```
import os
import numpy as np
import pandas as pd
from pathlib import Path
from tensorflow.python.keras.utils.np_utils import to_categorical


def read_painmonit_np(label):
    ''' Function to read the UzL dataset in form of numpys created by script "create_np_files".
    Parameters
    ----------

    label: String. Either 'covas' or 'heater' depending on the label you want to use.

    Returns
    -------
    X, y, subjects: np.
    '''
    np_dir = Path("datasets", "painmonit", "np-dataset")
    return np.load(Path(np_dir, "X.npy")), np.load(Path(np_dir, "y_{}.npy".format(label))), np.load(Path(np_dir, "subjects.npy"))


def read_biovid_np():
    ''' Function to read the BioVid dataset in form of numpys created by script "create_np_files".
    Parameters
    ----------

    label: String. Either 'covas' or 'heater' depending on the label you want to use.

    Returns
```

```
        -------

        X, y, subjects: np.

        '''

        np_dir = Path("datasets", "biovid", "np-dataset")

        return np.load(Path(np_dir, "X.npy")), np.load(Path(np_dir, "y.npy")), np.load(Path(np_dir,
"subjects.npy"))


def get_initials(file_name, n= 4):

        """

        Returns the initials in a given filename.


        Returns the first 'n' characters after the last os directory seperator (windows: '\\') inside the
string.

        Example: '\\data-files\\Ad Wa-2019-12-19_11-03-14'' -> 'Ad Wa' with n=5



        Parameters

        ----------

        file_name: String. Example: "\\synchronised-data-files\\Ad Wa-2019-12-19_11-03-14"


        n: Int. Number of characters to extract after last os seperator.


        Returns

        -------

        string: Initials. "Ad Wa"

        """


        # Find last occurrence of "\"

        index = file_name.rfind(os.path.sep)
```

```
        # Get initials -> assumes that the first 5 letters of the filename describe the initials (e.g. "***\\Ad Wa***")
        initials = file_name[index + 1: index + 1 + n]

        return initials

def from_categorical(x):
        """Returns the class vector for a given one-hot vector.

        Parameters
        ----------
        x: np. One-hot vector.

        Returns
        -------
        np: np.argmax(x, axis= 1)
        """
        return np.argmax(x, axis= 1)

def unison_shuffled(x):
        ''' Shuffles several numpy/pandas arrays (given as list) unison.
        Parameters
        ----------

        x: list of np.

        Returns
        -------
        y: list of np.
```

```
    '''
    # Check all elements have same length
    assert len({len(i) for i in x if i is not None}) == 1
    # Create a random permutation
    p = np.random.permutation(len(x[0]))
    # Apply permutation to all elements
    return [i[p] if isinstance(i, np.ndarray) else i.reset_index().reindex(p).drop(["index"], axis = 1) if isinstance(i, pd.DataFrame) else None for i in x]


def pick_classes(data, y, classes, input_is_categorical= False):
    """Function to pick certain classes for given data.

    Parameters
    ----------
    data: list. Data to pick from.
    y: np. Label of the dataset.
    classes: list/np. List of lists with class labels. For example, [[0], [4]], or [[0], [3, 4]]
    input_is_categorical: bool. Should be set True if y is given as categorical. Otherwise False. Default is False.

    Returns
    -------
    list: data for the two class problem
    np: label for the two class problem
    """
    if input_is_categorical:
        y = from_categorical(y)

    # Create a dict to convert class labels: picked_classes [[0], [4]] -> 0:0, 4:1 or [[0], [3, 4]] -> 0:0, 3:1, 4:1
```

```
        d = {}
        for class_idx, class_group in enumerate(classes):
                for class_name in class_group:
                        d[class_name]= class_idx

        # Convert initial class label to new ones and set not used ones to '-1'
        y = np.array([d[x] if x in d else -1 for x in y])

        # Create index list for elements according the picked classes
        indices = [False] * len(y)
        for c in np.unique(list(d.values())):
                indices = indices | (y == c)

        # Add label vector to the data list
        data.append(y)

        # Select elements absed on the index list for all dataframes in 'data'
        data = pick_data(data= data, condition= indices)

        # if input was categorical, convert 'y' to categorical again
        if input_is_categorical:
                data[-1] = to_categorical(data[-1])

        return data

def pick_data(data, condition):
        """Function to pick certain data for given data under given condition.

        Parameters
```

```
    ----------
    data: list. Data to pick from
    condition: list of bool.

    Returns
    -------
    list: selected data -> return [i[condition] for i in data]
    """
    return [i[condition] if i is not None else None for i in data]


def min_max_scaling(x):
    return (x - x.min()) / (x.max() - x.min())


def normalize_features(x):
    if not isinstance(x, pd.DataFrame):
        raise NotImplementedError("'normalize_features' expects pandas dataframe but received type: '{}'".format(type(x)))

    for column in x.columns:
        x[column]= min_max_scaling(x[column])

    return x


def normalize(x):
    """Function to normalize the data of given to a CNN.
    Expects a np dataframe in shape (num samples, time series data, sensors, channels).
    Normalizes each 'times series data' for each sample independent.

    Parameters
```

```
        ----------
        x: np.

        Returns
        -------
        normalized data.
        """

        if x is None:
            return None

        for i, data in enumerate(x):
            for y in np.arange(data.shape[1]):
                max_value = np.max(data[:, y])
                min_value = np.min(data[:, y])
                if max_value != min_value:
                    data[:, y] = (data[:, y] - min_value) / (max_value - min_value)
                else:
                    if max_value > 1:
                        data[:, y] = 1
            x[i] = data

        return x

def resample_by_interpolation(signal, input_fs, output_fs):

    scale = output_fs / input_fs
    # calculate new length of sample
    n = round(len(signal) * scale)
```

```
    # use linear interpolation
    # endpoint keyword means than linspace doesn't go all the way to 1.0
    # If it did, there are some off-by-one errors
    # e.g. scale=2.0, [1,2,3] should go to [1,1.5,2,2.5,3,3]
    # but with endpoint=True, we get [1,1.4,1.8,2.2,2.6,3]
    # Both are OK, but since resampling will often involve
    # exact ratios (i.e. for 44100 to 22050 or vice versa)
    # using endpoint=False gets less noise in the resampled sound
    resampled_signal = np.interp(
        np.linspace(0.0, 1.0, n, endpoint=False),  # where to interpret
        np.linspace(0.0, 1.0, len(signal), endpoint=False),  # known positions
        signal,  # known data points
    )
    return resampled_signal

def resample_axis(X, input_fs, output_fs, axis= 1):
        """Resamples an axis of a given numpy to have a given sample size. Uses resample from scipy

        Parameters
        ----------
        X: numpy. The given numpy to resample.

        sample_size: int. The sample size to resample the axis.

        axis: int. The axis to downsample.

        Returns
        -------
```

```
        np: Resampled numpy. False if not possible.
        """
        X = np.apply_along_axis(func1d= resample_by_interpolation, axis= axis, arr=X, input_fs= input_fs, output_fs= output_fs)


        return X
```