Assignment 2 Katie Briggs DSC650 302

December 13, 2020

Assignment 2 Katie Briggs DSC 650 302

1 Assignment 2

2 Import Data to be used for Assignments

For this assignment, we will be working with the CSV data found in the data/external/tidynomicon folder. Specifically, we will be using with the measurements.csv, person.csv, site.csv, and visited.csv files.

[135]:
```
import json
from pathlib import Path
import os
import pandas as pd
import s3fs

file_path = 'data/external/tidynomicon'

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'))
```

3 Assignment 2.1

Complete the code in kvdb.ipynb to implement a basic key-value database that saves its state to a json file. Use that code to create databases that store each of CSV files by key. The json files should be stored in the dsc650/assignments/assignment02/results/kvdb/ folder.

[136]:
```
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')
```

[137]:
```
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)
```

[138]:
```
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_person = read_cluster_csv('data/external/tidynomicon/person.csv')
    for person_id, group_df in df_person.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_visit = read_cluster_csv('data/external/tidynomicon/visited.csv')
```

```
    for visit_id, group_df in df_visit.groupby('visit_id'):
        db.set_value(visit_id, group_df.to_dict(orient='records')[0])
    db.save()


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

[139]:
```
create_sites_kvdb()
create_people_kvdb()
create_visits_kvdb()
create_measurements_kvdb()
```

kvdb_path = 'visits.json' kvdb = KVDB(kvdb_path) key = (619, 'DR-1') value = dict(visit_id=619, site_id='DR-1', visit_date='1927-02-08') kvdb.set_value(key, value) retrieved_value = kvdb.get_value(key)

[140]:
```
kvdb_path = 'measurements.json'
kvdb = KVDB(kvdb_path)
key = (619, 'dryer')
value = dict(
    visit_id=619,
    person_id='dyer',
    quantity='rad'
)
kvdb.set_value(key, value)
retrieved_value = kvdb.get_value(key)
```

[141]:
```
kvdb_path = 'visits.json'
kvdb = KVDB(kvdb_path)
key = (619, 'DR-1')
value = dict(
    visit_id=619,
    site_id='DR-1',
    visit_date='1927-02-08'
)
kvdb.set_value(key, value)
retrieved_value = kvdb.get_value(key)
```

4 Assignment 2.2

Now we will create a simple document database using the tinydb library. TinyDB stores its data as a JSON file. For this assignment, you will store the TinyDB database in

dsc650/assignments/assignment02/results/patient-info.json. You will store a document for each person in the database which should look like this.

[142]:
```
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)
```

[143]:
```
def _load_json(json_path):
    with open(json_path) as f:
        return json.load(f)
```

[144]:
```
class DocumentDB(object):

    def __init__(self, db_path):

        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._visit_lookup = _load_json(visited_json)
        self._site_lookup = _load_json(sites_json)
        self._measurements_lookup = _load_json(measurements_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([value 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 = str(visit['site_id'])
        site = self._site_lookup(site_id)
        visit['site'] = site
        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)
```

[145]:
```
db_path = results_dir.joinpath('patient-info.json')
if db_path.exists():
    os.remove(db_path)

db = DocumentDB(db_path)
```

5 Assignment 2.3

In this part, you will create a SQLite database that you will store in dsc650/assignments/assignment02/results/patient-info.db. The dsc650/assignments/assignment02/rdbms.ipynb file should contain code to assist you in the creation of this database.

[146]:
```
from pathlib import Path
import os
import sqlite3

import s3fs
import pandas as pd

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)

#file_path = 'data/external/tidynomicon'

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'))
```

[147]:
```
# Create and Load

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

[148]:
```
# Create and load people


def create_people_table(conn):
    sql = """
    CREATE TABLE IF NOT EXISTS people (
        people_id text NOT NULL,
        personal_name text,
```

```
        family_name text
        );
    """

    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)
```

[149]:
```
# Create and load sites table

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

[150]:
```
# Create and load visits

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

[151]:
```
# Create DB and Load

db_path = results_dir.joinpath('patient-info.db')
conn = sqlite3.connect(str(db_path))
# TODO: Uncomment once functions completed

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

sql = """SELECT * FROM visits;"""

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

result = c.fetchall

print(result)
conn.commit()
conn.close()
```

```
<built-in method fetchall of sqlite3.Cursor object at 0x7f1fbc61b490>
```

6 Assignemnt 2.4

Modify the query so that the column order is date, event, and eventLabel instead of event, eventLabel, and date. Download the results as a JSON file and copy the results to dsc650/assignments/assignment02/results/wikidata-query.json.

#Recent Events SELECT ?date ?event ?eventLabel WHERE { # find events ?event wdt:P31/wdt:P279* wd:Q1190554. # with a point in time or start date OPTIONAL { ?event wdt:P585 ?date. } OPTIONAL { ?event wdt:P580 ?date. } # but at least one of those FILTER(BOUND(?date) && DATATYPE(?date) = xsd:dateTime). # not in the future, and

not more than 31 days ago BIND(NOW() - ?date AS ?distance). FILTER(0 <= ?distance && ?distance < 31). # and get a label as well OPTIONAL { ?event rdfs:label ?eventLabel. FILTER(LANG(?eventLabel) = "en"). } } #limit to 10 results so we don't timeout LIMIT 10

[{"date":"2020-12-03T00:00:00Z","event":"http://www.wikidata.org/entity/Q61439180","eventLabel":"2019–20 Biathlon World Cup – Stage 8"},{"date":"2020-12-09T00:00:00Z","event":"http://www.wikidata.org/entity/Q6
12-09T00:00:00Z","event":"http://www.wikidata.org/entity/Q65204057"},{"date":"2020-12-09T00:00:00Z","event":"http://www.wikidata.org/entity/Q65272745","eventLabel":"True Beauty"},{"date":"2020-12-09T00:00:00Z","event":"http://www.wikidata.org/entity/Q65486271"},{"date":"2020-
12-09T00:00:00Z","event":"http://www.wikidata.org/entity/Q65486272"},{"date":"2020-12-04T00:00:00Z","event":"http://www.wikidata.org/entity/Q68249892","eventLabel":"Selena: The Series"},{"date":"2020-12-06T00:00:00Z","event":"http://www.wikidata.org/entity/Q76733622","eventLabel":"T
fourth Conference on Neural Information Processing Systems"},{"date":"2020-12-13T00:00:00Z","event":"http://www.wikidata.org/entity/Q79768715","eventLabel":"2020 European Cross Country Championships"},{"date":"2020-11-23T00:00:00Z","event":"http://www.wikidata.org/entity/
Narcissus"}]