Příloha 15

Kódové provedení shaker_modul

prepare_data – úprava dat pro vstup do STUDNA (včetně odstranění duplikátů apod.)
matdf_to_datetime – umožňuje převod času v string na python formát datetime
check_timesample – kontrola konstantního vzorkování
take_care_of_duplicates – odstranění duplikátů v datech
resample – zajištění konstantního vzorkování dat

```
import datetime
import pandas as pd
import matplotlib.pyplot as plt


def prepare_data(data, reduction=2):
    """
    This function uses all the functions below in order to prepare data
from
    shaker for futhure use.

    """
    data = data.copy()

    # Taking care of timestamp
    data = matdf_to_datetime(data)

    # Taking care of duplicates
    repaired_data, sample_rate = take_care_of_duplicates(data)

    # Take care of sample rate
    finished_data = resample(repaired_data, 'Time', reduction,
sample_rate=sample_rate)

    return finished_data


def matdf_to_datetime(data, column='Time', prefix='2022-05-17 00:00:0'):
    """
    Data acquired from matlab are edited and transformed to dataframe.

    Returns
    ----------
    dataframe : pandas
        Inseted dataframe with change in selected column, that is
        now datetime column.
    """

    # Initial Time column contains string "10 s"
    data[column] = data[column].apply(lambda x: float(x[:-4]))

    # Transforimg data to to_datetime friendly format
    data[column] = data[column].apply(lambda x: prefix + str(format(x,
'f')))

    data[column] = pd.to_datetime(data[column]).apply(lambda x:
```

```
datetime.datetime.timestamp(x))
    return data


def check_timesample(data, column='Time', visualise=False):
    """
    Returns dataframe of TimeStamp sample times and dataframe of data
description.

    Parameters
    ----------
    data : pandas.DataFrame
        Data with column that contains timestamp data.

    column : str, Optional
        Name of the column, that contains timestamp data, else 'TimeStamp'
column
        will be searched for.

    visualise : boolean, Optional
        If true, data will be plotted

    """
    out1 = data[column] - data[column].shift(1)
    out2 = pd.DataFrame(out1.describe())

    if visualise:
        plt.figure(figsize=(15, 5))
        plt.title('Time spacing between samples'); plt.xlabel('Sample');
plt.ylabel('Sec')
        plt.plot(out1)
        plt.show()

    return out1, out2



def take_care_of_duplicates(data, level_of_removal=2, threshold_p=0.5,
remove_dropdowns=True):
    """
    Takes care of duplicates in data.

    Returns
    ---------
    new_data : pandas DataFrame
        With reduced length

    df_stats : pandas DataFrame
        Mean sample
    """

    sampling_times, _ = check_timesample(data, 'Time')

    # Get rid of non valid data
    if remove_dropdowns:
        if level_of_removal == 0:
            filt2 = [False]*len(data)

        if level_of_removal >= 1:
            filt1 = (sampling_times==0)
            filt2 = filt1
```

```
        if level_of_removal >= 2:
            filt2 = filter_surrounding(filt1, 1)

        if level_of_removal >= 3:
            filt2 = filter_surrounding(filt1, 2)

        if level_of_removal >= 4:
            filt2 = filter_surrounding(filt1, 3)

        if level_of_removal >= 5:
            threshold = max(sampling_times.dropna())*threshold_p
            filt1 = (sampling_times<=threshold)
            filt2 = filter_surrounding(filt1, 2)

        sampling_times.drop(index=sampling_times[filt2].index,
inplace=True)
        new_data = data.drop(index=data[filt2].index)

        #print(f'Data removed: {sum(filt2)} ({round(sum(filt2)/len(filt2),
2)*100} %)')

        sampling_times, df_stats = check_timesample(new_data, 'Time')

        return new_data, df_stats.loc['mean'].values[0]


def resample(data, column, n, sample_rate = None):
    """
    Resample data with mean filtering.

    Parameters
    ------------
    data : ...
    column : str
        Columns where is timestamp

    n : int
        Number of reduciton

    """

    _, df_stats = check_timesample(data, column)

    if isinstance(sample_rate, type(None)):
        sample_rate = df_stats.loc['mean'].values[0]

    sample= str(format(sample_rate*n, 'f')) + 'S'

    data['DateTime'] = data[column].apply(lambda x:
datetime.datetime.fromtimestamp(x))
    data_resampled = data.resample(sample, on='DateTime',
kind='timestamp').mean()
    return data_resampled.reset_index(drop=True)
```