ML5G-PS-009

Synthetic Observability Data Generation using GANs

authors: Beijing Quant Evolution Inc.（北京宽客进化科技有限公司）, China Mobile Research Institute（中国移动研究院）

In [1]:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from collections import namedtuple
import matplotlib.pyplot as plt
from scipy.stats import wasserstein_distance
from fastdtw import fastdtw
from tqdm.auto import tqdm
from typing import *
import pandas as pd
import numpy as np
from PyEMD import EMD
import Sample
import pickle
import torch
import os
import warnings

warnings.filterwarnings('ignore')
```

Data analysis and preprocess

Training dataset

Four datasets are used in our train from the two labs: linux foundation lab and china mobile lab,
mixedbag-2hours-cpuper-node5（node5）
mixedbag-2hours-cpuper-node4（node4）
k8s-worker-2（worker2）
k8s-worker-1（worker1）

Other two datasets with 30min sampling window are excluded from our training set because their sampling time is not synchronized together.

Name	Last Modified
mixedbag-2hours-cpuper...	3 months ago
mixedbag-2hours-cpuper...	3 months ago
k8s-worker-2	4 months ago
k8s-worker-1	4 months ago

The description of the features in the original dataset,

指标类别	指标维度	指标维度值的含义/特征	数据类型
Memory	memory-buffered	内存中buffer的空间大小	整数
	memory-cached	内存中cache的空间大小	
	memory-free	内存剩余空间大小	
	memory-used	内存已用空间大小	
	memory-slab_recl	可回收的内存量	
	memory-slab_unrecl	不可回收的内存量	
CPU	percent-user	用户进程使用cpu的时间	浮点数
	percent-system	内核进程使用cpu的时间	
	percent-nice	用户进程空间内改变过优先级的进程使用的cpu时间	
	percent-idle	空闲的cpu时间	
	percent-wait	等待io完成的cpu时间	
	percent-steal	丢失的cpu时间	
	percent-softirq	系统处理软中断使用的cpu时间	
	percent-interrupt	中断模式的cpu	
interface	if_dropped	网卡接口接收的丢弃的数据包总数，rx（接收） 网卡接口发送的丢弃的数据包总数，tx（发送）	整数
	if_errors	网卡接口接收的错误数据包总数 网卡接口发送的错误数据包总数	
	if_octets	网卡接口接的数据包总数 网卡接口发送的数据包总数	
	if_packets	网卡接口接收的数据包总数 网卡接口发送的数据包总数	
disk	disk_io_time	当前文件系统I/O花费的总秒数 进行I/O所花费的加权秒数	整数
	disk_octets	磁盘读取操作的总数 磁盘写入操作的总数	
	disk_ops	平均每秒随机读取 I/O 操作数, 平均每秒随机写入 I/O 操作数	
	disk_time	磁盘读取操作耗时 磁盘写入操作耗时	
	pending_operations	每秒等待的I/O操作数	
	disk_merged	合并在单个请求中的相邻读请求；合并在单个请求中的相邻写请求	
processes	ps_state-paging	系统中分页操作的进程数	整数
	ps_state-sleeping	系统中挂起的进程数	
	ps_state-zombies	系统中的僵尸进程数	
	ps_state-blocked	系统中被阻塞的任务数	
	ps_state-running	系统中正在运行中的进程数	
	ps_state-stopped	系统中停止的进程数	
	fork_rate	每秒产生的进程数	
df	df_complex-free	当前文件系统的挂载点剩余的空间大小	浮点数
	df_complex-reserved	当前文件系统的挂载点总共可用的空间大小	
	df_complex-used	当前文件系统的挂载点已使用的空间大小	
irp	irp	从系统启动开始到当前时刻，进程的硬中断次数	整数
load	load	CPU过去1分钟、5分钟、15分钟的平均负载	浮点数

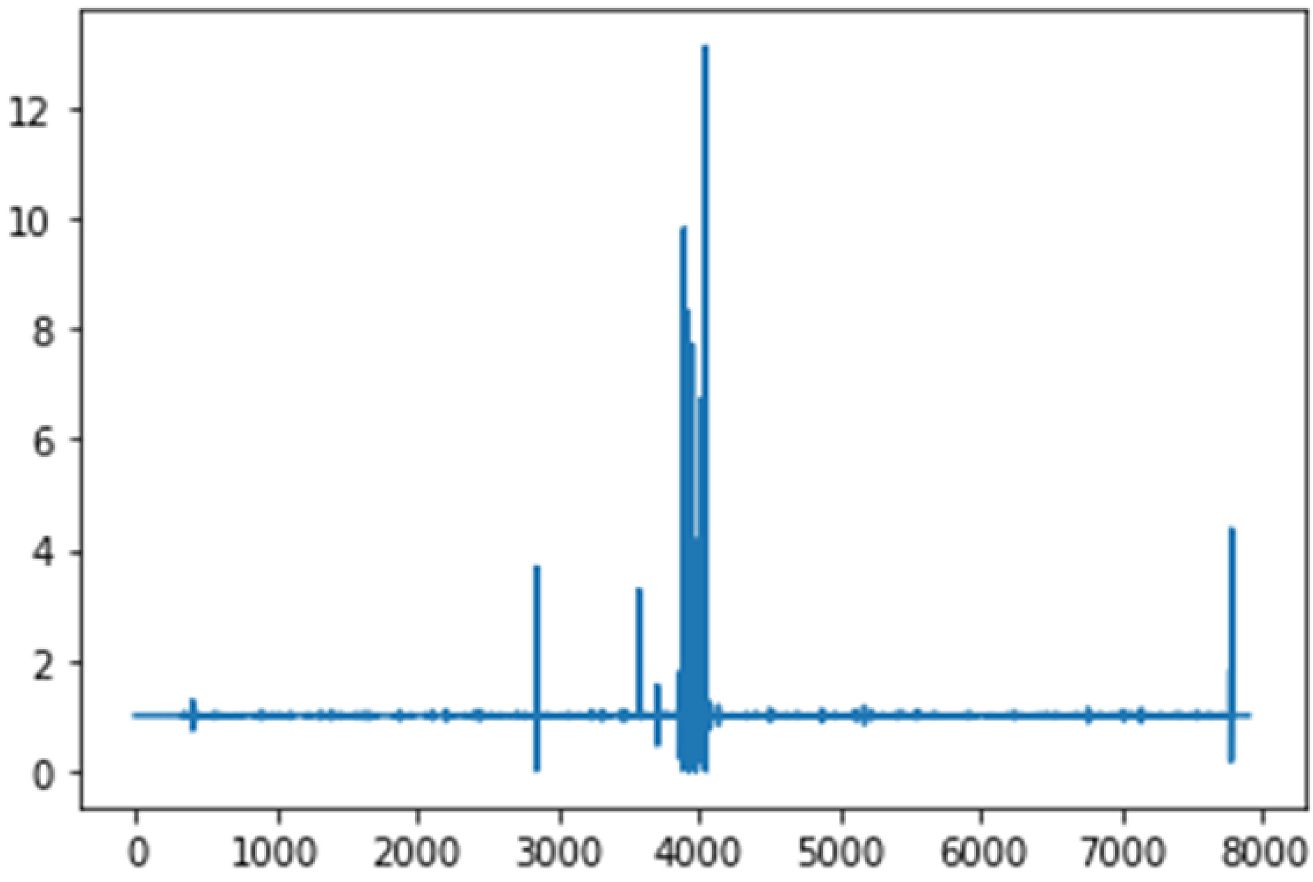
The features are required to generate in our GAN,

指标类别	指标维度	指标维度值的含义/特征	数据类型
Memory	memory-free	内存剩余空间大小	整数
	memory-used	内存已用空间大小	
CPU	percent-user	用户进程使用cpu的时间	浮点数
	percent-system	内核进程使用cpu的时间	
	percent-idle	空闲的cpu时间	
interface	if_dropped	网卡接口接收的丢弃的数据包总数，rx（接收） 网卡接口发送的丢弃的数据包总数，tx（发送）	整数
	if_errors	网卡接口接收的错误数据包总数 网卡接口发送的错误数据包总数	
	if_octets	网卡接口接的数据包总数 网卡接口发送的数据包总数	
	if_packets	网卡接口接收的数据包总数 网卡接口发送的数据包总数	
load	load	CPU过去1分钟、5分钟、15分钟的平均负载	浮点数

Data facts

A. Sampling time is not uniform

The sampling period is centered in 1 sec but widely spreaded as shown below. So we only use the sampling data with 1 sec in our task.

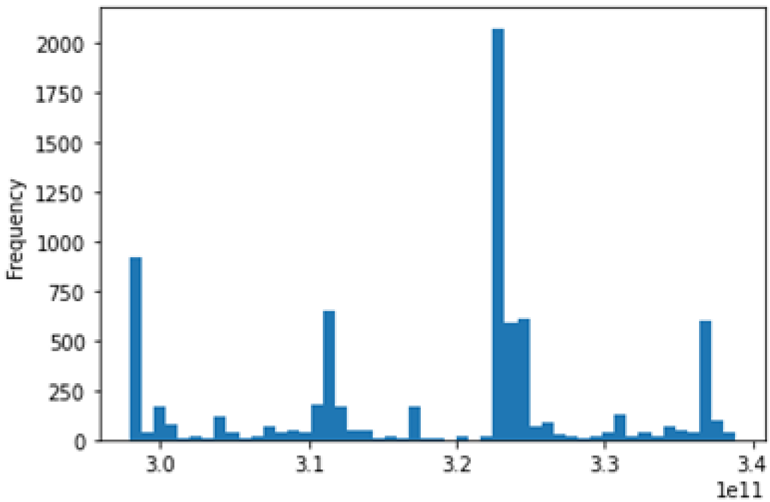


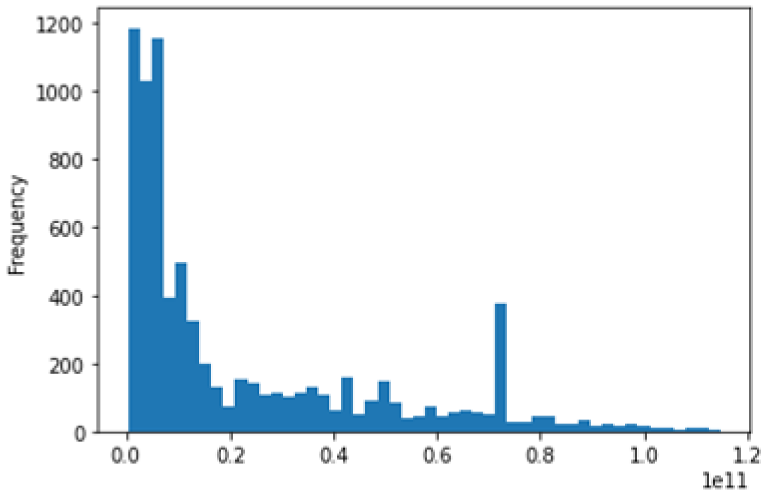
B. The features

In the training dataset, the number of data files is various under the same features because the data is sampled from the different CPUs, disks, interfaces installed on one dividual node. We take a summation of the corresponding hardware data and then learn the CPU and interface total load in one node.

文件数量	node4	node5	worker1	worker2
CPU	96	96	56	56
disk	18	18	9	9
interface	120	120	31	15
load	1	1	1	1
memory	1	1	1	1

The features are sampled from the four different nodes: node4, node5, worker1, worker2. It is found that the features are not identically distributed from the different nodes. The distributions of the feature "memory_free" from worker1 and node4 are shown below. So we train the different GAN models for the node4, node4, worker1, worker2, individually.





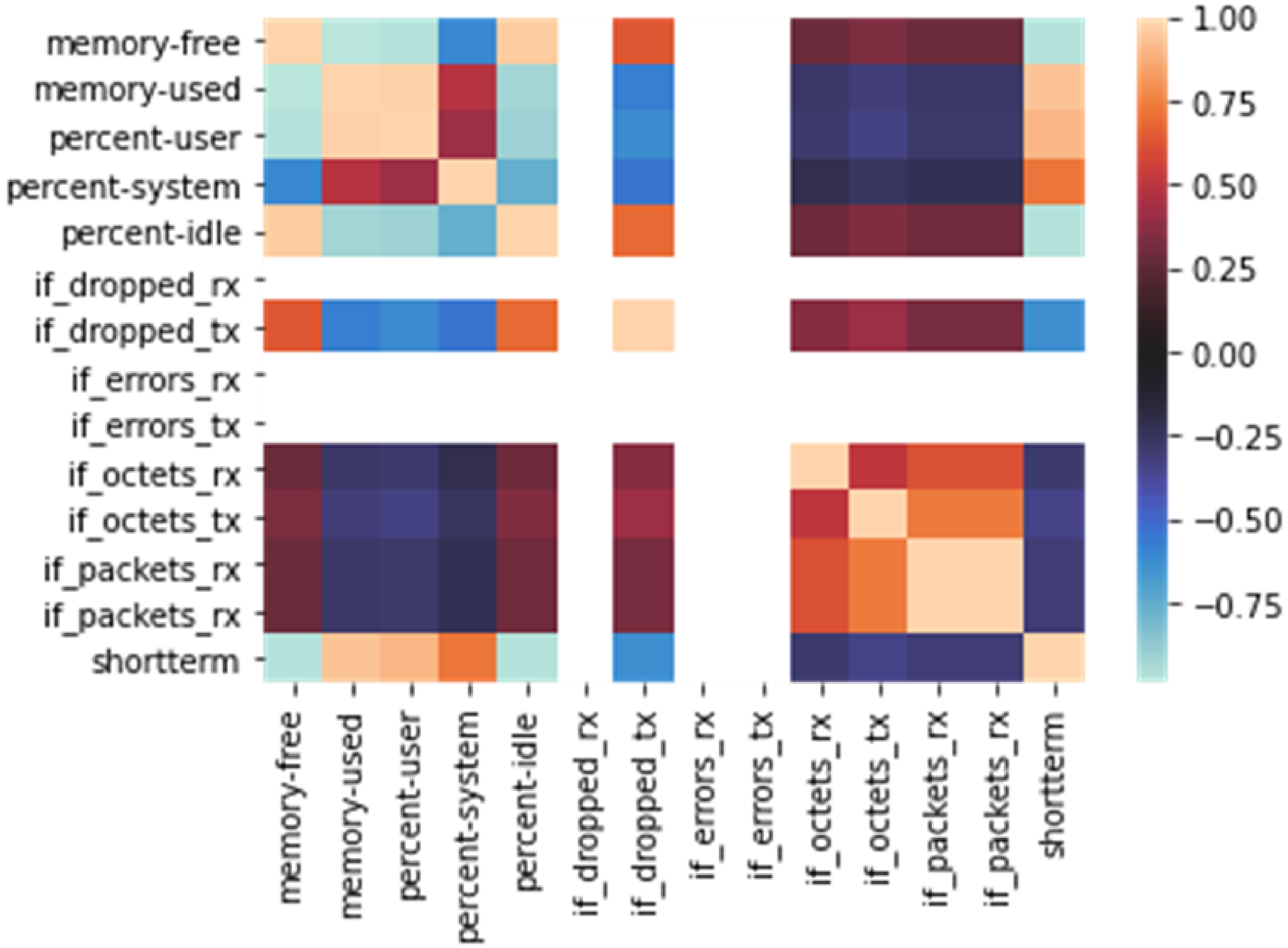
C. The time series of features are not stationary

We find the time series of features are not stationary, so the temporal order is ignored in our generative process. The time series can be rebuilt from the load data after the generation.

D. load is the dominant factor to determine the tendency of CPU, memory features

From the heatmaps below, we find the CPU, memory and load are strongly correlated while the correlations are weak between the interface, disk and load. So we design two generation processes for the group of CPU, memory and load and for the interface, respectively.

The load determines the tendency of CPU, memory. However, our training data is sampled only in two hours which greatly restricts the generated data patterns. The test load as an input condition is usually required in our generation to capture the tendency.



Data cleansing

According to the data facts we have found from the original data, we clean and filter our training dataset,

1. The foreign key epoch is converted to the integers and removes the repeated ones;
2. The sampling data with 1 sec is used in our task;
3. Sum up the corresponding CPU, interface data in one node;
4. The GAN model is individually trained for the node4, node4, worker1, worker2;
5. Remove the data rows with the value 'nan';
6. The GAN model can be conditioned by the tendency of the load.

In [64]:

```
# scientific notation is closed
pd.set_option('display.float_format', lambda x : '%.2f' % x)

def choose_filename(path,startwith):
    """
    locate the files starting with startwith
    :param path: str folder path
    :param startwith: str substring start with
    :return: list filenames
    """
    return [os.path.join(path, i) for i in os.listdir(path) if i.startswith(startwith)]


def pd_concat(lists):
    """
    concatence a list of pandas dataframes into one dataframe
    :param lists: a list of pandas dataframes
    :return: joined dataframe
    """
    df = lists[0]
    for i in lists[1:]:
        df = df.join(i, how='left')
    return df


class DataCleansing():
    """
    The utility collections for the data preprocessing
    """
    def __init__(self, node_file):
        self.node_file = node_file
        self.feature_dict = [
            ['cpu', ['per']],
            ['memory', ['memory']],
            ['interface', ['if']],
            ['load', ['load']],
            # ['process', ['ps', 'fork']],
            # ['disk', ['disk']],
            # ['df', ['df']],
        ]

    def preprocess(self):
        """
        The entrance of data preprocess
        """
        data_list = []
        for i in self.feature_dict:
            data = self.concat_file(file_start=i[0], second_start_list = i[1])
            data_sum = self.sum_data(data)
            data_list.append(data_sum)
        cleaned_data = pd.concat(data_list,axis=1)
        return cleaned_data

    def sum_data(self, data_list, save_more=True):
        """
        sum the hardware data from the various sources in one node
        """
```

```
        data0 = data_list[0]
        for datai in data_list[1:]:
            if save_more:
                datat = data0 + datai
                datat.fillna(data0, inplace=True)
                datat.fillna(datai, inplace=True)
                data0 = datat
            else:
                data0 += datai
        return data0

    def concat_file(self, file_start: str = 'df', second_start_list: List[str]=['df']):
        """
        concatence the hardware data from the various sources in one node
        """
        data_list = []
        for path in tqdm(choose_filename(self.node_file, file_start)):
            file_list = []
            for start in second_start_list:
                file_list += choose_filename(path, start)
            # join the files into one dataframe
            con_data = self._concat_data(file_list, second_start_list)
            # epoch is converted to the integer type
            con_data.index = con_data.index.astype(int)
            con_data = con_data[~con_data.index.duplicated(keep='first')]
            data_list.append(con_data)
        return data_list

    def _concat_data(self, file_list, start_list=['per']):
        if 'if' in start_list:
            lists = [self._interface_rename(file_list_id) for file_list_id in file_list]
        elif 'disk' in start_list:
            lists = [self._disk_rename(file_list_id) for file_list_id in file_list]
        else:
            lists = [self._general_rename(file_list_id) for file_list_id in file_list]
        return pd_concat(lists)

    def _interface_rename(self, file_list_id):
        interface_df = pd.read_csv(file_list_id).set_index('epoch')
        return interface_df.rename(columns={'rx': file_list_id.split(os.sep)[-1][:-11] + '_rx',
'tx': file_list_id.split(os.sep)[-1][:-11] + '_tx'})

    def _general_rename(self, file_list_id):
        df = pd.read_csv(file_list_id).set_index('epoch')
        return df.rename(columns={'value': file_list_id.split(os.sep)[-1][:-11]})

    def _disk_rename(self, file_list_id):
        disk_df = pd.read_csv(file_list_id).set_index('epoch')
        files = file_list_id.split(os.sep)[-1]
        if files.startswith('disk_ops'):
            disk_df.columns = ['ops_read', 'ops_write']
        elif files.startswith('disk_time'):
            disk_df.columns = ['time_read', 'time_write']
        elif files.startswith('disk_octets'):
            disk_df.columns = ['octets_read', 'octets_write']
        return disk_df
```

In [65]:

```
# Read the original files and  preprocess the data
foldername = 'node4'

if foldername == 'node4':
    node_file = 'metadata/mixedbag-2hours-cpuper-node4/pod18-node4'
    pkl_str = 'node4.pkl'
elif foldername == 'node5':
    node_file = 'metadata/mixedbag-2hours-cpuper-node4/pod18-node4'
    pkl_str = 'node5.pkl'
elif foldername == 'worker1':
    node_file = 'metadata/k8s-worker-1'
    pkl_str = 'worker1.pkl'
elif foldername == 'worker2':
    node_file = 'metadata/k8s-worker-2'
    pkl_str = 'worker2.pkl'
else:
    pass


meta_data_ = DataCleansing(node_file).preprocess()
meta_data = meta_data_
```

The data rows with the value 'nan' are dropped,

In [66]:

```
# The data rows with the value 'nan' are dropped
columns = ['memory-free', 'memory-used', 'percent-user', 'percent-system', 'percent-idle', 'shor
tterm', 'midterm', 'longterm']
meta_data = meta_data[columns][1:].dropna()
meta_data.index = range(len(meta_data))
```

Data generation

Generation 1: Load,CPU,memory

Load as the input condition

We recommend to provide a test load time series as the input condition of our generation process.

Otherwise, the training load is provided by default.

In [67]:

```
def get_load(load=None):
    if load is None:
        load = meta_data["shortterm"]
    else:
        pass
    return load
```

In [68]:

```
meta_load = get_load()
```

Conditional generation (shutdown the comment if the condition is provided)

In [69]:

```
# load = meta_data["shortterm"].sort_index(ascending=False)
# load.index = range(len(load))

# meta_load = get_load(load)
```

Call our generation module to generate load,cpu,memory

In [70]:

```
CTGAN = Sample.QEGAN
Table = Sample.Table
CTGANSynthesizer = Sample.CTGANSynthesizer
DataTransformer = Sample.DataTransformer
DataSampler = Sample.DataSampler
Generator = Sample.Generator
Residual = Sample.Residual
```

In [71]:

```
# Setup the parameters for our generation module
SpanInfo = namedtuple('SpanInfo', ['dim', 'activation_fn'])
ColumnTransformInfo = namedtuple('ColumnTransformInfo', [
        'column_name', 'column_type', 'transform', 'output_info', 'output_dimensions'])

# Open our trained generative models
with open(pkl_str, 'rb') as f:
    model = pickle.loads(f.read())
```

In [72]:

```
nentry = 7000  # the number of synthetic entries
syn_data = model.sample(num_rows=nentry)
```

In [73]:

```
# plt.plot(syn_data['shortterm'])
# plt.ylim(0)
```

Rebuliding the time series

In [74]:

```
# meta_load.nsmallest(3)
```

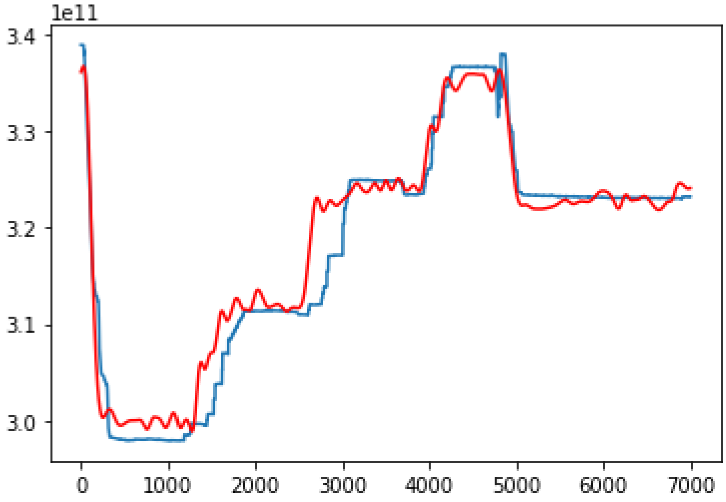
In [75]:

```
# capture the tendency from the conditioned load
meta_load = meta_load[:nentry].sort_values()

syn_data = syn_data.sort_values('shortterm')
syn_data.index = meta_load.index
syn_data = syn_data.sort_index()

syn_data.index = range(len(syn_data))
```

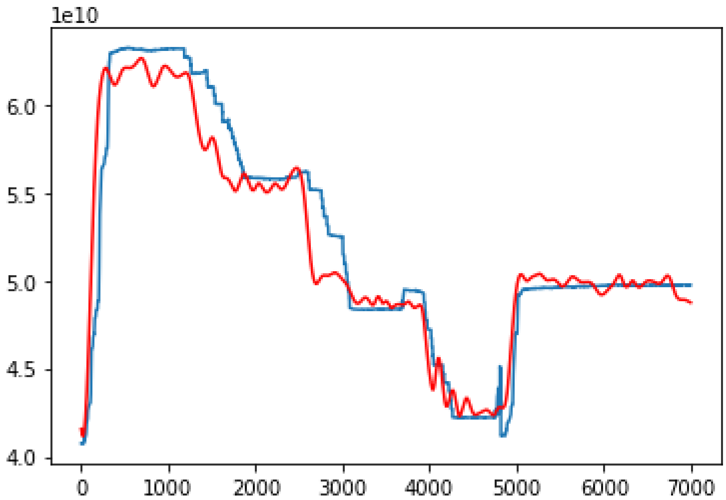
In [76]:

```
def sort_syn(syn_data=None, level=7):
    """
    Clean the noise due to the time series rearrangement
    """
    trend = []
    res = []
    t = np.arange(len(syn_data))
    columns_ = ['memory-free', 'memory-used', 'percent-user', 'percent-system', 'percent-idle']
# 'shortterm'

    for i in columns_:
        EMD_ = EMD()
        IMF = EMD_.emd(np.array(syn_data[i]), t, level)
        trend_ = IMF[level, :]
        res_ = sum(IMF[0: level-1, :])

        trend.append(trend_)
        res.append(res_)

    trend = pd.DataFrame(trend).T
    res = pd.DataFrame(res).T
    trend.columns = columns_
    res.columns = columns_

    return trend, res

trend, res = sort_syn(syn_data=syn_data)

for i in ['memory-free', 'memory-used', 'percent-user', 'percent-system', 'percent-idle'] :
    print(i)
    plt.plot(meta_data[i][:nentry])
    plt.plot(trend[i], 'r')
    plt.show()
    syn_data[i] = trend[i]

plt.plot(syn_data["shortterm"], 'r')
plt.plot(meta_data["shortterm"][:nentry])
plt.show()
```

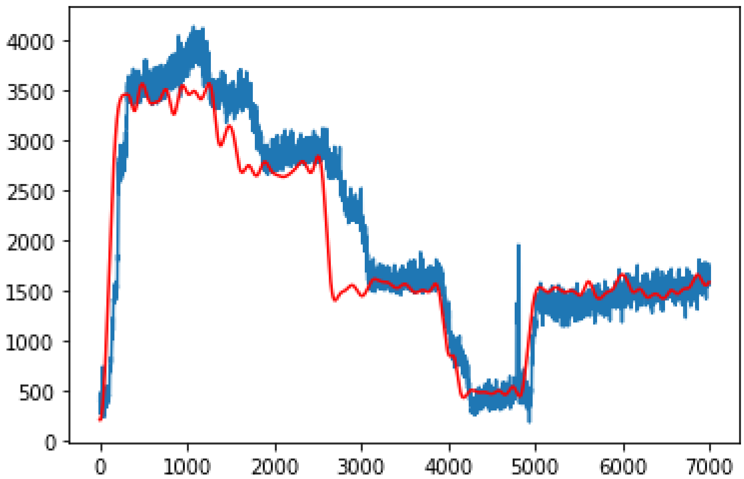
memory-free



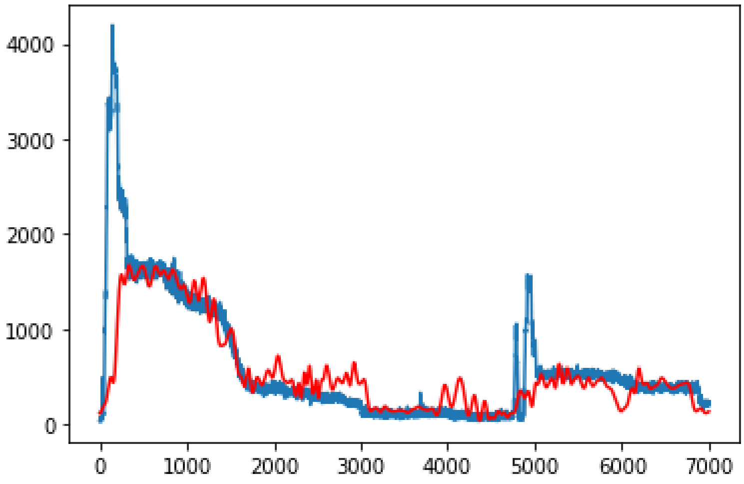
memory-used



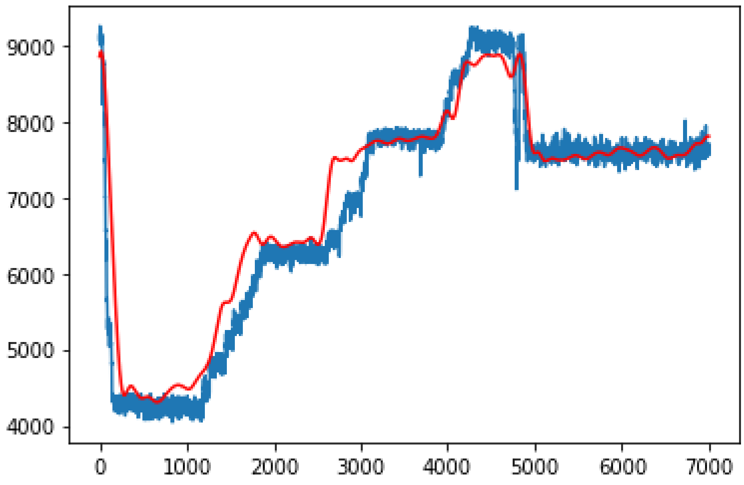
percent-user

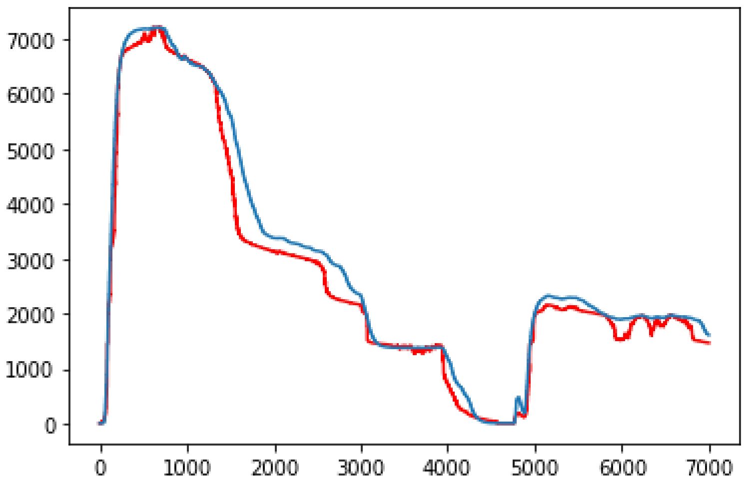


percent-system



percent-idle

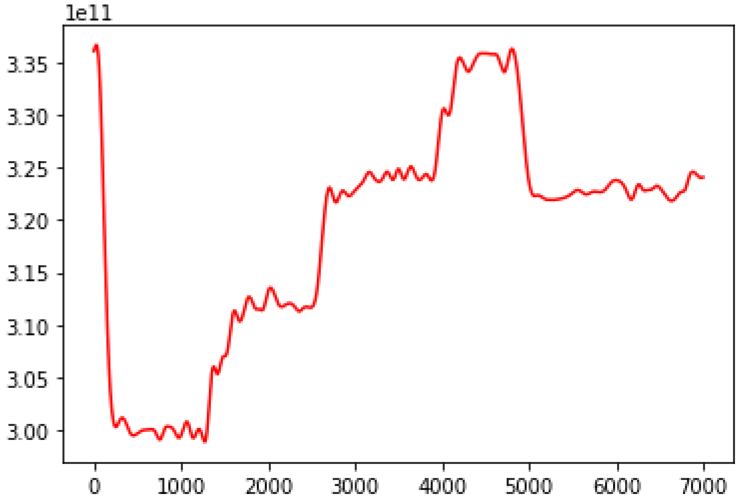




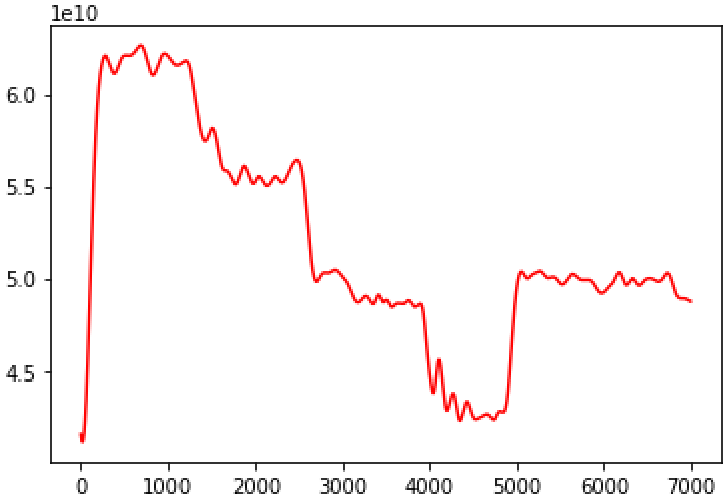
In [77]:

```
for i in ['memory-free', 'memory-used', 'percent-user', 'percent-system', 'percent-idle'] :
    print(i)
    # plt.plot(meta_data[i])
    plt.plot(trend[i], 'r')
    plt.show()
    syn_data[i] = trend[i]

plt.plot(syn_data["shortterm"], 'r')
# plt.plot(meta_data["shortterm"])
plt.show()
```

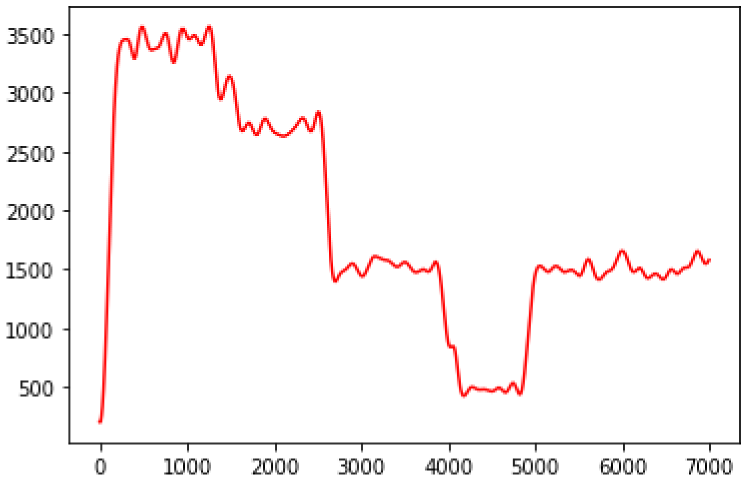
memory-free



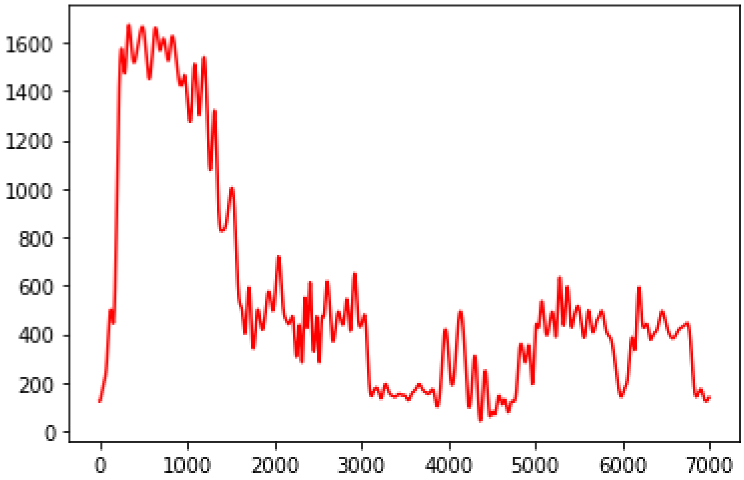
memory-used



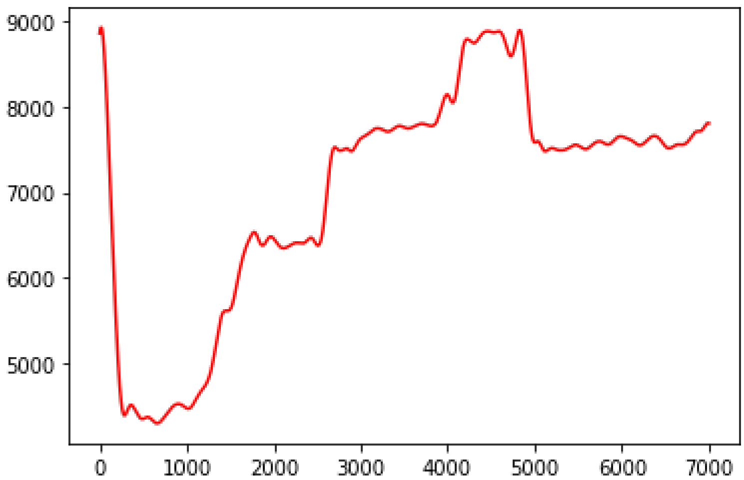
percent-user

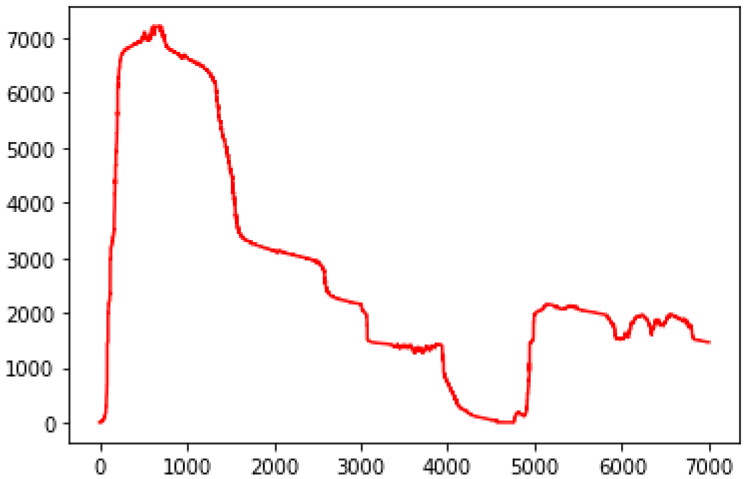


percent-system



percent-idle





In [78]:

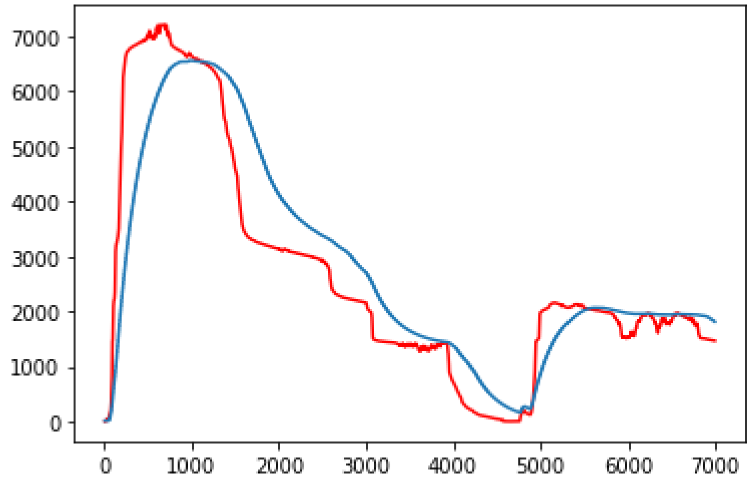
```
def runavg(x, width):
    """
    generate the midterm, longterm of the load
    """
    n = len(x)
    x = np.append(x, np.append(x, x))
    x_smooth = np.convolve(x, np.ones(width)/width, mode='same')
    xs = x_smooth[n:2*n]
    return xs
```

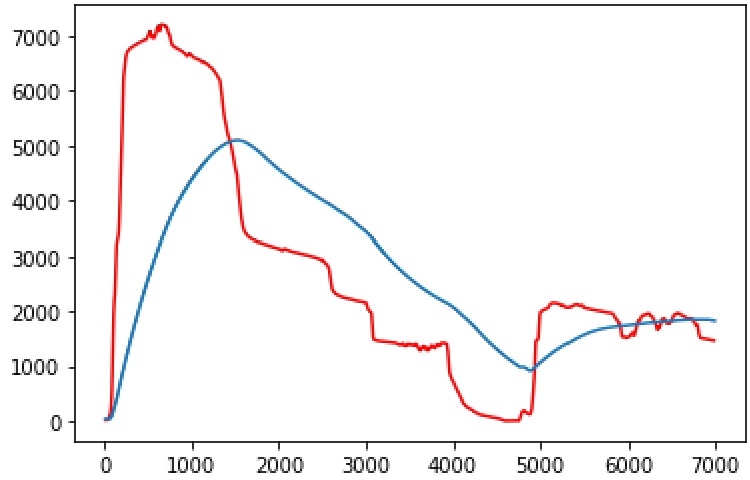
In [79]:

```
syn_data['midterm'] = runavg(syn_data["shortterm"], 5)
syn_data['longterm'] = runavg(syn_data["shortterm"], 15)
```

In [96]:

```
plt.plot(syn_data['midterm'][4:6996], 'r')
plt.plot(meta_data['midterm'][4:6996])
plt.show()

plt.plot(syn_data['longterm'][14:6986], 'r')
plt.plot(meta_data['longterm'][4:6996])
plt.show()
```





Generation 2：interface

In [81]:

```
# The generative utility functions for interface
def clean_data(data):
    """
    clean the noise from the original interface dataset
    """
    mean = data.mean()
    std = data.std()
    interface_df = (data-mean)/std
    return interface_df[(interface_df>-1)&(interface_df<1)]*std + mean

def linear_regress(data):
    linreg = LinearRegression()
    linreg.fit(data.index.values.reshape(-1,1),data.values)
    return linreg.predict(data.index.values.reshape(-1,1))

def generate_if0(data_oct):
    """
    generate the interface data for Packets/Octets
    """
    data = clean_data(data_oct)
    return pd.DataFrame(linear_regress(data),index=data.index,columns=[data.name])

def clean_data1(datas):
    return pd.DataFrame([i.max() for i in datas.rolling(4)])

def stepwise_dropped(data_drop,init_num= 8743,inteval= 1,want_jump_count=2):
    """
    generate stepwise function for the dropped interface
    """
    dropped_shape = data_drop.shape[0]
    data = (np.random.uniform(0, 1, (dropped_shape,))<np.array((want_jump_count/dropped_shape)))
.reshape(-1,1)
    dropped_if = pd.DataFrame(np.zeros_like(data_drop))
    data_mask = dropped_if.mask(data,1)
    return init_num + data_mask.cumsum()*inteval

def generate_if1(data):
    """
    generate the interface data for Dropped/Errors
    """
    init_num = data.iloc[0]
    cdata = clean_data1(data)
    datap = cdata.diff()
    dataq = datap[datap>0].dropna()
    if dataq.empty:
        return data
    else:
        jump_counts = len(dataq)
        jump_interval = dataq.mean()*0.85
        dropped_data = stepwise_dropped(cdata,init_num= init_num,inteval= jump_interval,want_jum
p_count=jump_counts)
        dropped_data.index = data.index
        return dropped_data
```

In [82]:

```
# generate the interface data: syn_interface
syn_interface = pd.DataFrame()

for i in ['if_octets_rx', 'if_octets_tx', 'if_packets_rx', 'if_packets_tx']:
    if0 = generate_if0(meta_data_[i])
    syn_interface = pd.concat([syn_interface, if0], axis=1)

for i in ['if_errors_rx', 'if_errors_tx']:
    syn_interface[i] = 0.0

for i in ['if_dropped_rx', 'if_dropped_tx']:
    if1 = generate_if1(meta_data_[i].fillna(meta_data_[i].mean()))
    syn_interface[i] = if1
```

Assessment

In [83]:

```
epoch = syn_interface.index
syn_interface.index = range(len(syn_interface))
syn_interface = syn_interface.iloc[:nentry, :]
```

In [84]:

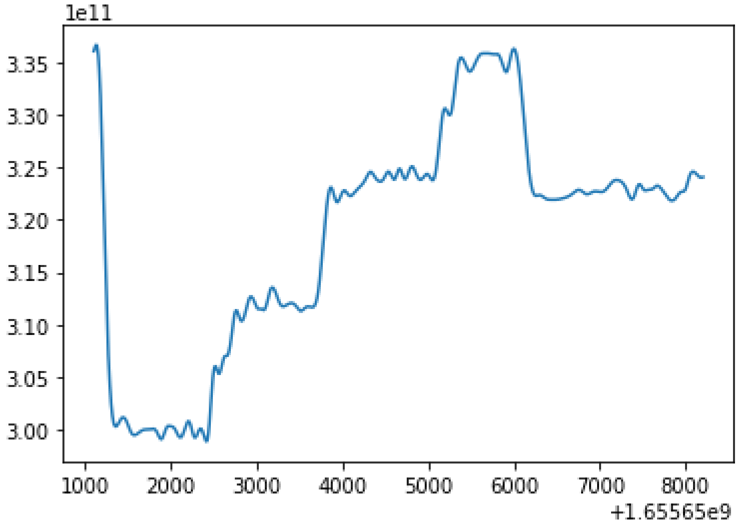
```
syn_data_ = pd.concat([syn_data, syn_interface], axis=1)
syn_data_.index = epoch[:nentry]
```

Generated time series

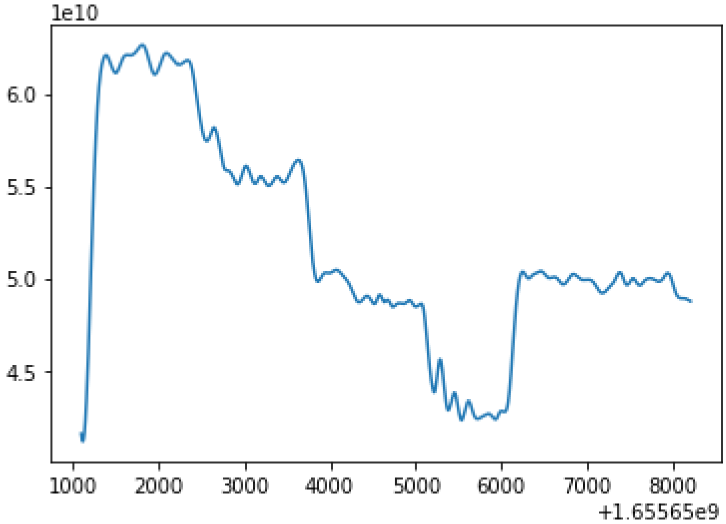
In [85]:

```
for i in syn_data_.columns:
    print(i)
    plt.plot(syn_data_[i])
    plt.show()
```

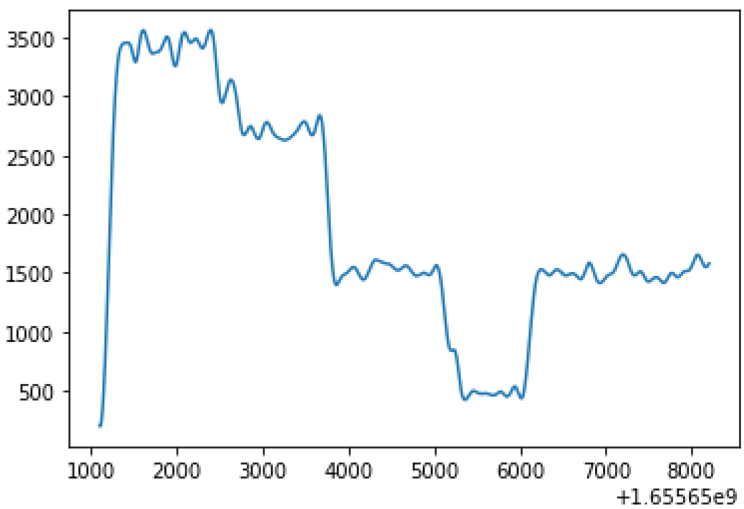
memory-free



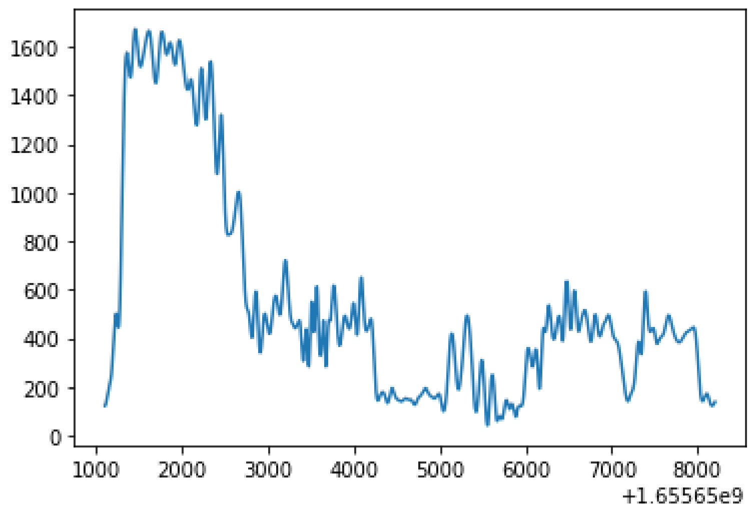
memory-used



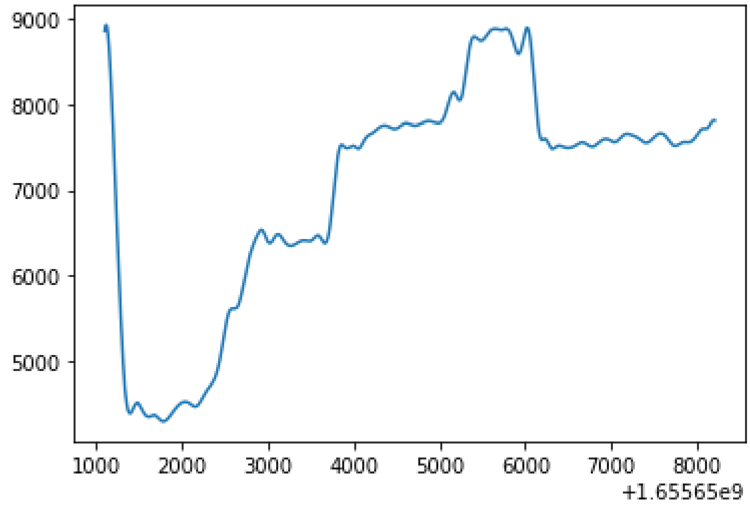
percent-user



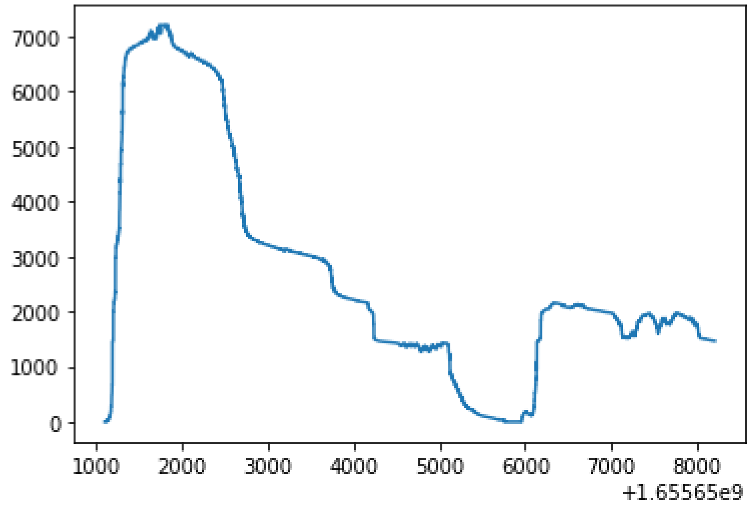
percent-system



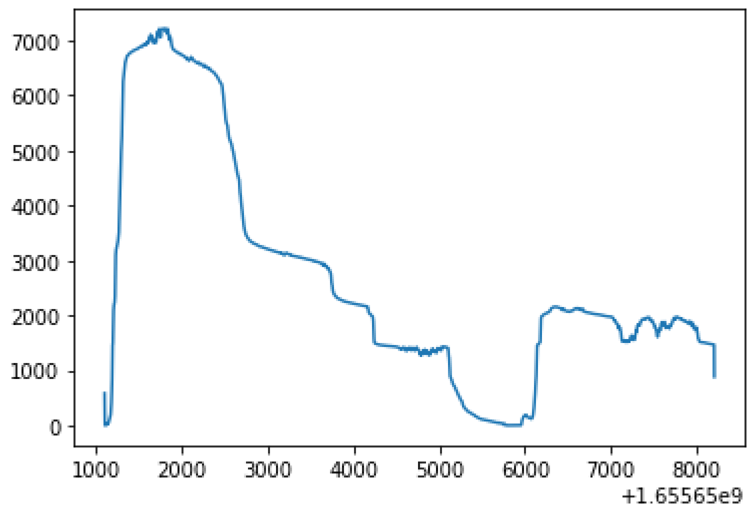
percent-idle



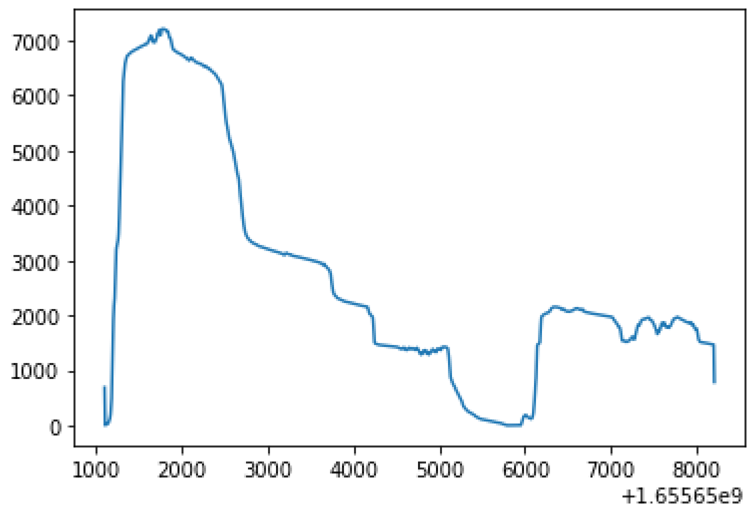
shortterm



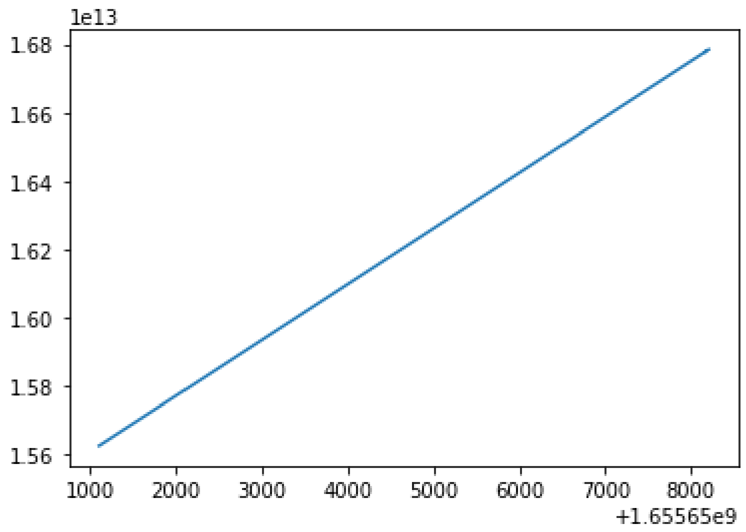
midterm



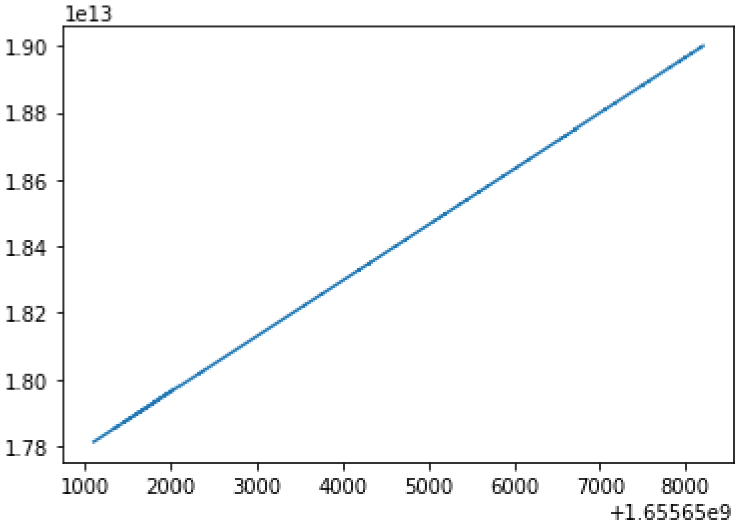
longterm



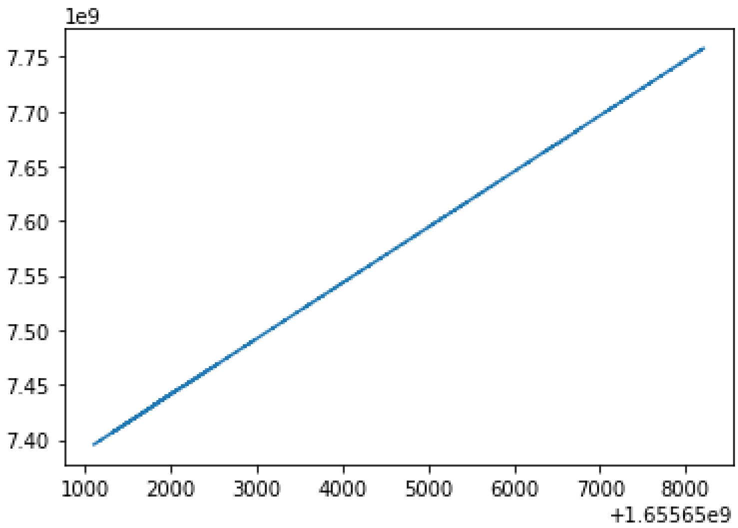
if_octets_rx



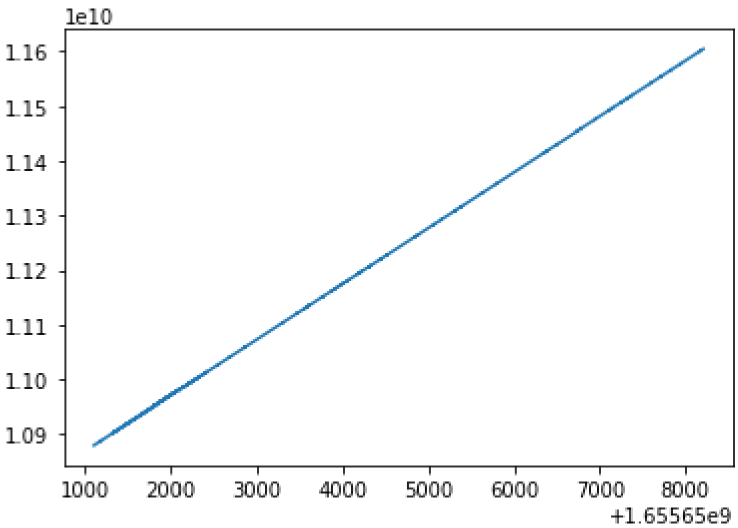
if_octets_tx



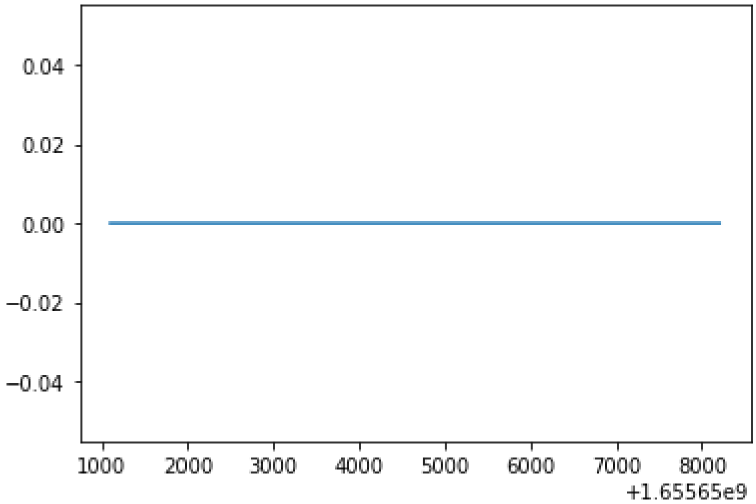
if_packets_rx



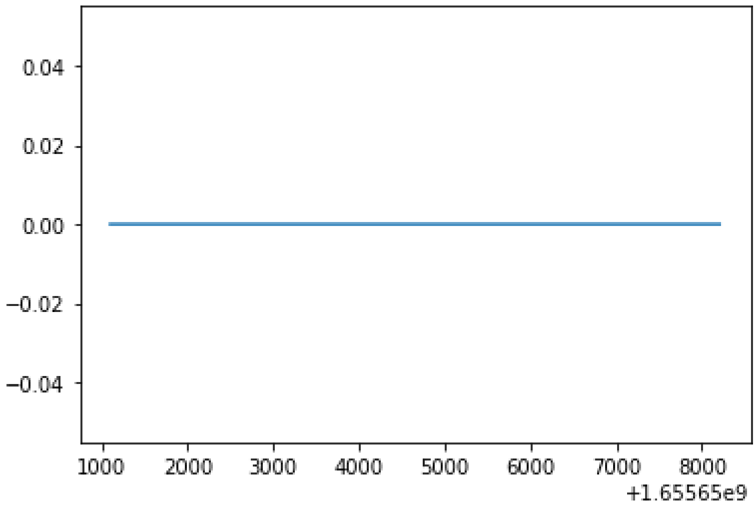
if_packets_tx



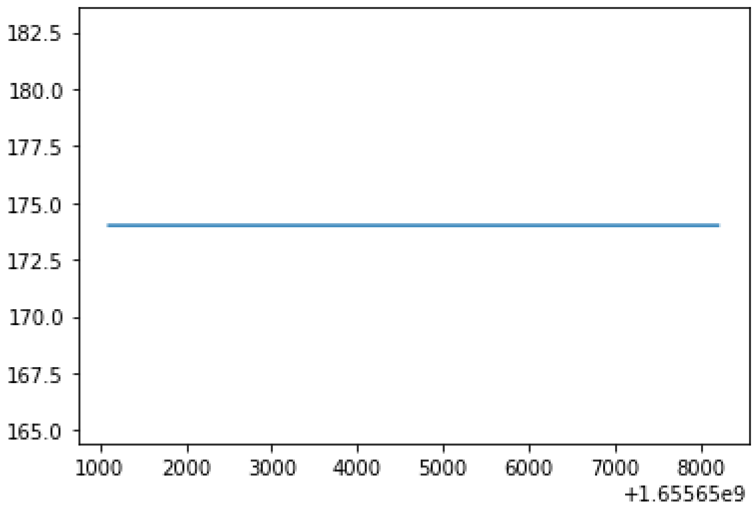
if_errors_rx



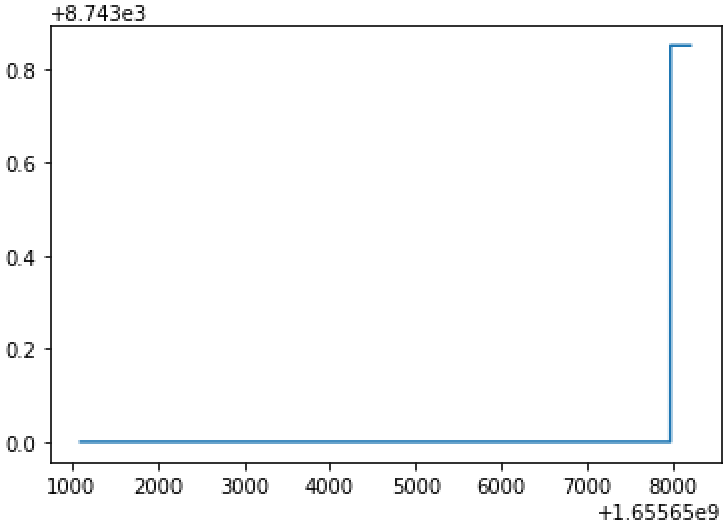
if_errors_tx



if_dropped_rx



if_dropped_tx



Save data

In [49]:

```
# ['memory-free', 'memory-used', 'percent-user', 'percent-system', 'percent-idle', 'shortterm',
 'midterm', 'longterm']

for i in ['memory-free', 'memory-used', 'percent-user', 'percent-system', 'percent-idle', 'if_o
ctets_rx',
       'if_octets_tx', 'if_packets_rx', 'if_packets_tx', 'if_errors_rx',
       'if_errors_tx', 'if_dropped_rx', 'if_dropped_tx']:
    syn_data_[i].to_csv('./syndata/'+foldername+'/'+str(i)+'.csv')

syn_data_[['shortterm', 'midterm', 'longterm']].to_csv('./syndata/'+foldername+'/'+'load.csv')
```

Evaluate synthetic data

In [86]:

```
syn_data
```

Out[86]:

	memory-free	memory-used	percent-user	percent-system	percent-idle	shortterm	midterm	lor
0	336055604353.79	41584173659.54	204.33	123.16	8854.12	6.61	590.36	
1	336076969454.89	41542263180.54	203.04	123.05	8861.50	6.61	298.46	
2	336099092746.45	41502447271.60	202.10	123.15	8868.48	6.61	6.61	
3	336122018446.69	41464751910.75	201.51	123.45	8875.06	6.61	6.61	
4	336145784700.27	41429203075.99	201.28	123.93	8881.22	6.61	6.61	
...	...	...	...	...	...	...	...	
6995	324068324777.12	48755698128.93	1575.00	136.43	7811.22	1466.13	1466.48	1
6996	324071967274.24	48754320124.35	1576.26	137.12	7810.82	1466.44	1466.39	1
6997	324075734043.42	48753072464.12	1577.54	137.83	7810.35	1466.48	1466.20	
6998	324079619045.85	48751957375.31	1578.83	138.55	7809.82	1466.09	1174.30	
6999	324083616242.72	48750975721.41	1580.14	139.30	7809.22	1465.88	882.33	

7000 rows × 8 columns

In [87]:

```
random_ = pd.DataFrame(columns=syn_data.columns)

for i in syn_data.columns:
    random_[i] = np.random.randint(syn_data[i].min(), syn_data[i].max(), 7000)
```

In [88]:

```
random_
```

Out[88]:

	memory-free	memory-used	percent-user	percent-system	percent-idle	shortterm	midterm	longterm
0	311681447961	46331142421	1157	56	8846	532	3918	694(
1	320520985462	49224457202	205	1468	5800	6103	3175	6968
2	307437779402	49593342739	3173	1047	7303	4978	2689	687
3	317777576993	48397232570	3126	819	5816	1696	4728	2558
4	318249950473	46399691025	2833	63	7744	1937	2070	609(
...	...	...	...	...	...	...	...	..
6995	319554210018	55870266082	999	1058	7519	6080	2213	4305
6996	328309956077	60952652094	3418	833	4948	5034	2434	1243
6997	306122120566	52603298260	663	105	6228	1145	1224	7194
6998	316068444552	53484386549	2433	1073	4492	133	5772	410(
6999	333460424291	48468876679	767	557	7026	1096	6444	2154

7000 rows × 8 columns

In [89]:

```
def zscore(Series):
    return (Series-Series.mean())/Series.std()
# meta_data = meta_data[syn_data.columns][1:nentry+1]
syn_data_df = syn_data.apply(zscore)
meta_data_df = meta_data.apply(zscore)
random_df = random_.apply(zscore)
```

Wasserstein distance

In [59]:

```
def w_distance(real_data, syn_data):
    return wasserstein_distance(real_data, syn_data)
```

In [93]:

```
for i in range(len(meta_data.columns)):
    print(meta_data.columns[i])
    df = meta_data_df.iloc[:, i]
    df = df.sample(3000)
    print(w_distance(meta_data_df.iloc[:, i], df))
    print("************")
```

```
memory-free
0.0076891317443744485
************
memory-used
0.01230232317885029
************
percent-user
0.02545174452130164
************
percent-system
0.020868995075387895
************
percent-idle
0.014969145947818514
************
shortterm
0.014040142499199198
************
midterm
0.014684479897679114
************
longterm
0.027106840744678687
************
```

RMSE

In [98]:

```
def RMSE(real_data, syn_data):
    # numpy 格式 均方根误差
    return np.sqrt(np.mean((real_data-syn_data)**2))
```

In [99]:

```
for i in range(len(meta_data.columns)):
    print(meta_data.columns[i])
    print(RMSE(meta_data.iloc[:, i], syn_data.iloc[:, i]))
    print("************")
```

```
memory-free
2743906124.594964
************
memory-used
1966123810.3928025
************
percent-user
402.195276260801
************
percent-system
462.63173121588807
************
percent-idle
417.0348692016774
************
shortterm
346.3028504845145
************
midterm
926.3678267403942
************
longterm
1757.193165520007
************
```

Mutual Information

In [32]:

```
def multal_info(real_data,syn_data):
    # 必须为1D 如Series
    from sklearn.metrics import mutual_info_score
    return mutual_info_score(real_data,syn_data)
```

In [33]:

```
for i in range(len(meta_data.columns)):
    print(meta_data.columns[i])
    print(multal_info(meta_data.iloc[:, i], syn_data.iloc[:, i]))
    print("************")
```

```
memory-free
8.843169199303256
************
memory-used
8.820539991116398
************
percent-user
8.85366542803745
************
percent-system
8.85366542803745
************
percent-idle
8.85366542803745
************
shortterm
7.108542923133963
************
midterm
7.143167996116443
************
longterm
7.172169316269114
************
```

Distribution

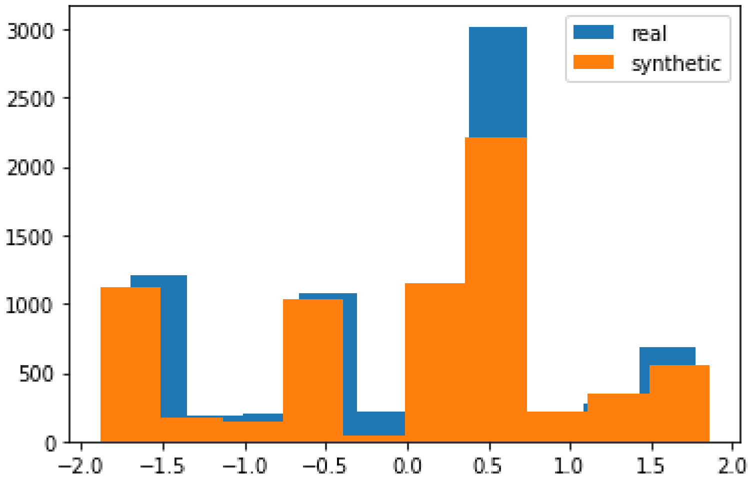
In [34]:

```
def distribution(data, bins=20):
    return np.histogram(data, bins=bins)
```

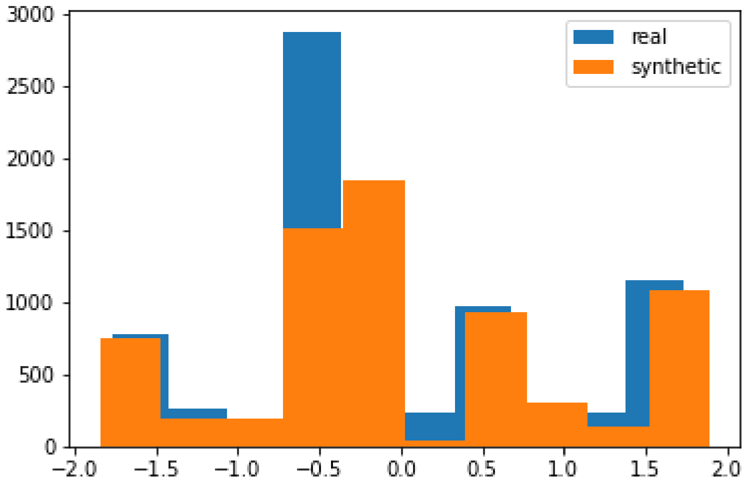
In [35]:

```
for i in range(len(meta_data.columns)):
    print(meta_data.columns[i])
    pic1 = plt.hist(meta_data.iloc[:, i])
    pic2 = plt.hist(syn_data.iloc[:, i])
    plt.legend(['real','synthetic'])
    plt.show()
    print("************")
```

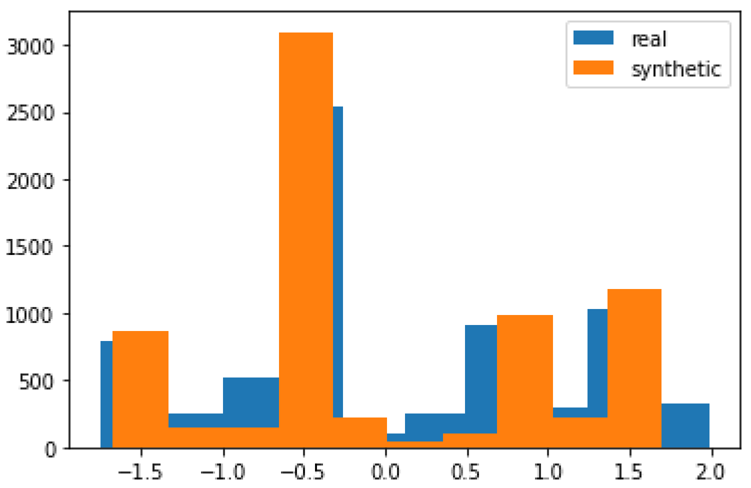
memory-free



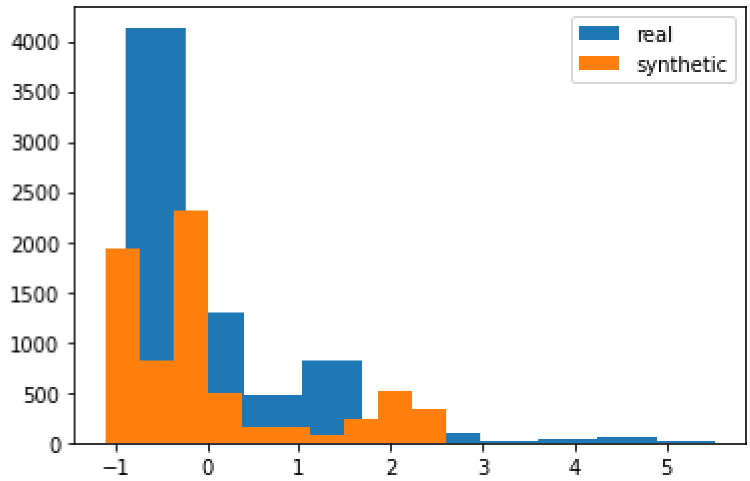
****
memory-used



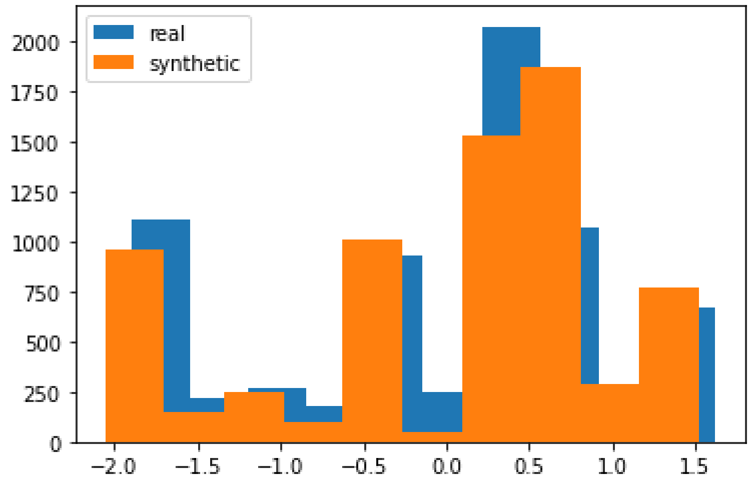
****
percent-user



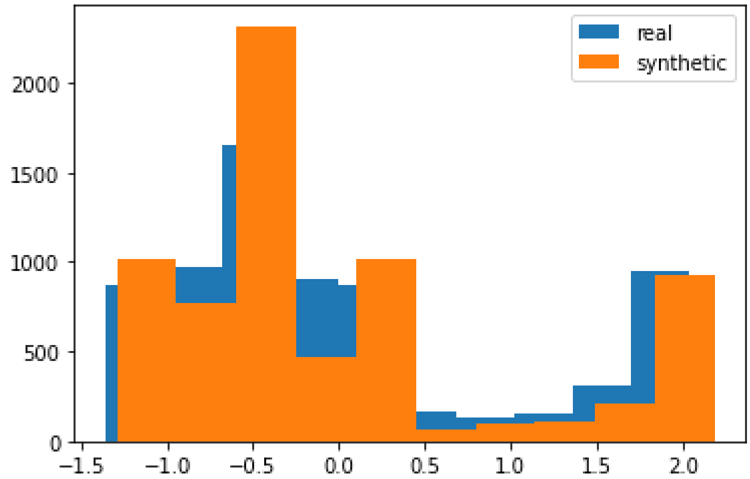
****
percent-system



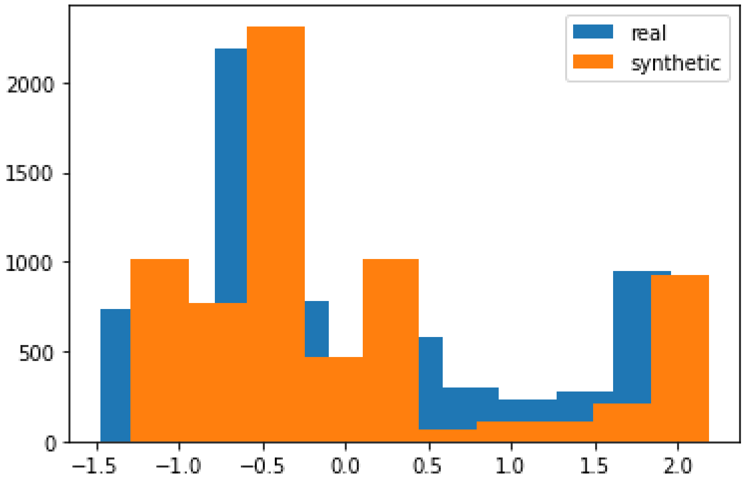
****
percent-idle



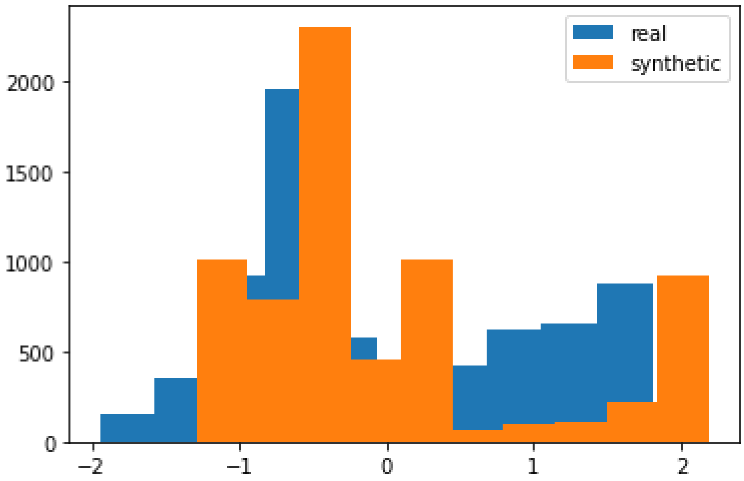
****
shortterm



****
midterm




```
************
longterm
```




```
************
```

Autocorrelation

In [36]:

```
def autocorrelation(data,maxLags=100):
    auto_corr = []

    for i in range(1, maxLags):
        corr = np.corrcoef(
            np.array([np.abs(data[:-i]), np.abs(data[i:])]))[0, 1]
        auto_corr.append(corr)
    return auto_corr
```

In [37]:

```
for i in range(len(meta_data.columns)):
    print(meta_data.columns[i])
    print('real:',np.mean(autocorrelation(meta_data.iloc[:, i])))
    print('synthetic:',np.mean(autocorrelation(syn_data.iloc[:, i])))
    print("************")
```

```
memory-free
real: 0.954167942557257
synthetic: 0.9509577156639127
************
memory-used
real: 0.9607131859216127
synthetic: 0.9427402728616521
************
percent-user
real: 0.9221069077779183
synthetic: 0.9462028751588831
************
percent-system
real: 0.8601232665551508
synthetic: 0.9240819528700717
************
percent-idle
real: 0.9498526122391491
synthetic: 0.9571187083131528
************
shortterm
real: 0.9712574800488744
synthetic: 0.9662112307480802
************
midterm
real: 0.9811904710746432
synthetic: 0.9661889191749201
************
longterm
real: 0.9833545373936153
synthetic: 0.966245087070808
************
```

DTW

In [38]:

```
def DTW(real_data, syn_data):
    x = real_data
    y = syn_data
    distance, path = fastdtw(x,y)
    return distance
```

In [39]:

```
for i in range(len(meta_data.columns)):
    print(meta_data.columns[i])
    print(DTW(meta_data.iloc[:, i], syn_data.iloc[:, i]))
    print("************")
```

```
memory-free
358.3893302750555
************
memory-used
389.13824149822256
************
percent-user
502.6980332474464
************
percent-system
1322.7314495253872
************
percent-idle
489.88110165102
************
shortterm
185.7336018920596
************
midterm
285.0658464705385
************
longterm
755.7896965763512
************
```

MMD

In [40]:

```
def MMD(real_data, syn_data, kernel='multiscale', device='cpu'):
    """
    calculate the distribution distance using MMD
    :param real_data: original data
    :param syn_data: synthetic data
    :param kernel: str kernel method ('multiscale','rbf')
    :param device: str device('cpu','cuda:0')
    :return: MMD
    """
    x = torch.tensor(real_data)
    y = torch.tensor(syn_data)

    xx, yy, zz = torch.mm(x, x.t()), torch.mm(y, y.t()), torch.mm(x, y.t())
    rx = (xx.diag().unsqueeze(0).expand_as(xx))
    ry = (yy.diag().unsqueeze(0).expand_as(yy))

    dxx = rx.t() + rx - 2. * xx  # Used for A in (1)
    dyy = ry.t() + ry - 2. * yy  # Used for B in (1)
    dxy = rx.t() + ry - 2. * zz  # Used for C in (1)

    XX, YY, XY = (torch.zeros(xx.shape).to(device),
                  torch.zeros(xx.shape).to(device),
                  torch.zeros(xx.shape).to(device))

    if kernel == "multiscale":

        bandwidth_range = [0.2, 0.5, 0.9, 1.3]
        for a in bandwidth_range:
            XX += a ** 2 * (a ** 2 + dxx) ** -1
            YY += a ** 2 * (a ** 2 + dyy) ** -1
            XY += a ** 2 * (a ** 2 + dxy) ** -1

    if kernel == "rbf":

        bandwidth_range = [10, 15, 20, 50]
        for a in bandwidth_range:
            XX += torch.exp(-0.5 * dxx / a)
            YY += torch.exp(-0.5 * dyy / a)
            XY += torch.exp(-0.5 * dxy / a)

    return torch.mean(XX + YY - 2. * XY).item()
```

In [41]:

```
print('MMD between the real and synthetic dataset:', MMD(meta_data.values, syn_data.values))
```

MMD between the real and synthetic dataset: 0.30100196599960327

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```