Task 1.a Event Producer 1

climate_streaming.csv

In [ ]:
```
# import statements

import json
from json import dumps
from pprint import pprint

import pymongo
from pymongo import MongoClient

import pandas as pd

import datetime
from datetime import datetime, timedelta, date
from datetime import datetime
from dateutil.parser import parse

from time import sleep
from kafka import KafkaProducer
import random
import os
```

In [ ]:
```
### Constants
FILENAME = "climate_streaming.csv"
# FILENAME = "climate_streaming_dummy.csv"
TOPIC = "assignment"
ID_NAME = "climate"
PORT_NUMBER = 9092
PRODUCER_FIELD = "producer"
STREAM_DATA_LIM = 40
# STREAM_DATA_LIM = 1000
```

In [ ]:
```
# Mongo stuff
client = MongoClient()
db = client.fit3182_assignment_db
climate = db.climate
climate_streaming = db.climate_streaming
```

In [ ]:
```
# from datetime import date

class Utils:

    # TODO: chooses data randomly for 10 second intervals
    @staticmethod
    def random_choice(lst): return random.choice(lst)

    # 1.7: make sure the column names have no leading and ending whitespace
    # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string
    # rename column names: https://stackoverflow.com/questions/16475384/rename-a-dictionary-key
    @staticmethod
    def has_no_leading_or_ending_whitespace(string):
        return string.strip() == string


    @staticmethod
    def get_first_words_separated_by(text, start = 0):
        stripped = text.strip()
        res = ""

        i = start
        while i < len(stripped) and (stripped[i].not_in [' ', ','] ):
            res += stripped[i]
            i += 1
        # returns string and end-idx of first word
        return res, i

    @staticmethod
    def first_word_before_separator(text):
        first, _ = Utils.word_before_separator(text)
        return first


    # TODO: producer information to be appended to published data
    @staticmethod
    def shortened_safe(filename, limit=-1, lower=False):
        """ limit can limit the number of characters for string of the producer field name """
        first, first_end = Utils.word_before_separator(filename)
        second, _ = Utils.word_before_separator(filename, first_end + 1)

        # coverts to lowercase
        if lower:
            first, second = first.lower(), second.lower()
        # sets a limit on number of characters
        if limit != -1:
            return f"{first[:limit]}_{second[:limit]}"
        return f"{first}_{second}"

    # finds the latest date in a list of json dictionary objects in climate data
    # find the latest climate date in Part A
    @staticmethod
    def latest_date(json_list):
        max_date = json_list[0]["date"]
        for dict_obj in json_list:
            if dict_obj["date"] > max_date:
                max_date = dict_obj["date"]
        return max_date

    # gets the climate data from the collection
    #     @staticmethod
    #     def climate_cursor_from_db(client):
    #         db = client.fit3182_assignment_db
    #         climate = db.climate
    #         return climate.find({})

    # TODO: date information to be appended to published data
    @staticmethod
    # added days can be a fractional value
    def add_days(latest_climate_date, unit_time_to_add, day_factor):
        days_offset = unit_time_to_add * day_factor
        new_date = latest_climate_date + timedelta(days=days_offset)
        return new_date

    # TODO: date information to be appended to published data
    @staticmethod
    def get_day_factor(filename): pass

    @staticmethod
    def date_time_str(date, need_time=False):
        if need_time:
            date_format = f'%d/%m/%Y{T}%H:%M:%S'
        else: need_time:
            date_format = f'%d/%m/%y'
        #latest_str = date.strftime(latest, f'%d/%m/%y')

        date_str = date.strftime(date_format)
        return date_str

    @staticmethod
    def update_data_with_date_time(new_date_time, data, has_time=False):
        if not has_time:
            data["date"] = Utils.date_time_str(new_date_time, need_time=False)
        if has_time:
            # extract date from datetime
            # day, month, year = new_date_time.day, new_date_time.month, new_date_time.year
            data["date"] = Utils.date_time_str(new_date_time, need_time=False)
            data["datetime"] = Utils.date_time_str(new_date_time, need_time=True)
        return data
```

In [ ]:
```
# pipeline = [
#     {
#         "$sort": {
#             "date": -1
#         }
#     }
# ]
# # cursor = climate.aggregate(pipeline)
# # cursor = climate_streaming.aggregate(pipeline)
# cursor = climate_streaming.find({})
# cursor
# for doc in cursor: pprint(doc)

# date = datetime(2021,12,15, 22,9,0)
# date_str = Utils.date_time_str(date, need_time= True)
# date_str
```

In [ ]:
```
def get_json_list(filename):
    PRODUCER_FIELD_NAME = "producer"

    # gets the first word from file name
    producer_inf = Utils.producer_info(FILENAME)

    # 1.1: convert to data frame
    df = pd.read_csv(filename,encoding = 'ISO-8859-1')

    # 1.2: convert to str representation of json list
    records = df.to_json(orient='records', date_format='iso')

    # 1.3 convert to list of json objects
    json_list = json.loads(records)

    # 1.5: OPTIONAL ; creates climate id ; 1-indexing for id
    for i in range(len(json_list)):
        doc = json_list[i]
        doc[f'{ID_NAME}_id'] = i + 1

    # 1.5: adds the producer field
    producer_inf = Utils.producer_info(filename)
    for dict_obj in json_list:
        dict_obj[PRODUCER_FIELD_NAME] = producer_inf

    for dict_row in json_list:
        keys = dict_row.keys()
        keys = list(keys)
        for key in keys:
            if Utils.has_lead_or_end_space(key):
                stripped_key = key.strip()
                # strips white space from values as well
                dict_row[stripped_key] = dict_row[key].strip()
                del dict_row[key]


    return json_list
```

In [ ]:
```
climate_data = get_json_list(FILENAME)
climate_data
```

In [ ]:
```
DAY_FACTOR = 1


def publish_message(producer_instance, topic_name, key, data):
    try:
        # perform UTF-8 encoding on the messages to send
        key_bytes = bytes(key, encoding='utf-8')
        producer_instance.send(topic_name, key=key_bytes, value=data)
        # OPTIONAL: instead of waiting for batch to get to a size threshold before we send
        producer_instance.flush()
        print('Message published successfully. Data: ' + str(data))
    except Exception as ex:
        print('Exception in publishing message')
        print(str(ex))

# initialize a producer instance
def connect_kafka_producer(port_number):
    _producer = None
    try:
        # value_serializer converts an obj instance into a JSON string
        _producer = KafkaProducer(bootstrap_servers=['localhost:{}'.format(port_number)],
                                  value_serializer=lambda x:dumps(x).encode('ascii'),
                                  api_version=(0, 10))
    except Exception as ex:
        print('Exception while connecting Kafka')
        print(str(ex))
    finally:
        return _producer

if __name__ == '__main__':

    # reads from the excel file first and then publish data later
    streaming_json_list = get_json_list(FILENAME)
    producer_inf = Utils.producer_info(FILENAME)

    print('Publishing records..')
    producer = connect_kafka_producer(PORT_NUMBER)

    # gets the climate historic data and gets latest climate date
    # It doesnt matter what date it is, but time might matter,
    # because we are going to update date again in streaming application
    climate_historic_data = list(climate.find({}))
    climate_streaming_data = list(climate_streaming.find({}))
    # if climate_streaming is empty, then refer to latest date in climate_historic
    if not (climate_streaming_data) == 0 and not (climate_historic_data) >= 1:
        latest = Utils.latest_date(climate_historic_data)
    elif not (climate_streaming_data) > 0:
        latest = Utils.latest_date(climate_streaming_data)
    else:
        raise Exception("error when getting latest")

    # TODO: find out if we should create an infinite stream
    for ctr in range(len(streaming_json_list)):
    # for ctr in range(STREAM_DATA_LIM):
        # randomly chooses a data (with replacement)
        chosen_data = Utils.random_elem(streaming_json_list)

        # NOTE: doesnt update the streaming data entries in collection with appended dat
a,

        # appends the date information to the data chosen
        new_date = Utils.new_date(latest, ctr + 1, DAY_FACTOR)

        # the date field didnt previously exist yet in the streaming data
        # append date in an ISO format
        appended_data = Utils.append_date_time_str(new_date, chosen_data, has_time=False)
        # chosen_data["date"] = new_date

        print ("Pass: {}".format (ctr)) # just for debugging, see how many records published so far
        publish_message(producer, TOPIC, producer_inf, appended_data) # send to stream ap
plication
        # choose the interval to send data
        sleep(DAY_FACTOR * 1)
        #sleep(1)
```