from sklearn: Preprocessing

Time limit for program runtime	**10 seconds**
Memory limit	**128Mb**
Input	**standard input or input.txt**
Output	**standard output or output.txt**

Prepare the data for subsequent work with the machine learning model. To do this, use ready-made methods from the libraries: `sklearn.preprocessing.MinMaxScaler` and `sklearn.model_selection.train_test_split`.

Write a `preprocessing(X: np.ndarray, y: np.ndarray, test_size=0.33)` function with the following input parameters:

- X: NumPy matrix of object features.
- y: NumPy vector of true labels (target).
- `test_size`: Specifies the size of the test dataset as `test_size` · 100% of the entire sample (set the default value of the keyword argument to `test_size`=0.33).

Note that the data must be shuffled before splitting. In the `sklearn.model_selection.train_test_split` method, set the default value of the keyword argument `random_state`=1234 for correct function validation.

The function returns data scaled using the `MinMaxScale` method and split into training and test datasets, where the length of the test dataset is `test_size` of the entire sample. The output must be a tuple of 4 items: `(X_train: np.ndarray, y_train: np.ndarray, X_test: np.ndarray, y_test: np.ndarray)`.

*Make sure to follow the correct order of operations for processing data.

Sample

Input	Output

`import numpy as np`
 	
`X, y = np.array([[1, 2], [3, 4], [1, 2]]), np.array([1, 2, 3])`
`X_train, y_train, X_test, y_test = preprocessing(X, y, test_size=0.33)`
`print(X_train, y_train, X_test, y_test, sep='\n')`
	`[[1. 1.]`
	` [0. 0.]]`
	`[2 3]`
	`[[0. 0.]]`
	`[1]`

Notes

The file submitted to the testing system must only contain the function described in the task and, possibly, helper functions. Make sure to also import the libraries that you've used.

Hold-Out validation

Time limit for program runtime	**1 second**
Memory limit	**64 Mb**
Input	**standard input or input.txt**
Output	**standard output or output.txt**

Implement `Hold-Out validation`, replicating the behavior of the method `sklearn.model_selection.train_test_split`. You can't use the ready-made method from the library.

Write a `train_test_split(X: np.ndarray, y: np.ndarray, test_size=0.33)` function with the following input parameters:

- X: NumPy matrix of object features.
- y: NumPy vector of true labels (target).
- `test_size`: Specifies the size of the test dataset as `test_size` · 100% of the entire sample (set the default value of the keyword argument to `test_size=0.33`).

The function returns data split into training and test datasets, where the length of the test dataset is `test_size` of the entire sample. The size of the test dataset must be rounded according to the logic implemented by the `round()` function. The output must be a tuple of 4 items: `(X_train: np.ndarray, y_train: np.ndarray, X_test: np.ndarray, y_test: np.ndarray)`.

Note that the data must be shuffled before splitting. For `np.random.seed`, use the parameter value `seed=1234`.

Attention! You need to shuffle the indexes of the objects.

The shuffled data will be split into subsamples in such a way that the first (test_size of the data size) shuffled indexes belong to the test sample, and the rest to the training sample. This is necessary for correct function validation in the checking system.

Sample

Input	Output

`import numpy as np`
 	
`X, y = np.array([[2, 3], [2, 3], [2, 3]]), np.array([1, 1, 1])`
`X_train, y_train, X_test, y_test = train_test_split(X, y, test_size=0.33)`
`print(X_train, y_train, X_test, y_test, sep='\n')`
	`[[2 3]`
	` [2 3]]`
	`[1 1]`
	`[[2 3]`
	`[1]`

Notes

The file submitted to the testing system must only contain the function described in the task and, possibly, helper functions. Make sure to also import the libraries that you've used.

OneHot Encoder

Time limit for program runtime	**1 second**
Memory limit	**64Mb**
Input	**standard input or input.txt**
Output	**standard output or output.txt**

Implement a simplified version of `one_hot_encoding` transformation without using `pd.get_dummies` and `sklearn.preprocessing.OneHotEncoder`.

Write a `onehot_encoding(X: np.ndarray) -> np.ndarray` function that takes feature column X as input, represented as a one-dimensional NumPy array of a size matching the number of objects with categorical values. The function must return a NumPy integer matrix of the size (number of objects · number of unique feature values) filled with `one_hot_encoding` 0 or 1 depending on the feature value.

The binary vectors in the matrix are sorted by categorical feature value in ascending order: if the feature takes the values b, a, and c, then the leftmost vector in the one_hot matrix will be the vector for a, and the rightmost vector will be the vector for c.

Sample

Input	Output

`import numpy as np`
 	
`x = np.array([3, 2, 2, 1])`
`print(onehot_encoding(x))`
	`[[0 0 1]`
	` [0 1 0]`
	` [0 1 0]`
	` [1 0 0]]`

Notes

The file submitted to the testing system must only contain the function described in the task and, possibly, helper functions. Make sure to also import the libraries that you've used.

MinMax Scaler

Time limit for program runtime	**1 second**
Memory limit	**64Mb**
Input	**standard input or input.txt**
Output	**standard output or output.txt**

Implement `MinMaxScaler` transformation, replicating the behavior of the `sklearn.preprocessing.MinMaxScaler` method. You can't use the ready-made method from the library.

Write a `minmax_scale(X: np.ndarray) -> np.ndarray` function that takes feature matrix X in NumPy array format and returns a matrix scaled using the `MinMaxScaler` method in NumPy array format.

*Make sure to account for cases where the feature takes on a single value. In such scenarios, the function should return the minimum possible normalized value.

Sample

Input	Output

`import numpy as np`
 	
`X = np.array([[1, 2],`
`[2, 1]])`
`print(minmax_scale(X))`
	`[[0. 1.]`
	`[1. 0.]]`

Notes

The file submitted to the testing system must only contain the function described in the task and, possibly, helper functions. Make sure to also import the libraries that you've used.