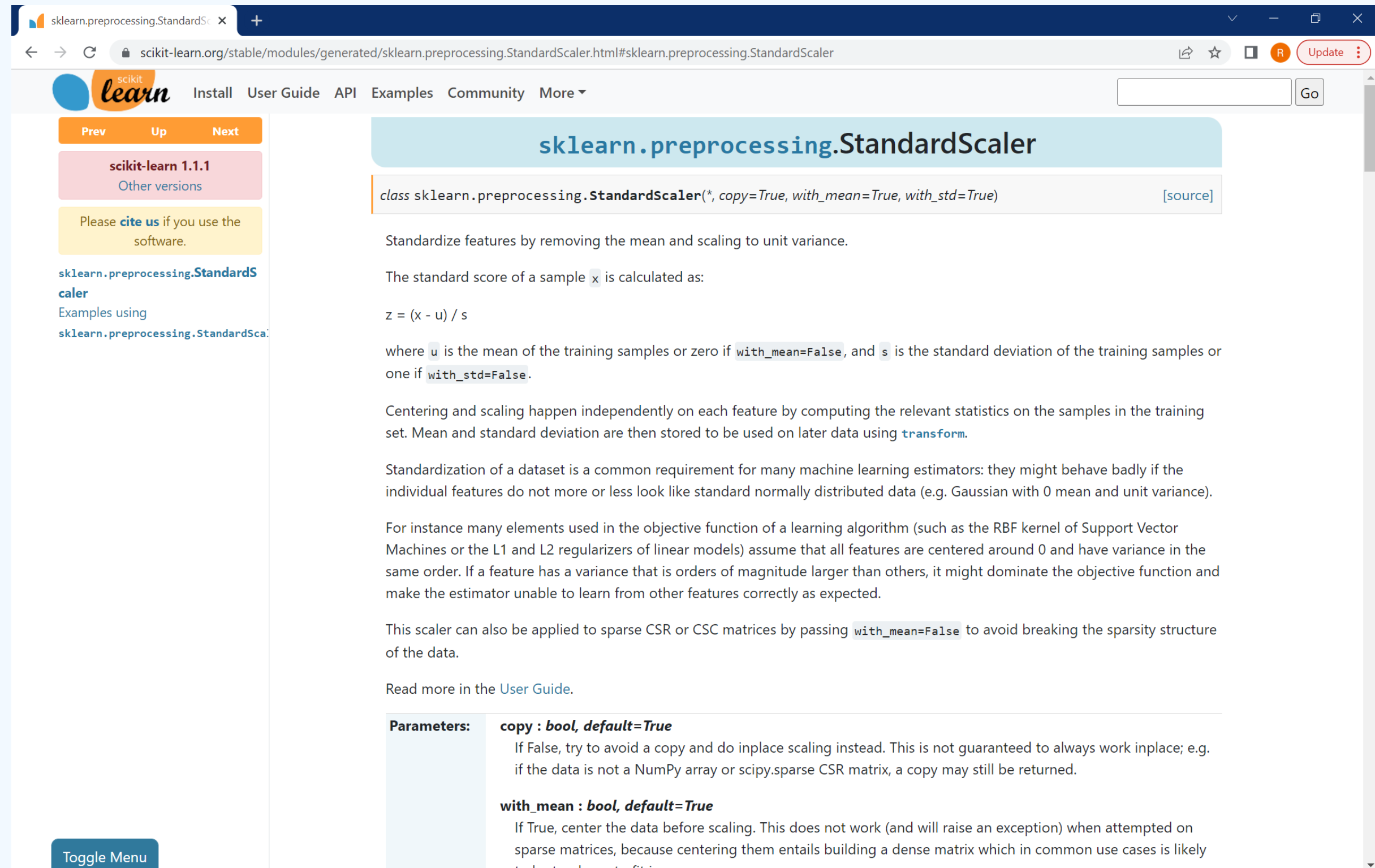
- Reescala os dados de forma a buscar que a média seja igual a 0 e o desvio padrão igual a 1;
- Assim como a normalização, indicada para quando os dados apresentam escalas (muito) diferentes;
- Assume que os dados obedecem uma distribuição Gaussiana, mas pode ser aplicada também quando isso não ocorre;
- É indicado que os valores de média e desvio padrão utilizados sejam da base de treino, sendo aplicados nas demais bases;

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

$$\bar{x} = \frac{\sum_{i=0}^n x}{n}$$

$$\sigma = \sqrt{\frac{\sum_{i=0}^n (x - \bar{x})^2}{n - 1}}$$

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS STANDARDIZATION



The screenshot shows the official documentation for `sklearn.preprocessing.StandardScaler` on the scikit-learn website. The page is titled "sklearn.preprocessing.StandardScaler" and includes a navigation bar with links for "Install", "User Guide", "API", "Examples", "Community", and "More". The main content area describes the class and its usage.

sklearn.preprocessing.StandardScaler

```
class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True)
```

Standardize features by removing the mean and scaling to unit variance.

The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

where u is the mean of the training samples or zero if `with_mean=False`, and s is the standard deviation of the training samples or one if `with_std=False`.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using `transform`.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

For instance many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

This scaler can also be applied to sparse CSR or CSC matrices by passing `with_mean=False` to avoid breaking the sparsity structure of the data.

Read more in the [User Guide](#).

Parameters:

- copy : bool, default=True**
If False, try to avoid a copy and do inplace scaling instead. This is not guaranteed to always work inplace; e.g. if the data is not a NumPy array or scipy.sparse CSR matrix, a copy may still be returned.
- with_mean : bool, default=True**
If True, center the data before scaling. This does not work (and will raise an exception) when attempted on sparse matrices, because centering them entails building a dense matrix which in common use cases is likely to be too large to fit in memory.

CONVERSÃO DE VARIÁVEIS NUMÉRICAS

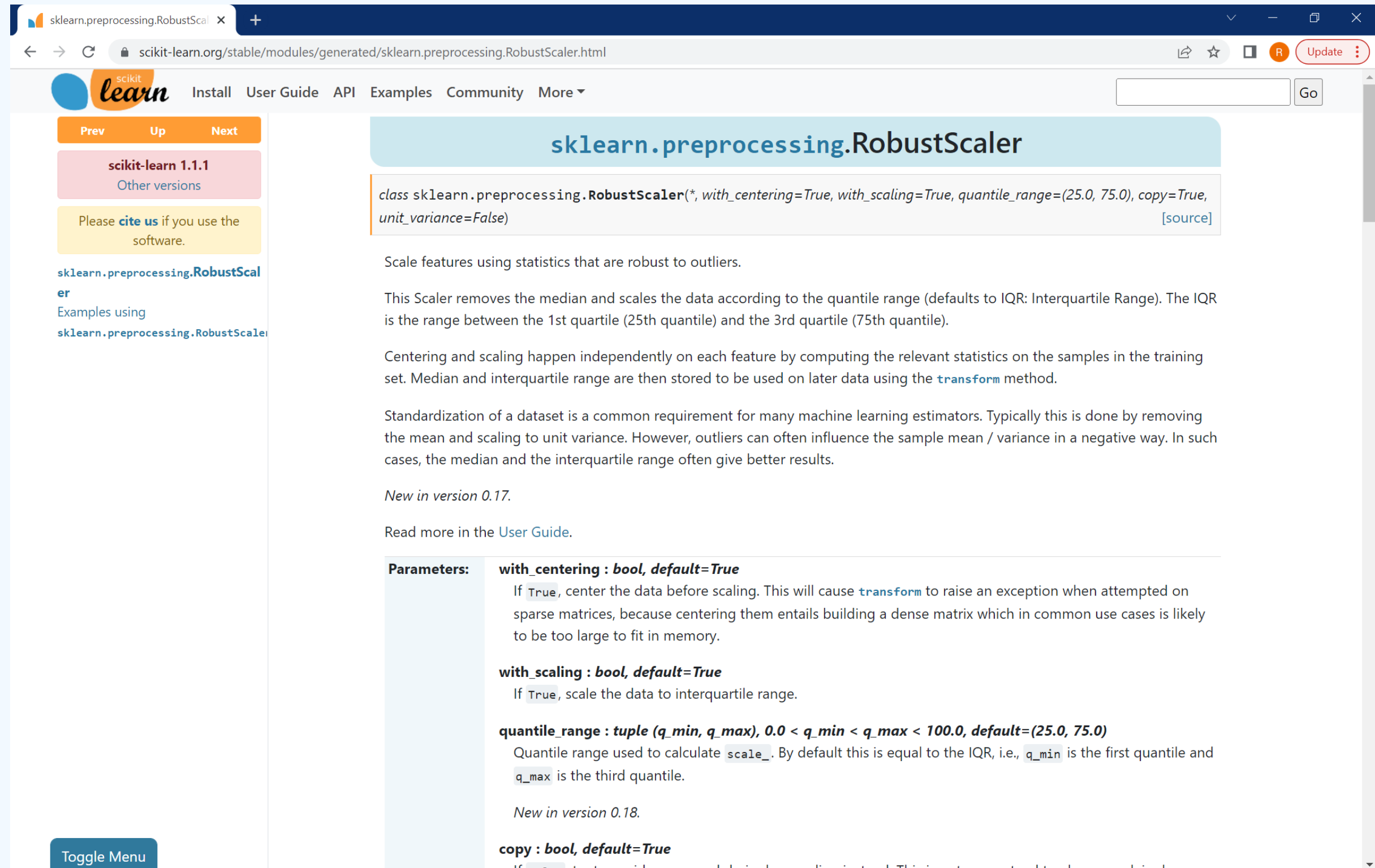
ROBUST SCALING

- É usado para reescalar variáveis com outliers;
- Faz uso da mediana e da distância interquartis para o processo;
- É matematicamente muito similar ao processo de *normalization*, mudando apenas o ponto de centralidade da média para mediana e o domínio sendo reduzido para p_{25} e p_{75} .

$$y = \frac{x - p_{50}}{p_{75} - p_{25}}$$

CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

ROBUS SCALING



The screenshot shows the official documentation for `sklearn.preprocessing.RobustScaler` on the scikit-learn website. The page includes a navigation bar with links for 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. A sidebar on the left contains links for 'Prev', 'Up', 'Next', 'scikit-learn 1.1.1', 'Other versions', a 'cite us' notice, and a link to 'Examples using sklearn.preprocessing.RobustScaler'. The main content area features the class name `sklearn.preprocessing.RobustScaler` in a light blue header, followed by its class signature: `class sklearn.preprocessing.RobustScaler(*, with_centering=True, with_scaling=True, quantile_range=(25.0, 75.0), copy=True, unit_variance=False)`. Below this, a brief description states: 'Scale features using statistics that are robust to outliers.' The text explains that the scaler removes the median and scales the data according to the quantile range (IQR), which is the range between the 1st and 3rd quartiles. It also notes that centering and scaling happen independently on each feature by computing relevant statistics on the training set. A section titled 'Parameters:' lists the following options:

- with_centering** : *bool, default=True*. If `True`, center the data before scaling. This will cause `transform` to raise an exception when attempted on sparse matrices, because centering them entails building a dense matrix which in common use cases is likely to be too large to fit in memory.
- with_scaling** : *bool, default=True*. If `True`, scale the data to interquartile range.
- quantile_range** : *tuple (q_min, q_max), 0.0 < q_min < q_max < 100.0, default=(25.0, 75.0)*. Quantile range used to calculate `scale_`. By default this is equal to the IQR, i.e., `q_min` is the first quantile and `q_max` is the third quantile.
- copy** : *bool, default=True*. If `False`, try to avoid a copy and do inplace scaling instead. This is not guaranteed to always work inplace; e.g.

CONVERSÃO DE VARIÁVEIS NUMÉRICAS

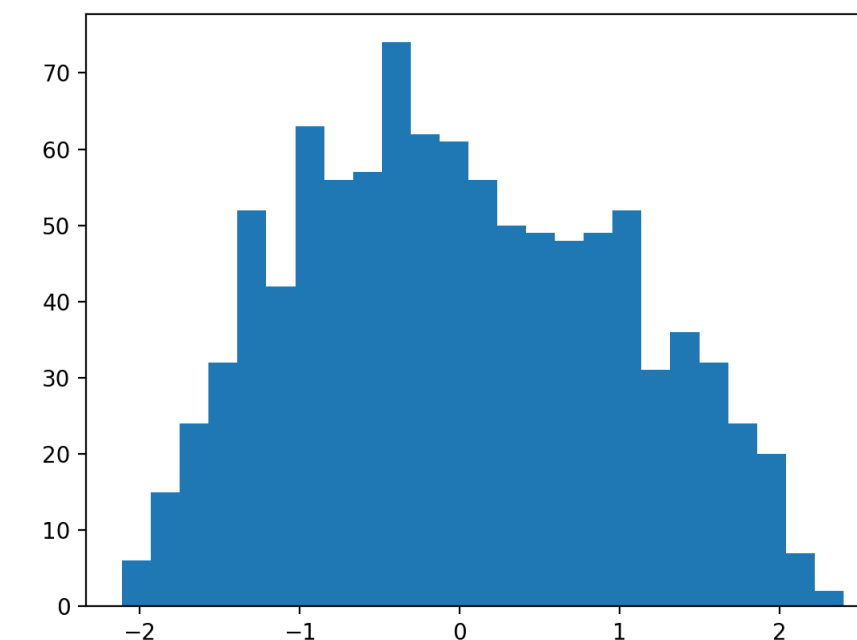
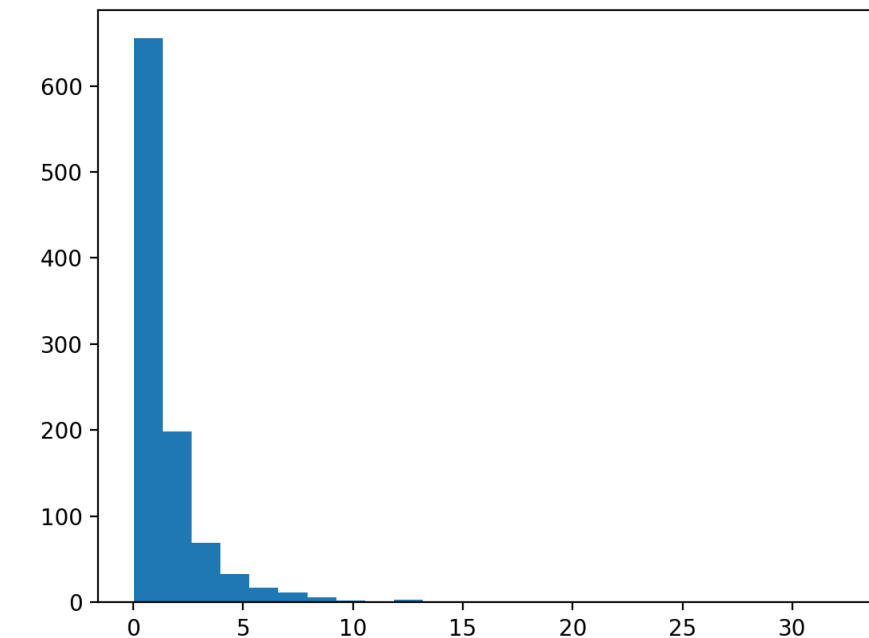
POWER TRANSFORM

- Busca diminuir a assimetria da distribuição;
- Efetivamente busca fazer com que a distribuição se aproxime de uma Gaussiana;
- Pode ser obtida de algumas maneiras, entre elas calculando-se o logaritmo ou a raiz quadrada dos valores. Contudo, essas transformações podem não ser as melhores para uma dada variável;
- Há a possibilidade de se buscar a melhor transformação a partir, geralmente, de duas abordagens:
 - Box-Cox,
 - Yeo-Johnson;
- Ambas buscam determinar um fator λ que melhor se adequa aos dados.

CONVERSÃO DE VARIÁVEIS NUMÉRICAS

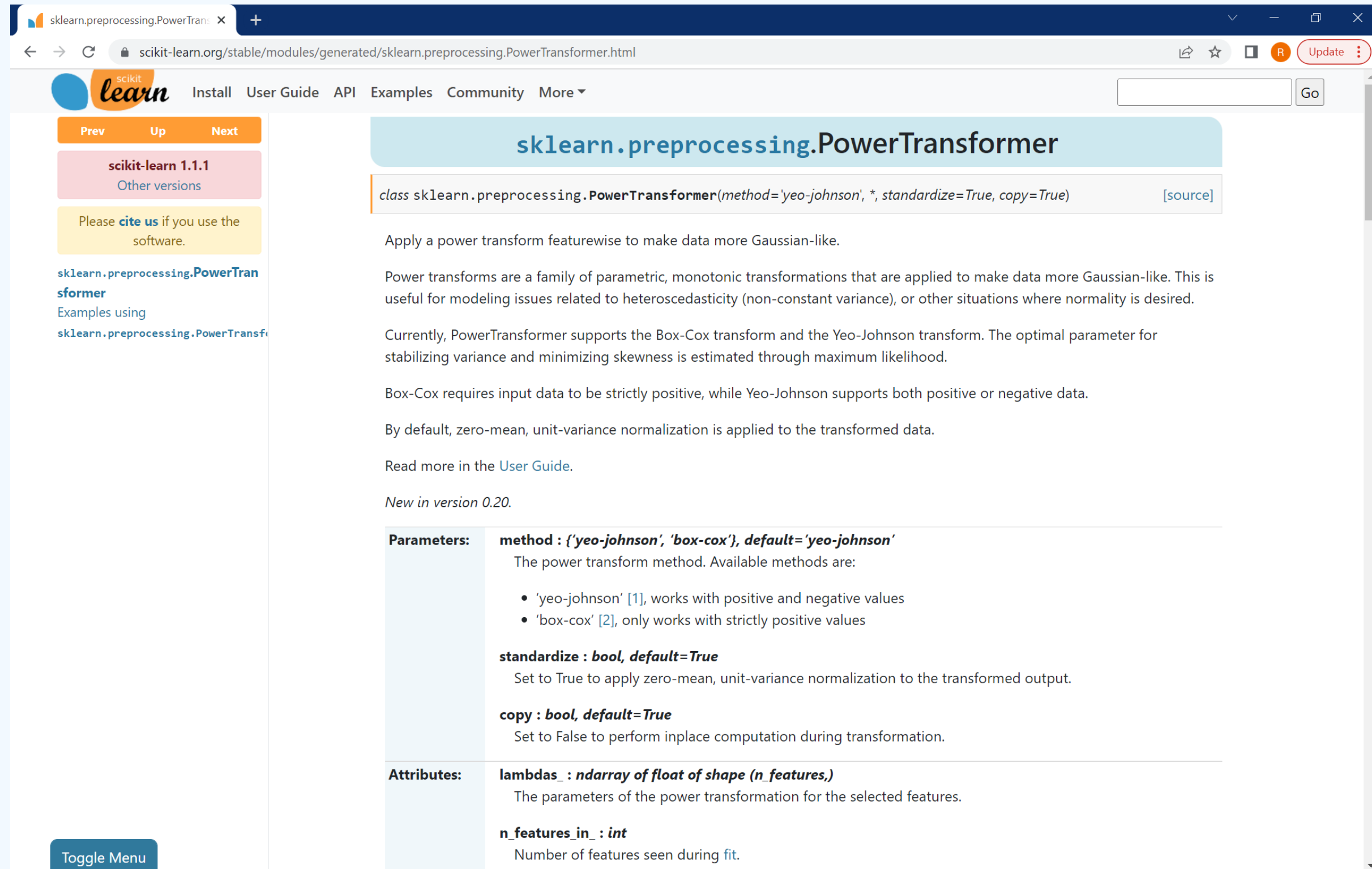
POWER TRANSFORM

- $\lambda = -1 \rightarrow$ transformação recíproca/inversa;
- $\lambda = -0,5 \rightarrow$ transformação recíproca por raiz quadrada;
- $\lambda = 0 \rightarrow$ transformação logarítmica;
- $\lambda = 0,5 \rightarrow$ transformação por raiz quadrada;
- $\lambda = 1 \rightarrow$ não há transformação;



CONVERSÃO DE VARIÁVEIS CATEGÓRICAS

POWER TRANSFORM



The screenshot shows the official documentation for `sklearn.preprocessing.PowerTransformer` on the scikit-learn website. The page includes navigation links, version information (scikit-learn 1.1.1), and a sidebar with a 'Toggle Menu' button. The main content area provides a detailed description of the transformer, its parameters, and its attributes.

sklearn.preprocessing.PowerTransformer

```
class sklearn.preprocessing.PowerTransformer(method='yeo-johnson', *, standardize=True, copy=True) \[source\]
```

Apply a power transform featurewise to make data more Gaussian-like.

Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like. This is useful for modeling issues related to heteroscedasticity (non-constant variance), or other situations where normality is desired.

Currently, PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.

Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive or negative data.

By default, zero-mean, unit-variance normalization is applied to the transformed data.

Read more in the [User Guide](#).

New in version 0.20.

Parameters:	<p>method : {'<i>yeo-johnson</i>', '<i>box-cox</i>'}, default='yeo-johnson'</p> <p>The power transform method. Available methods are:</p> <ul style="list-style-type: none">'yeo-johnson' [1], works with positive and negative values'box-cox' [2], only works with strictly positive values <p>standardize : <i>bool</i>, default=True</p> <p>Set to True to apply zero-mean, unit-variance normalization to the transformed output.</p> <p>copy : <i>bool</i>, default=True</p> <p>Set to False to perform inplace computation during transformation.</p>
Attributes:	<p>lambdas_ : <i>ndarray of float of shape (n_features,)</i></p> <p>The parameters of the power transformation for the selected features.</p> <p>n_features_in_ : <i>int</i></p> <p>Number of features seen during fit.</p>

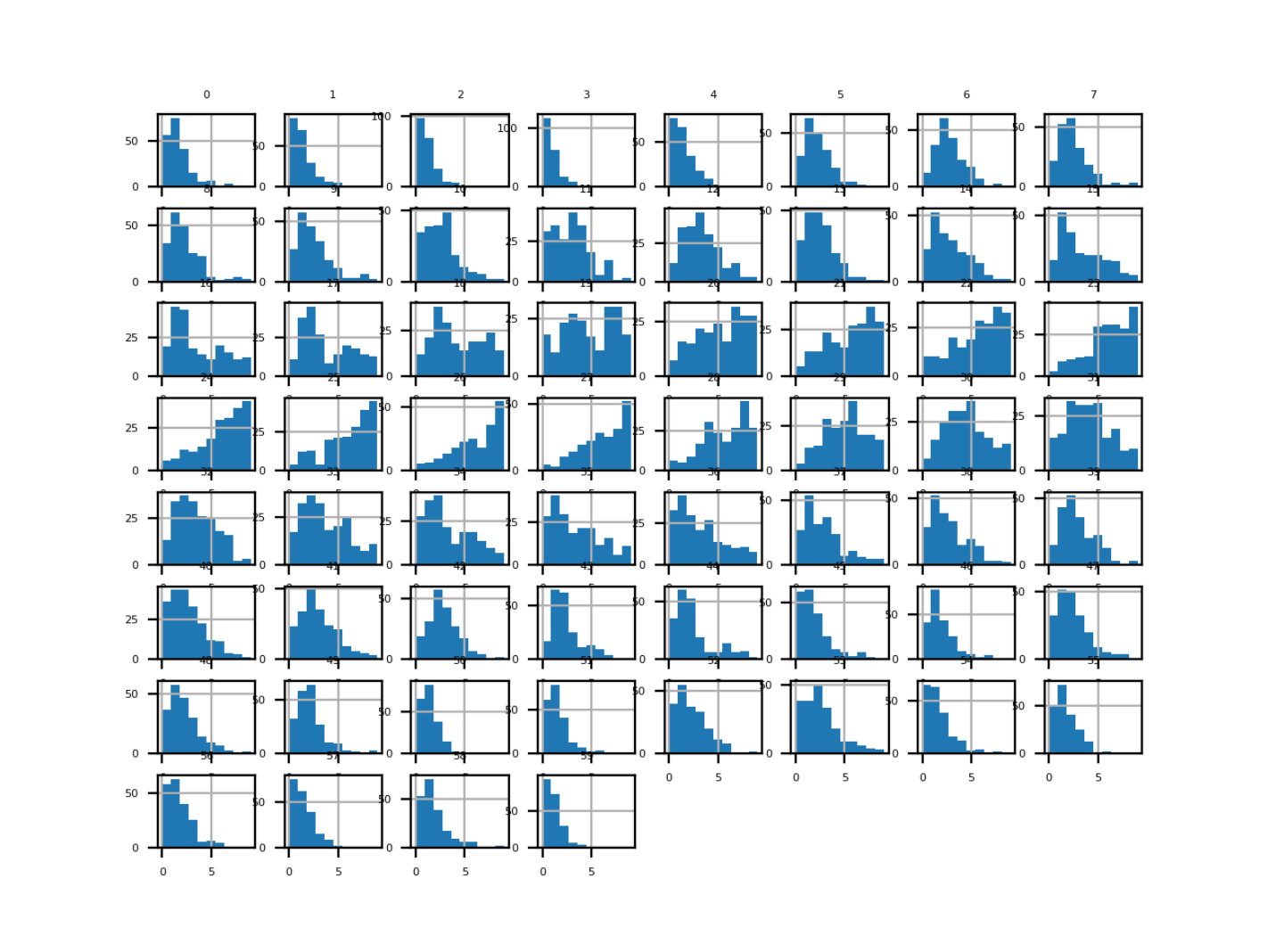
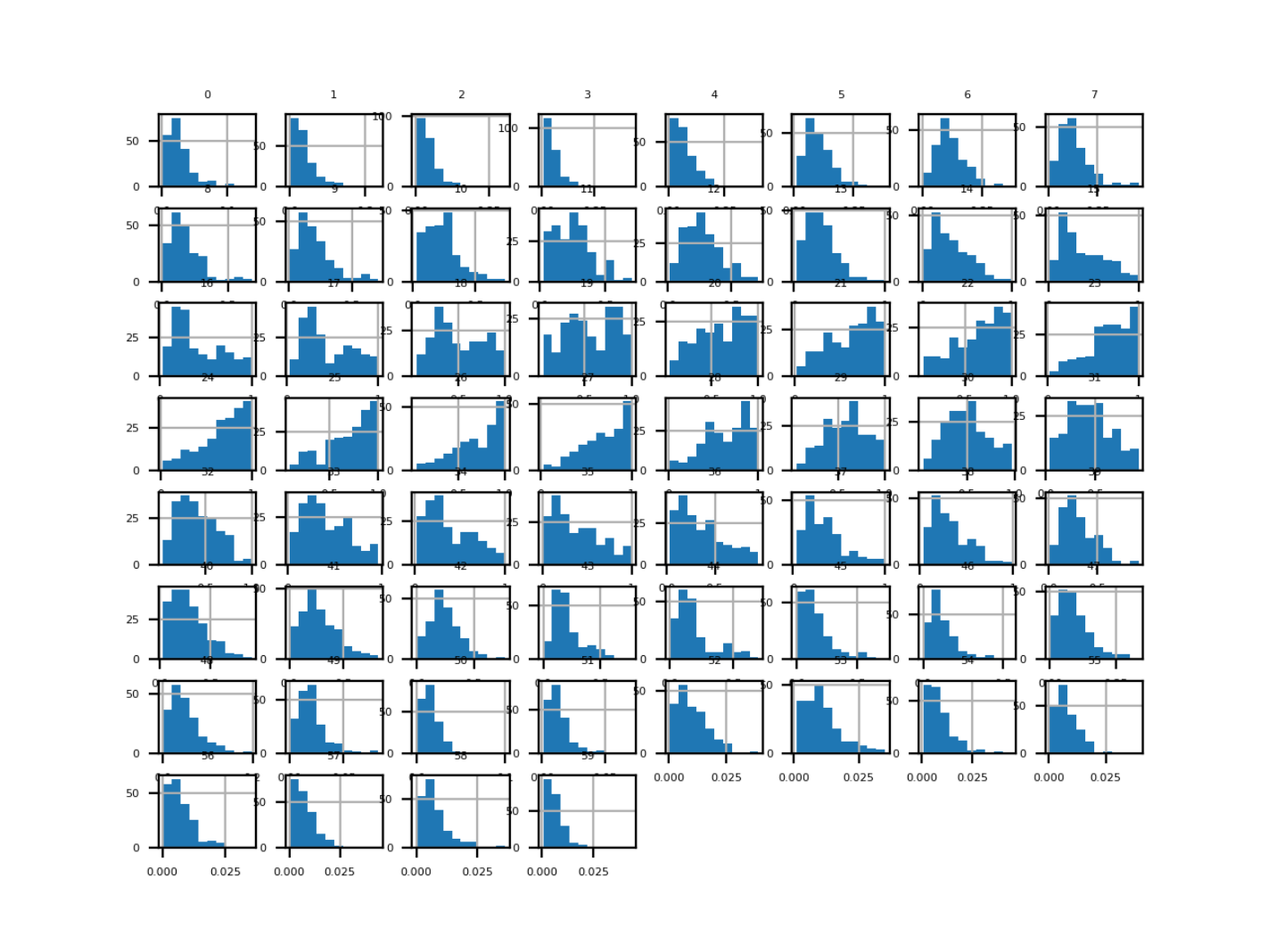
CONVERSÃO DE VARIÁVEIS NUMÉRICAS

DISCRETIZATION

- Algoritmos tendem a não performar bem com dados muito assimétricos ou com distribuição não-normal;
- Uma opção é a discretização de dados numéricos;
- *Binning*: os dados são alocados em *bins*, identificados por números inteiros que mantenham sua ordem;
- As faixas (*bins*) podem ser definidas de 3 formas:
 - *Uniform*: cada faixa tem a mesma largura,
 - *Quantile*: cada faixa tem o mesmo número de observações (limites definidos pelos percentis),
 - *Clustered*: clusters são identificados e observações são agrupadas

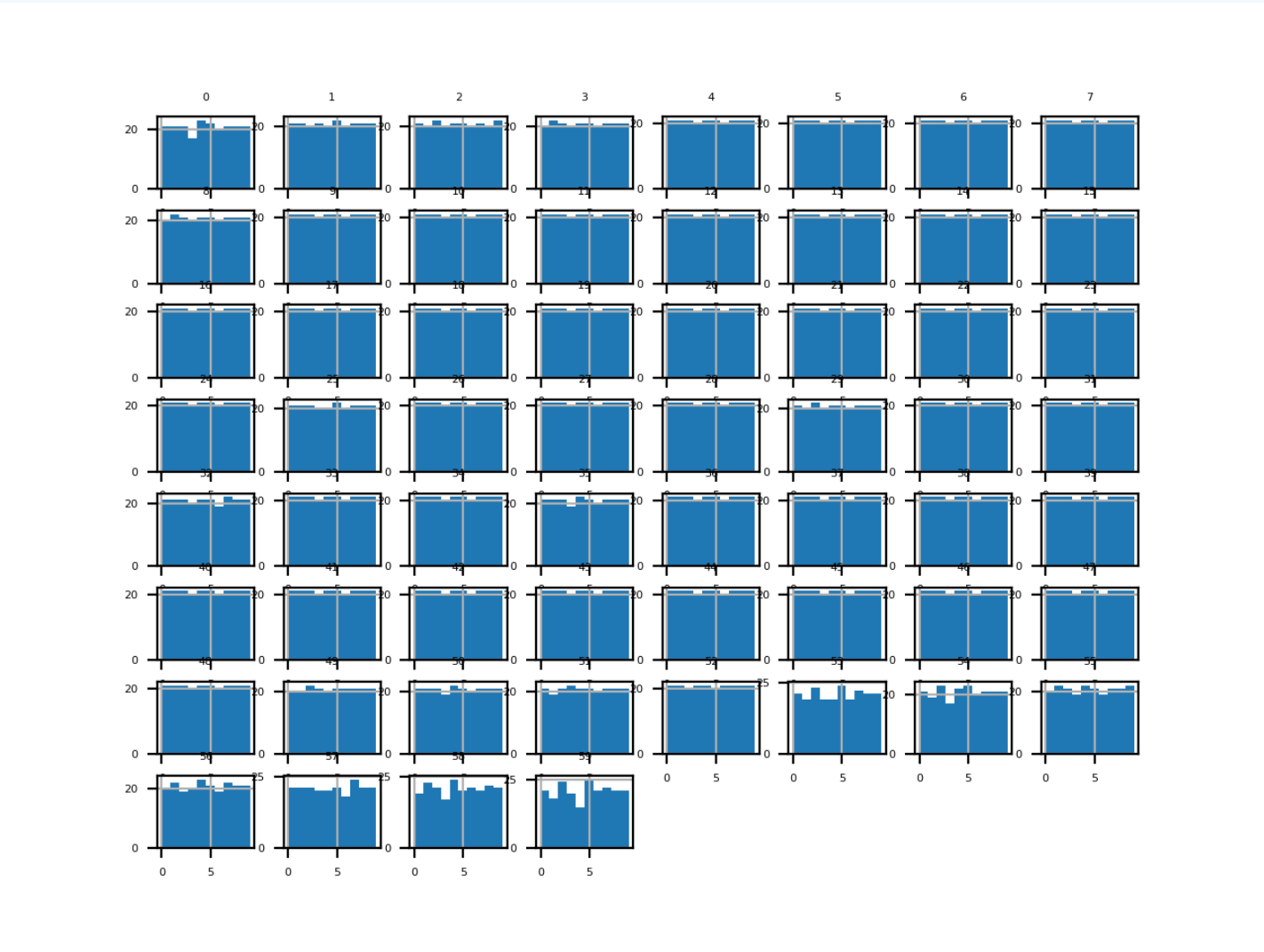
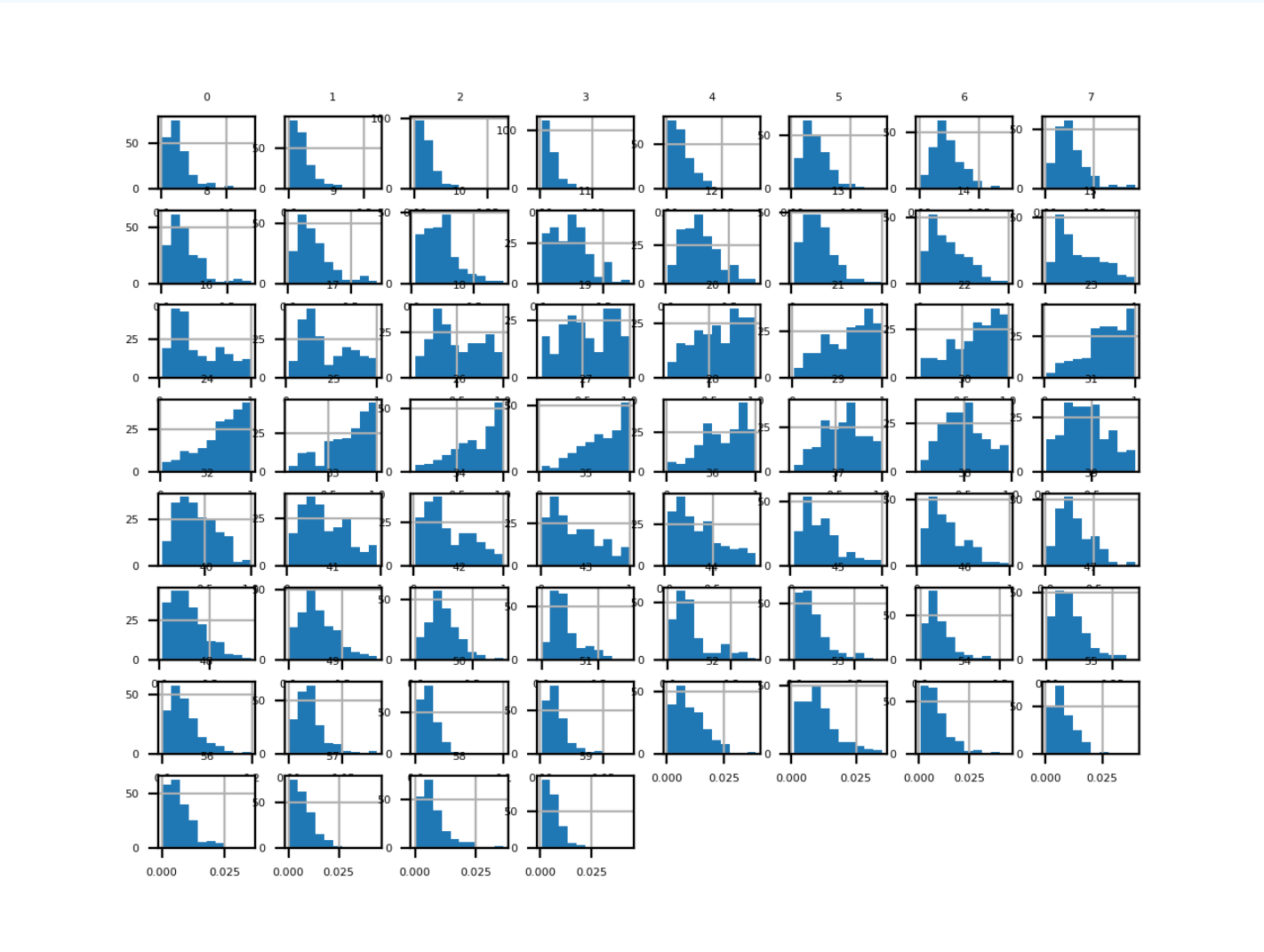
CONVERSÃO DE VARIÁVEIS NUMÉRICAS

DISCRETIZATION: UNIFORM



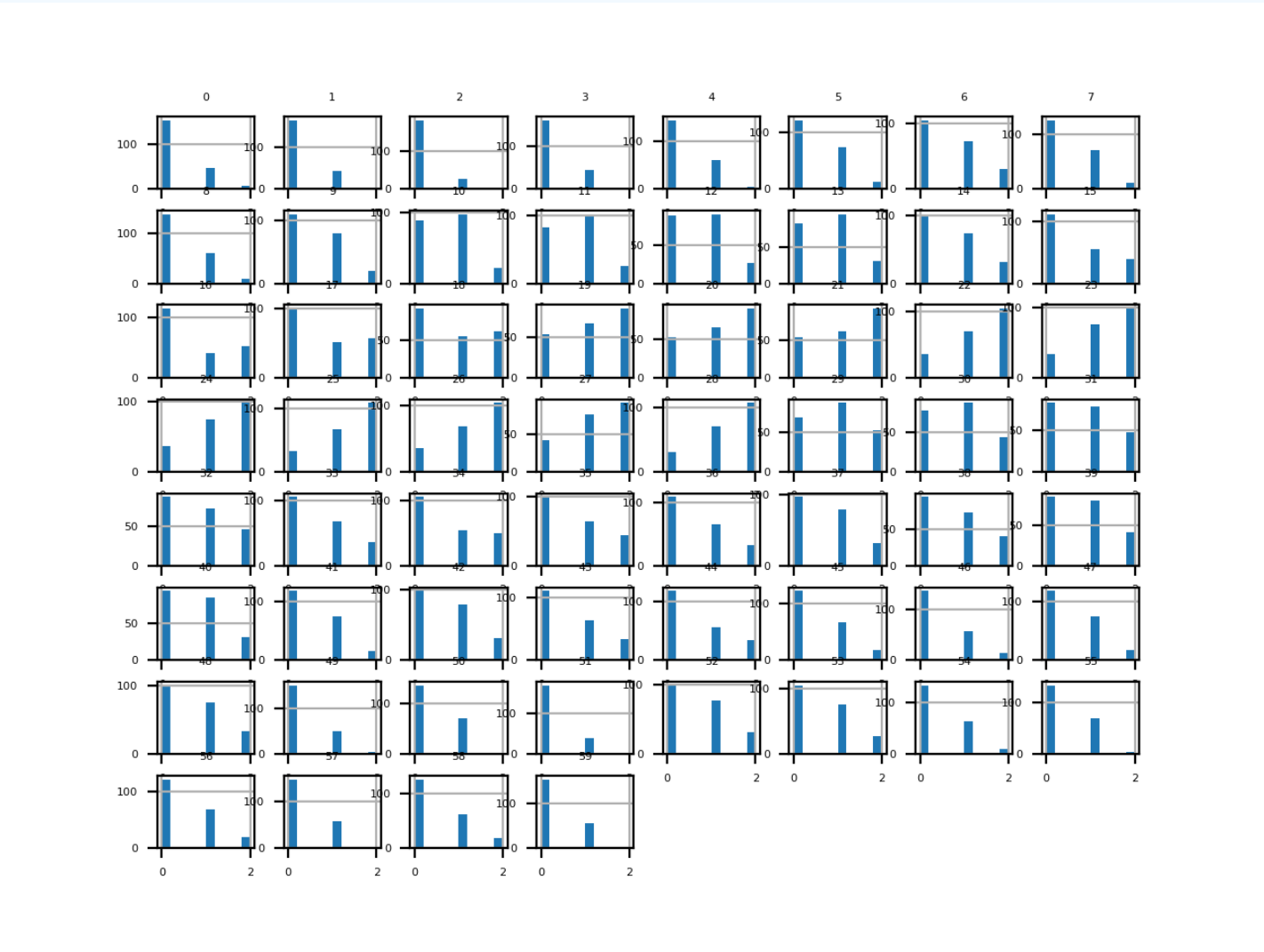
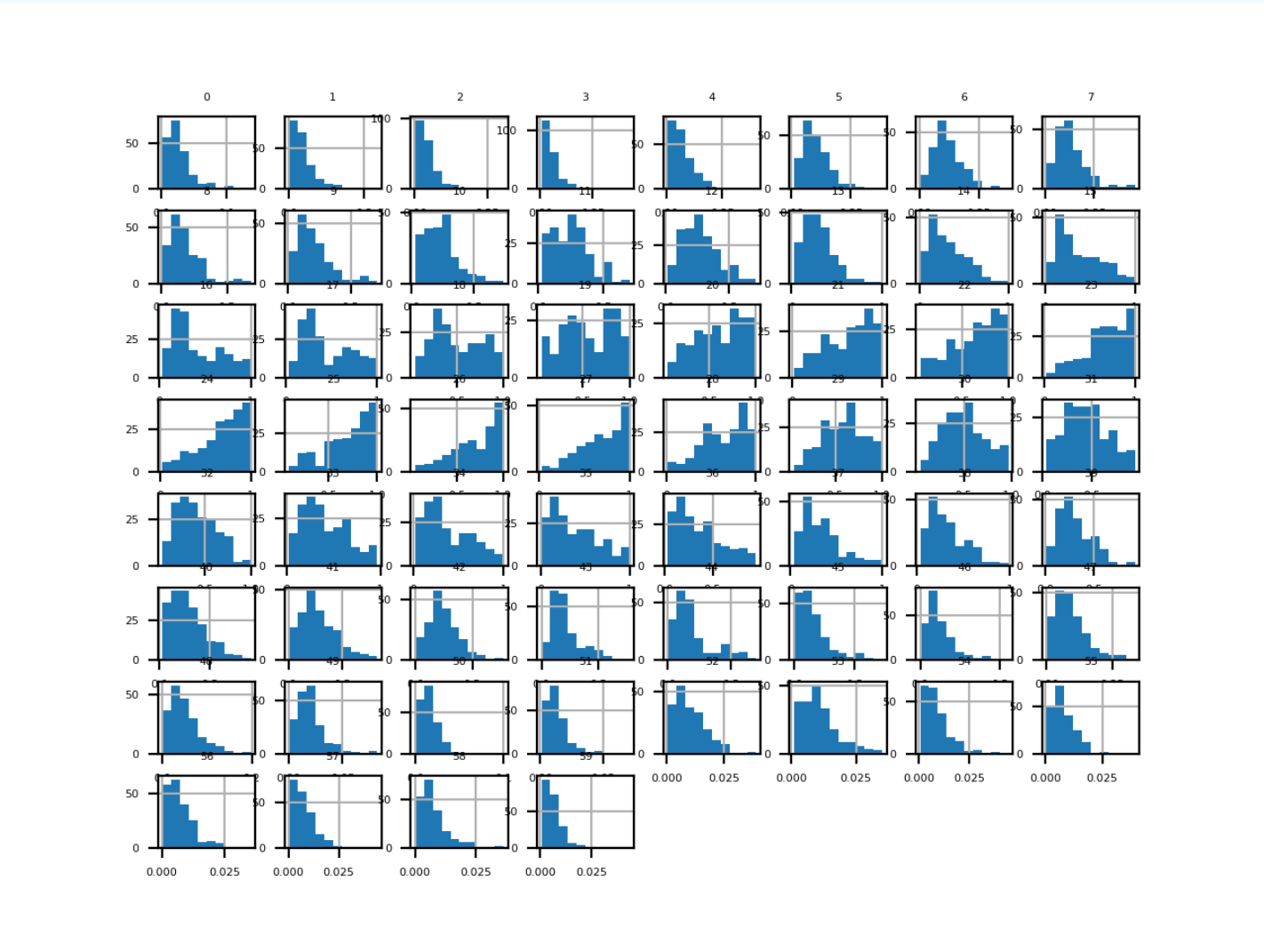
CONVERSÃO DE VARIÁVEIS NUMÉRICAS

DISCRETIZATION: QUANTILE

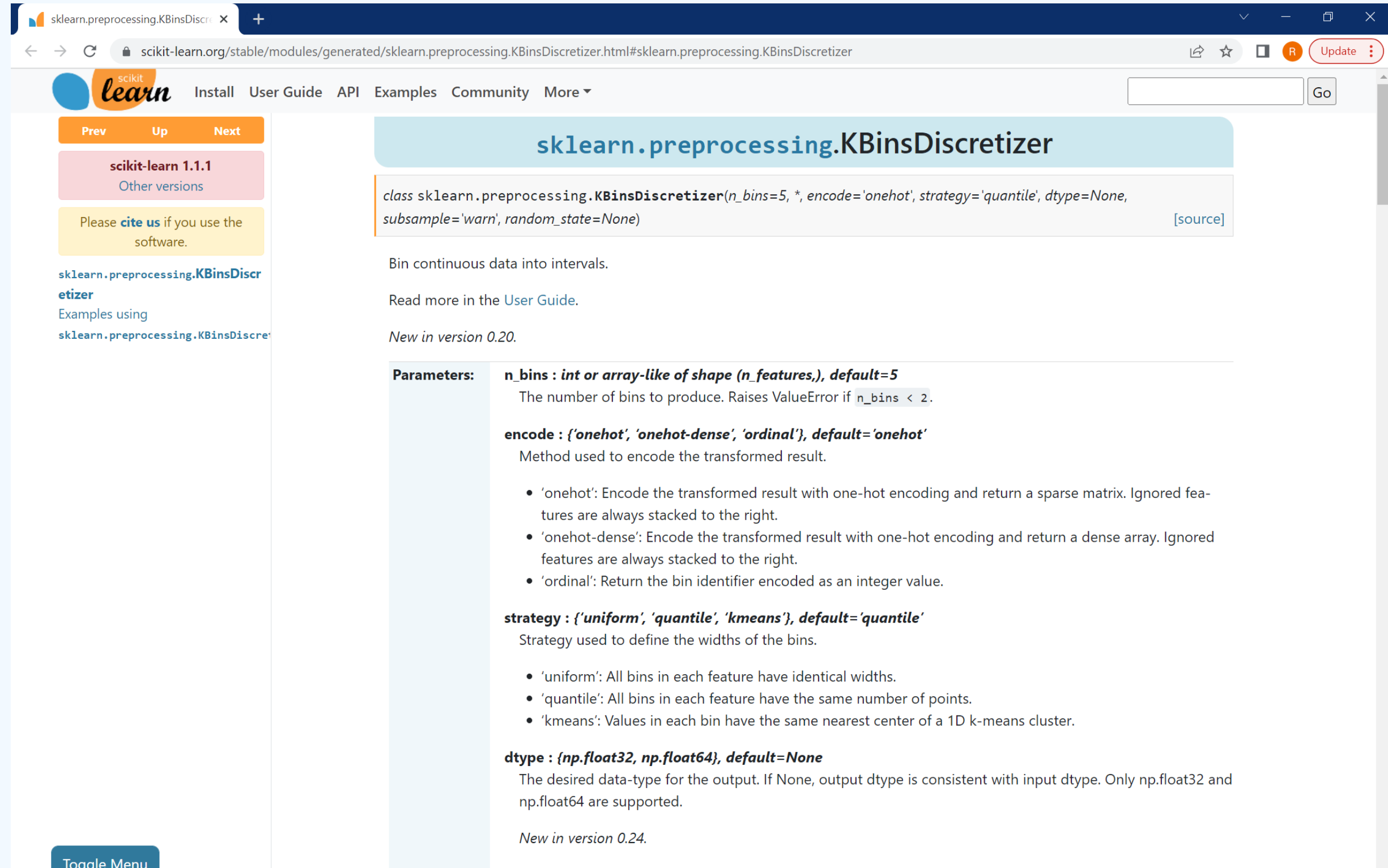


CONVERSÃO DE VARIÁVEIS NUMÉRICAS

DISCRETIZATION: CLUSTERED



CONVERSÃO DE VARIÁVEIS CATEGÓRICAS DISCRETIZATION



The screenshot shows the official documentation for `sklearn.preprocessing.KBinsDiscretizer` on the scikit-learn website. The page is titled "sklearn.preprocessing.KBinsDiscretizer" and includes the following information:

- Class Definition:** `class sklearn.preprocessing.KBinsDiscretizer(n_bins=5, *, encode='onehot', strategy='quantile', dtype=None, subsample='warn', random_state=None)`
- Description:** "Bin continuous data into intervals."
- Read more:** "Read more in the [User Guide](#)."
- Version:** "New in version 0.20."
- Parameters:**
 - n_bins :** *int or array-like of shape (n_features,)*, **default=5**
The number of bins to produce. Raises `ValueError` if `n_bins < 2`.
 - encode :** *{'onehot', 'onehot-dense', 'ordinal'}*, **default='onehot'**
Method used to encode the transformed result.
 - 'onehot': Encode the transformed result with one-hot encoding and return a sparse matrix. Ignored features are always stacked to the right.
 - 'onehot-dense': Encode the transformed result with one-hot encoding and return a dense array. Ignored features are always stacked to the right.
 - 'ordinal': Return the bin identifier encoded as an integer value.
 - strategy :** *{'uniform', 'quantile', 'kmeans'}*, **default='quantile'**
Strategy used to define the widths of the bins.
 - 'uniform': All bins in each feature have identical widths.
 - 'quantile': All bins in each feature have the same number of points.
 - 'kmeans': Values in each bin have the same nearest center of a 1D k-means cluster.
 - dtype :** *{np.float32, np.float64}*, **default=None**
The desired data-type for the output. If `None`, output dtype is consistent with input dtype. Only `np.float32` and `np.float64` are supported.
New in version 0.24.