Assignment 3

April 24, 2023

1 Assignment 3

Import libraries and define common helper functions

[ ]:
```
import os
import sys
import gzip
import json
from pathlib import Path
import pandas as pd
import s3fs
import pyarrow as pa
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()]

    src_data_path = '../../../data/processed/openflights/routes.jsonl.gz'
    with gzip.open(src_data_path, '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

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

1.1 3.1

1.1.1 3.1.a JSON Schema

[ ]:
```
def create_json_schema(records):
    import genson

    schema = genson.Schema()

    for record in records:
        schema.add_object(record)

    json_schema = schema.to_dict()

    schema_path = schema_dir.joinpath('routes-schema.json')
    with open(schema_path, 'w') as f:
        json.dump(json_schema, f, indent=2)

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/json_validation.md')
    with open(validation_csv_path, 'w') as f:
        f.write("JSONL lines with failed validation\n")
        f.write("----------------------------------\n")
        for i, record in enumerate(records):
            try:
                jsonschema.validate(record, schema)
            except ValidationError as e:
                f.write(f"Failed Entry: {i}\n")

# create_json_schema(records)
```

```
validate_jsonl_data(records)
```

1.1.2 3.1.b Avro

[ ]:
```
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:
        parsed_schema = json.load(f)

    with open(data_path, 'wb') as out:
        fastavro.writer(out, parsed_schema, records)

    # used to test file output
    # with open(data_path, 'rb') as fo:
    #     avro_reader = fastavro.reader(fo)
    #     for record in avro_reader:
    #         print(record)

create_avro_dataset(records)
```

1.1.3 3.1.c Parquet

[ ]:
```
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 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
            record_data = pa.array([json.loads(line) for line in f.readlines()])
    table = pa.Table.from_arrays([record_data], names=['Flight Info'])
    pq.write_table(table, parquet_output_path)

create_parquet_dataset()
```

1.1.4 3.1.d Protocol Buffers

[ ]:
```
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('city'):
        obj.city = airline.get('alias')
    if airline.get('iata'):
        obj.iata = airline.get('iata')
    if airline.get('icao'):
        obj.icao = airline.get('icao')
    if airline.get('altitude'):
        obj.altitude = airline.get('callsign')
    if airline.get('timezone'):
        obj.timezone = airline.get('country')
    if airline.get('active'):
        obj.active = airline.get('active')
    else:
        obj.active = False
    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
        route.airline.CopyFrom(_airline_to_proto_obj(record["airline"]))
        if record.get('src_airport'):
            route.src_airport.
 ↪CopyFrom(_airport_to_proto_obj(record["src_airport"]))
        if record.get('dst_airport'):
            route.dst_airport.
 ↪CopyFrom(_airport_to_proto_obj(record["dst_airport"]))
        # if record.get('codeshare'):
        #     route.codeshare = record["codeshare"]
        if 'codeshare' in record and record['codeshare'] is not None:
            route.codeshare = record['codeshare']
        else:
            route.codeshare = False

        if record.get('stops'):
            route.stops = record["stops"]
        if record.get('equipment'):
            route.equipment.append("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)
```

1.1.5 3.1.e Output Sizes

[ ]:
```
import os
import gzip
import pandas as pd


def get_file_size(file_path):
    """Get the size of a file in bytes"""
    return os.stat(file_path).st_size

def get_gzip_size(filepath):
    with open(filepath, 'rb') as f_in:
        with gzip.open(filepath + ".gz", 'wb') as f_out:
            f_out.write(f_in.read())

    size = os.stat(filepath + ".gz").st_size
    os.remove(filepath + ".gz")
    return size



def get_snappy_size(filepath):
    if not os.path.isfile(filepath +'.snappy'):
        with open(filepath, 'rb') as infile:
            data = infile.read()
            compressed_data = snappy.compress(data)
            with open(filepath +'.snappy', 'wb') as outfile:
                outfile.write(compressed_data)
        size = os.stat(filepath + ".snappy").st_size
        os.remove(filepath + ".snappy")
        return size
    return os.stat(filepath + ".snappy").st_size
```

```
# File paths
avro_file = "results/routes.avro"
pb_file = "results/routes.pb"
json_file = "results/json_validation.md"
parquet_file = "results/routes.parquet"

output_file = "results/comparison.csv"


entries = []
entries.append({"format" : "avro file","uncompressed" :␣
 ↪get_file_size(avro_file),"gzip" : get_gzip_size(avro_file),"snappy" :␣
 ↪get_snappy_size(avro_file)})
entries.append({"format" : "protocol buffer file","uncompressed" :␣
 ↪get_file_size(pb_file),"gzip" : get_gzip_size(pb_file),"snappy" :␣
 ↪get_snappy_size(pb_file)})
entries.append({"format" : "json Schema file","uncompressed" :␣
 ↪get_file_size(json_file),"gzip" : get_gzip_size(json_file),"snappy" :␣
 ↪get_snappy_size(json_file)})
entries.append({"format" : "parquet file","uncompressed" :␣
 ↪get_file_size(parquet_file),"gzip" : get_gzip_size(parquet_file),"snappy" :␣
 ↪get_snappy_size(parquet_file)})
sizes_df = pd.DataFrame(entries)

sizes_df.to_csv(output_file, sep=',')

print("Comparison results saved to:", output_file)
```

```
Comparison results saved to: results/comparison.csv
```

1.2 3.2

1.2.1 3.2.a Simple Geohash Index

[ ]:
```
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:
        if record.get('src_airport'):
            hashes.append(pygeohash.
 ↪encode(record["src_airport"]["longitude"],record["src_airport"]["latitude"]))
        json_dir = f"/results/geoindex/{hashes[-1][:1]}/{hashes[-1][:2]}/
 ↪{hashes[-1][:3]}.jsonl.gz"
        cwd = os.getcwd()
```

```
        os.makedirs(os.path.dirname(cwd+json_dir), exist_ok=True)

        with gzip.open(cwd+json_dir, "ab") as f:
            json_str = json.dumps(record).encode("utf-8")
            f.write(json_str + b"\n")

create_hash_dirs(records)
```

1.2.2 3.2.b Simple Search Feature

[ ]:
```
def airport_search(latitude, longitude):
    ## TODO: Create simple search to return nearest airport
    src_loc = pygeohash.encode(latitude, longitude, precision=5)
    records = read_jsonl_data()
    short_distance = 20000000
    short_record = {}
    for record in records:
        if record.get('src_airport'):
            temp_loc = pygeohash.encode(record["src_airport"]["latitude"],␣
 ↪record["src_airport"]["longitude"], precision=5)
            if pygeohash.geohash_approximate_distance(src_loc, temp_loc,␣
 ↪check_validity=False) < short_distance:
                short_distance = pygeohash.
 ↪geohash_approximate_distance(src_loc, temp_loc, check_validity=False)
                short_record = record

    print(f"Closest airport is :  {short_record['src_airport']['name']}")
    print(f"Distance of:  {short_distance/1000} Km")

# airport_search(48.04261750114304, 2.8295283335661248)
airport_search(41.1499988, -95.91779)
```

```
Closest airport is :  Eppley Airfield
Distance of:  19.545 Km
```