TheDataLytics

scikit learn

sklearn.preprocessing.OneHotEncoder

sklearn.preprocessing.OneHotEncoder¶

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

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 a neural network or an unregularized regression.

However, dropping one category breaks the symmetry of the original representation and can therefore induce a bias in downstream models, for instance for penalized linear classification or regression models.

- None : retain all features (the default).
- 'first' : drop the first category in each feature. If only one category is present, the feature will be dropped entirely.
- 'if_binary' : drop the first category in each feature with two categories. Features with 1 or more than 2 categories are left intact.
- array : `drop[i]` is the category in feature `X[:, i]` that should be dropped.

New in version 0.21: The parameter `drop` was added in 0.21.
Changed in version 0.23: The option `drop='if_binary'` was added in 0.23.

sparse*bool, default=True*
Will return sparse matrix if set True else will return an array.

dtype*number type, default=float*
Desired dtype of output.

handle_unknown*{'error', 'ignore'}, default='error'*
Whether to raise an error or ignore if an unknown categorical feature is present during transform (default is to raise). When this parameter is set to 'ignore' and an unknown category is encountered during transform, the resulting one-hot encoded columns for this feature will be all zeros. In the inverse transform, an unknown category will be denoted as None.

Attributes

categories_*list of arrays*
The categories of each feature determined during fitting (in order of the features in X and corresponding with the output of `transform`). This includes the category specified in `drop` (if any).

drop_idx_*array of shape (n_features,)*

- `drop_idx_[i]` is the index in `categories_[i]` of the category to be dropped for each feature.
- `drop_idx_[i] = None` if no category is to be dropped from the feature with index `i`, e.g. when `drop='if_binary'` and the feature isn't binary.
- `drop_idx_ = None` if all the transformed features will be retained.

Changed in version 0.23: Added the possibility to contain `None` values.

n_features_in_*int*

Number of features seen during fit.

New in version 1.0.

feature_names_in_*ndarray of shape (n_features_in_,)*

Names of features seen during fit. Defined only when `X` has feature names that are all strings.

New in version 1.0.

See also

`OrdinalEncoder`

Performs an ordinal (integer) encoding of the categorical features.

`sklearn.feature_extraction.DictVectorizer`

Performs a one-hot encoding of dictionary items (also handles string-valued features).

`sklearn.feature_extraction.FeatureHasher`

Performs an approximate one-hot encoding of dictionary items or strings.

`LabelBinarizer`

Binarizes labels in a one-vs-all fashion.

`MultiLabelBinarizer`

Transforms between iterable of iterables and a multilabel format, e.g. a (samples x classes) binary matrix indicating the presence of a class label.

Examples

Given a dataset with two features, we let the encoder find the unique values per feature and transform the data to a binary one-hot encoding.

```
>>>
>>> from sklearn.preprocessing import OneHotEncoder
```

One can discard categories not seen during `fit`:

```
>>>
>>> enc = OneHotEncoder(handle_unknown='ignore')
>>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
>>> enc.fit(X)
```

```
OneHotEncoder(handle_unknown='ignore')
>>> enc.categories_
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
>>> enc.transform([['Female', 1], ['Male', 4]]).toarray()
array([[1., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.]])
>>> enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])
array([['Male', 1],
       [None, 2]], dtype=object)
>>> enc.get_feature_names_out(['gender', 'group'])
array(['gender_Female', 'gender_Male', 'group_1', 'group_2', 'group_3'], ...)
```

One can always drop the first column for each feature:

```
>>>
>>> drop_enc = OneHotEncoder(drop='first').fit(X)
>>> drop_enc.categories_
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
>>> drop_enc.transform([['Female', 1], ['Male', 2]]).toarray()
array([[0., 0., 0.],
       [1., 1., 0.]])
```

Or drop a column for feature only having 2 categories:

```
>>>
>>> drop_binary_enc = OneHotEncoder(drop='if_binary').fit(X)
>>> drop_binary_enc.transform([['Female', 1], ['Male', 2]]).toarray()
array([[0., 1., 0., 0.],
       [1., 0., 1., 0.]])
```

Methods

`fit`(X[, y])	Fit OneHotEncoder to X.
`fit_transform`(X[, y])	Fit OneHotEncoder to X, then transform X.
`get_feature_names`([input_features])	DEPRECATED: get_feature_names is deprecated in 1.0 and will be removed in 1.2.
`get_feature_names_out`([input_features])	Get output feature names for transformation.
`get_params`([deep])	Get parameters for this estimator.
`inverse_transform`(X)	Convert the data back to the original representation.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X)	Transform X using one-hot encoding.

fit(*X*, *y=None*)[source]

Fit OneHotEncoder to X.

Parameters

X ***array-like of shape (n_samples, n_features)***

The data to determine the categories of each feature.

y ***None***

Ignored. This parameter exists only for compatibility with `Pipeline`.

Returns

self

Fitted encoder.

fit_transform(*X*, *y=None*)[source]

Fit OneHotEncoder to X, then transform X.

Equivalent to fit(X).transform(X) but more convenient.

Parameters

X ***array-like of shape (n_samples, n_features)***

The data to encode.

y ***None***

Ignored. This parameter exists only for compatibility with `Pipeline`.

Returns

X_out ***{ndarray, sparse matrix} of shape (n_samples, n_encoded_features)***

Transformed input. If `sparse=True`, a sparse matrix will be returned.

get_feature_names(*input_features=None*)[source]

DEPRECATED: get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.

Return feature names for output features.

Parameters

input_features ***list of str of shape (n_features,)***

String names for input features if available. By default, “x0”, “x1”, … “xn_features” is used.

Returns

output_feature_names ***ndarray of shape (n_output_features,)***

Array of feature names.

get_feature_names_out(*input_features=None*)[source]

Get output feature names for transformation.

Parameters

input_features ***array-like of str or None, default=None***

Input features.

- If `input_features` is `None`, then `feature_names_in_` is used as feature names in. If `feature_names_in_` is not defined, then names are generated: `[x0, x1, ..., x(n_features_in_)]`.
- If `input_features` is an array-like, then `input_features` must match `feature_names_in_` if `feature_names_in_` is defined.

Returns

feature_names_out ***ndarray of str objects***

Transformed feature names.

get_params(*deep=True*)[source]

Get parameters for this estimator.

Parameters

deep ***bool, default=True***

If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params ***dict***

Parameter names mapped to their values.

inverse_transform(*X*)[source]

Convert the data back to the original representation.

When unknown categories are encountered (all zeros in the one-hot encoding), `None` is used to represent this category. If the feature with the unknown category has a dropped category, the dropped category will be its inverse.

Parameters

X *{array-like, sparse matrix} of shape (n_samples, n_encoded_features)*

The transformed data.

Returns

X_tr *ndarray of shape (n_samples, n_features)*

Inverse transformed array.

set_params(***params*)[source]

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params*** *dict*

Estimator parameters.

Returns

self *estimator instance*

Estimator instance.

transform(*X*)[source]

Transform X using one-hot encoding.

Parameters

X *array-like of shape (n_samples, n_features)*

The data to encode.

Returns

X_out *{ndarray, sparse matrix} of shape (n_samples, n_encoded_features)*

Transformed input. If `sparse=True`, a sparse matrix will be returned.

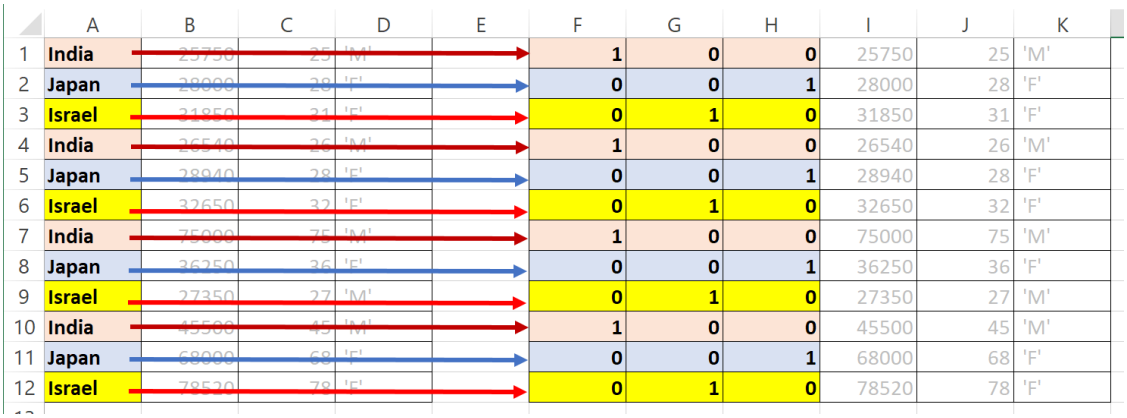
#Case 1

```
import pandas as pd
import numpy as np
```

Location	Salary	Age	Gender	Purchase
India	25750	25	M	Y
Japan	28000	28	M	N
Israel	31850	31	F	N
India	26540	26	F	Y
Japan	28940	28	F	Y
Israel	32650	32	M	N
India	75000	75	M	N
Japan	36250	36	F	Y
Israel	27350	27	F	Y
India	45500	45	F	N
Japan	68000	68	M	N
Israel	78520	78	F	Y

```
dataset = pd.read_csv('data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X,y
```

```
(array([['India', 25750, 25, 'M'],
        ['Japan', 28000, 28, 'F'],
        ['Israel', 31850, 31, 'F'],
        ['India', 26540, 26, 'M'],
        ['Japan', 28940, 28, 'F'],
        ['Israel', 32650, 32, 'F'],
        ['India', 75000, 75, 'M'],
        ['Japan', 36250, 36, 'F'],
        ['Israel', 27350, 27, 'M'],
        ['India', 45500, 45, 'M'],
        ['Japan', 68000, 68, 'F'],
        ['Israel', 78520, 78, 'F']], dtype=object),
 array(['Y', 'N', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'N', 'Y'],
       dtype=object))
```



	A	B	C	D	E	F	G	H	I	J	K
1	India	25750	25	'M'		1	0	0	25750	25	'M'
2	Japan	28000	28	'F'		0	0	1	28000	28	'F'
3	Israel	31850	31	'F'		0	1	0	31850	31	'F'
4	India	26540	26	'M'		1	0	0	26540	26	'M'
5	Japan	28940	28	'F'		0	0	1	28940	28	'F'
6	Israel	32650	32	'F'		0	1	0	32650	32	'F'
7	India	75000	75	'M'		1	0	0	75000	75	'M'
8	Japan	36250	36	'F'		0	0	1	36250	36	'F'
9	Israel	27350	27	'M'		0	1	0	27350	27	'M'
10	India	45500	45	'M'		1	0	0	45500	45	'M'
11	Japan	68000	68	'F'		0	0	1	68000	68	'F'
12	Israel	78520	78	'F'		0	1	0	78520	78	'F'

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),
[0])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
X
```

```
array([[1.0, 0.0, 0.0, 25750, 25, 'M'],
       [0.0, 0.0, 1.0, 28000, 28, 'F'],
       [0.0, 1.0, 0.0, 31850, 31, 'F'],
       [1.0, 0.0, 0.0, 26540, 26, 'M'],
       [0.0, 0.0, 1.0, 28940, 28, 'F'],
       [0.0, 1.0, 0.0, 32650, 32, 'F'],
       [1.0, 0.0, 0.0, 75000, 75, 'M'],
       [0.0, 0.0, 1.0, 36250, 36, 'F'],
       [0.0, 1.0, 0.0, 27350, 27, 'M'],
       [1.0, 0.0, 0.0, 45500, 45, 'M'],
       [0.0, 0.0, 1.0, 68000, 68, 'F'],
       [0.0, 1.0, 0.0, 78520, 78, 'F']], dtype=object)
```

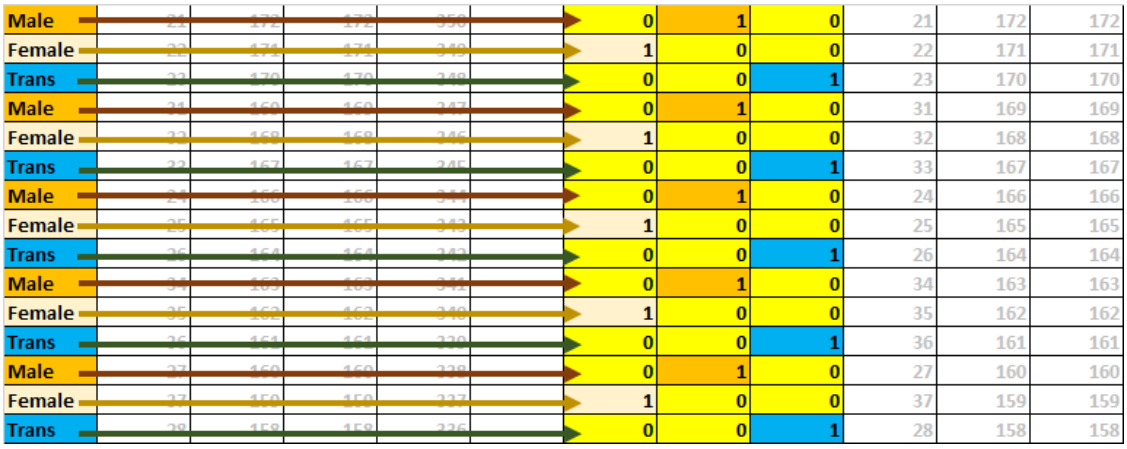
#Case 2

```
import pandas as pd
import numpy as np
```

Gender	Age	Height	Weight	BMI
Male	21	172	172	350
Female	22	171	171	349
Trans	23	170	170	348
Male	31	169	169	347
Female	32	168	168	346
Trans	33	167	167	345
Male	24	166	166	344
Female	25	165	165	343
Trans	26	164	164	342
Male	34	163	163	341
Female	35	162	162	340
Trans	36	161	161	339
Male	27	160	160	338
Female	37	159	159	337
Trans	28	158	158	336

```
dataset = pd.read_csv('BMI-GenAge.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X,y
```

```
(array([['Male', 21, 172, 172],
        ['Female', 22, 171, 171],
        ['Trans', 23, 170, 170],
        ['Male', 31, 169, 169],
        ['Female', 32, 168, 168],
        ['Trans', 33, 167, 167],
        ['Male', 24, 166, 166],
        ['Female', 25, 165, 165],
        ['Trans', 26, 164, 164],
        ['Male', 34, 163, 163],
        ['Female', 35, 162, 162],
        ['Trans', 36, 161, 161],
        ['Male', 27, 160, 160],
        ['Female', 37, 159, 159],
        ['Trans', 28, 158, 158]], dtype=object),
 array([350, 349, 348, 347, 346, 345, 344, 343, 342, 341, 340, 339,
338,
        337, 336]))
```




```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),
[0])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
X
```

```
array([[0.0, 1.0, 0.0, 21, 172, 172],
       [1.0, 0.0, 0.0, 22, 171, 171],
       [0.0, 0.0, 1.0, 23, 170, 170],
       [0.0, 1.0, 0.0, 31, 169, 169],
       [1.0, 0.0, 0.0, 32, 168, 168],
       [0.0, 0.0, 1.0, 33, 167, 167],
       [0.0, 1.0, 0.0, 24, 166, 166],
       [1.0, 0.0, 0.0, 25, 165, 165],
       [0.0, 0.0, 1.0, 26, 164, 164],
       [0.0, 1.0, 0.0, 34, 163, 163],
       [1.0, 0.0, 0.0, 35, 162, 162],
       [0.0, 0.0, 1.0, 36, 161, 161],
       [0.0, 1.0, 0.0, 27, 160, 160],
       [1.0, 0.0, 0.0, 37, 159, 159],
       [0.0, 0.0, 1.0, 28, 158, 158]], dtype=object)
```

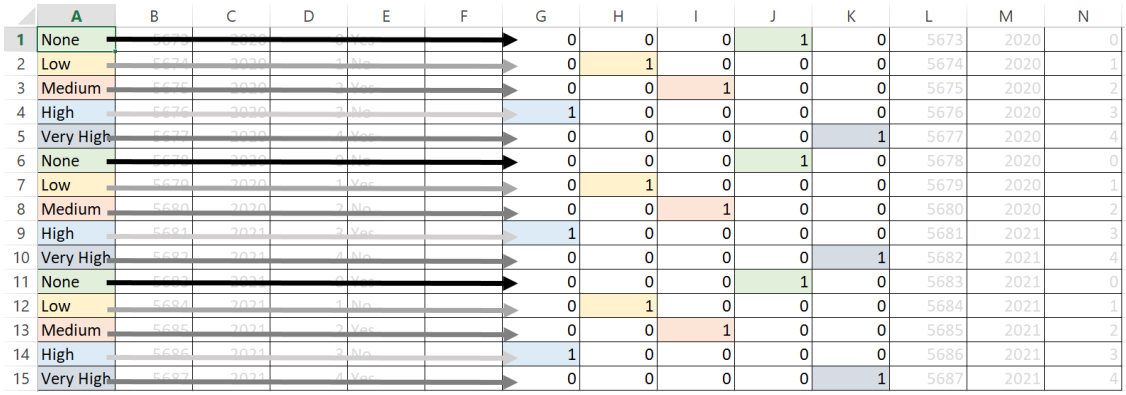
#Case 3

```
import pandas as pd
import numpy as np
```

Level	task	Year	Priority	Solved
None	5673	2020	0	Yes
Low	5674	2020	1	No
Medium	5675	2020	2	Yes
High	5676	2020	3	No
Very High	5677	2020	4	Yes
None	5678	2020	0	No
Low	5679	2020	1	Yes
Medium	5680	2020	2	No
High	5681	2021	3	Yes
Very High	5682	2021	4	No
None	5683	2021	0	Yes
Low	5684	2021	1	No
Medium	5685	2021	2	Yes
High	5686	2021	3	No
Very High	5687	2021	4	Yes

```
dataset = pd.read_csv('severity_levels.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X,y
```

```
(array([['None', 5673, 2020, 0],
        ['Low', 5674, 2020, 1],
        ['Medium', 5675, 2020, 2],
        ['High', 5676, 2020, 3],
        ['Very High', 5677, 2020, 4],
        ['None', 5678, 2020, 0],
        ['Low', 5679, 2020, 1],
        ['Medium', 5680, 2020, 2],
        ['High', 5681, 2021, 3],
        ['Very High', 5682, 2021, 4],
        ['None', 5683, 2021, 0],
        ['Low', 5684, 2021, 1],
        ['Medium', 5685, 2021, 2],
        ['High', 5686, 2021, 3],
        ['Very High', 5687, 2021, 4]], dtype=object),
 array(['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes',
'No',
        'Yes', 'No', 'Yes', 'No', 'Yes'], dtype=object))
```



	A	G	H	I	J	K	L	M	N
1	None	0	0	0	1	0	5673	2020	0
2	Low	0	1	0	0	0	5674	2020	1
3	Medium	0	0	1	0	0	5675	2020	2
4	High	1	0	0	0	0	5676	2020	3
5	Very High	0	0	0	0	1	5677	2020	4
6	None	0	0	0	1	0	5678	2020	0
7	Low	0	1	0	0	0	5679	2020	1
8	Medium	0	0	1	0	0	5680	2020	2
9	High	1	0	0	0	0	5681	2021	3
10	Very High	0	0	0	0	1	5682	2021	4
11	None	0	0	0	1	0	5683	2021	0
12	Low	0	1	0	0	0	5684	2021	1
13	Medium	0	0	1	0	0	5685	2021	2
14	High	1	0	0	0	0	5686	2021	3
15	Very High	0	0	0	0	1	5687	2021	4

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),
[0])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
X
```

```
array([[0.0, 0.0, 0.0, 1.0, 0.0, 5673, 2020, 0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 5674, 2020, 1],
       [0.0, 0.0, 1.0, 0.0, 0.0, 5675, 2020, 2],
       [1.0, 0.0, 0.0, 0.0, 0.0, 5676, 2020, 3],
       [0.0, 0.0, 0.0, 0.0, 1.0, 5677, 2020, 4],
       [0.0, 0.0, 0.0, 1.0, 0.0, 5678, 2020, 0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 5679, 2020, 1],
       [0.0, 0.0, 1.0, 0.0, 0.0, 5680, 2020, 2],
       [1.0, 0.0, 0.0, 0.0, 0.0, 5681, 2021, 3],
       [0.0, 0.0, 0.0, 0.0, 1.0, 5682, 2021, 4],
       [0.0, 0.0, 0.0, 1.0, 0.0, 5683, 2021, 0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 5684, 2021, 1],
       [0.0, 0.0, 1.0, 0.0, 0.0, 5685, 2021, 2],
       [1.0, 0.0, 0.0, 0.0, 0.0, 5686, 2021, 3],
       [0.0, 0.0, 0.0, 0.0, 1.0, 5687, 2021, 4]], dtype=object)
```

#Case 4

```
import pandas as pd
import numpy as np
```

Stream	Vertical	Dept	Rank	Bridge
IT	1	Sales	5	Yes
Finance	1	Sales	4	No
Mechnical	1	Sales	3	Yes
Education	1	Sales	2	No
Pharma	1	Sales	1	Yes
Medical	1	Sales	6	No
Banking	1	Sales	7	Yes
IT	2	Purchase	7	No
Finance	2	Purchase	6	Yes
Mechnical	2	Purchase	5	No
Education	2	Purchase	4	Yes
Pharma	2	Purchase	3	No
Medical	2	Purchase	2	Yes
Banking	2	Purchase	1	No
IT	3	HR	1	Yes
Finance	3	HR	2	No
Mechnical	3	HR	3	Yes
Education	3	HR	4	No
Pharma	3	HR	5	Yes
Medical	3	HR	6	No
Banking	3	HR	7	Yes

```
dataset = pd.read_csv('stream_vertical.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X,y
```

```
(array([['IT', 1, 'Sales', 5],
        ['Finance', 1, 'Sales', 4],
        ['Mechnical', 1, 'Sales', 3],
        ['Education', 1, 'Sales', 2],
        ['Pharma', 1, 'Sales', 1],
        ['Medical', 1, 'Sales', 6],
        ['Banking', 1, 'Sales', 7],
        ['IT', 2, 'Purchase', 7],
        ['Finance', 2, 'Purchase', 6],
        ['Mechnical', 2, 'Purchase', 5],
```

```
        ['Education', 2, 'Purchase', 4],
        ['Pharma', 2, 'Purchase', 3],
        ['Medical', 2, 'Purchase', 2],
        ['Banking', 2, 'Purchase', 1],
        ['IT', 3, 'HR', 1],
        ['Finance', 3, 'HR', 2],
        ['Mechnical', 3, 'HR', 3],
        ['Education', 3, 'HR', 4],
        ['Pharma', 3, 'HR', 5],
        ['Medical', 3, 'HR', 6],
        ['Banking', 3, 'HR', 7]], dtype=object),
 array(['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes',
'No',
        'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes',
'No',
        'Yes'], dtype=object))
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),
[0])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
X
```

```
array([[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1, 'Sales', 5],
       [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1, 'Sales', 4],
       [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1, 'Sales', 3],
       [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1, 'Sales', 2],
       [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1, 'Sales', 1],
       [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1, 'Sales', 6],
       [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1, 'Sales', 7],
       [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 2, 'Purchase', 7],
       [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2, 'Purchase', 6],
       [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 2, 'Purchase', 5],
       [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2, 'Purchase', 4],
       [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 2, 'Purchase', 3],
       [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 2, 'Purchase', 2],
       [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2, 'Purchase', 1],
       [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 3, 'HR', 1],
       [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 3, 'HR', 2],
       [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 3, 'HR', 3],
       [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3, 'HR', 4],
       [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 3, 'HR', 5],
       [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 3, 'HR', 6],
       [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3, 'HR', 7]], dtype=object)
```

IT	1	Sales	5	Yes	0	0	0	1	0	0	0	1	'Sales'	5
Finance	1	Sales	4	No	0	0	1	0	0	0	0	1	'Sales'	4
Mechnical	1	Sales	3	Yes	0	0	0	0	1	0	0	1	'Sales'	3
Education	1	Sales	2	No	0	1	0	0	0	0	0	1	'Sales'	2
Pharma	1	Sales	1	Yes	0	0	0	0	0	0	1	1	'Sales'	1
Medical	1	Sales	6	No	0	0	0	0	0	1	0	1	'Sales'	6
Banking	1	Sales	7	Yes	1	0	0	0	0	0	0	1	'Sales'	7
IT	2	Purchase	7	No	0	0	0	1	0	0	0	2	'Purchase	7
Finance	2	Purchase	6	Yes	0	0	1	0	0	0	0	2	'Purchase	6
Mechnical	2	Purchase	5	No	0	0	0	0	1	0	0	2	'Purchase	5
Education	2	Purchase	4	Yes	0	1	0	0	0	0	0	2	'Purchase	4
Pharma	2	Purchase	3	No	0	0	0	0	0	0	1	2	'Purchase	3
Medical	2	Purchase	2	Yes	0	0	0	0	0	1	0	2	'Purchase	2
Banking	2	Purchase	1	No	1	0	0	0	0	0	0	2	'Purchase	1
IT	3	HR	1	Yes	0	0	0	1	0	0	0	3	'HR'	1
Finance	3	HR	2	No	0	0	1	0	0	0	0	3	'HR'	2
Mechnical	3	HR	3	Yes	0	0	0	0	1	0	0	3	'HR'	3
Education	3	HR	4	No	0	1	0	0	0	0	0	3	'HR'	4
Pharma	3	HR	5	Yes	0	0	0	0	0	0	1	3	'HR'	5
Medical	3	HR	6	No	0	0	0	0	0	1	0	3	'HR'	6
Banking	3	HR	7	Yes	1	0	0	0	0	0	0	3	'HR'	7