Assignment 3

April 17, 2023

1 Assignment 3

Import libraries and define common helper functions

[1]:
```
pip install fastavro==1.5.1
```

```
Requirement already satisfied: fastavro==1.5.1 in
/opt/conda/lib/python3.10/site-packages (1.5.1)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
pip install iteration_utilities
```

```
Requirement already satisfied: iteration_utilities in
/opt/conda/lib/python3.10/site-packages (0.11.0)
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```
pip install avro
```

```
Requirement already satisfied: avro in /opt/conda/lib/python3.10/site-packages
(1.11.1)
Note: you may need to restart the kernel to use updated packages.
```

[4]:
```
from iteration_utilities import unique_everseen
import os
import sys
import gzip
import json
from pathlib import Path
import csv
import avro.schema
import fastavro
import pandas as pd
import s3fs
# from fs_s3fs import S3FS
import pyarrow as pa
from pyarrow.json import read_json
import pyarrow.parquet as pq
#import fastavro
```

```
from fastavro.schema import load_schema
from fastavro import writer, reader, parse_schema
#import pygeohashpisr d
import pygeohash
import snappy
import jsonschema
from jsonschema import validate

from jsonschema.exceptions import ValidationError


endpoint_url='/home/jovyan/'

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)

print(current_dir)
print(schema_dir)
print(results_dir)

def read_jsonl_data():
    f_gz = '/home/jovyan/assignment03_Vayuvegula_SomaShekar/data/processed/
 ↪openflights/routes.jsonl.gz'
    with gzip.open(f_gz, 'rb') as f:
        records = [json.loads(line) for line in f.readlines()]

    return records
```

```
/home/jovyan/assignment03_Vayuvegula_SomaShekar
/home/jovyan/assignment03_Vayuvegula_SomaShekar/schemas
/home/jovyan/assignment03_Vayuvegula_SomaShekar/results
```

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

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

1.1 3.1

1.1.1 3.1.a JSON Schema

[6]:
```
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('validation-results.csv')
    # Open the validation CSV
```

```
    with open(validation_csv_path, 'w', encoding="utf-8") as f:
        # Create column names
        fieldnames = ['row', 'valid', 'msg']
        # Assign CSV writer object
        csv_writer = csv.DictWriter(f, fieldnames=fieldnames, lineterminator =␣
↪'\n')
        csv_writer.writeheader()
        for i, record in enumerate(records):
            try:
                ## TODO: Validate record
                result = dict(
                    row = i,
                    valid = True,
                    msg = record
                )
                pass
            except ValidationError as e:
                ## Print message if invalid record
                result = dict(
                    row = i,
                    valid = False,
                    msg = record
                )
                pass
            finally:
                # Write the line to the CSV.
                csv_writer.writerow(result)

validate_jsonl_data(records)
```

1.1.2 3.1.b Avro

[7]:
```
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:
        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

[8]:
```
def create_parquet_dataset():
    src_data_path = '/home/jovyan/assignment03_Vayuvegula_SomaShekar/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 gzip.open(src_data_path, 'rb') as f:
        records = [json.loads(line) for line in f.readlines()]
    print(records)
    df = pd.DataFrame(records)
    ## TODO: Use Apache Arrow to create Parquet table and save the dataset
    table = pa.Table.from_pandas(df)
    print(table)
    pq.write_table(table, parquet_output_path, compression='none')

create_parquet_dataset()
```

```
IOPub data rate exceeded.
The Jupyter server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--ServerApp.iopub_data_rate_limit`.

Current values:
ServerApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
ServerApp.rate_limit_window=3.0 (secs)


pyarrow.Table
airline: struct<active: bool, airline_id: int64, alias: string, callsign:
string, country: string, iata: string, icao: string, name: string>
  child 0, active: bool
  child 1, airline_id: int64
  child 2, alias: string
  child 3, callsign: string
  child 4, country: string
  child 5, iata: string
  child 6, icao: string
  child 7, name: string
```

```
src_airport: struct<airport_id: int64, altitude: int64, city: string, country:
string, dst: string, iata: string, icao: string, latitude: double, longitude:
double, name: string, source: string, timezone: double, type: string, tz_id:
string>
  child 0, airport_id: int64
  child 1, altitude: int64
  child 2, city: string
  child 3, country: string
  child 4, dst: string
  child 5, iata: string
  child 6, icao: string
  child 7, latitude: double
  child 8, longitude: double
  child 9, name: string
  child 10, source: string
  child 11, timezone: double
  child 12, type: string
  child 13, tz_id: string
dst_airport: struct<airport_id: int64, altitude: int64, city: string, country:
string, dst: string, iata: string, icao: string, latitude: double, longitude:
double, name: string, source: string, timezone: double, type: string, tz_id:
string>
  child 0, airport_id: int64
  child 1, altitude: int64
  child 2, city: string
  child 3, country: string
  child 4, dst: string
  child 5, iata: string
  child 6, icao: string
  child 7, latitude: double
  child 8, longitude: double
  child 9, name: string
  child 10, source: string
  child 11, timezone: double
  child 12, type: string
  child 13, tz_id: string
codeshare: bool
equipment: list<item: string>
  child 0, item: string
----
airline: [
  -- is_valid: all not null
  -- child 0 type: bool
[true,true,true,true,true,…,true,true,true,true,true]
  -- child 1 type: int64
[410,410,410,410,410,…,4178,19016,19016,19016,19016]
  -- child 2 type: string
["ANA All Nippon Airways","ANA All Nippon Airways","ANA All Nippon Airways","ANA
```

```
All Nippon Airways","ANA All Nippon Airways",…,"Qantas
Airways","Apache","Apache","Apache","Apache"]
  -- child 3 type: string
["AEROCONDOR","AEROCONDOR","AEROCONDOR","AEROCONDOR","AEROCONDOR",…,"REX","APA
CHE","APACHE","APACHE","APACHE"]
  -- child 4 type: string
["Portugal","Portugal","Portugal","Portugal","Portugal",…,"Australia","United
States","United States","United States","United States"]
  -- child 5 type: string
["2B","2B","2B","2B","2B",…,"ZL","ZM","ZM","ZM","ZM"]
  -- child 6 type: string
["ARD","ARD","ARD","ARD","ARD",…,"RXA","IWA","IWA","IWA","IWA"]
  -- child 7 type: string
["Aerocondor","Aerocondor","Aerocondor","Aerocondor","Aerocondor",…,"Regional
Express","Apache Air","Apache Air","Apache Air","Apache Air"]]
src_airport: [
  -- is_valid:  [true,true,true,true,true,…,true,true,true,true,true]
  -- child 0 type: int64
[2965,2966,2966,2968,2968,…,6334,4029,2912,2912,2913]
  -- child 1 type: int64
[89,-65,-65,769,769,…,41,588,2058,2058,2927]
  -- child 2 type: string
["Sochi","Astrakhan","Astrakhan","Chelyabinsk","Chelyabinsk",…,"Whyalla","Mosc
ow","Bishkek","Bishkek","Osh"]
  -- child 3 type: string
["Russia","Russia","Russia","Russia","Russia",…,"Australia","Russia","Kyrgyzst
an","Kyrgyzstan","Kyrgyzstan"]
  -- child 4 type: string
["N","N","N","N","N",…,"O","N","U","U","U"]
  -- child 5 type: string
["AER","ASF","ASF","CEK","CEK",…,"WYA","DME","FRU","FRU","OSS"]
  -- child 6 type: string
["URSS","URWA","URWA","USCC","USCC",…,"YWHA","UUDD","UAFM","UAFM","UAFO"]
  -- child 7 type: double
[43.449902,46.2832984924,46.2832984924,55.305801,55.305801,…,-
33.05889892578125,55.40879821777344,43.0612983704,43.0612983704,40.6090011597]
  -- child 8 type: double
[39.9566,48.0063018799,48.0063018799,61.5033,61.5033,…,137.51400756835938,37.9
0629959106445,74.4776000977,74.4776000977,72.793296814]
  -- child 9 type: string
["Sochi International Airport","Astrakhan Airport","Astrakhan
Airport","Chelyabinsk Balandino Airport","Chelyabinsk Balandino
Airport",…,"Whyalla Airport","Domodedovo International Airport","Manas
International Airport","Manas International Airport","Osh Airport"]
  -- child 10 type: string
["OurAirports","OurAirports","OurAirports","OurAirports","OurAirports",…,"OurA
irports","OurAirports","OurAirports","OurAirports","OurAirports"]
  -- child 11 type: double
```

```
[3,4,4,5,5,…,9.5,3,6,6,6]
  -- child 12 type: string
["airport","airport","airport","airport","airport",…,"airport","airport","airp
ort","airport","airport"]
  -- child 13 type: string
["Europe/Moscow","Europe/Samara","Europe/Samara","Asia/Yekaterinburg","Asia/Yeka
terinburg",…,"Australia/Adelaide","Europe/Moscow","Asia/Bishkek","Asia/Bishkek
","Asia/Bishkek"]]
dst_airport: [
  -- is_valid:  [true,true,true,true,true,…,true,true,true,true,true]
  -- child 0 type: int64
[2990,2990,2962,2990,4078,…,3341,2912,4029,2913,2912]
  -- child 1 type: int64
[411,411,1054,411,365,…,20,2058,588,2927,2058]
  -- child 2 type: string
["Kazan","Kazan","Mineralnye
Vody","Kazan","Novosibirsk",…,"Adelaide","Bishkek","Moscow","Osh","Bishkek"]
  -- child 3 type: string
["Russia","Russia","Russia","Russia","Russia",…,"Australia","Kyrgyzstan","Russ
ia","Kyrgyzstan","Kyrgyzstan"]
  -- child 4 type: string
["N","N","N","N","N",…,"O","U","N","U","U"]
  -- child 5 type: string
["KZN","KZN","MRV","KZN","OVB",…,"ADL","FRU","DME","OSS","FRU"]
  -- child 6 type: string
["UWKD","UWKD","URMM","UWKD","UNNT",…,"YPAD","UAFM","UUDD","UAFO","UAFM"]
  -- child 7 type: double
[55.606201171875,55.606201171875,44.22510147094727,55.606201171875,55.0125999450
68,…,-34.945,43.0612983704,55.40879821777344,40.6090011597,43.0612983704]
  -- child 8 type: double
[49.278701782227,49.278701782227,43.08190155029297,49.278701782227,82.6507034301
76,…,138.53100600000002,74.4776000977,37.90629959106445,72.793296814,74.477600
0977]
  -- child 9 type: string
["Kazan International Airport","Kazan International Airport","Mineralnyye Vody
Airport","Kazan International Airport","Tolmachevo Airport",…,"Adelaide
International Airport","Manas International Airport","Domodedovo International
Airport","Osh Airport","Manas International Airport"]
  -- child 10 type: string
["OurAirports","OurAirports","OurAirports","OurAirports","OurAirports",…,"OurA
irports","OurAirports","OurAirports","OurAirports","OurAirports"]
  -- child 11 type: double
[3,3,3,3,7,…,9.5,6,3,6,6]
  -- child 12 type: string
["airport","airport","airport","airport","airport",…,"airport","airport","airp
ort","airport","airport"]
  -- child 13 type: string
["Europe/Moscow","Europe/Moscow","Europe/Moscow","Europe/Moscow","Asia/Krasnoyar
```

```
sk",…,"Australia/Adelaide","Asia/Bishkek","Europe/Moscow","Asia/Bishkek","Asia
/Bishkek"]]
codeshare: [[false,false,false,false,false,…,false,false,false,false,false]]
equipment: [[["CR2"],["CR2"],…,["734"],["734"]]]
```

1.1.4 3.1.d Protocol Buffers

[9]:
```
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')

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

    return obj


def _airline_to_proto_obj(airline):
```

```
    obj = routes_pb2.Airline()
    ## TODO: Create an Airline obj using Protocol Buffers API
    if airline is None:
        return None
    if airline.get('airline_id') is None:
        return None

    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
        for key, value in record.items():
            if key=='airline':
                airline = _airline_to_proto_obj(value)
                airin = route.airline
                airin.name = airline.name
                airin.airline_id = airline.airline_id
                airin.active = airline.active
            if key=='src_airport' and value is not None:
                src_airport = _airport_to_proto_obj(value)
                srcairin = route.src_airport
                srcairin.name = src_airport.name
                srcairin.airport_id = src_airport.airport_id
                srcairin.latitude = src_airport.latitude
                srcairin.longitude = src_airport.longitude

            if key=='dst_airport' and value is not None:
```

```
                dst_airport = _airport_to_proto_obj(value)
                dstairin = route.dst_airport
                dstairin.name = dst_airport.name
                dstairin.airport_id = dst_airport.airport_id
                dstairin.latitude = dst_airport.latitude
                dstairin.longitude = dst_airport.longitude

            if key=='codeshare':
                route.codeshare = value

        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

[10]:
```
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:
                ## TODO: use pygeohash.encode() to assign geohashes to the␣
↪records and complete the hashes list
                hashes.append(pygeohash.
↪encode(longitude=longitude,latitude=latitude,precision=3))
                record['geohash'] = pygeohash.
↪encode(longitude=longitude,latitude=latitude,precision=3)
    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'))

create_hash_dirs(records)
```

1.2.2 3.2.b Simple Search Feature

[11]:
```
def airport_search(latitude, longitude):
    src_geohash_code = pygeohash.
 ↪encode(longitude=longitude,latitude=latitude,precision=15)
    # load the data from that file
    geoindex_dir = results_dir.joinpath('geoindex')
    # building my search area
    my_geo_dir = geoindex_dir.joinpath(str(src_geohash_code[0]))
    my_geo_dir = my_geo_dir.joinpath(str(src_geohash_code[:2]))
    my_geo_dir = my_geo_dir.joinpath(str(src_geohash_code[:3])+".jsonl.gz")

    # print(my_geo_dir)
    with open(my_geo_dir, 'rb') as f_gz:
        with gzip.open(f_gz, 'rb') as f:
            geo_records = [json.loads(line) for line in f.readlines()]
            current_dis = 0
            closest_airport = {}
            for distance in geo_records:
                if distance['src_airport']:
                    temp_lat = distance["src_airport"]["latitude"]
                    temp_lon = distance["src_airport"]["longitude"]
                    dst_geohash_code = pygeohash.
 ↪encode(longitude=temp_lon,latitude=temp_lat,precision=15)
                    dis = pygeohash.
 ↪geohash_approximate_distance(str(src_geohash_code), str(dst_geohash_code))
                    if current_dis == 0 or dis <= current_dis:
                        current_dis = dis
                        closest_airport = distance["airline"]["name"]

            print("The closest airport is:" + closest_airport + ", which is: "+␣
 ↪str(current_dis) + " km away")
```

```
airport_search(41.1499988, -95.91779)
```

```
The closest airport is:Southwest Airlines, which is: 19545 km away
```

[ ]:

[ ]: