Reading the files from CSV files and load to fit3182_assignment_db

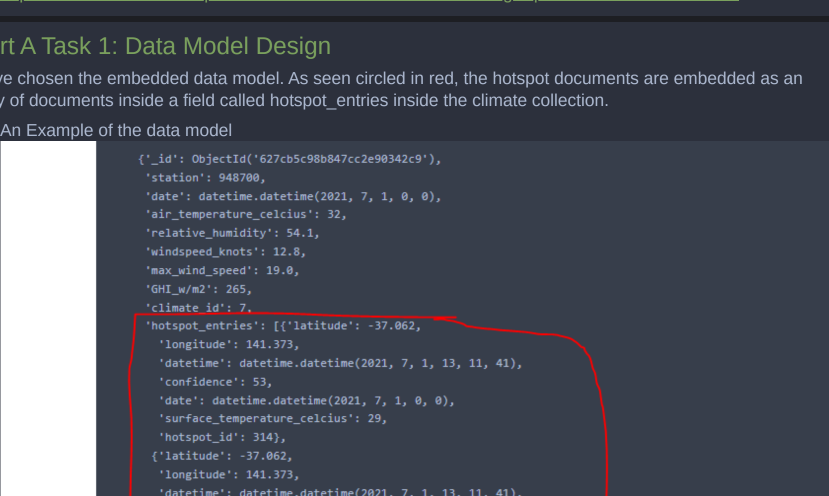
references:

- https://www.geeksforgeeks.org/how-to-convert-pandas-dataframe-into-json-in-python/
- https://stackoverflow.com/questions/1894269/how-to-convert-string-representation-of-list-to-a-list

Part A Task 1: Data Model Design

I have chosen the embedded data model. As seen circled in red, the hotspot documents are embedded as an array of documents inside a field called hotspot_entries inside the climate collection.

- An Example of the data model



```
{'_id': ObjectId('627cb5c98b847cc2e90342c9'),
 'station': 948700,
 'date': datetime.datetime(2021, 7, 1, 0, 0),
 'air_temperature_celcius': 32,
 'relative_humidity': 54.1,
 'windspeed_knots': 12.8,
 'max_wind_speed': 19.0,
 'GHI_w/m2': 265,
 'climate_id': 7,
 'hotspot_entries': [{'latitude': -37.062,
   'longitude': 141.373,
   'datetime': datetime.datetime(2021, 7, 1, 13, 11, 41),
   'confidence': 53,
   'date': datetime.datetime(2021, 7, 1, 0, 0),
   'surface_temperature_celcius': 29,
   'hotspot_id': 314},
  {'latitude': -37.062,
   'longitude': 141.373,
   'datetime': datetime.datetime(2021, 7, 1, 13, 11, 41),
   'confidence': 53,
   'date': datetime.datetime(2021, 7, 1, 0, 0),
   'surface_temperature_celcius': 29,
   'hotspot_id': 315},
  {'latitude': -36.779,
   'longitude': 146.108,
   'datetime': datetime.datetime(2021, 7, 1, 3, 46, 8),
   'confidence': 61,
   'date': datetime.datetime(2021, 7, 1, 0, 0),
   'surface_temperature_celcius': 32,
   'hotspot_id': 316},
  {'latitude': -36.779,
   'longitude': 146.108,
   'datetime': datetime.datetime(2021, 7, 1, 3, 46, 8),
   'confidence': 61,
   'date': datetime.datetime(2021, 7, 1, 0, 0),
   'surface_temperature_celcius': 32,
   'hotspot_id': 317}],
 'precipitation': '0.00I'}
```

justification for choosing the data model

Space issues and JOIN costs

The embedded data model is certainly a very space-intensive data model.

However, the main advantage to using the embedded data model is that JOIN operations are very costly, so if we were to use a reference data model, there are certain queries that we would have to perform a JOIN operation between the two collections of climate and hotspot.

Instead of maintaing a reference like in the reference model, the embedded data model eliminates the expensive cost of JOINing, as well as keeps things simple as we could perform aggregate queries easily by using the pymongo API.

Issue with complicated queries using reference model

Due to mongoDB not supporting normalization like so in RDMS, we will find that queries are more complicated and tedious at the programming-level if we use the reference model.

maximum document size

I also took into account of the maximum document size of 16Mb which is the default maximum document size of MongoDB. However, it is very very unlikely in this case, for a specific hotspot_entries document to be exceeding the limit.

This is because we wouldn't be making any updates to a climate document after inserting it. Moreover, every unique climate document is based on a unique date, and there couldnt possible be that many fire incidents which directly correlates to number of records of hotspot for a given unique date, to exceed the maximum document size of 16Mb.

Part A Task 2.1

In [ ]:
```python
import json
import pymongo
import pandas as pd
from pymongo import MongoClient
from pprint import pprint
import ast

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


# 1: read from hotspot_historic.csv & climate_historic.csv
# 1.1: convert to data frame
hotspot_df = pd.read_csv('hotspot_historic.csv',encoding = 'ISO-8859-1')
climate_df = pd.read_csv('climate_historic.csv',encoding = 'ISO-8859-1')


# convert to proper datetime format and cleaning up some column names
hotspot_df["datetime"]= pd.to_datetime(hotspot_df["datetime"], infer_datetime_format=True
)
hotspot_df["date"]= pd.to_datetime(hotspot_df["date"], infer_datetime_format=True)

climate_df["date"]= pd.to_datetime(climate_df["date"], infer_datetime_format=True)

# 1.2: convert to str representation of json list
hotspot_records = hotspot_df.to_json(orient="records", date_format="iso")
climate_records = climate_df.to_json(orient="records", date_format="iso")

# 1.3 convert to list of json objects
hotspot_json_list = json.loads(hotspot_records)
climate_json_list = json.loads(climate_records)

# 1.4: convert string json back to date time object to store into mongoDB
for dict_obj in hotspot_json_list:
    dict_obj['datetime'] = parse(dict_obj['datetime'], ignoretz=True, dayfirst=True)
    dict_obj['date'] = parse(dict_obj['date'],ignoretz=True, dayfirst=True)

for dict_obj in climate_json_list:
    dict_obj['date'] = parse(dict_obj['date'],ignoretz=True, dayfirst=True)




# 1.5: OPTIONAL ; creates climate id and climate_producer ; 1-indexing for id
for i in range(len(climate_json_list)):
    c_doc = climate_json_list[i]
    c_doc["climate_id"] = i + 1

    # FIXME: This is mainly for debugging purposes for Part B of the assignment,
    # to know where this data came from
    c_doc["producer"] = "climate_historic"

for i in range(len(hotspot_json_list)):
    h_doc = hotspot_json_list[i]
    h_doc["hotspot_id"] = i + 1


# 1.6: embedding
for c_doc in climate_json_list:
    c_doc_date = c_doc["date"]
    # find entries in hotspot that has same c_doc_date
    temp_lst = []
    for hotspot_doc in hotspot_json_list:
        if hotspot_doc["date"] == c_doc_date:
            temp_lst.append(hotspot_doc)
    c_doc["hotspot_entries"] = temp_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-k
ey
def has_lead_or_end_space(string):
    return string.strip() != string

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

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


# 2: connect on the default host and port
client = MongoClient() # connect on the default host and port

# 3: create mongoDB database
# client.drop_database('fit3182_assignment_db')
db = client.fit3182_assignment_db

# 4: create 2 new collections
# hotspot = db.hotspot
climate = db.climate
# drop any existing climate collection first
climate.drop()

# 5: insert stuff into the collections
# result_hotspot = hotspot.insert_many(hotspot_json_list)
result_climate = climate.insert_many(climate_json_list)

print(result_climate.acknowledged)
```

In [ ]:
```python
# hotspot_cursor = hotspot.find({})
# hotspot_data = list(hotspot_cursor)
# len(hotspot_data)
# hotspot_data[-1]

climate_cursor = climate.find({})
# climate_data = list(climate_cursor)
# len(climate_data)

for doc in climate_cursor:
#     pprint(doc)
    doc["producer"] = "updated producer"
    pprint(doc)

# climate_data = list(climate_cursor)

# climate_data

# climate_data
```

Justification for Data model used

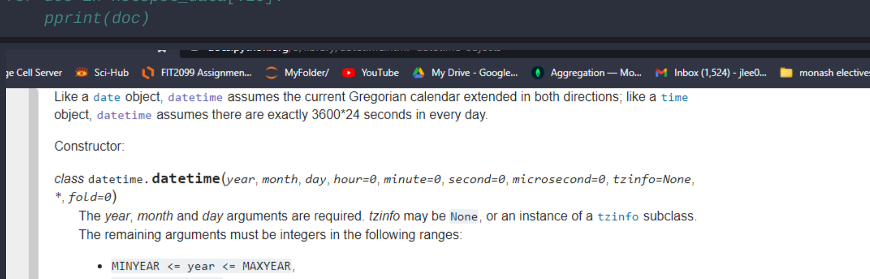
- Just to keep it simple, I have chosen the embedded data model
- this is because I can easily embedded data model when using queries

In [ ]:
```python
"""
- using a data reference model build a reference from hotspot to climate
- store the climate_id inside hotspot
"""
# # 1: loop through each entry in hotspot
# hotspot_cursor = hotspot.find({})
# hotspot_data = list(hotspot_cursor)

# # for each hotspot entry, find a climate entry that has same date with hotspot and form
 the link
# for hs_doc in hotspot_data:
#     # find a climate entry that has same date with hotspot
#     query = {"date": hs_doc["date"]}
#     c_data = climate.find(query)
#     c_data_len = climate.count_documents(query)

#     # create a new attribute called climate_id\
#     if c_data_len == 1:
#         c_doc = c_data[0]
#         hs_doc["climate_id"] = c_doc["_id"]
#     elif c_data_len == 0:
#         hs_doc["climate_id"] = "null"
#     else:
#         raise Exception("c_data_len is more than 0 or 1")

# for doc in hotspot_data[:10]:
#     pprint(doc)
```



Like a `date` object, `datetime` assumes the current Gregorian calendar extended in both directions; like a `time` object, `datetime` assumes there are exactly 3600*24 seconds in every day.

Constructor:

class `datetime.`**`datetime`**(*year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None, *, fold=0*)

The *year*, *month* and *day* arguments are required. *tzinfo* may be `None`, or an instance of a `tzinfo` subclass. The remaining arguments must be integers in the following ranges:

- `MINYEAR <= year <= MAXYEAR`,
- `1 <= month <= 12`,
- `1 <= day <= number of days in the given month and year`,
- `0 <= hour < 24`,
- `0 <= minute < 60`,
- `0 <= second < 60`,
- `0 <= microsecond < 1000000`,
- `fold in [0, 1]`.

If an argument outside those ranges is given, `ValueError` is raised.

New in version 3.6: Added the `fold` argument.

a. find the climate data on 12th December 2021

In [ ]:
```python
# hotspot.count_documents({"datetime": "/(2021-12-12)/"})
from datetime import datetime
climate.count_documents({"date": datetime(2021,12,12)})

climate_cursor = climate.find({"date": datetime(2021,12,12)})

for doc in climate_cursor:
    pprint(doc)

#[doc for doc in cliamte_cursor]
```

b. Find the latitude, longitude, surface temperature (°C), and confidence

- when the surface temperature (°C) was between 65 °C and 100 °C.
- NOTE: double checked in excel to be correct

In [ ]:
```python
# cursor = climate.find(
#     {
#         "hotspot_entries": {
#             "$elemMatch": {
#                 "surface_temperature_celcius": {"$gte":65, "$lte":100}
#             }
#         }
#     }
# )

pipeline = [
#     {
#         "$unwind": "$hotspot_entries"
#     },
    {
        "$match":  {
            "hotspot_entries.surface_temperature_celcius": {"$gte":65, "$lte":100}
        }
    },
    {
        "$project":{
            "climate_id": 1,
            "hotspot_entries.hotspot_id": 1,
            "hotspot_entries.latitude": 1,
            "hotspot_entries.longitude": 1,
            "hotspot_entries.surface_temperature_celcius": 1,
            "hotspot_entries.confidence": 1
        }
    }
]


cursor = climate.aggregate(pipeline)
for doc in cursor: pprint(doc)
```

c. Find date, surface temperature (°C), air temperature (°C), relative humidity and max wind speed

- on 15th and 16th of December 2021.

In [ ]:
```python
from datetime import datetime

# pipeline = [

#     {
#         "$lookup": {
#             "from": "hotspot",
#             "localField": "date",
#             "foreignField": "date",
#             "as": "hotspot_dates"
#         }
#     },
# ]

"""
1: match on 15 and 16 dec 2021
2: lookup
3: project
"""

# datetime(yr,month,day)
both_days = [datetime(2021,12,15), datetime(2021,12,16)]

pipeline = [
    {
        "$match":{"date": {"$in": both_days}}
    },
    {
        "$project": {
            "climate_id": 1,
            "date": 1,
            "air_temperature_celcius": 1,
            "max_wind_speed": 1,
            "relative_humidity": 1,
            "hotspot_entries.surface_temperature_celcius": 1,
            "hotspot_entries.hotspot_id": 1
        }
    }
]

# climate_join_hotspot is a cursor object
cursor = climate.aggregate(pipeline)

for doc in cursor:
    pprint(doc)
# pprint(type(climate_join_hotspot))
```

d. Find datetime, air temperature (°C), surface temperature (°C) and confidence

- when the confidence is between 80 and 100.

In [ ]:
```python
pipeline = [
     # doing unwind, because, if we do $match without $unwind, we can still see records o
f
    # embedded hotspot_entries NOT within this range in climate_data, because $match retu
rns TRUE if
    # any 1 of the hotspot_entries fulfill this range
    {
        "$unwind": "$hotspot_entries"
    },
    {
        "$match": {"hotspot_entries.confidence": {"$gte":80, "$lte":100}}
    },
    {
        "$project": {
            "climate_id": 1,
            "air_temperature_celcius": 1,
            "hotspot_entries.hotspot_id": 1,
            "hotspot_entries.datetime": 1,
            "hotspot_entries.surface_temperature_celcius": 1,
            "hotspot_entries.confidence": 1
        }
    }
]

# climate is embedded with hotspot entries
cursor = climate.aggregate(pipeline)
for doc in cursor: pprint(doc)
```

e. Find the top 10 records with the highest surface temperature (°C).

In [ ]:
```python
pipeline = [
    # technically will work without unwind, but will return all embedded hotspot_entries
    # in the climate datas that have top 10 highest surface_temp
    # inside any 1 the embedded hotspot_data
    {
        "$unwind": "$hotspot_entries"
    },
    {
        "$sort": {
            "hotspot_entries.surface_temperature_celcius": -1
        }
    },
    {
        "$limit": 10
    }
]

cursor = climate.aggregate(pipeline)
data = list(cursor)
for doc in data: pprint(doc)
# len(data)
```

f. Find the number of fires each day. You are required to only display the total number of fires and the date in the output.

In [ ]:
```python
pipeline = [
    {
        "$unwind": "$hotspot_entries"
    },
    {
        "$group": {
            "_id": {
                "climate_id": "$climate_id",
                "date": "$date"    # we included it here, because we want to project it l
ater
            },
            "number_of_fires": {  # output column field name
                "$sum": 1    # summing an expression <> for each record
                             # e.g.  $sum: { $multiply: [ "$price", "$quantity" ]
            }
        }
    },
    {
        "$project": {
            "climate_id": 1,
            "date": 1,
            "number_of_fires": 1
        }
    }
]

cursor = climate.aggregate(pipeline)
for doc in cursor: pprint(doc)
```

g. Find the records of fires where the confidence is below 70.

In [ ]:
```python
pipeline = [
    {
        "$unwind": "$hotspot_entries"
    },
    {
        "$match": {
            "hotspot_entries.confidence": {"$lt": 70}
        }
    }
]

cursor = climate.aggregate(pipeline)
data = list(cursor)
# for doc in data: pprint(doc)
len(data)
```

h. Find the average surface temperature (°C) for each day. You are required to only display average surface temperature (°C) and the date in the output.

In [ ]:
```python
pipeline = [
    {
        "$unwind": "$hotspot_entries"
    },
    {
        "$group": {
            "_id": {
                "climate_id": "$climate_id",
                "date": "$date"    # we included it here, because we want to project it l
ater
            },
            "avg_surface_temp": {  # output column field name
                "$avg": "$hotspot_entries.surface_temperature_celcius"
            }
        }
    },
    {
        "$project": {
            "avg_surface_temp": 1,
            "date": 1
        }
    }
]

cursor = climate.aggregate(pipeline)
data = list(cursor)
for doc in data: pprint(doc)
# len(data)
```

i. Find the top 10 records with the lowest GHI.

In [ ]:
```python
pipeline = [
    {
        "$sort": {"GHI_w/m2": 1}
    },
    {
        "$limit": 10
    }
]

cursor = climate.aggregate(pipeline)
data = list(cursor)
for doc in data: pprint(doc)
#len(data)
```

j. Find the records with a 24-hour precipitation recorded between 0.20 to 0.35.

In [ ]:
```python
# 0.2 to 0.35
remove_last = lambda x: x[:len(x)-1]

pipeline = [
    {
        "$project": {
            "climate_id": 1,
            "precipitation": 1,
            # extracting precip type
            "precipitation_type":{
                # building a substring to extract the type of the precipitation, val+type
                 "$substr": [
                     "$precipitation",
                     {
                         "$subtract": [ {"$strLenCP": "$precipitation"}, 1 ]
                     },
                     1
                 ]
            },
            # extracting precip value and convert string to double type
            "precipitation_value":{
                "$toDouble": {
                    "$substr": [
                        "$precipitation",
                        0,
                        {"$subtract": [ {"$strLenCP": "$precipitation"}, 1 ] }
                    ]
                }
            }
        }
    },
    # perform matching against precip type 'G' and value between 0.2 to 0.35
    {
        "$match":{
            "precipitation_type": 'G',
            "precipitation_value": {"$gte": 0.2, "$lte": 0.35}
        }
    }
]

# I have slightly modified the document after the projection,
# notice the precipitation type and value are now split into 2 fields after the projectio
n
cursor = climate.aggregate(pipeline)
data = list(cursor)
for doc in data: pprint(doc)
# len(data)
```

In [ ]:
```python
# Testing for Part B of assignment

# lst = list(climate.find({}))
# lst[-2:]
```