DSC650-wk2

March 31, 2021

[2]:
```
import pandas as pd
import s3fs
```

[3]:
```
import json
from pathlib import Path
import os


def read_cluster_csv(file_path, endpoint_url='https://storage.budsc.
 ↪midwest-datascience.com'):
    s3 = s3fs.S3FileSystem(
        anon=True,
        client_kwargs={
            'endpoint_url': endpoint_url
        }
    )
    return pd.read_csv(s3.open(file_path, mode='rb'))

current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')
kv_data_dir = results_dir.joinpath('kvdb')
kv_data_dir.mkdir(parents=True, exist_ok=True)

people_json = kv_data_dir.joinpath('people.json')
visited_json = kv_data_dir.joinpath('visited.json')
sites_json = kv_data_dir.joinpath('sites.json')
measurements_json = kv_data_dir.joinpath('measurements.json')
```

[4]:
```
class KVDB(object):
    def __init__(self, db_path):
        self._db_path = Path(db_path)
        self._db = {}
        self._load_db()

    def _load_db(self):
        if self._db_path.exists():
            with open(self._db_path) as f:
                self._db = json.load(f)

    def get_value(self, key):
        return self._db.get(key)

    def set_value(self, key, value):
        self._db[key] = value

    def save(self):
        with open(self._db_path, 'w') as f:
            json.dump(self._db, f, indent=2)
```

[5]:
```
def create_sites_kvdb():
    db = KVDB(sites_json)
    df = read_cluster_csv('data/external/tidynomicon/site.csv')
    for site_id, group_df in df.groupby('site_id'):
        db.set_value(site_id, group_df.to_dict(orient='records')[0])
    db.save()


def create_people_kvdb():
    db = KVDB(people_json)
    df = read_cluster_csv('data/external/tidynomicon/person.csv')
    for person_id, group_df in df.groupby('person_id'):
        db.set_value(person_id, group_df.to_dict(orient='records')[0])
    db.save()


def create_visits_kvdb():
    db = KVDB(visited_json)
    df = read_cluster_csv('data/external/tidynomicon/visited.csv')
    df.fillna(' ',inplace=True)
    for key, group_df in df.groupby(['visit_id', 'site_id']):
        db.set_value(str(key), group_df.to_dict(orient='records')[0])
    db.save()


def create_measurements_kvdb():
    db = KVDB(measurements_json)
    df = read_cluster_csv('data/external/tidynomicon/measurements.csv')
    for key, group_df in df.groupby(['person_id', 'visit_id', 'quantity']):
        db.set_value(str(key), group_df.to_dict(orient='records')[0])
    db.save()
```

[6]:
```
create_sites_kvdb()
create_people_kvdb()
create_visits_kvdb()
```

```
create_measurements_kvdb()
```

[7]:
```
from pathlib import Path
import json
import os
from tinydb import TinyDB
current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')
kv_data_dir = results_dir.joinpath('kvdb')
kv_data_dir.mkdir(parents=True, exist_ok=True)

def _load_json(json_path):
    with open(json_path) as f:
        return json.load(f)

class DocumentDB(object):
    def __init__(self, db_path):
        ## You can use the code from the previous exmaple if you would like
        people_json = kv_data_dir.joinpath('people.json')
        visited_json = kv_data_dir.joinpath('visited.json')
        sites_json = kv_data_dir.joinpath('sites.json')
        measurements_json = kv_data_dir.joinpath('measurements.json')
        self._db_path = Path(db_path)
        self._db = None
        self._person_lookup = _load_json(people_json)
        self._measurements_lookup = _load_json(measurements_json)
        self._visit_lookup = _load_json(visited_json)
        self._load_db()

    def _get_site(self, site_id):
        return self._site_lookup[str(site_id)]
    def _get_measurements(self, person_id):
        measurements = []
        for values in self._measurements_lookup.values():
            measurements.extend([values for value in values if␣
 ↪str(['person_id']) == str(person_id)])
        return measurements
    def _get_visit(self, visit_id):
        visit = self._visit_lookup.get(str(visit_id))
        site_id = visit['site_id']
        site = visit
        visit['site'] = visit
        return visit
    def _load_db(self):
        self._db = TinyDB(self._db_path)
        persons = self._person_lookup.items()
        for person_id, record in persons:
            measurements = self._get_measurements(person_id)
            visit_ids = set([measurement['visit_id'] for measurement in␣
 ↪measurements])
            visits = []
            for visit_id in visit_ids:
                visit = self._get_visit(visit_id)
                visit['measurements'] = [
                    measurement for measurement in measurements
                    if visit_id == measurement['visit_id']
                ]
                visits.append(visit)
            record['visits'] = visits
            self._db.insert(record)
```

[8]:
```
db_path = results_dir.joinpath('patient-info.json')
if db_path.exists():
    os.remove(db_path)

db = DocumentDB(db_path)
```

[28]:
```
import sqlite3

def create_measurements_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS measurements (
        visit_id integer NOT NULL,
        person_id text NOT NULL,
        quantity text,
        reading real,
        FOREIGN KEY (visit_id) REFERENCES visits (visit_id),
        FOREIGN KEY (person_id) REFERENCES people (people_id)
        );
    """

    c = conn.cursor()
    c.execute(sql)

def load_measurements_table(conn):
    create_measurements_table(conn)
    df = read_cluster_csv('data/external/tidynomicon/measurements.csv')
    measurements = df.values
    c = conn.cursor()
    c.execute('DELETE FROM measurements;') # Delete data if exists
    c.executemany('INSERT INTO measurements VALUES (?,?,?,?)', measurements)
```

[29]:
```
def create_people_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS people (
        person_id text NOT NULL,
        personal_name text,
        family_name text,
        FOREIGN KEY (person_id) REFERENCES measurements (people_id)
        );
    """
    c = conn.cursor()
    c.execute(sql)

def load_people_table(conn):
    create_people_table(conn)
    df = read_cluster_csv('data/external/tidynomicon/person.csv')
    people = df.values
    c = conn.cursor()
    c.execute('DELETE FROM people;') # Delete data if exists
    c.executemany('INSERT INTO people VALUES (?,?,?)', people)
```

[33]:

```
def create_sites_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS sites (
        site_id text PRIMARY KEY,
        latitude double NOT NULL,
        longitude double NOT NULL
        );
    """

    c = conn.cursor()
    c.execute(sql)

def load_sites_table(conn):
    create_sites_table(conn)
    df = read_cluster_csv('data/external/tidynomicon/site.csv')
    sites = df.values
    c = conn.cursor()
    c.execute('DELETE FROM sites;') # Delete data if exists
    c.executemany('INSERT INTO sites VALUES (?,?,?)', sites)
```

[34]:

```
def create_visits_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS visits (
        visit_id integer PRIMARY KEY,
        site_id text NOT NULL,
        visit_date text,
        FOREIGN KEY (site_id) REFERENCES sites (site_id)
        );
    """
    c = conn.cursor()
    c.execute(sql)

def load_visits_table(conn):
    create_visits_table(conn)
    df = read_cluster_csv('data/external/tidynomicon/visited.csv')
    visits = df.values
    c = conn.cursor()
    c.execute('DELETE FROM visits;') # Delete data if exists
    c.executemany('INSERT INTO visits VALUES (?,?,?)', visits)
```

[36]:
```
db_path = results_dir.joinpath('patient-info.db')
conn = sqlite3.connect(str(db_path))

load_people_table(conn)
load_sites_table(conn)
load_visits_table(conn)
load_measurements_table(conn)

conn.commit()
conn.close()
```

[ ]:
```
```