이건,이동현
https://github.com/donghn5/ai_crypto

```
#! /usr/bin/env python
# rdate -s time.nist.gov

import sys
import datetime
import json
import csv
import os
import requests
import time
import pandas as pd
import argparse

import timeit
import urllib3
import numpy as np
import collections

def agg_order_book (bids, asks):

    group_bid = (bids.groupby('price').sum()).reset_index()
    group_bid = group_bid.sort_values('price', ascending=False)

    group_ask = (asks.groupby('price').sum()).reset_index()
    group_ask = group_ask.sort_values('price', ascending=True)
    group_ask['type'] = 1

    return group_bid, group_ask

def bithumb_live_book(data, req_timestamp):
    #timestamp,price,type,quantity
    data = data['data']
    bids = (pd.DataFrame(data['bids'])).apply(pd.to_numeric,errors='ignore')

    bids.sort_values('price', ascending=False, inplace=True)
    bids = bids.reset_index(); del bids['index']
    bids['type'] = 0

    asks = (pd.DataFrame(data['asks'])).apply(pd.to_numeric,errors='ignore')

    asks.sort_values('price', ascending=True, inplace=True)
    asks['type'] = 1

    df = bids.append(asks)
    df['quantity'] = df['quantity'].round(decimals=4)
    df['timestamp'] = req_timestamp

    return df

def agg_diff_trade (diff):

    df = diff
    df['count'] = ''
    if df.empty:
        df = df.append ({'price':0,  'total':0,  'transaction_date':0,  'type':0,  'units_traded':0,  'count':0},
ignore_index=True)
        return df

    group_bid = df[(df.type == 0)].copy().reset_index()
    group_ask = df[(df.type == 1)].copy().reset_index()

    if not group_bid.empty:
        quant = group_bid['units_traded'].sum()
        w_price = int(group_bid['total'].sum() / quant)

        group_bid.loc[0, 'units_traded'] = quant
        group_bid.loc[0, 'price'] = w_price
        group_bid.loc[0, 'type'] = 0
        group_bid.loc[0, 'count'] = len(group_bid.index)
```

```
    if not group_ask.empty:
        quant = group_ask['units_traded'].sum()
        w_price = int(group_ask['total'].sum() / quant)

        group_ask.loc[0, 'units_traded'] = quant
        group_ask.loc[0, 'price'] = w_price
        group_ask.loc[0, 'type'] = 1
        group_ask.loc[0, 'count'] = len(group_ask.index)

    df = (group_bid.head(1)).append(group_ask.head(1))
    df['total'] = df['total'].astype(int)
    df['price'] = df['price'].astype(int)
    df['type'] = df['type'].astype(int)
    df['count'] = df['count'].astype(int)
    del df['index']

    return df


first_seq = True
df1 = ''
bithumb_empty_df = pd.DataFrame(columns=['price', 'total', 'transaction_date', 'type', 'units_traded'])
def bithumb_live_trade(data, req_timestamp):

    global df1
    global first_seq

    df = (pd.DataFrame(data['data'])).apply(pd.to_numeric,errors='ignore')

    df.loc[df['type'] == 'bid', 'type'] = 0
    df.loc[df['type'] == 'ask', 'type'] = 1
    df = (df.sort_values(by=['transaction_date'], ascending=False)).reset_index()

    if first_seq:
        df1 = df
        first_seq = False
        return None, None

    df2 = df

    ###
    #print df1
    #print df2
    #print req_timestamp
    #print df2.isin(df1.head(1))

    _index = 50
    if not df1.empty:
        _h = df1.head(1)
        _l_index = df2[(df2['price']==_h['price'].values[0]) & (df2['units_traded']==_h['units_traded'].values[0]) &
(df2['transaction_date']==_h['transaction_date'].values[0]) & (df2['type']==_h['type'].values[0])].index.tolist()
        if _l_index:
            _index = _l_index[0]
        #print _index
        #print '\n'

    diff = bithumb_empty_df
    if _index > 0:
        diff = df2[0:_index]

    df1 = df

    diff = agg_diff_trade (diff)
    diff['timestamp'] = req_timestamp
    df['timestamp'] = req_timestamp

    return diff[['price', 'total', 'transaction_date', 'type', 'units_traded', 'timestamp', 'count']], df
```

```
def write_csv(fn, df):
    should_write_header = os.path.exists(fn)
    if should_write_header == False:
        df.to_csv(fn, index=False, header=True, mode = 'a')
    else:
        df.to_csv(fn, index=False, header=False, mode = 'a')

def http_get(url):
    return (session.get(url, headers={ 'User-Agent': 'Mozilla/5.0' }, verify=False, timeout=1)).json()

def get_book_trade(ex, url, req_timestamp):

    book = trade = {}
    try:
        book = (session.get(url[0], headers={ 'User-Agent': 'Mozilla/5.0' }, verify=False, timeout=1)).json()
        trade = (session.get(url[1], headers={ 'User-Agent': 'Mozilla/5.0' }, verify=False, timeout=1)).json()
    except:
        return None, None

    return book, trade

def pull_csv_book_trade():

    timestamp = last_update_time = datetime.datetime.now()

    while 1:

        timestamp = datetime.datetime.now()
        if ((timestamp-last_update_time).total_seconds() < 5.0):
            continue
        last_update_time = timestamp

        req_timestamp = timestamp.strftime('%Y-%m-%d %H:%M:%S.%f')
        req_time = req_timestamp.split(' ')[0]

        _dict_book_trade = {}
        #start_time = timeit.default_timer()
        _err = False
        for ex in _list_ex:
            book, trade = get_book_trade (ex, _dict_url[ex], req_timestamp)
            if book is None or trade is None:
                _err = True
                break
            if not book or not trade:
                _err = True
                break
            _dict_book_trade.update ({ex: [book, trade]})

        #delay = timeit.default_timer() - start_time
        #print 'fetch delay: %.2fs' % delay

        if _err == True:
            continue

        for ex in _list_ex:
            book_fn = '%s/%s-only-%s-%s-book.csv'% (csv_dir, req_time, ex, currency)
            trade_fn = '%s/%s-only-%s-%s-trade.csv'% (csv_dir, req_time, ex, currency)

            book_df = bithumb_live_book (book, req_timestamp)
            trade_df, raw_trade_df = bithumb_live_trade (trade, req_timestamp)

            if trade_df is None:
                continue

            write_csv(book_fn, book_df)
            write_csv(trade_fn, trade_df)


from requests.adapters import HTTPAdapter
from requests.packages.urllib3.util.retry import Retry
```

```
def init_session():
    session = requests.Session()
    retry = Retry (connect=1, backoff_factor=0.1)
    adapter = HTTPAdapter (max_retries=retry)
    session.mount('http://', adapter)
    session.mount('https://', adapter)

    return session

def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--currency', help="choose crypto-currency", choices = ('BTC','ETH'), dest='currency', action="store")

    return parser.parse_args()


_list_ex = ['bithumb']
csv_dir ='./raw'
_dict_url = {}
currency = ''

session = init_session()

def write_csv(fn, df):
    # Check if directory exists, if not create it
    os.makedirs(os.path.dirname(fn), exist_ok=True)

    should_write_header = not os.path.exists(fn)
    if should_write_header:
        df.to_csv(fn, index=False, header=True, mode='a')
    else:
        df.to_csv(fn, index=False, header=False, mode='a')

def main():
    global _dict_url
    global currency

    urllib3.disable_warnings()
    args = parse_args()
    currency = args.currency


    if currency == 'BTC':
        _dict_url = {'bithumb': ['https://api.bithumb.com/public/orderbook/BTC_KRW/?count=5',
                                 'https://api.bithumb.com/public/transaction_history/BTC_KRW/?count=50']}
    elif currency == 'ETH':
        _dict_url = {'bithumb': ['https://api.bithumb.com/public/orderbook/ETH_KRW/?count=5',
                                 'https://api.bithumb.com/public/transaction_history/ETH_KRW/?count=50']}
    else:
        print("Invalid currency selected.")
        return

    pull_csv_book_trade()



if __name__ == '__main__':
    main()

pd.set_option('mode.chained_assignment',  None)
```