DSC 650 Alyssa Weber 6/26/23

Assignment 3

Import libraries and define common helper functions

In [1]:
```
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 gzip.open(src_data_path, 'rb') as f:records = [json.loads(line) for line i

    return records
```

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

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

In [3]:
```
# show first record
# records[1]
```

Note: I copied this first record into https://www.liquid-technologies.com/online-json-to-schema-converter to create my JSON schema. There were errors with the boolean items in 'active' and 'codeshare'. I adjusted them to strings in the website, then back to bookleans in the code in 3.1a

3.1

3.1.a JSON Schema

I first tried to write the schema using the genson package. The following code was used to write the routes-schema.json file.

In [5]:
```
#%pip install genson
from genson import SchemaBuilder

schema_path = schema_dir.joinpath('routes-schema.json')

builder = SchemaBuilder()
builder.add_object(records )
routes_schema = builder.to_schema()

with open(schema_path, "w") as f:
    json.dump(routes_schema, f)
```

I used a generator to write a different schema to use for validation and manually added it since the genson schema was throwing too many errors. To create the schema, I copied the first record into https://www.liquid-technologies.com/online-json-to-schema-converter. There were errors with the boolean items in 'active' and 'codeshare'. I adjusted them to strings in the website, then back to bookleans in the code in 3.1a

In [6]:
```
def validate_jsonl_data(records):

    validation_csv_path = results_dir.joinpath('routes-schema.json')
    from jsonschema import validate

    schema = {
  "$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"
  ]
}
```

```
    with open(validation_csv_path, 'w') as f:
        count = 0
        total = 0
        for i, record in enumerate(records):
            total = total + 1
            try:
                validate(record, schema)
                pass
            except ValidationError as e:
                count = count + 1
                pass

        print('There are', count, 'invalid records out of' , total, 'records.')


validate_jsonl_data(records)
```

```
There are 892 invalid records out of 67663 records.
```

3.1.b Avro

In [7]:
```
# version 1.7 was throwing errors in the next cell with <none> values
# pip install fastavro==1.3.0
```

In [8]:
```
from fastavro import writer, reader, parse_schema

def create_avro_dataset(records):
    schema_path = schema_dir.joinpath('routes.avsc')
    data_path = results_dir.joinpath('routes.avro')

    with open(schema_path) as f:
        schema = json.loads(f.read())
        avro_schema = fastavro.parse_schema(schema)
    with open(data_path, 'wb') as dataset:
        fastavro.writer(dataset, avro_schema, records)

create_avro_dataset(records)
```

I was unable to open the routes.avro file due to a UTF-8 error. I used https://towardsdatascience.com/csv-files-for-storage-absolutely-not-use-apache-avro-instead-7b7296149326 to help me view the file as a pandas dataframe.

In [9]:
```
import pandas as pd

# 1. List to store the records
avro_records = []

# 2. Read the Avro file
data_path = results_dir.joinpath('routes.avro')
with open(data_path, 'rb') as fo:
    avro_reader = reader(fo)
    for record in avro_reader:
        avro_records.append(record)

# 3. Convert to pd.DataFrame
df_avro = pd.DataFrame(avro_records)

# Print the first couple of rows
df_avro.head()
```

Out[9]:

	airline	src_airport	dst_airport	codeshare	stops	equipment
0	{'airline_id': 410, 'name': 'Aerocondor', 'ali...	{'airport_id': 2965, 'name': 'Sochi Internatio...	{'airport_id': 2990, 'name': 'Kazan Internatio...	False	0	[CR2]
1	{'airline_id': 410, 'name': 'Aerocondor', 'ali...	{'airport_id': 2966, 'name': 'Astrakhan Airpor...	{'airport_id': 2990, 'name': 'Kazan Internatio...	False	0	[CR2]
2	{'airline_id': 410, 'name': 'Aerocondor', 'ali...	{'airport_id': 2966, 'name': 'Astrakhan Airpor...	{'airport_id': 2962, 'name': 'Mineralnyye Vody...	False	0	[CR2]
3	{'airline_id': 410, 'name': 'Aerocondor', 'ali...	{'airport_id': 2968, 'name': 'Chelyabinsk Bala...	{'airport_id': 2990, 'name': 'Kazan Internatio...	False	0	[CR2]
4	{'airline_id': 410, 'name': 'Aerocondor', 'ali...	{'airport_id': 2968, 'name': 'Chelyabinsk Bala...	{'airport_id': 4078, 'name': 'Tolmachevo Airpo...	False	0	[CR2]

3.1.c Parquet

In [10]:
```
def create_parquet_dataset():
    src_data_path = 'routes.jsonl.gz'
    parquet_output_path = results_dir.joinpath('routes.parquet')

    with gzip.open(src_data_path, 'rb') as f:
        df = pd.read_json(src_data_path, lines=True)
        table = pa.Table.from_pandas(df)

    pq.write_table(table, parquet_output_path)
```

```
    pass
    ## TODO: Use Apache Arrow to create Parquet table and save the dataset

create_parquet_dataset()
```

Again, the routes.parquet file will not open due to a UTF-8 encoded error. The following code checks that the file was populated correctly. (Method found in a Teams thread by Matthew Fikes on 9/15/2021)

In [11]:
```
pq_output = results_dir.joinpath('routes.parquet')
parquet_table = pq.read_table(pq_output)
parquet_table.to_pandas()
```

Out[11]:

	airline	src_airport	dst_airport	codeshare	equipment
0	{'active': True, 'airline_id': 410, 'alias': '...	{'airport_id': 2965.0, 'altitude': 89.0, 'city...	{'airport_id': 2990.0, 'altitude': 411.0, 'cit...	False	[CR2]
1	{'active': True, 'airline_id': 410, 'alias': '...	{'airport_id': 2966.0, 'altitude': -65.0, 'cit...	{'airport_id': 2990.0, 'altitude': 411.0, 'cit...	False	[CR2]
2	{'active': True, 'airline_id': 410, 'alias': '...	{'airport_id': 2966.0, 'altitude': -65.0, 'cit...	{'airport_id': 2962.0, 'altitude': 1054.0, 'ci...	False	[CR2]
3	{'active': True, 'airline_id': 410, 'alias': '...	{'airport_id': 2968.0, 'altitude': 769.0, 'cit...	{'airport_id': 2990.0, 'altitude': 411.0, 'cit...	False	[CR2]
4	{'active': True, 'airline_id': 410, 'alias': '...	{'airport_id': 2968.0, 'altitude': 769.0, 'cit...	{'airport_id': 4078.0, 'altitude': 365.0, 'cit...	False	[CR2]
...	...	...	...	...	...
67658	{'active': True, 'airline_id': 4178, 'alias': ...	{'airport_id': 6334.0, 'altitude': 41.0, 'city...	{'airport_id': 3341.0, 'altitude': 20.0, 'city...	False	[SF3]
67659	{'active': True, 'airline_id': 19016, 'alias':...	{'airport_id': 4029.0, 'altitude': 588.0, 'cit...	{'airport_id': 2912.0, 'altitude': 2058.0, 'ci...	False	[734]
67660	{'active': True, 'airline_id': 19016, 'alias':...	{'airport_id': 2912.0, 'altitude': 2058.0, 'ci...	{'airport_id': 4029.0, 'altitude': 588.0, 'cit...	False	[734]
67661	{'active': True, 'airline_id': 19016, 'alias':...	{'airport_id': 2912.0, 'altitude': 2058.0, 'ci...	{'airport_id': 2913.0, 'altitude': 2927.0, 'ci...	False	[734]
67662	{'active': True, 'airline_id': 19016, 'alias':...	{'airport_id': 2913.0, 'altitude': 2927.0, 'ci...	{'airport_id': 2912.0, 'altitude': 2058.0, 'ci...	False	[734]

67663 rows × 5 columns

3.1.d Protocol Buffers

In [12]:
```
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()
    if airline is None:
        return None
    if not airline.get('name'):
        return None
    if not airline.get('airline_id'):
        return None
    if not airline.get('active'):
        return None

    obj.airline_id = airline.get('airline_id')
    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()
        airline = _airline_to_proto_obj(record.get('airline', {}))
        if airline:
            route.airline.CopyFrom(airline)
        src_airport = _airport_to_proto_obj(record.get('src_airport', {}))
        if src_airport:
            route.src_airport.CopyFrom(src_airport)
        dst_airport = _airline_to_proto_obj(record.get('dst_airport', {}))
        if dst_airport:
            route.dst_airport.CopyFrom(dst_airport)
        route.codeshare = record.get('codeshare', False)
        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)
```

3.1.e Output Sizes

In [13]:
```
import os

# routes.avro
file_path = results_dir.joinpath('routes.avro')
file_size = os.path.getsize(file_path)
print("File Size is :", file_size, "bytes")
```

```
File Size is : 19646227 bytes
```

In [14]:
```
# routes.parquet
file_path = results_dir.joinpath('routes.parquet')
file_size = os.path.getsize(file_path)
print("File Size is :", file_size, "bytes")
```

```
File Size is : 1977853 bytes
```

In [15]:
```
# routes.pb
file_path = results_dir.joinpath('routes.pb')
file_size = os.path.getsize(file_path)
print("File Size is :", file_size, "bytes")
```

```
File Size is : 13401314 bytes
```

In [16]:
```
# routes.pb.snappy
file_path = results_dir.joinpath('routes.pb.snappy')
file_size = os.path.getsize(file_path)
print("File Size is :", file_size, "bytes")
```

```
File Size is : 1900412 bytes
```

3.2

3.2.a Simple Geohash Index

In [17]:
```
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:
                geohash = pygeohash.encode(latitude, longitude)
                record['geohash'] = geohash
                hashes.append(geohash)
    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)
```

3.2.b Simple Search Feature

In [18]:
```
import pygeohash

def airport_search(latitude, longitude):
    # create geohash based on input of lat and long
    geohash = pygeohash.encode(latitude, longitude)

    # reference geoindex
    geoindex_dir = results_dir.joinpath('geoindex')
```

```
    # set the closest distance to a large value
    closest_distance = float('inf')

    # iterate through the records
    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:
                # for each record create the geohash based on the lat and long
                src_geohash = pygeohash.encode(latitude, longitude)
                # calculate the distance between the record and the original input
                distance = pygeohash.geohash_approximate_distance(geohash, src_geoh
                # if the distance is the new smallest...
                if distance < closest_distance:
                    # save it as the closest distance and save the airport informat
                    closest_distance = distance
                    airport = src_airport

    #print the results
    print(airport['name'], 'is the closest airport to your search.' )

airport_search(41.1499988, -95.91779)
```

```
Eppley Airfield is the closest airport to your search.
```

In [ ]: