"Install a pip package in the current Jupyter kernel\n", "Memo to self: Install modules with 'sys.executable' and then restart kernel"

In [6]:
```
import sys
#!{sys.executable} -m pip install --upgrade  nbconvert[webpdf]
```

In [25]:
```
#************** Start importing python modules
import time
import os # For
import pandas as pd
import numpy as np
from functools import partial # For partial functions
#******* Start importing imputation tools
from scipy.interpolate import CubicSpline
#******* End importing imputation tools
#********* Start modules clustering Frequency domain
from scipy.fft import fft,fftfreq # To calculate Fourier transform
#********* End modules clustering Frequency domain
#***** Start
import scipy.cluster.hierarchy as spc
#****** End

#************ Start GUI modules
import io
import traitlets
import ipywidgets as widgets
from IPython.display import display
from tkinter import Tk, filedialog
import matplotlib.pyplot as plt
#************ End GUI modules
#************** End importing python modules
# Start
```

Start importing data First ask the user to select the json file containing

In [8]:
```
btn_upload = widgets.FileUpload(
    accept = '*.json',
    multiple = False
)
display(btn_upload)
```

```
FileUpload(value=(), accept='*.json', description='Upload')
```

Extract the content of the json-file and convert the json to a data frame

In [9]:
```
stringContent = btn_upload.value[0]['content'].tobytes().decode("utf-8")
df =  pd.read_json(io.StringIO(stringContent),orient = 'index')
print(f'top part is {df.head()}')
print(f'bottom part is {df.tail()}')
```

```
top part is                       series_0  series_1  series_2  series_3  series_4
2022-06-01 00:00:00       0.0 -0.012866       0.0  0.009272 -0.943774  \
2022-06-01 00:01:00       0.1  0.106740       0.1 -0.004306 -0.734072
2022-06-01 00:02:00       0.2  0.209939       0.2  0.005588  3.272961
2022-06-01 00:03:00       0.3  0.293082       0.3 -0.005641  0.832609
2022-06-01 00:04:00       0.4  0.391814       0.4 -0.006649 -2.225293

                     series_5  series_6
2022-06-01 00:00:00       NaN       NaN
2022-06-01 00:01:00       NaN       NaN
2022-06-01 00:02:00       NaN       NaN
2022-06-01 00:03:00     228.0       1.0
2022-06-01 00:04:00       NaN       1.0
bottom part is                       series_0    series_1    series_2  series_3  s
eries_4
2022-06-02 23:55:00     287.5  287.498961  287.323143 -0.165455  2.017107  \
2022-06-02 23:56:00     287.6  287.599466  287.600000 -0.010588 -1.490229
2022-06-02 23:57:00     287.7  287.719646  287.700000 -0.005317 -0.667750
2022-06-02 23:58:00     287.8  287.790522  287.800000  0.005404 -3.185175
2022-06-02 23:59:00     287.9  287.910746  287.900000 -0.006259 -0.760504

                     series_5  series_6
2022-06-02 23:55:00    2375.0       3.0
2022-06-02 23:56:00       NaN       3.0
2022-06-02 23:57:00       NaN       3.0
2022-06-02 23:58:00       NaN       3.0
2022-06-02 23:59:00       NaN       3.0
```

End importing dataWe start by imputing missing values using cubic splines

In [10]:

```
# v needs to be a single column/vector
def find_index_missing(v,include_nonmissing = True):
    index_missing_values = [index for index in range(len(v)) if np.isnan(v[index
    returnValue = index_missing_values
    if(include_nonmissing):
        index_nonmissing = [index for index in range(len(v)) if index not in inc
        returnValue = index_missing_values,index_nonmissing
    #
    return returnValue
#
# By default 'extrapolate' is set to False since the behavior of cubic splines c
# and last spline can sometimes be erratic
def imputeCubicSpline(x,y,extrapolate = False):
    index_missing_values,index_nonmissing = find_index_missing(
        y,include_nonmissing=True)
    #
    if len(index_missing_values) > 0 :
        cs = CubicSpline(x[index_nonmissing],y[index_nonmissing],extrapolate = e
        y = cs(x)
    #
    return y
#
# If 'timeCol' is  None, then the index is assumed to be a timestamp. Oth
# This function imputes missing values by fitting a cubic spline to the non-miss
# If "imputationCols" is a string (single column),then it is converted to a list
# I 'imputationCols" is None,then every column in the dataframe is imputed
def df_impute_cubic(df,imputationCols=None,timeCols = None,crop = False,**kwargs
    if type(imputationCols) is str:
        imputationCols = [imputationCols]
    elif imputationCols is None:
        imputationCols = list(df.columns)
```

```
    #Datetimes need to be converted to unix time
    listOfTimes = None
    if type(timeCols) is str:
        listOfTimes = df[timeCol].tolist()
    elif timeCols is None:
        listOfTimes = list(df.index)
    #
    timeValues =  np.asarray([ts.timestamp() for ts in listOfTimes])
    #
    imputer = partial(imputeCubicSpline,timeValues,**kwargs)
    df[imputationCols] = df[imputationCols].apply(imputer)
    #
    if crop:
        df = df.dropna()
    return df
#
```

Do the imputation

In [11]:
```
df = df_impute_cubic(df,crop=True)
```

In [12]:
```
# Because frequency domain is symmetrical, take only positive frequencies
def df_fft(df,fft_cols=None,only_positive = True):
    if type(fft_cols) is str:
        fft_cols = [fft_cols]
    elif fft_cols is None:
        fft_cols = list(df.columns)
    #
    freqs = fftfreq(df.shape[0])
    # Memo to self: Need to first convert to np.array before applying fft
    df[fft_cols] = df[fft_cols].apply(lambda x: fft(x.to_numpy(),norm = "forwarc
    if only_positive:
        df = df.apply(np.abs)
        df = df[freqs > 0]
    #
    return df
#
```

In [13]:
```
# Calculate Fourier transform
df_fft_vals = df_fft(df.copy())
#*******
```

In [15]:
```
print('Earliest 5 values (after imputation and cropping) are')
print(df.head())
print(df_fft_vals.head())
print('Latest 5 values (after imputation and cropping)  are')
print(df.tail())
print(df_fft_vals.tail())
```

```
Earliest 5 values (after imputation and cropping) are
                     series_0  series_1  series_2  series_3  series_4
2022-06-01 00:03:00       0.3  0.293082  0.300000 -0.005641  0.832609  \
2022-06-01 00:04:00       0.4  0.391814  0.400000 -0.006649 -2.225293
2022-06-01 00:05:00       0.5  0.493612  2.182942  1.699679  0.584138
2022-06-01 00:06:00       0.6  0.599261  0.600000  0.009936 -1.128122
2022-06-01 00:07:00       0.7  0.688582  0.700000  0.009095  0.622713

                        series_5  series_6
2022-06-01 00:03:00   228.000000       1.0
2022-06-01 00:04:00 -1032.523994       1.0
2022-06-01 00:05:00 -1452.087743       1.0
2022-06-01 00:06:00 -1235.189494       1.0
2022-06-01 00:07:00  -586.327496       1.0
                      series_0   series_1   series_2  series_3  series_4
2022-06-01 00:04:00  45.725224  45.725269  45.725229  0.001322  0.018984  \
2022-06-01 00:05:00  22.862626  22.862566  22.862636  0.001321  0.036497
2022-06-01 00:06:00  15.241766  15.241796  15.241780  0.001403  0.041671
2022-06-01 00:07:00  11.431340  11.431322  11.431360  0.001221  0.036394
2022-06-01 00:08:00   9.145089   9.145023   9.145113  0.001235  0.042650

                      series_5  series_6
2022-06-01 00:04:00  79.089684  0.024779
2022-06-01 00:05:00  46.341023  0.029669
2022-06-01 00:06:00  55.470473  0.003957
2022-06-01 00:07:00  59.887749  0.031867
2022-06-01 00:08:00  52.813043  0.040652
Latest 5 values (after imputation and cropping)  are
                     series_0    series_1    series_2  series_3  series_4
2022-06-02 23:51:00     287.1  287.107576  287.100000 -0.003254  1.753111  \
2022-06-02 23:52:00     287.2  287.199356  287.200000  0.008578 -0.517878
2022-06-02 23:53:00     287.3  287.303431  287.300000  0.016632 -0.906323
2022-06-02 23:54:00     287.4  287.410192  287.400000  0.012792  1.437542
2022-06-02 23:55:00     287.5  287.498961  287.323143 -0.165455  2.017107

                        series_5  series_6
2022-06-02 23:51:00  1860.634963       1.0
2022-06-02 23:52:00  2601.000000       3.0
2022-06-02 23:53:00   466.000000       1.0
2022-06-02 23:54:00 -1194.701194       1.0
2022-06-02 23:55:00  2375.000000       3.0
                     series_0  series_1  series_2  series_3  series_4
2022-06-01 23:55:00  0.050001  0.050223   0.04959  0.000211  0.049747  \
2022-06-01 23:56:00  0.050000  0.049925   0.04959  0.000317  0.031375
2022-06-01 23:57:00  0.050000  0.050203   0.04959  0.000511  0.014581
2022-06-01 23:58:00  0.050000  0.049964   0.04959  0.000544  0.037133
2022-06-01 23:59:00  0.050000  0.049953   0.04959  0.000404  0.021299

                     series_5  series_6
2022-06-01 23:55:00  0.556567  0.005653
2022-06-01 23:56:00  2.856571  0.011339
2022-06-01 23:57:00  5.363970  0.009402
2022-06-01 23:58:00  2.874175  0.007480
2022-06-01 23:59:00  4.720977  0.009655
```

For single numerical columns x and y (of equal length) we choose aboluste value of the correlation og x and y as our distance metrix i,e,

$$dist(x,y) = 1 - |corr(x,y)|.$$

(1)

When either x or y have missing values we choose to calculate the correlation as the

The reason why we are using the absolute value of the ccrrelation is that the task is for asking for "relation", which we interpret as grouping columns that have a strong connection regardless of whether they are positively or negatively correlated. For gorups of colums we choose method = 'complete' as our distance metric From documentation: method='complete' assigns

$$d(u,v) = \max(dist(u[i], v[j])).$$

(2)

for all points (in our case columns) `i` in cluster u and `j` in cluster `v`., This is also known by the Farthest Point Algorithm or Voor Hees Algorithm.Description algorithm: 1. Calculate (1) defined above for each pair of columns in the data frame ("condensed values") 2. Do Agglomerative Hiearchical Clustering based

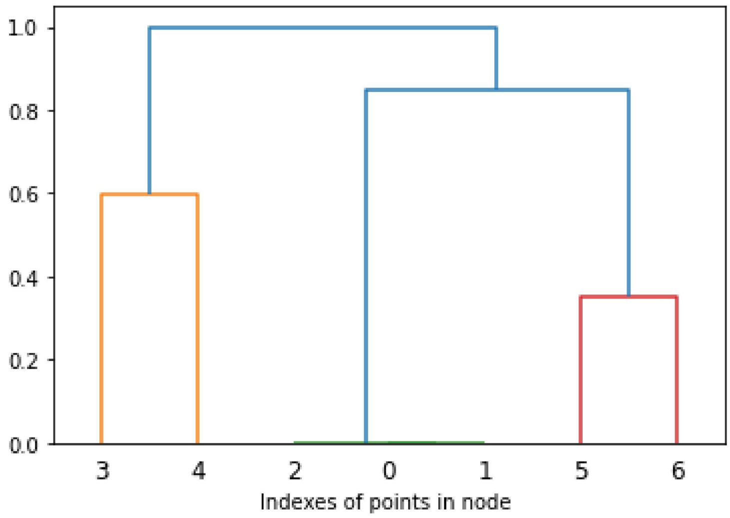
In [17]:
```
def clusterByCorr(df):
    corr = abs(df.corr(method = "pearson").values)
    #
    pdist_uncondensed = 1.0 - abs(corr)
    pdist_condensed = np.concatenate([row[i+1:] for i, row in enumerate(pdist_ur
    #
    linkage = spc.linkage(pdist_condensed, method='complete')
    idx = spc.fcluster(linkage, 0.5 * pdist_condensed.max(), 'distance')
    # Put the cluster information into a data frame. The first column is the nam
    # The second columns is an indicator that says which cluster group each colu
    groupingFrame = pd.DataFrame(list(df.columns),columns=['Series'])
    groupingFrame['Group'] = list(idx)
    #
    clusterDict = {}
    #
    for group in list(set(list(idx))):
        clusterDict['_'.join(['group',str(group)])] = [
            list(df.columns)[col] for col in list(groupingFrame[groupingFrame['G
            ]
        #
    #
    return linkage,clusterDict
```

In [18]:
```
linkage,clusterDict = clusterByCorr(df_fft_vals)
```

In [20]:
```
print(clusterDict)
```

```
{'group_1': ['series_3'], 'group_2': ['series_4'], 'group_3': ['series_0', 'serie
s_1', 'series_2'], 'group_4': ['series_5', 'series_6']}
```

In [26]:
```
spc.dendrogram(linkage)
plt.xlabel("Indexes of points in node ")
plt.show()
```




```
In [ ]:
```