Assignment 3

April 28, 2023

1 Assignment 3

Import libraries and define common helper functions

[3]:
```
import os
import sys
import gzip
import json
from pathlib import Path
import csv

import genson
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(endpoint_path='/Users/mithilpatel/Desktop/DSC_650/'):
    # s3 = s3fs.S3FileSystem(
    #     anon=True,
    #     client_kwargs={
    #         'endpoint_url': endpoint_url
    #     }
```

```
    # )

    src_data_path = endpoint_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

[4]:
```
records = read_jsonl_data()
```

[ ]:
```

```

1.1 3.1

1.1.1 3.1.a JSON Schema

[ ]:
```
def validate_jsonl_data(records):

    # Building Schema using Genson library
    def create_schema(records):
        builder = genson.SchemaBuilder()
        for i in records:
            builder.add_object(i)
        schema = builder.to_schema()

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

    # Calling the function to create a json file for schema (Uncommend to␣
 ↪create one)
    # create_schema(records)

    schema_path = schema_dir.joinpath('routes-schema.json')
    with open(schema_path) as f:
        schema = json.load(f)

    with open("results/schema_validation.txt", 'w') as f:
        for i, record in enumerate(records):
            try:
                jsonschema.validate(record, schema)
            except jsonschema.exceptions.ValidationError as e:
                print("Validation failed: ", e)
        print("Validation successful!")
```

```
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')

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

    # Wrdata_pathg to Avro file
    with open(data_path, "wb") as f:
        fastavro.writer(f, schema, records)


create_avro_dataset(records)
```

1.1.3 3.1.c Parquet

[ ]:
```
def create_parquet_dataset(endpoint_path='/Users/mithilpatel/Desktop/DSC_650/'):

    parquet_output_path = results_dir.joinpath('routes.parquet')
    src_data_path = endpoint_path + 'data/processed/openflights/routes.jsonl.gz'

    # Reading the json data and storing into an array
    with gzip.open(src_data_path, 'rb') as f:
        records = [json.loads(line) for line in f.readlines()]

    # Saving the data as the Parquet dataset using pyarrow
    table = pa.Table.from_pydict({key: [row[key] for row in records] for key in␣
 ↪records[0]}, schema=None)

    # Writing the dataset into a table
    pq.write_table(table, parquet_output_path)
create_parquet_dataset()
```

1.1.4 3.1.d Protocol Buffers

[54]:
```
sys.path.insert(0, os.path.abspath('routes_pb2'))

import routes_pb2

def _airport_to_proto_obj(airport):
    # Creating an instance of Airport class with no set values
```

```
    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()

    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[:4]:
        route = routes_pb2.Route()

        route.airline.CopyFrom(_airline_to_proto_obj(record.get("airline")))
        if record.get('src_airport'):
            route.src_airport.CopyFrom(_airport_to_proto_obj(record.
 ↪get("src_airport")))

        if record.get('dst_airport'):
            route.dst_airport.CopyFrom(_airport_to_proto_obj(record.
 ↪get("dst_airport")))

        if record.get('codeshare') is not None:
            route.codeshare = record.get('codeshare')

        if record.get('equipment'):
            CR2_unicode = ''.join(record.get('equipment'))
            route.equipment.append(CR2_unicode)

        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)
print("Done!")
```

```
Done!
```

2 3.1.e Output Sizes

[101]:
```
import bz2

# Getting uncompressed file size
def file_size(path):
    return os.path.getsize(path)

# Function to get zip file size (.gz)
def zip_file_size(path):
    with open(path, 'rb') as zip_file:
        with gzip.open(path+'.gz', 'wb') as file:
            file.write(zip_file.read())

    zip_size = os.path.getsize(path+'.gz')
    os.remove(path+'.gz')
    return zip_size

# Function to get zip file size (.bz2)
def zip2_file_size(path):
        # compress the text file with bz2
    with open(path, 'rb') as zip2_file:
        with bz2.open(path +'.bz2', 'wb') as file:
            file.write(zip2_file.read())

    zip2_size = os.path.getsize(path+'.bz2')
    os.remove(path+'.bz2')
    return zip2_size

# Function to get zip file size (snappy)
def snappy_file_size(path):
    if path.endswith('.snappy'):
        return os.path.getsize(path)
    else:
        with open(path, 'rb') as f_in:
            with open('file_name.snappy', 'wb') as f_out:
                f_out.write(snappy.compress(f_in.read()))
        snappy_size = os.path.getsize('file_name.snappy')
        os.remove('file_name.snappy')
        return snappy_size
```

```
# Getting file path inside the 'result' folder
path_list = []

# Loop through each subdirectory in the directory
for root, dirs, files in os.walk(results_dir):
    # Loop through each file in the directory
    for file in files:
        # Print the file path
        if file.endswith('.gz') or file.endswith('.csv'):
            continue
        path_list.append(os.path.join(root,file))

# File paths
path_list = path_list[:-1]

# File names
extensions = [os.path.splitext(path)[1] for path in path_list]

result = []
for i, path in enumerate(path_list):
    if path.endswith('.snappy'):
        continue
    result.append({
        "Format": extensions[i][1:],
        "Uncompressed": file_size(path),
        "Compressed(.gz)": zip_file_size(path),
        "Compressed(.bz2)": zip2_file_size(path),
        "Compressed(snappy)" : snappy_file_size(path)
    })

# Creating Pandas dataframe to save the result
result_df = pd.DataFrame(result)

result_df.to_csv('results/comparison.csv', index= False)
print("Successfully saved!")
```

```
Successfully saved!
```

2.1 3.2

2.1.1 3.2.a Simple Geohash Index

[102]:
```
def create_hash_dirs(records):

    #Creating a geoindex folder to store indexes
    geoindex_dir = results_dir.joinpath('geoindex/')
    geoindex_dir.mkdir(exist_ok=True, parents=True)
    hashes = []
```

```
    for coord in records:
        #Checking whether source airport exist
        if coord.get('src_airport'):
            # Getting the geohash for each source airport & appending to a list
            src_airport = coord.get('src_airport')
            geohash = pygeohash.encode(src_airport['latitude'],␣
↪src_airport['longitude'])
            hashes.append(geohash)

            # Creating a geohash and source airport key-value pair
            output_dic = {}
            output_dic[geohash] = coord['src_airport']

            # Creating folders to store the dictionary
            index_folder = geoindex_dir.joinpath(f'{geohash[-3]}/{geohash[-3:
↪-1]}')
            index_folder.mkdir(exist_ok=True,parents=True)
            os.chdir(index_folder)

            # Writing the geohash as the key and src_airport as the value into␣
↪each files
            with open(f'{geohash[-3:]}.jsonl.gz','wb') as f:
                f.write(json.dumps(output_dic).encode("utf-8"))
                f.write(b"\n")

create_hash_dirs(records)
```

2.1.2 3.2.b Simple Search Feature

[103]:
```
def airport_search(latitude, longitude):
    input_geohash = pygeohash.encode(41.1499988,-95.91779, precision=5)
    shortest_dist = 1e100
    airport_name = ''
    for rec in records:
        if rec.get('src_airport'):
            src_airport = rec.get('src_airport')
            geohash = pygeohash.encode(src_airport['latitude'],␣
↪src_airport['longitude'])
            src_input_dist = pygeohash.
↪geohash_approximate_distance(input_geohash,geohash)/1000
            if src_input_dist < shortest_dist:
                airport_name = src_airport['name']
                shortest_dist = src_input_dist

    return f"Shortest airport is {airport_name} at {shortest_dist:2f} km"
```

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

[103]: 'Shortest airport is Eppley Airfield at 19.545000 km'

[ ]: