# Assignment 3

October 5, 2020

## 1 Assignment 3

Import libraries and define common helper functions

[1]:
```
import os
import errno
import sys
import gzip
import json
from pathlib import Path
import csv
import shutil
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
import math


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

[2]:
```
records = read_jsonl_data()
```

## 1.1 3.1

### 1.1.1 3.1.a JSON Schema

[3]:
```
def validate_jsonl_data(records):
    schema_path = schema_dir.joinpath('routes-schema.json')
    validation_csv_path = results_dir.joinpath('validation-results.csv')
    with open(schema_path) as f:
        schema = json.load(f)
    with open(validation_csv_path, 'w') as f:
        fieldnames = ['row_num', 'is_valid', 'msg']
        csv_writer = csv.DictWriter(f, fieldnames=fieldnames)
        csv_writer.writeheader()
        for i, record in enumerate(records):
            try:
                ## TODO
                # The JSON conforms to the schema
                jsonschema.validate(record, schema=schema)
                result = dict(
                    ## TODO
                    row_num = i,
                    is_valid = True,
                    msg = record
                )
            except ValidationError as e:
                # The JSON does not conform to the schema
                result = dict(
                    ## TODO
                    row_num = i,
                    is_valid = False,
                    msg = record
                )
                #print(record)
            finally:
                csv_writer.writerow(result)
validate_jsonl_data(records)
```

### 1.1.2 3.1.b Avro

[4]:
```
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) as f:
        # Make sure that schema is correct
        schema = json.load(f)
    with open(data_path, 'wb') as out:
        fastavro.writer(out, schema, records)


create_avro_dataset(records)
```

### 1.1.3 3.1.c Parquet

[5]:
```
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:
            ## TODO: Use Apache Arrow to create Parquet table and save the␣
 ↪dataset
            table = pa.json.read_json(f)
            pq.write_table(table, parquet_output_path)

create_parquet_dataset()
```

### 1.1.4 3.1.d Protocol Buffers

[6]:
```
sys.path.insert(0, os.path.abspath('routes_pb2'))

import routes_pb2
```

```python
def _airport_to_proto_obj(airport):
    obj = routes_pb2.Airport()
    if airport is None:
        return None
        # return obj
    if airport.get('airport_id') is None:
        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):

    # This part was so frustrating
    obj = routes_pb2.Airline()
    ## TODO: Create an Airline obj using Protocol Buffers API
    if airline is None:
        # returning None was throwing a type error
        # return None
        return obj
    if airline.get('airline_id') is None:
        # return None
        return obj

    # Pull some values from the json
    obj.airline_id = airline.get('airline_id')
    if airline.get('name'):
        obj.name = airline.get('name')
    if airline.get('alias'):
        obj.alias = airline.get('alias')
    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')

    return obj


def create_protobuf_dataset(records):
    routes = routes_pb2.Routes()
    for record in records:
        route = routes_pb2.Route()
        ## TODO: Implement the code to create the Protocol Buffers Dataset
        src_airport = _airport_to_proto_obj(record.get('src_airport'))
        dst_airport = _airport_to_proto_obj(record.get('dst_airport'))
        airline = _airline_to_proto_obj(record.get('airline'))
        codeshare = record.get('codeshare')
        equipment = record.get('equipment')
        if not src_airport == None:
            route.src_airport.CopyFrom(src_airport)
        else:
            # Required values
            route.src_airport.airport_id = 0
            route.src_airport.latitude = 0
            route.src_airport.longitude = 0

        if not dst_airport == None:
            route.dst_airport.CopyFrom(dst_airport)
        else:
            # Required values
            route.dst_airport.airport_id = 0
            route.dst_airport.latitude = 0
            route.dst_airport.longitude = 0

        if not airline == None:
            route.airline.CopyFrom(airline)
        else:
            # Required values
            route.airline.airline_id = 0
            route.airline.name = None
            route.airline.active = False

        route.codeshare = codeshare
        route.equipment.extend(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

[7]:
```
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:
        try:
            src_lat = record.get('src_airport').get('latitude')
            src_long = record.get('src_airport').get('longitude')
        except:
            # Lat and long can't be none, so "null island" it is
            src_lat = 0
            src_long = 0

        location_hash = pygeohash.encode(src_lat, src_long)
        hashes.append(dict(hash = location_hash,record = record))

    for hashed_location in hashes:
        geo_hash = hashed_location.get('hash')
        dir_name = results_dir.joinpath('geoindex').joinpath(geo_hash[0]).
↪joinpath(geo_hash[1])
        file_name = geo_hash[0:3] + '.jsonl'
        path_name = os.path.join(dir_name, file_name)
        if not os.path.exists(os.path.dirname(path_name)):
            try:
                # Make the directory if not exists
                os.makedirs(os.path.dirname(path_name))
            except OSError as exc: # Guard against race condition
                if exc.errno != errno.EEXIST:
                    raise
        with open(path_name, "a") as f: # Append mode
            # save the record to a .jsonl file.
            f.write(json.dumps(hashed_location.get('record')) + '\n')

    # Zip up all the .jsonl files we just made
    # It is probably more efficient to write directly to .gz files as bytes,␣
↪but this works too
    for root, dirs, files in os.walk(geoindex_dir):
        for file in files:
            fname = os.path.join(root,file)
            with open(fname, 'rb') as f_in:
                with gzip.open(fname + '.gz', 'wb') as f_out:
                    # Make a zipped copy
                    shutil.copyfileobj(f_in, f_out)
            # Delete original .jsonl file
            os.remove(fname)


create_hash_dirs(records)
```

### 1.2.2 3.2.b Simple Search Feature

[8]:
```
def airport_search(latitude, longitude):

    ## TODO: Create simple search to return nearest airport
    search_hash = pygeohash.encode(latitude, longitude)
    search_dir = results_dir.joinpath('geoindex').joinpath(search_hash[0]).
↪joinpath(search_hash[1])
    search_file = search_hash[0:3] + '.jsonl.gz'
    search_path = os.path.join(search_dir, search_file)
    # No error checking to see if the file does not exist
    # In this case it does, so lets keep it "simple"
    with open(search_path, 'rb') as f_gz:
        with gzip.open(f_gz, 'rb') as f:
            closest_airport_distance = math.inf
            for line in f:
                route = json.loads(line.decode("utf-8"))
                lookup_lat = route['src_airport']['latitude']
                lookup_long = route['src_airport']['longitude']
                # How far apart are the two geohashes?
                dist = pygeohash.geohash_approximate_distance(search_hash,␣
↪pygeohash.encode(lookup_lat,lookup_long))
                # Save the closest for later, if we have a new closest
                if dist < closest_airport_distance:
                    closest_airport = route
    # A print statement works here, but a return statement would work better
    print(json.dumps(closest_airport['src_airport']))


airport_search(41.1499988, -95.91779)
```

```
{"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"}
```

[ ]: