Assignment 3

June 29, 2021

1 Assignment 3

Import libraries and define common helper functions

[4]:
```
import os
import sys
import gzip
import json
from pathlib import Path
import csv

import pandas as pd
import s3fs
import pyarrow as pa
from pyarrow.json import read_json
import pyarrow.parquet as pq
import fastavro
import pygeohash
import snappy
import jsonschema
from jsonschema.exceptions import ValidationError


endpoint_url='https://storage.budsc.midwest-datascience.com'

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


def read_jsonl_data():
    s3 = s3fs.S3FileSystem(
        anon=True,
        client_kwargs={
            'endpoint_url': endpoint_url
        }
    )
```

```
    src_data_path = 'data/processed/openflights/routes.jsonl.gz'
    with s3.open(src_data_path, 'rb') as f_gz:
        with gzip.open(f_gz, 'rb') as f:
            records = [json.loads(line) for line in f.readlines()]


    return records
```

Load the records from https://storage.budsc.midwest-datascience.com/data/processed/openflights/routes.jsonl.gz

[5]:
```
records = read_jsonl_data()
```

[13]:
```
print(json.dumps(records[10], indent =4))
```


```
{
    "airline": {
        "airline_id": 410,
        "name": "Aerocondor",
        "alias": "ANA All Nippon Airways",
        "iata": "2B",
        "icao": "ARD",
        "callsign": "AEROCONDOR",
        "country": "Portugal",
        "active": true
    },
    "src_airport": {
        "airport_id": 6156,
        "name": "Belgorod International Airport",
        "city": "Belgorod",
        "country": "Russia",
        "iata": "EGO",
        "icao": "UUOB",
        "latitude": 50.643798828125,
        "longitude": 36.5900993347168,
        "altitude": 735,
        "timezone": 3.0,
        "dst": "N",
        "tz_id": "Europe/Moscow",
        "type": "airport",
        "source": "OurAirports"
    },
    "dst_airport": {
        "airport_id": 2990,
        "name": "Kazan International Airport",
        "city": "Kazan",
        "country": "Russia",
        "iata": "KZN",
        "icao": "UWKD",
```

```
        "latitude": 55.606201171875,
        "longitude": 49.278701782227,
        "altitude": 411,
        "timezone": 3.0,
        "dst": "N",
        "tz_id": "Europe/Moscow",
        "type": "airport",
        "source": "OurAirports"
    },
    "codeshare": false,
    "equipment": [
        "CR2"
    ]
}
```

1.1 3.1

1.1.1 3.1.a JSON Schema

[24]:
```
def validate_jsonl_data(records):
    schema_path = schema_dir.joinpath('routes-schema.json')
    with open(schema_path) as f:
        schema = json.load(f)

    validation_csv_path = results_dir.joinpath('json_schema_validation_results.
↪csv')
    with open(validation_csv_path, 'w') as f:

        columnnames = ['row_num', 'record', 'is_valid']
        csv_writer = csv.DictWriter(f,fieldnames=columnnames, lineterminator␣
↪='\n')
        csv_writer.writeheader()
        for i, record in enumerate(records):
            try:
                ## TODO: Validate record
                #pass
                jsonschema.validate(instance=record, schema = schema)
                #result = dict(row_num = i, is_valid = True)
            except ValidationError as e:
                ## Print message if invalid record
                #pass

                result = dict(row_num = i, record = record, is_valid = False)
                csv_writer.writerow(result)
                #pass
            #finally:
```

```
validate_jsonl_data(records)
```

[46]:
```
pd.read_csv("results/json_schema_validation_results.csv")['is_valid'].
↪value_counts()
```

[46]:
```
False    892
Name: is_valid, dtype: int64
```

1.1.2 3.1.b Avro

[38]:
```
def create_avro_dataset(records):
    schema_path = schema_dir.joinpath('routes.avsc')
    data_path = results_dir.joinpath('routes.avro')
    ## TODO: Use fastavro to create Avro dataset

    with open(schema_path,'r') as f:
        schema = json.load(f)

    parsed_schema = fastavro.parse_schema(schema)

    #, encoding="utf-8"
    with open(data_path, 'wb') as out:
        fastavro.writer(out, parsed_schema , records)


create_avro_dataset(records)
```

1.1.3 3.1.c Parquet

[44]:
```
def create_parquet_dataset():
    src_data_path = 'data/processed/openflights/routes.jsonl.gz'
    parquet_output_path = results_dir.joinpath('routes.parquet')
    s3 = s3fs.S3FileSystem(
        anon=True,
        client_kwargs={
            'endpoint_url': endpoint_url
        }
    )

    with s3.open(src_data_path, 'rb') as f_gz:
        with gzip.open(f_gz, 'rb') as f:
            #pass
            ## TODO: Use Apache Arrow to create Parquet table and save the␣
↪dataset
```

```
            parquet_table = read_json(f)
            #print(parquet_table)
            pq.write_table(parquet_table , parquet_output_path)


create_parquet_dataset()
```

1.1.4 3.1.d Protocol Buffers

[15]:
```
sys.path.insert(0, os.path.abspath('routes_pb2'))

import routes_pb2
import google.protobuf.json_format as pbjf
#from proto import apiai_pb2

def _airport_to_proto_obj(airport):
    obj = routes_pb2.Airport()
    if airport is None:
        # Returning None is raising a TypeError, so returning empty object
        return None
#         return obj
    if airport.get('airport_id') is None:
        # Returning None is raising a TypeError, so returning empty object
        return None
#         return obj

    obj.airport_id = airport.get('airport_id')
    if airport.get('name'):
        obj.name = airport.get('name')
    if airport.get('city'):
        obj.city = airport.get('city')
    if airport.get('iata'):
        obj.iata = airport.get('iata')
    if airport.get('icao'):
        obj.icao = airport.get('icao')
    if airport.get('altitude'):
        obj.altitude = airport.get('altitude')
    if airport.get('timezone'):
        obj.timezone = airport.get('timezone')
    if airport.get('dst'):
        obj.dst = airport.get('dst')
    if airport.get('tz_id'):
        obj.tz_id = airport.get('tz_id')
    if airport.get('type'):
        obj.type = airport.get('type')
    if airport.get('source'):
        obj.source = airport.get('source')
```

```
    obj.latitude = airport.get('latitude')
    obj.longitude = airport.get('longitude')

    return obj


def _airline_to_proto_obj(airline):
    obj = routes_pb2.Airline()
    if not airline.get('name') is None:
        return None
    if not airline.get('airline_id') is None:
        return None
    obj.airline_id = airline.get('airline_id')
    obj.name = airline.get('name')
    if airline.get('alias'):
        obj.alias = airline.get('alias')
    ## TODO: Create an Airline obj using Protocol Buffers API
    """
    Added by Tinto
    """
    if airline.get('iata'):
        obj.iata = airline.get('iata')
    if airline.get('icao'):
        obj.icao = airline.get('icao')
    if airline.get('callsign'):
        obj.callsign = airline.get('callsign')
    if airline.get('country'):
        obj.country = airline.get('country')
    if airline.get('active'):
        obj.active = airline.get('active')
    """
    Added by Tinto end
    """
    return obj


def create_protobuf_dataset(records):
    routes = routes_pb2.Routes()
    for record in records:
        route = routes_pb2.Route()

        airline = _airline_to_proto_obj(record.get('airline', {}))
        if airline:
            route.airline.CopyFrom(airline)
        src_airport = _airport_to_proto_obj(record.get('src_airport', {}))
        ## TODO
```

```
        if src_airport:
            route.src_airport.CopyFrom(src_airport)

        dst_airport = _airport_to_proto_obj(record.get('dst_airport', {}))
        if dst_airport:
            route.dst_airport.CopyFrom(dst_airport)

        route.codeshare = record['codeshare']
#         route.equipment[:] = record['equipment']

        stops = _airport_to_proto_obj(record.get('stops', {}))
        if stops:
            route.dst_airport.CopyFrom(stops)

        equipment = record.get('equipment')

        routes.route.append(route)

    data_path = results_dir.joinpath('routes.pb')

    with open(data_path, 'wb') as f:
        f.write(routes.SerializeToString())

    compressed_path = results_dir.joinpath('routes.pb.snappy')

    with open(compressed_path, 'wb') as f:
        f.write(snappy.compress(routes.SerializeToString()))

create_protobuf_dataset(records)
```

1.2 3.2

1.2.1 3.2.a Simple Geohash Index

[68]:
```
def create_hash_dirs(records):
    geoindex_dir = results_dir.joinpath('geoindex')
    geoindex_dir.mkdir(exist_ok=True, parents=True)
    hashes = []
    ## TODO: Create hash index
    for record in records:
        src_airport = record.get('src_airport', {})
        if src_airport:
            latitude = src_airport.get('latitude')
            longitude = src_airport.get('longitude')
            if latitude and longitude:
#                 hashes.append(pygeohash.encode(latitude, longitude, precision␣
 ↪= 3))
```

```
                geohash = pygeohash.encode(latitude, longitude)
                record["geohash"] = geohash
#                 geohash = record.set('geohash')
                hashes.append(geohash)
#
    hashes.sort()
#     print(hashes)
    three_letter = sorted(list(set([entry[:3] for entry in hashes])))
#     print(three_letter)
    hash_index = {value: [] for value in three_letter}
    for record in records:
        geohash = record.get('geohash')
        if geohash:
            hash_index[geohash[:3]].append(record)
#     print(hash_index)
    for key, values in hash_index.items():
        output_dir = geoindex_dir.joinpath(str(key[:1])).joinpath(str(key[:2]))
        output_dir.mkdir(exist_ok=True, parents=True)
        output_path = output_dir.joinpath('{}.jsonl.gz'.format(key))
        with gzip.open(output_path, 'w') as f:
            json_output = '\n'.join([json.dumps(value) for value in values])
            f.write(json_output.encode('utf-8'))

create_hash_dirs(records)
```

[73]:
```
"""
Test to see if the data is getting generated with the files in the index
"""

def read_idx_data(src_data_path1):
#     src_data_path1 = 'results/geoindex/9/9z/9z7.jsonl.gz'
#     with open(src_data_path, 'rb') as f_gz:
    with gzip.open(src_data_path1, 'rb') as f:
        idx_records = [json.loads(line) for line in f.readlines()]


    return idx_records
```

[74]:
```
idx_records = read_idx_data('results/geoindex/9/9z/9z7.jsonl.gz')
print(json.dumps(idx_records[0]))
```

```
{"airline": {"airline_id": 24, "name": "American Airlines", "alias": "\\N",
"iata": "AA", "icao": "AAL", "callsign": "AMERICAN", "country": "United States",
"active": true}, "src_airport": {"airport_id": 3454, "name": "Eppley Airfield",
"city": "Omaha", "country": "United States", "iata": "OMA", "icao": "KOMA",
"latitude": 41.3032, "longitude": -95.89409599999999, "altitude": 984,
"timezone": -6.0, "dst": "A", "tz_id": "America/Chicago", "type": "airport",
```

"source": "OurAirports"}, "dst_airport": {"airport_id": 3876, "name": "Charlotte Douglas International Airport", "city": "Charlotte", "country": "United States", "iata": "CLT", "icao": "KCLT", "latitude": 35.2140007019043, "longitude": -80.94309997558594, "altitude": 748, "timezone": -5.0, "dst": "A", "tz_id": "America/New_York", "type": "airport", "source": "OurAirports"}, "codeshare": true, "equipment": ["CR9"], "geohash": "9z7fczh09de3"}

1.2.2 3.2.b Simple Search Feature

[95]:
```
def airport_search(latitude, longitude):
    ## TODO: Create simple search to return nearest airport
#     pass
    geohash1 = pygeohash.encode(latitude, longitude)
#     print(geohash)
    geoindex_dir = results_dir.joinpath('geoindex')
    search_dir = geoindex_dir.joinpath(str(geohash1[:1])).
 ↪joinpath(str(geohash1[:2]))
#     print(search_dir)
    search_file = search_dir.joinpath('{}.jsonl.gz'.format(geohash1[:3]))
#     print(search_file)

    idx_records = read_idx_data(search_file)
#     print(json.dumps(idx_records[0]))
    for record in idx_records:
        src_airport = record.get('src_airport', {})
        geohash = record.get('geohash')

#         print(json.dumps(src_airport))
        if src_airport:
            airport_id = src_airport.get('airport_id')
            airport_name = src_airport.get('name')
#             print(airport_name)
            if geohash:
#                 print(geohash)
                if (pygeohash.geohash_approximate_distance(geohash, geohash1)/
 ↪1000) < 100:
                    airport_id = airport_id
                    airport_name = airport_name

    print(airport_id)
    print(airport_name)


airport_search(41.1499988, -95.91779)
```

```
3454
Eppley Airfield
```

[ ]:

[ ]: