Assignment 3

June 26, 2021

1 Assignment 3

Import libraries and define common helper functions

[15]:
```python
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

[16]:
```
records = read_jsonl_data()
print(json.dumps(records[0], sort_keys=False, 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": 2965,
        "name": "Sochi International Airport",
        "city": "Sochi",
        "country": "Russia",
        "iata": "AER",
        "icao": "URSS",
        "latitude": 43.449902,
        "longitude": 39.9566,
        "altitude": 89,
        "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

[17]:
```python
def validate_jsonl_data(records):
    schema_path = schema_dir.joinpath('routes-schema.json')
    with open(schema_path) as f:
        _schema = json.load(f)

    print(_schema)

    validation_csv_path = results_dir.joinpath('validation-results.csv')
    with open(validation_csv_path, 'w') as f:
        for i, record in enumerate(records):
            try:
                ## TODO: Validate record
                jsonschema.validate(record, _schema)
                ##pass
            except ValidationError as e:
                ## Print message if invalid record
                detail = e.message
                print(detail)
                f.write(str(detail))
                return detail

validate_jsonl_data(records)
```

```
{'$schema': 'http://json-schema.org/draft-04/schema#', 'type': 'object',
'properties': {'airline': {'type': 'object', 'properties': {'airline_id':
{'type': 'integer'}, 'name': {'type': 'string'}, 'alias': {'type': 'string'},
'iata': {'type': 'string'}, 'icao': {'type': 'string'}, 'callsign': {'type':
'string'}, 'country': {'type': 'string'}, 'active': {'type': 'boolean'}},
'required': ['airline_id', 'name', 'alias', 'iata', 'icao', 'callsign',
'country', 'active']}, 'src_airport': {'type': 'object', 'properties':
{'airport_id': {'type': 'integer'}, 'name': {'type': 'string'}, 'city': {'type':
'string'}, 'country': {'type': 'string'}, 'iata': {'type': 'string'}, 'icao':
{'type': 'string'}, 'latitude': {'type': 'number'}, 'longitude': {'type':
'number'}, 'altitude': {'type': 'integer'}, 'timezone': {'type': 'number'},
'dst': {'type': 'string'}, 'tz_id': {'type': 'string'}, 'type': {'type':
'string'}, 'source': {'type': 'string'}}, 'required': ['airport_id', 'name',
'city', 'country', 'iata', 'icao', 'latitude', 'longitude', 'altitude',
'timezone', 'dst', 'tz_id', 'type', 'source']}, 'dst_airport': {'type':
'object', 'properties': {'airport_id': {'type': 'integer'}, 'name': {'type':
'string'}, 'city': {'type': 'string'}, 'country': {'type': 'string'}, 'iata':
{'type': 'string'}, 'icao': {'type': 'string'}, 'latitude': {'type': 'number'},
'longitude': {'type': 'number'}, 'altitude': {'type': 'integer'}, 'timezone':
{'type': 'number'}, 'dst': {'type': 'string'}, 'tz_id': {'type': 'string'},
'type': {'type': 'string'}, 'source': {'type': 'string'}}, 'required':
['airport_id', 'name', 'city', 'country', 'iata', 'icao', 'latitude',
'longitude', 'altitude', 'timezone', 'dst', 'tz_id', 'type', 'source']},
'codeshare': {'type': 'boolean'}, 'equipment': {'type': 'array', 'items':
[{'type': 'string'}]}}, 'required': ['airline', 'src_airport', 'dst_airport',
'codeshare', 'equipment']}
None is not of type 'object'
```

[17]:
```
"None is not of type 'object'"
```

1.1.2 3.1.b Avro

[18]:
```python
from fastavro import writer, parse_schema
from fastavro.schema import load_schema
def create_avro_dataset(records):
    schema_path = schema_dir.joinpath('routes.avsc')
    print(schema_path)
    data_path = results_dir.joinpath('routes.avro')
    print(data_path)
    ## TODO: Use fastavro to create Avro dataset
    parsed_schema = load_schema(schema_path)
    with open(data_path, 'wb') as out:
        writer(out, parsed_schema, records)
```

[19]:
```python
create_avro_dataset(records)
```

```
/home/jovyan/dsc650/dsc650/assignments/assignment03/schemas/routes.avsc
/home/jovyan/dsc650/dsc650/assignments/assignment03/results/routes.avro
```

1.1.3 3.1.c Parquet

[20]:
```python
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
            records= [json.loads(line) for line in f.readlines()]
            df= pd.DataFrame(records)
            table= pa.Table.from_pandas(df)
            pq.write_table(table, parquet_output_path)
            return parquet_output_path



create_avro_dataset(records)
```

```
/home/jovyan/dsc650/dsc650/assignments/assignment03/schemas/routes.avsc
/home/jovyan/dsc650/dsc650/assignments/assignment03/results/routes.avro
```

[21]:
```python
create_parquet_dataset()
```

[21]:
```
PosixPath('/home/jovyan/dsc650/dsc650/assignments/assignment03/results/routes.pa
rquet')
```

1.1.4 3.1.d Protocol Buffers

[47]:
```python
sys.path.insert(0, os.path.abspath('routes_pb2'))

import routes_pb2

def _airport_to_proto_obj(airport):
    obj = routes_pb2.Airport()
    if airport is None:
        return None
    if airport.get('airport_id') is None:
        return None
    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')
    if airport.get('active'):
        obj.active = airline.get('active')

    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'):
        return None
    if not airline.get('airline_id'):
        return None
    obj.airline_id = airline.get('airline_id')
    obj.name = airline.get('name')
    obj.active = airline.get('active')
    if airline.get('alias'):
        obj.alias = airline.get('alias')

    ## TODO
    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)



        if _airport_to_proto_obj(record['src_airport']) is not None:
            src_airport = _airport_to_proto_obj(record.get('src_aiport', {}))
            route.airline.CopyFrom(airline)

        if _airport_to_proto_obj(record['dst_airport']) is not None:
            dst_airport = _airport_to_proto_obj(record.get('dst_aiport', {}))
            route.airline.CopyFrom(airline)

        route.codeshare = record['codeshare']

        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')


create_protobuf_dataset(records)
```

1.1.5 3.2

1.1.6 3.2.a Simple Geohash Index

[56]:
```
def create_hash_dirs(records):
    geoindex_dir = results_dir.joinpath('geoindex')
    geoindex_dir.mkdir(exist_ok=True, parents=True)
    hashes = []
    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:
                pygeohash.encode(latitude, longitude)
    hashes.sort()
    three_letter = sorted(list(set([entry[:3] for entry in hashes])))
    hash_index = {value: [] for value in three_letter}
    for record in records:
        geohash = record.get('geohash')
        if geohash:
            hash_index[geohash[:3]].append(record)
    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'))
```

[57]:
```
create_hash_dirs(records)
```

1.1.7 3.2.b Simple Search Feature

[59]:
```
from iteration_utilities import unique_everseen

def airport_search(latitude, longitude, distm):
    ## TODO: Create simple search to return nearest airport
    distm = distm / 1000
    srcgeoval = pygeohash.encode(latitude, longitude, precision=3)
    AirportDistances = []
    airrecord = []

    for record in records:
        for key, value in record.items():
            if key == 'src_airport' and value is not None:
                if value not in airrecord:
                    airrecord.append(value)


    for record in airrecord:
        dstname = record['name']
        dstlat = record['latitude']
        dstlong = record['longitude']
        geohval = pygeohash.encode(dstlat, dstlong, precision=3)
        distm_dstgeo1 = pygeohash.geohash_approximate_distance(srcgeoval,␣
 ↪geohval) / 1000
        airport_dist = {
            "Airport Name": dstname,
            "Latitude": dstlat,
            "Longitude": dstlong,
            "Distance(m)": distm_dstgeo1
        }
        AirportDistances.append(airport_dist)

    DistList = list(unique_everseen(AirportDistances))
    print("Meters From Location: "+str(distm))
    print("LAT: "+str(latitude))
    print("LONG: "+str(longitude))
    for i in range(len(DistList)):
        for a, b in DistList[i].items():
            if a == 'Distance(m)':
                if b < distm:
                    print(DistList[i])
    ##pass

distm = 625441
airport_search(41.1499988, -95.91779, distm)
```

```
Meters From Location: 625.441
LAT: 41.1499988
LONG: -95.91779
{'Airport Name': 'Eppley Airfield', 'Latitude': 41.3032, 'Longitude':
-95.89409599999999, 'Distance(m)': 123.264}
{'Airport Name': 'Lincoln Airport', 'Latitude': 40.85100173950195, 'Longitude':
-96.75920104980469, 'Distance(m)': 123.264}
```

```
[ ]:
```