```
"""
Advances in Financial Machine Learning, Marcos Lopez de Prado
Chapter 2: Financial Data Structures

This module contains the functions to help users create structured financial data from
raw unstructured data,
in the form of time, tick, volume, and dollar bars.

These bars are used throughout the text book (Advances in Financial Machine Learning, By
Marcos Lopez de Prado, 2018, pg 25)
to build the more interesting features for predicting financial time series data.

These financial data structures have better statistical properties when compared to those
based on fixed time interval sampling.
A great paper to read more about this is titled: The Volume Clock: Insights into the high
frequency paradigm, Lopez de Prado, et al

Many of the projects going forward will require Dollar and Volume bars.
"""


# Imports
import pandas as pd
import numpy as np


def _update_counters(cache, flag):
    """
    Updates the counters by resetting them or making use of the cache to update them
based on a previous batch.

    :param cache: Contains information from the previous batch that is relevant in this
batch.
    :param flag: A flag which signals to use the cache.
    :return: Updated counters - cum_ticks, cum_dollar_value, cum_volume, high_price,
low_price
    """
    # Check flag
    if flag and cache:
        # Update variables based on cache
        cum_ticks = int(cache[-1][6])
        cum_dollar_value = np.float(cache[-1][5])
        cum_volume = cache[-1][4]
        low_price = np.float(cache[-1][2])
        high_price = np.float(cache[-1][3])
    else:
        # Reset counters
        cum_ticks, cum_dollar_value, cum_volume, high_price, low_price = 0, 0, 0, -
np.inf, np.inf

    return cum_ticks, cum_dollar_value, cum_volume, high_price, low_price


def _extract_bars(data, metric, threshold=50000, cache=None, flag=False):
    """
    For loop which compiles the various bars: dollar, volume, or tick.

    We did investigate the use of trying to solve this in a vectorised manner but found
that a For loop worked well.

    :param data: Contains 3 columns - date_time, price, and volume.
    :param metric: cum_ticks, cum_dollar_value, cum_volume
    :param threshold: A cumulative value above this threshold triggers a sample to be
taken.
    :param cache: contains information from the previous batch that is relevant in this
batch.
    :param flag: A flag which signals to use the cache.
    :return: The financial data structure with the cache of short term history.
    """
    if cache is None:
```

```
            cache = []

        list_bars = []
        cum_ticks, cum_dollar_value, cum_volume, high_price, low_price =
    _update_counters(cache, flag)

        # Iterate over rows
        for row in data.values:
            # Set variables
            date_time = row[0]
            price = np.float(row[1])
            volume = row[2]

            # Calculations
            cum_ticks += 1
            dollar_value = price * volume
            cum_dollar_value = cum_dollar_value + dollar_value
            cum_volume = cum_volume + volume

            # Check min max
            if price > high_price:
                high_price = price
            if price <= low_price:
                low_price = price

            # Update cache
            cache.append([date_time, price, low_price, high_price, cum_volume,
    cum_dollar_value, cum_ticks])

            # If threshold reached then take a sample
            if eval(metric) >= threshold:   # pylint: disable=eval-used
                # Create bars
                open_price = cache[0][1]
                low_price = min(low_price, open_price)
                close_price = price

                # Update bars & Reset counters
                list_bars.append([date_time, open_price, high_price, low_price, close_price,
                                  cum_volume, cum_dollar_value, cum_ticks])
                cum_ticks, cum_dollar_value, cum_volume, cache, high_price, low_price = 0, 0,
    0, [], -np.inf, np.inf

        return list_bars, cache


    def _assert_dataframe(test_batch):
        """
        Tests that the csv file read has the format: date_time, price, & volume.
        If not then the user needs to create such a file. This format is in place to remove
    any unwanted overhead.

        :param test_batch: DataFrame which will be tested.
        """
        assert test_batch.shape[1] == 3, 'Must have only 3 columns in csv: date_time, price,
    & volume.'
        assert isinstance(test_batch.iloc[0, 1], float), 'price column in csv not float.'
        assert isinstance(test_batch.iloc[0, 2], np.int64), 'volume column in csv not int.'

        try:
            pd.to_datetime(test_batch.iloc[0, 0])
        except ValueError:
            print('csv file, column 0, not a date time format:', test_batch.iloc[0, 0])


    def _batch_run(file_path, metric, threshold=50000, batch_size=20000000):
        """
        Reads a csv file in batches and then constructs the financial data structure in the
    form of a DataFrame.
```

```
    The csv file must have only 3 columns: date_time, price, & volume.

    :param file_path: File path pointing to csv data.
    :param metric: cum_ticks, cum_dollar_value, cum_volume
    :param threshold: A cumulative value above this threshold triggers a sample to be
taken.
    :param batch_size: The number of rows per batch. Less RAM = smaller batch size.
    :return: Financial data structure
    """
    print('Reading data in batches:')

    # Variables
    count = 0
    flag = False  # The first flag is false since the first batch doesn't use the cache
    cache = None
    final_bars = []

    # Read in the first row & assert format
    _assert_dataframe(pd.read_csv(file_path, nrows=1))

    # Read csv in batches
    for batch in pd.read_csv(file_path, chunksize=batch_size):

        print('Batch number:', count)
        list_bars, cache = _extract_bars(data=batch, metric=metric, threshold=threshold,
cache=cache, flag=flag)

        # Append to bars list
        final_bars += list_bars
        count += 1

        # Set flag to True: notify function to use cache
        flag = True

    # Return a DataFrame
    cols = ['date_time', 'open', 'high', 'low', 'close', 'cum_vol', 'cum_dollar',
'cum_ticks']
    bars_df = pd.DataFrame(final_bars, columns=cols)
    print('Returning bars \n')
    return bars_df


def get_dollar_bars(file_path, threshold=70000000, batch_size=20000000):
    """
    Creates the dollar bars: date_time, open, high, low, close, cum_vol, cum_dollar, and
cum_ticks.

    Following the paper "The Volume Clock: Insights into the high frequency paradigm" by
Lopez de Prado, et al,
    it is suggested that using 1/50 of the average daily dollar value, would result in
more desirable statistical
    properties.

    :param file_path: File path pointing to csv data.
    :param threshold: A cumulative value above this threshold triggers a sample to be
taken.
    :param batch_size: The number of rows per batch. Less RAM = smaller batch size.
    :return: Dataframe of dollar bars
    """
    return _batch_run(file_path=file_path, metric='cum_dollar_value',
threshold=threshold, batch_size=batch_size)


def get_volume_bars(file_path, threshold=28224, batch_size=20000000):
    """
    Creates the volume bars: date_time, open, high, low, close, cum_vol, cum_dollar, and
cum_ticks.

    Following the paper "The Volume Clock: Insights into the high frequency paradigm" by
```

```
Lopez de Prado, et al,
    it is suggested that using 1/50 of the average daily volume, would result in more
desirable statistical properties.

    :param file_path: File path pointing to csv data.
    :param threshold: A cumulative value above this threshold triggers a sample to be
taken.
    :param batch_size: The number of rows per batch. Less RAM = smaller batch size.
    :return: Dataframe of volume bars
    """
    return _batch_run(file_path=file_path, metric='cum_volume', threshold=threshold,
batch_size=batch_size)


def get_tick_bars(file_path, threshold=2800, batch_size=20000000):
    """
    Creates the tick bars: date_time, open, high, low, close, cum_vol, cum_dollar, and
cum_ticks.

    :param file_path: File path pointing to csv data.
    :param threshold: A cumulative value above this threshold triggers a sample to be
taken.
    :param batch_size: The number of rows per batch. Less RAM = smaller batch size.
    :return: Dataframe of tick bars
    """
    return _batch_run(file_path=file_path, metric='cum_ticks', threshold=threshold,
batch_size=batch_size)
```