```python
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 23 16:32:22 2019

@author: Alan.Toppen
"""

import pandas as pd
import numpy as np
import sqlalchemy as sq
import io
import boto3
from multiprocessing import get_context, Pool
import itertools
import re
import os
import sys
from datetime import datetime, timedelta
import time
import random

s3 = boto3.client('s3')
events_bucket = 'gdot-spm'
config_bucket = events_bucket


def read_atspm_query(query):
    engine = sq.create_engine('mssql+pyodbc://atspm',
                pool_size=20)

    with engine.connect() as con:
        df = pd.read_sql_query(query, con=con)
    return df


def get_eventlog_data_db(signalid, date_str):

    start_date = date_str  # date_.strftime('%Y-%m-%d %H:%M:%S.%f')[:-5]
    end_date = (pd.Timestamp(date_str) + pd.DateOffset(days=1) - pd.DateOffset(seconds=0.1))\
                .strftime('%Y-%m-%d %H:%M:%S.%f')[:-5]

    df = read_atspm_query("""
            SELECT * FROM Controller_Event_Log
            WHERE SignalID = '{}'
            AND Timestamp BETWEEN '{}' AND '{}'
            ORDER BY SignalID, Timestamp, EventCode, EventParam
            """.format(signalid.zfill(5),
                       start_date,
                       end_date))
    return df


def get_eventlog_data(bucket, signalid, dates):
    for date_ in dates:
        date_str = date_.strftime('%F')
        df = pd.read_parquet('s3://{b}/atspm/date={d}/atspm_{s}_{d}.parquet'.format(
                b=bucket, d=date_str, s=signalid))
        df.Timestamp = df.Timestamp.dt.tz_localize(None)
        df.SignalID = df.SignalID.astype('str')
        df['date'] = date_
        yield df
```

```python
def get_det_config(bucket, date_, leading_zeros=False):
    date_str = date_.strftime('%F')
    objs = s3.list_objects(Bucket=bucket, Prefix=f'config/atspm_det_config_good/date={date_str}/')
    keys = [obj['Key'] for obj in objs['Contents']]

    def f(bucket, key):
        with io.BytesIO() as data:
            s3.download_fileobj(
                    Bucket=bucket,
                    Key=key,
                    Fileobj=data)

            dc = pd.read_feather(data)\
                .assign(SignalID = lambda x: x.SignalID.astype('str'))\
                .assign(Detector = lambda x: x.Detector.astype('int64'))\
                .reset_index(drop=True)

            dc['date'] = date_

            if leading_zeros:
                dc['SignalID'] = dc['SignalID'].str.zfill(5)

        return dc

    return pd.concat(map(lambda k: f(bucket, k), keys))
            #.rename(columns={'CallPhase': 'Call Phase'})\


def get_det_configs(bucket, dates, leading_zeros=False):
    return pd.concat([get_det_config(bucket, date_, leading_zeros) for date_ in dates])


def get_det_config_local(filename):

    dc = pd.read_feather(filename)\
        .assign(SignalID = lambda x: x.SignalID.astype('str'))\
        .assign(Detector = lambda x: x.Detector.astype('int64'))\
        .reset_index(drop=True)
        #.rename(columns={'CallPhase': 'Call Phase'})\
    return dc


def get_det_config_future(bucket, date_str):
    key = 's3://{b}/atspm_det_config_good/date={d}/ATSPM_Det_Config_Good_Ozark.parquet'.format(
            b=bucket, d=date_str)
    print(key)
    dc = pd.read_parquet(key).reset_index(drop=True)
    return dc


# Works. Doesn't copy down. Use this for new grouping variable and copy_down to apply value across and down
def create_new_grouping_field(df, eventcodes, grouping_field, new_grouping_field, transform_func = lambda x: x):

    if type(eventcodes) is list:
        mask = df.EventCode.isin(eventcodes)
    else:
        eventcode = eventcodes
        mask = df.EventCode==eventcode

    df.loc[mask, new_grouping_field] = df.loc[mask, grouping_field].apply(transform_func)

    return df
```

```python
# Works. Two-step create new field and copy down. May need just a copy down.
def copy_updown(
        df, eventcodes, new_field_name, group_fields, copy_field,
        off_eventcode=None, direction='down', apply_to_timestamp='all'):
    '''
    df - eventlog dataframe
    eventcodes - EventCode(s) signifying event(s) to carry forward to subsequent events, e.g., 0 for PhaseStart
    new_field_name - name of Event corresponding to EventCode, e.g., PhaseStart
    group_fields - grouping(s) to which eventcode applies, e.g., [SignalID, EventParam] (Phase) for PhaseStart
    copy_field - field identifying eventcode, e.g., Timestamp for PhaseStart
    off_eventcode - optional value for where to stop copying up or down, otherwise goes to next value in
    eventcodes
    direction - 'up' for copy up, 'down' for copy down new_field_name
    apply_to_timestamp - 'all' to fill all rows with timestamps of the eventcodes before copying up or down,
                           Example would be 31-Barrier which should renew with all events at that same
                           timestamp,
                           of which there are many starts and ends to phase intervals
                         'group' to fill all rows at the timestamp in the group
                           Example would be Recorded Split
                         None to not fill all rows at the timestamp.
                           Example detector off (81) or call off (44) events
    '''
    if type(eventcodes) is list:
        if sum(df.EventCode.isin(eventcodes)) == 0:
            #print('Event Codes {} not in data frame'.format(','.join(map(str, eventcodes))))
            return df
        else:
            df.loc[df.EventCode.isin(eventcodes), new_field_name] = df.loc[df.EventCode.isin(eventcodes),
            copy_field]
    else:
        eventcode = eventcodes
        if sum(df.EventCode==eventcode) == 0:
            #print('Event Code {} not in data frame'.format(eventcode))
            return df
        else:
            df.loc[df.EventCode==eventcode, new_field_name] = df.loc[df.EventCode==eventcode, copy_field]

    if apply_to_timestamp=='all':
        df[new_field_name] = df.groupby(['SignalID','Timestamp'], group_keys=False)[new_field_name].transform(
        'max') ## This seems to work
    elif apply_to_timestamp=='group':
        group_vars = list(set(['SignalID','Timestamp'] + group_fields))
        df[new_field_name] = df.groupby(group_vars, group_keys=False)[new_field_name].transform('max') ## This
        seems to work

    if off_eventcode is not None:
        df.loc[df.EventCode==off_eventcode, new_field_name] = -1

    if direction == 'down':
        df[new_field_name] = df.groupby(group_fields)[new_field_name].ffill()
    elif direction == 'up':
        df[new_field_name] = df.groupby(group_fields)[new_field_name].bfill()

    if off_eventcode is not None:
        df.loc[df[new_field_name]==-1, new_field_name] = None

    return df
```

```
def copy_down(
        df, eventcodes, new_field_name, group_fields, copy_field,
        off_eventcode=None,
        apply_to_timestamp='all'):
    return copy_updown(
            df, eventcodes, new_field_name, group_fields, copy_field,
            off_eventcode=off_eventcode,
            apply_to_timestamp=apply_to_timestamp,
            direction='down')


def copy_up(
        df, eventcodes, new_field_name, group_fields, copy_field,
        off_eventcode=None,
        apply_to_timestamp='all'):
    return copy_updown(
            df, eventcodes, new_field_name, group_fields, copy_field,
            off_eventcode=off_eventcode,
            apply_to_timestamp=apply_to_timestamp,
            direction='up')


def widen(s, date_, det_config=None, source='s3'): # or source = 'db'

    signalid = s
    date0_ = date_ - pd.Timedelta(1, unit='D')
    # date0_str = date0_.strftime('%F')
    date_str = date_.strftime('%F')

    print('{} | {} started.'.format(date_str, s))

    if det_config is None:
        det_config = get_det_configs(config_bucket, [date0_, date_])

    dc = det_config[['SignalID','Detector','CallPhase','TimeFromStopBar','date']]
    dc = dc[dc.SignalID==s]

    if source=='s3':
        df = pd.concat(get_eventlog_data(events_bucket, signalid, [date0_, date_]))
    elif source=='db':
        df = get_eventlog_data_db(signalid, date_str)

    df = df[~df.EventCode.isin([43, 44])]

    df = df.rename(columns={'TimeStamp':'Timestamp'})
    df = df.sort_values(['SignalID','Timestamp','EventCode','EventParam']).reset_index(drop=True)

    print('{} | {} data queried from database.'.format(date_str, s))

    # Map Detectors to Phases.
    # Replace EventParam with Phase to align with other event types
    # Add new field for Detector from original EventParam field
    detector_codes = list(range(81,89))
    dc2 = pd.concat([dc.assign(EventCode=d).rename(columns={'Detector':'EventParam'}) for d in detector_codes])

    df = pd.merge(
            left=df,
            right=dc2,
            on=['SignalID','EventCode','EventParam','date'],
            how='left')\
        .reset_index(drop=True)

    # Adjust Timestamp for detectors by adding TimeFromStopBar
    df.Timestamp = df.Timestamp + pd.to_timedelta(df.TimeFromStopBar.fillna(0), 's')
    df = df.sort_values(['SignalID','Timestamp','EventCode','EventParam'])
```

```
256
257     codes = df.EventCode.drop_duplicates().values.tolist()
258
259     # Rename Detector, Phase columns
260     df.loc[df.EventCode.isin(detector_codes), 'Detector'] = df.loc[df.EventCode.isin(detector_codes),
        'EventParam']
261     df.loc[df.EventCode.isin(detector_codes), 'Phase'] = df.loc[df.EventCode.isin(detector_codes), 'CallPhase']
262     df = df.drop(columns=['CallPhase','TimeFromStopBar'])
263
264     df = create_new_grouping_field(df, list(range(83,89)), ['SignalID', 'Detector'], 'DetectorFault')
265     df = copy_down(df, list(range(84,89)), 'DetectorFault', ['SignalID','Detector'], 'EventParam', off_eventcode=
        83)
266
267     ped_input_codes = [89, 90]
268     df.loc[df.EventCode.isin(ped_input_codes), 'PedInput'] = df.loc[df.EventCode.isin(ped_input_codes),
        'EventParam']
269
270     # Global (Signal-wide) copy-downs. Uses two-step function to create new field and copy down
271     if 31 in codes:
272         df = copy_down(df, 31, 'Ring', ['SignalID'], 'EventParam')
273         df = copy_down(df, 31, 'CycleStart', ['SignalID'], 'Timestamp')
274     else:
275         df['Ring'] = None
276         df['CycleStart'] = None
277     if 131 in codes:
278         df = copy_down(df, 131, 'CoordPattern', ['SignalID'], 'EventParam')
279     else:
280         df['CoordPattern'] = None
281     if 132 in codes:
282         df = copy_down(df, 132, 'CycleLength', ['SignalID'], 'EventParam')
283     else:
284         df['CycleLength'] = None
285     if 316 in codes:
286         df = copy_down(df, 316, 'ActualCycleLength', ['SignalID'], 'EventParam') # New 7/20/21
287     else:
288         df['ActualCycleLength'] = None
289     if 317 in codes:
290         df = copy_down(df, 317, 'ActualNaturalCycleLength', ['SignalID'], 'EventParam') # New 7/20/21
291     else:
292         df['ActualNaturalCycleLength'] = None
293     if 133 in codes:
294         df = copy_down(df, 133, 'CycleOffset', ['SignalID'], 'EventParam')
295     else:
296         df['CycleOffset'] = None
297     if 318 in codes:
298         df = copy_down(df, 318, 'ActualCycleOffset', ['SignalID'], 'EventParam') # New 7/20/21
299     else:
300         df['ActualCycleOffset'] = None
301     if 150 in codes:
302         df = copy_down(df, 150, 'CoordState', ['SignalID'], 'EventParam')
303     else:
304         df['CoordState'] = None
305     if 173 in codes:
306         df = copy_down(df, 173, 'FlashStatus', ['SignalID'], 'EventParam')
307     else:
308         df['FlashStatus'] = None
309
310     split_eventcodes = list(range(134,150))
311     df = create_new_grouping_field(df, split_eventcodes, 'EventCode', 'Phase', lambda x: x-133)
312
313     actual_split_eventcodes = list(range(300, 316))
314     df = create_new_grouping_field(df, actual_split_eventcodes, 'EventCode', 'Phase', lambda x: x-299)
315
316     ped_wait_eventcodes = list(range(612, 652))
317     df = create_new_grouping_field(df, ped_wait_eventcodes, 'EventCode', 'Phase', lambda x: x-611)
```

```
318
319         phase_eventcodes = list(range(0,25)) + list(range(41,50)) + [151]
320         df = create_new_grouping_field(df, phase_eventcodes, 'EventParam', 'Phase')
321         df = copy_down(df, 0, 'PhaseStart', ['SignalID','Phase'], 'Timestamp')
322         df = copy_down(df, [1,8,10], 'Interval', ['SignalID','Phase'], 'EventCode', apply_to_timestamp='group')
323         df.loc[df.EventCode==4, 'TermType'] = 4
324         df.loc[df.EventCode==5, 'TermType'] = 5
325         df.loc[df.EventCode==6, 'TermType'] = 6
326
327         # TODO: See if we can get mapping between Vehicle Detector ID and Phase using (82, 81) and (43, 44).
328         #       Seems we can.
329         # TODO: See if we can get mapping between Pedestrian Detector ID and Phase using (90), (45)
330
331         df = copy_down(df, split_eventcodes, 'ProgrammedSplit', ['SignalID','Phase'], 'EventParam',
            apply_to_timestamp='group')
332         df = copy_up(df, actual_split_eventcodes, 'RecordedSplit', ['SignalID','Phase'], 'EventParam',
            apply_to_timestamp='group')
333         df = copy_down(df, ped_wait_eventcodes, 'PedWait', ['SignalID','Phase'], 'EventParam', apply_to_timestamp=
            'all')
334
335
336         df = copy_down(df, [183, 184], 'PowerFailure', ['SignalID'], 'EventParam', off_eventcode=182)
337
338         # Pair up detector on/offs under eventcode 82
339         df = copy_up(df, 81, 'DetectorOff', ['SignalID','Detector'], 'Timestamp', apply_to_timestamp=None)
340         df.loc[df.EventCode != 82, 'DetectorOff'] = np.nan
341         df['DetectorOff'] = pd.to_datetime(df['DetectorOff'])
342         df['DetectorDuration'] = (df['DetectorOff'] - df['Timestamp'])/pd.Timedelta(1, 's')
343
344         # Pair up ped input on/offs under eventcode 90
345         df = copy_up(df, 89, 'PedInputOff', ['SignalID','Detector'], 'Timestamp', apply_to_timestamp=None)
346         df.loc[df.EventCode != 90, 'PedInputOff'] = np.nan
347         df['PedInputOff'] = pd.to_datetime(df['PedInputOff'])
348         df['PedInputDuration'] = (df['PedInputOff'] - df['Timestamp'])/pd.Timedelta(1, 's')
349
350         df = df[~df.EventCode.isin([43, 44, 81, 89])]
351
352         # Pair up phase call on/offs under eventcode 43
353         #df = copy_up(df, 44, 'PhaseCallOff', ['SignalID','Detector'], 'Timestamp', apply_to_timestamp=None)
354         #df.loc[df.EventCode != 43, 'PhaseCallOff'] = np.nan
355         #df['PhaseCallDuration'] = (df['PhaseCallOff'] - df['Timestamp'])/pd.Timedelta(1, 's')
356         #df = df[~df.EventCode.isin([43, 44,81,89])]
357
358         # TODO: Need a way to account for multiple detectors, inputs, etc. that overlap.
359         #       Add a new column for each? e.g., Detector1, Detector2, etc.?
360
361         # Possible update to copy_down for phase interval status. but needs to be grouped by phase
362         #df.loc[df.EventCode.isin(eventcodes), 'Interval'] = df.loc[df.EventCode.isin(eventcodes), 'EventCode']
363
364         df = df[df['Timestamp'].dt.date==date_.date()].reset_index(drop=True)
365         df = df[['Timestamp','SignalID','EventCode','EventParam','date',
366                  'Ring','CycleStart','CoordPattern','CoordState',
367                  'CycleLength','ActualCycleLength','ActualNaturalCycleLength','CycleOffset','ActualCycleOffset',
368                  'Phase','PhaseStart','Interval','TermType','ProgrammedSplit','RecordedSplit',
369                  'Detector','DetectorFault','DetectorOff','DetectorDuration',
370                  'PedInput','PedWait','PedInputOff','PedInputDuration']]
371
372         print('{} | {} done.'.format(date_str, s))
373
374         df.to_parquet(f's3://{events_bucket}/atspm_wide/date={date_str}/atspm_wide_{s}_{date_str}.parquet')
375
376         return df
377
378
379
```

```python
def get_signalids(bucket, prefix):

    s3 = boto3.client('s3')
    paginator = s3.get_paginator('list_objects')

    # Create a PageIterator from the Paginator
    page_iterator = paginator.paginate(
        Bucket=bucket,
        Prefix=prefix)

    for contents in [page['Contents'] for page in page_iterator]:
        keys = [content['Key'] for content in contents]
        for key in keys:
            try:
                signalid = re.search('atspm_(.+?)_', key).group(1)
            except AttributeError:
                signalid = ''

            yield signalid



def get_channel_phase_mapping(df, channel_num=82, phase_num=43):

    df = df.rename(columns={'TimeStamp': 'Timestamp'})
    df = df.sort_values(['SignalID', 'Timestamp', 'EventCode', 'EventParam']).reset_index(drop=True)

    is_phase = df.EventCode==phase_num
    is_channel = df.EventCode==channel_num
    df.loc[is_phase, 'Timestamp'] = df.loc[is_phase, 'Timestamp'] - timedelta(seconds=0.1)

    dfc = df[is_channel].set_index(['SignalID', 'Timestamp'])['EventParam']
    dfp = df[is_phase].set_index(['SignalID', 'Timestamp'])['EventParam']

    dfcp = (pd.merge(left=dfc, right=dfp, on=['SignalID', 'Timestamp'], how='outer', suffixes=['_c', '_p'])
            .dropna('EventParam_p')
            .apply(tuple, axis=1))

    dfcp = (dfcp[dfcp.groupby(level=['SignalID', 'Timestamp']).transform('count') == 1]
         .reset_index(level='Timestamp', drop=True)
         .drop_duplicates()
         .sort_values())

    dfcp = (pd.DataFrame.from_records(
                dfcp,
                columns=['Detector', 'CallPhase'],
                index=dfcp.index)
           .astype(int)
           .reset_index(drop=False))

    return dfcp
```

```
if __name__=='__main__':

    if len(sys.argv) > 1:
        start_date = sys.argv[1]
        end_date = sys.argv[2]
    else:
        #start_date = '2021-07-14'
        #end_date = '2021-07-14'
        sys.exit('Need start_date and end_date as command line parameters')

    if start_date == 'yesterday':
        start_date = (datetime.today() - timedelta(days=1)).strftime('%Y-%m-%d')
    if end_date == 'yesterday':
        end_date = (datetime.today() - timedelta(days=1)).strftime('%Y-%m-%d')


    dates = pd.date_range(start_date, end_date, freq='1D')

    for date_ in dates:
        t0 = time.time()

        date0_ = date_ - pd.Timedelta(1, unit='D')
        date_str = date_.strftime('%Y-%m-%d')
        print(date_str)

        signalids = get_signalids(events_bucket, prefix=f'atspm/date={date_str}')
        det_config = get_det_configs(config_bucket, [date0_, date_])

        # with get_context('spawn').Pool(processes=os.cpu_count()-1) as pool:
        with Pool(os.cpu_count()-1) as pool:
            pool.starmap_async(widen, itertools.product(signalids, [date_], [det_config], ['s3']))
            pool.close()
            pool.join()

        print(f'{len(signalids)} signals in {round(time.time()-t0, 1)} seconds.')

#
df[~df.DetectorDuration.isna()].groupby(['SignalID','CycleStart','Phase','Interval']).count()['Timestamp'].unstac
k('Interval', fill_value=0)
# df[~df.TermType.isna()][['SignalID','CycleStart','Phase','TermType']]

```