Кластеризация малой выборки по DTW

In [1]:
```python
import numpy as np
import pandas as pd
import json
import os
import time
import glob
import datetime
import matplotlib.pyplot as plt
from tqdm import tqdm
from datetime import datetime
from dtaidistance import dtw
```

Нам нужна относительно маленькая выборка, чтобы по ней можно было найти центры кластеров и сравнивать их уже со всей большой выборкой. Отберем наблюдения с дневным диапазоном >50% - более яркие представители пампов и дампов.

In [2]:
```python
with open("train_list_2.txt", "r") as fp:
    train_list = json.load(fp)
```

In [3]:
```python
len(train_list)
```

Out[3]:
```
9304
```

Соберем датаесет из рядов по этой выборке:

In [5]:
```python
time_series = []
for i in tqdm(range(len(train_list))):
    parent_dir_train = 'C:/Users/Kuanysh/Downloads/pump_and_dump/train2'
    ticker_name = train_list[i][0]
    day = train_list[i][1]
    ts = pd.read_csv(os.path.join(parent_dir_train, ticker_name + '_'+ day +'.csv')).to_numpy().ravel()
    time_series.append(ts)
```

```
100%|████████████████████████████████████████████████████████████| 9304/9304 [00:06<00:00, 136
1.42it/s]
```

In [6]:
```python
with open('ts_list.npy', 'wb') as fp:
    np.save(fp, time_series)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\_asarray.py:171: VisibleDeprecationWarning: Creating an n
darray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different len
gths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarr
ay.
  return array(a, dtype, copy=False, order=order, subok=True)
```

In [4]:
```python
ts_list = np.load("ts_list.npy", allow_pickle=True)
```

In [5]:
```python
len(ts_list)
```

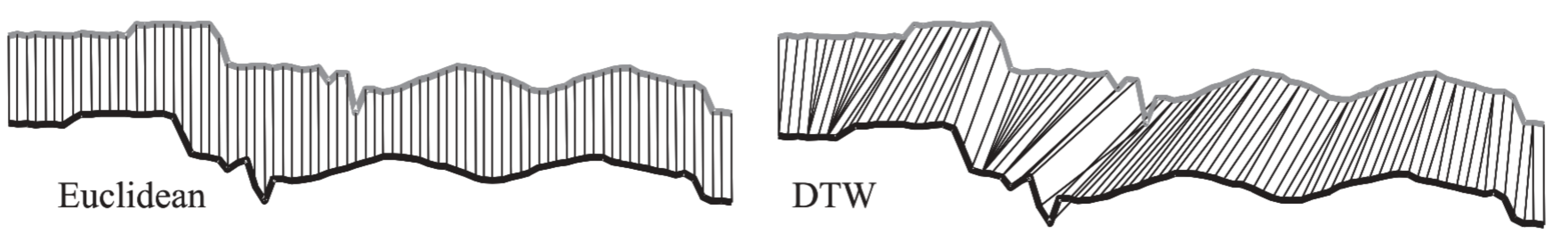
Out[5]:
```
9304
```

Теперь посчитаем матрицу попарных евклидовых расстояний между рядами отного датасета для определения их схожести - Dynamic Time Warping

(cs.ucr.edu/~eamonn/KAIS_2004_warping.pdf)

In [6]:
```python
from IPython.display import Image
Image('dtw_eucl.png')
```

Out[6]:



Есть простой алгоритм, который считает все расстояния между рядами, сложности $O(nm)$ где n и m длины рядов:

In [7]:
```python
def dtw1(s1, s2):
    DTW={}
    for i in range(len(s1)):
        DTW[(i, -1)] = float('inf')
    for i in range(len(s2)):
        DTW[(-1, i)] = float('inf')
    DTW[(-1, -1)] = 0

    for i in range(len(s1)):
        for j in range(len(s2)):
            dist= (s1[i]-s2[j])**2
            DTW[(i, j)] = dist + min(DTW[(i-1, j)],DTW[(i, j-1)], DTW[(i-1, j-1)])

    return sqrt(DTW[len(s1)-1, len(s2)-1])
```

Есть и более облегченная его версия с окном, которая позволяет значительно уменьшить вычислительную сложность:

In [8]:
```python
def dtw2(s1, s2, window):
    n, m = s1.shape[0], s2.shape[0]

    w = np.max(np.array([window, np.absolute(n-m)]))

    dtw_matrix = np.full((n + 1, m + 1), np.inf)
    dtw_matrix[0, 0] = 0.

    for i in range(1, n+1):
        for j in range(np.max(np.array([1, i-w])), np.min(np.array([m, i+w]))+1):
            cost = (s1[i-1] - s2[j-1])**2
            dtw_matrix[i, j] = cost + np.min(np.array([dtw_matrix[i-1, j], dtw_matrix[i, j-1], dtw_matrix[i-1,
    return np.sqrt(dtw_matrix[-1, -1])
```

Применим библиотеку dtaidistance, которая будет использовать код на Си и паралеллить подсчет матрицы:

In [9]:
```python
dtw.try_import_c() # проверим есть ли OMP для Си
```

In [7]:
```python
dist_matrix = dtw.distance_matrix_fast(time_series, window=60, compact=True)
```

In [ ]:
```python
# используем squareform для перехода на обычную матрицу
from scipy.spatial.distance import pdist, squareform
dist = squareform(dist_matrix)
```

In [35]:
```python
#np.save('ts_dist_matrix.npy', dist)
```

In [10]:
```python
ts_dist_matrix = np.load("ts_dist_matrix.npy")
ts_dist_matrix
```

Out[10]:
```
array([[0.        , 2.16297457, 6.21311378, ..., 3.20641225, 2.97714582,
        4.96603827],
       [2.16297457, 0.        , 7.28620543, ..., 6.53808198, 7.49029629,
        2.30455842],
       [6.21311378, 7.28620543, 0.        , ..., 2.307163  , 1.77260901,
        8.48498069],
       ...,
       [3.20641225, 6.53808198, 2.307163  , ..., 0.        , 1.19643093,
        7.33282846],
       [2.97714582, 7.49029629, 1.77260901, ..., 1.19643093, 0.        ,
        7.27461574],
       [4.96603827, 2.30455842, 8.48498069, ..., 7.33282846, 7.27461574,
        0.        ]])
```

Найдем linkage matrix:

In [11]:
```python
from scipy.cluster.hierarchy import ward, dendrogram, ClusterWarning
from warnings import simplefilter
```

In [10]:
```python
simplefilter("ignore", ClusterWarning)
#pdist(ts_dist_matrix)
linkage_matrix = ward(ts_dist_matrix)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\cluster\hierarchy.py:834: ClusterWarning: scipy.cluster: The s
ymmetric non-negative hollow observation matrix looks suspiciously like an uncondensed distance matrix
  return linkage(y, method='ward', metric='euclidean')
```

In [ ]:
```python
# построим дендрограмму:
fig = plt.figure(figsize=(20, 100))
dn = dendrogram(linkage_matrix, orientation='left', color_threshold=4, truncate_mode='level', p=500, labels=tra
plt.show()
```

In [12]:
```python
#np.save('linkage_matrix.npy', linkage_matrix)
```

In [14]:
```python
linkage_matrix = np.load("linkage_matrix.npy")
print(linkage_matrix.shape)
linkage_matrix
```

```
(9303, 4)
```

Out[14]:
```
array([[3.78000000e+02, 5.90200000e+03, 1.48850550e+01, 2.00000000e+00],
       [8.56000000e+02, 1.29800000e+03, 1.83981784e+01, 2.00000000e+00],
       [1.61100000e+03, 7.65800000e+03, 1.87199933e+01, 2.00000000e+00],
       ...,
       [1.86020000e+04, 1.86030000e+04, 8.90605951e+03, 5.71000000e+03],
       [1.86000000e+04, 1.86010000e+04, 1.33028215e+04, 3.59400000e+03],
       [1.86040000e+04, 1.86050000e+04, 2.78567638e+04, 9.30400000e+03]])
```

По linkage matrix сделаем иерархическую кластеризацию:

In [23]:
```python
# Обрежем дерево на значении 500
from scipy.cluster.hierarchy import fcluster

# hand-select an appropriate cut-off on the dendrogram
cluster_labels = fcluster(linkage_matrix, 500, criterion='distance')
unq = np.unique(cluster_labels, return_counts=True)
unq
```

Out[23]:
```
(array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
         14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
         27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
         40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
         53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
         66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
         79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
         92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103, 104,
        105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,
        118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
        131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
        144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
        157, 158, 159, 160, 161, 162, 163, 164, 165, 166], dtype=int32),
 array([108,  23,  78, 111, 119,  25,  62,  61,  59,  46,  76,  49,  58,
         56, 183, 121,  75,  93,   9,  26,  51,  86,  73,  58,  80, 103,
        196,  35,  64, 121,  79, 136,  32,  63,  57,  47,  63,  36,  54,
         78,  21,  42,  24,  34,  27,  53,  35,  71,  47,  58,  33,  38,
         35,  30,  14,  14,  35,  50,  39,  13,  25,  36,  58,  68,  46,
         67,  67,  29,  41, 124,  19,  82,  68,  60,  69,  63,  75, 109,
        121,  59, 105,  27,  26,  35,  52,  49,  40,  50,  78,  59,  53,
         25,  68,  57,  37,  70,  73, 111,  41,  56,  48,  77,  42,  85,
         67,  71,  69,  77,  54,  84,  85,  35,  64,  67,  25,  56,  22,
         57,  14,  29,  31,  23,  28,  22,  19,  25,  17,  44,  61,  48,
         37,  50,  50,  79,  45,  64,  77,  44,  44,  15,  79,  30,  33,
         49,  70,  43,  72,  38,  82,  40,  46,  74,  20,  94,  31,  66,
        100, 111,  14,  19,  34,  14,  10,  13,  31,  79], dtype=int64))
```

In [26]:
```python
cluster_labels.shape
```

Out[26]:
```
(9304,)
```

Создадим список из наблюдений и соответствующих им кластеров:

In [27]:
```python
new_list_arr = np.array(train_list)
cluster_labels = cluster_labels.reshape(len(cluster_labels),1)
```

In [ ]:
```python
new_arr = np.hstack((new_list_arr, cluster_labels))
#np.save('new_arr_after_cluster.npy', new_arr2)
```

In [6]:
```python
new_arr = np.load("new_arr_after_cluster.npy")
new_arr
```

Out[6]:
```
array([['SPI', '2020-09-23', '60'],
       ['GLSI', '2020-12-09', '94'],
       ['APM', '2020-09-29', '104'],
       ...,
       ['WTRH', '2020-03-11', '129'],
       ['ACRE', '2020-04-09', '97'],
       ['MVIS', '2020-10-08', '18']], dtype='<U11')
```

In [29]:
```python
#for ticker in unq[0].tolist():
#    parent_dir = 'C:/Users/Kuanysh/Downloads/pump_and_dump/clusters_1'
#    path = os.path.join(parent_dir, str(ticker))
#    os.makedirs(path)
```

Создадим графики всех этих наблюдений в кластерах:

In [ ]:
```python
for i in tqdm(range(len(new_arr))):
    ticker_name = new_arr[i][0]
    parent_dir = 'C:/Users/Kuanysh/Downloads/pump_and_dump/agg_tickers_1m'

    file = os.path.join(parent_dir, ticker_name + '.csv')
    df = pd.read_csv(file)
    df['date'] = pd.to_datetime(df['time']).dt.date
    df['times'] = pd.to_datetime(df['time']).dt.time
    df['date'] = pd.to_datetime(df['date'])
    df = df[df['date'] == str(new_arr[i][1])]
    df=df.sort_values(by='times', ascending=True)
    df.index = pd.DatetimeIndex(df['time'])
    pictures_dir = 'C:/Users/Kuanysh/Downloads/pump_and_dump/clusters_1' + '/' + new_arr[i][2]
    file2 = os.path.join(pictures_dir, ticker_name + '_'+ str(new_arr[i][1]) +'.png')
    print(new_arr[i])
    try:
        mpf.plot(df.iloc[:,1:6], figsize =(2048/100,1080/100), type='candle', volume=True, \
             warn_too_much_data=1000, title=str(new_arr[i]), \
             scale_padding=0.2, savefig=file2)
        plt.close()
    except IndexError:
           print("error")
           pass
```

Далее рассмотрим все подпапки и отберем нужные кластеры из 166 кластеров вручную, чтобы по ним можно было найти центры кластеров. Останется 2729 наблюдений.