1. Fragment OHLCA data to daily files.
2. Add this:

Class SummaryOHLCA (Summary):

pass

ohlcva\_list = List[SummaryOHLCA]

1. update read\_multi\_timeframe\_ohlca(date\_range\_str) with a date\_range\_str = ‘22-08-09.00-00T23-08-09.00-00’ By looking in ohlcva\_list find gaps which we do not have OHLCV data for given date\_range.
2. generate all the missing time period by calling (date\_range\_str) ->padera.DataFrame[OHLCV] and generate\_multi\_timeframe\_ohlca, store
3. store informatin of OHLCA of each day (GMT) in a separated file name like ohlc.23-08-09.12-00T23-08-10.00-00.zip

keep the summary information as a new row in ohlcv

change following code to every time we run this:

1. Add these:

Class Summary (pandera.DataFrameModel):

end: // real end = 23:59:59 of the day or the latest time we have its data. If we use 1minute updates the last time will be 23:59:00 of the day.

start: // real start = 12:00:00 PM yestarday of the day

rows = number of rows

hash= hash of original.csv file

zip\_hash= hash of zip file

Class SummaryOHLCV (Summary):

low,

high,

ohlcv\_list = List[SummaryOHLCV]

1. Load ohlcv\_list from os.path.join(file\_path, f'ohlca\_summary.zip') only if not loaded before (if is none).

# Fragment OHLC data to daily files:

Rename the existing read\_multi\_timeframe\_ohlc to old\_ read\_multi\_timeframe\_ohlc

update read\_multi\_timeframe\_ohlc(date\_range\_str, presice\_start\_date = false, precise\_end\_date = false) with a date\_range\_str like ‘22-08-09.00-01T23-08-09.00-00’

1. start, end = date\_range(date\_range\_str: str)
2. split date range from start to end day by day and for eachd day:
   1. create read\_daily\_multitimeframe\_ohlcv(day: datetime, timezone = GMT)-> padera.DataFrame [MultiTimeframeOHLCV]
   2. assign day\_daterange as YY-MM-DD.00-00TYY-MM-DD.23-59 for day.
   3. Read old\_ read\_multi\_timeframe\_ohlc(date\_range\_str)
3. Concatenate daily multitimeframe\_ohlcv and return

Date\_range(date\_range\_str: str) -> Tuple[datetime, datetime]:  
 start\_date\_string, end\_date\_string = date\_range\_str.split('T')  
 start\_date = datetime.strptime(start\_date\_string, '%y-%m-%d.%H-%M')  
 end\_date = datetime.strptime(end\_date\_string, '%y-%m-%d.%H-%M')  
 return start\_date, end\_date

def date\_range\_to\_string(end\_date: datetime = None, days: float = 60) -> str:  
 if end\_date is None:  
 end\_date = today\_morning()  
 start\_date = end\_date - timedelta(days=days)  
 return f'{start\_date.strftime("%y-%m-%d.%H-%M")}T' \  
 f'{end\_date.strftime("%y-%m-%d.%H-%M")}'

class MultiTimeframe(pandera.DataFrameModel):  
 timeframe: pt.Index[str]  
  
  
class OHLCV(pandera.DataFrameModel):  
 date: pt.Index[Timestamp]  
 open: pt.Series[float]  
 close: pt.Series[float]  
 high: pt.Series[float]  
 low: pt.Series[float]  
 volume: pt.Series[float]  
  
  
class MultiTimeframeOHLCV(OHLCV, MultiTimeframe):  
 pass  
  
  
class OHLCA(OHLCV):  
 ATR: pt.Series[float] = pandera.Field(nullable=True)  
  
  
class MultiTimeframeOHLCA(OHLCA, MultiTimeframe):  
 pass

def insert\_atr(single\_timeframe\_ohlc: pd.DataFrame) -> pd.DataFrame:  
 \_ATR = ta.ATR(high=single\_timeframe\_ohlc['high'].values, low=single\_timeframe\_ohlc['low'].values,  
 close=single\_timeframe\_ohlc['close'].values)  
 single\_timeframe\_ohlc['ATR'] = \_ATR  
 return single\_timeframe\_ohlc  
  
  
@measure\_time  
def generate\_ohlca(date\_range\_str: str, file\_path: str = config.path\_of\_data) -> None:  
 # if not input\_file\_path.startswith('ohlc') or input\_file\_path.startswith('ohlca'):  
 # raise Exception('input\_file expected to start with "ohlc" and does not start with "ohlca"!')  
 ohlc = read\_ohlc(date\_range\_str)  
 ohlca = insert\_atr(ohlc)  
 # plot\_ohlca(ohlca)  
 ohlca.to\_csv(os.path.join(file\_path, f'ohlca.{date\_range\_str}.zip'), compression='zip')  
  
  
@measure\_time  
def generate\_multi\_timeframe\_ohlca(date\_range\_str: str = None, file\_path: str = config.path\_of\_data) -> None:  
 if date\_range\_str is None:  
 date\_range\_str = config.under\_process\_date\_range  
 multi\_timeframe\_ohlc = read\_multi\_timeframe\_ohlc(date\_range\_str)  
 multi\_timeframe\_ohlca = pd.DataFrame()  
 for \_, timeframe in enumerate(config.timeframes):  
 \_single\_timeframe\_ohlca = insert\_atr(single\_timeframe(multi\_timeframe\_ohlc, timeframe))  
 \_single\_timeframe\_ohlca['timeframe'] = timeframe  
 \_single\_timeframe\_ohlca.set\_index('timeframe', append=True, inplace=True)  
 \_single\_timeframe\_ohlca = \_single\_timeframe\_ohlca.swaplevel()  
 multi\_timeframe\_ohlca = pd.concat([\_single\_timeframe\_ohlca, multi\_timeframe\_ohlca])  
 multi\_timeframe\_ohlc.sort\_index(level='date', inplace=True)  
 # plot\_multi\_timeframe\_ohlca(multi\_timeframe\_ohlca)  
 multi\_timeframe\_ohlca.to\_csv(os.path.join(file\_path, f'multi\_timeframe\_ohlca.{date\_range\_str}.zip'),  
 compression='zip')  
  
  
@measure\_time  
def generate\_multi\_timeframe\_ohlc(date\_range\_str: str, file\_path: str = config.path\_of\_data):  
 ohlc = read\_ohlc(date\_range\_str)  
 # ohlc['timeframe '] = config.timeframes[0]  
 multi\_timeframe\_ohlc = ohlc.copy()  
 multi\_timeframe\_ohlc.insert(0, 'timeframe', config.timeframes[0])  
 multi\_timeframe\_ohlc.set\_index('timeframe', append=True, inplace=True)  
 multi\_timeframe\_ohlc = multi\_timeframe\_ohlc.swaplevel()  
 for \_, timeframe in enumerate(config.timeframes[1:]):  
 if timeframe == '1W':  
 frequency = 'W-MON'  
 elif timeframe == 'M':  
 frequency = 'MS'  
 else:  
 frequency = timeframe  
 \_timeframe\_ohlc = ohlc.groupby(pd.Grouper(freq=frequency)) \  
 .agg({'open': 'first',  
 'close': 'last',  
 'low': 'min',  
 'high': 'max',  
 'volume': 'sum', })  
 \_timeframe\_ohlc.insert(0, 'timeframe', timeframe)  
 \_timeframe\_ohlc.set\_index('timeframe', append=True, inplace=True)  
 \_timeframe\_ohlc = \_timeframe\_ohlc.swaplevel()  
 multi\_timeframe\_ohlc = pd.concat([multi\_timeframe\_ohlc, \_timeframe\_ohlc])  
 multi\_timeframe\_ohlc.sort\_index(inplace=True)  
 # plot\_multi\_timeframe\_ohlc(multi\_timeframe\_ohlc, date\_range\_str)  
 multi\_timeframe\_ohlc.to\_csv(os.path.join(file\_path, f'multi\_timeframe\_ohlc.{date\_range\_str}.zip'),  
 compression='zip')  
  
  
def read\_multi\_timeframe\_ohlc(date\_range\_str: str = None) \  
 -> pt.DataFrame[MultiTimeframeOHLCV]:  
 result = read\_file(date\_range\_str, 'multi\_timeframe\_ohlc', generate\_multi\_timeframe\_ohlc,  
 MultiTimeframeOHLCV)  
 for timeframe in config.timeframes:  
 GLOBAL\_CACHE[f'valid\_times\_{timeframe}'] = \  
 single\_timeframe(result, timeframe).index.get\_level\_values('date').tolist()  
 return result  
  
  
def read\_multi\_timeframe\_ohlca(date\_range\_str: str = None) \  
 -> pt.DataFrame[MultiTimeframeOHLCA]:  
 result = read\_file(date\_range\_str, 'multi\_timeframe\_ohlca', generate\_multi\_timeframe\_ohlca,  
 MultiTimeframeOHLCA)  
 for timeframe in config.timeframes:  
 if f'valid\_times\_{timeframe}' not in GLOBAL\_CACHE.keys():  
 GLOBAL\_CACHE[f'valid\_times\_{timeframe}'] = \  
 single\_timeframe(result, timeframe).index.get\_level\_values('date').tolist()  
 return result  
  
  
def read\_ohlca(date\_range\_str: str = None) -> pt.DataFrame[OHLCA]:  
 result = read\_file(date\_range\_str, 'ohlca', generate\_ohlca, OHLCA)  
 return result  
  
  
def read\_ohlc(date\_range\_str: str = None) -> pt.DataFrame[OHLCV]:  
 if date\_range\_str is None:  
 date\_range\_str = config.under\_process\_date\_range  
 result = read\_file(date\_range\_str, 'ohlc', generate\_ohlc, OHLCV)  
 cast\_and\_validate(result, OHLCV)  
 return result  
  
  
@measure\_time  
def generate\_ohlc(date\_range\_str: str = None, file\_path: str = config.path\_of\_data):  
 if date\_range\_str is None:  
 date\_range\_str = config.under\_process\_date\_range  
 raw\_ohlcv = fetch\_ohlcv\_by\_range(date\_range\_str)  
 df = pd.DataFrame(raw\_ohlcv, columns=['timestamp', 'open', 'high', 'low', 'close', 'volume'])  
 df['date'] = pd.to\_datetime(df['timestamp'], unit='ms')  
 df.set\_index('date', inplace=True)  
 df.drop(columns=['timestamp'], inplace=True)  
 cast\_and\_validate(df, OHLCV)  
 df.to\_csv(os.path.join(file\_path, f'ohlc.{date\_range\_str}.zip'),  
 compression='zip')  
 MT.extract\_to\_data\_path(os.path.join(file\_path, f'ohlc.{date\_range\_str}.zip'))  
 MT. load\_rates()