Assignment 3

Import libraries and define common helper functions

In [24]:

```
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
from fastavro import parse_schema
from fastavro import writer
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.g
    #with s3.open(src_data_path, 'rb') as f_gz:
    with gzip.open(src_data_path, 'rb') as f:
        records = [json.loads(line) for line in f.readlines()]


    return records
```

In [25]:
```
os.getcwd()
```

Out[25]:
```
'/home/jovyan/dsc650/dsc650/assignments/assignment03'
```

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

In [26]:
```
records = read_jsonl_data()
```

In [27]:
```
records[0:1]
```

Out[27]:

```
[{'airline': {'airline_id': 410,
   'name': 'Aerocondor',
   'alias': 'ANA All Nippon Airways',
   'iata': '2B',
   'icao': 'ARD',
   'callsign': 'AEROCONDOR',
   'country': 'Portugal',
   'active': True},
  'src_airport': {'airport_id': 2965,
   'name': 'Sochi International Airport',
   'city': 'Sochi',
   'country': 'Russia',
   'iata': 'AER',
   'icao': 'URSS',
   'latitude': 43.449902,
   'longitude': 39.9566,
   'altitude': 89,
   'timezone': 3.0,
   'dst': 'N',
   'tz_id': 'Europe/Moscow',
   'type': 'airport',
   'source': 'OurAirports'},
  'dst_airport': {'airport_id': 2990,
   'name': 'Kazan International Airport',
   'city': 'Kazan',
   'country': 'Russia',
   'iata': 'KZN',
   'icao': 'UWKD',
   'latitude': 55.606201171875,
   'longitude': 49.278701782227,
   'altitude': 411,
   'timezone': 3.0,
   'dst': 'N',
   'tz_id': 'Europe/Moscow',
   'type': 'airport',
   'source': 'OurAirports'},
  'codeshare': False,
  'equipment': ['CR2']}]
```

3.1

3.1.a JSON Schema

In [15]:
```
pip install genson
```

```
Collecting genson
  Using cached genson-1.2.2-py2.py3-none-any.whl
Installing collected packages: genson
Successfully installed genson-1.2.2
Note: you may need to restart the kernel to use updated packages.
```

In [16]:
```
# Create the routes schema from the json file
from genson import SchemaBuilder

schema_path = schema_dir.joinpath('routes-schema.json')
builder = SchemaBuilder()
builder.add_schema({"type": "object", "properties": {}})
builder.add_object(records)
builder.to_schema()

print(builder.to_json(indent=2))

original_stdout = sys.stdout

with open(schema_path, 'w') as f:
  sys.stdout = f
  print(builder.to_json())
  sys.stdout = original_stdout
```

```
          },
          "src_airport": {
            "anyOf": [
              {
                "type": "null"
              },
              {
                "type": "object",
                "properties": {
                  "airport_id": {
                    "type": "integer"
                  },
                  "name": {
                    "type": "string"
                  },
                  "city": {
                    "type": "string"
                  },
                  "country": {
                    "type": "string"
```

In [18]:
```
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'

    with open(validation_csv_path, 'w', newline = '') as f:
        for i, record in enumerate(records):
            writer = csv.writer(f)

            try:
                ## TODO: Validate record
                jsonschema.validate(record, schema)
            except ValidationError as e:
                ## Print message if invalid record
                f.write(f"Error: {e.message}; failed validating {e.vali
                print(e)


validate_jsonl_data(records)
```

3.1.b Avro

In [19]:
```
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

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

    # parse schema
    parsed_schema = parse_schema(schema)

    # write record according to schema
    with open(data_path, 'wb') as out:
        writer(out, parsed_schema, records)


create_avro_dataset(records)
```

3.1.c Parquet

In [21]:

```
def create_parquet_dataset():
    src_data_path = '../../../data/processed/openflights/routes.jsonl.g
    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(src_data_path, 'rb') as f:
        ## TODO: Use Apache Arrow to create Parquet table and save the 

        # Use Apache Arrow to create Parquet table and save the dataset
        table = read_json(f)

    # write parquet table to '.parquet' file
    pq.write_table(table, parquet_output_path)

create_parquet_dataset()
```

3.1.d Protocol Buffers

In [22]:

```
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
    # check if name and airline_id are there
    if airline is None:
        return None
    if airline.get('airline_id') is None:
        return None

    # get airline info
    obj.airline_id = airline.get('airline_id')

    if airline.get('name'):
        obj.name = airline.get('name')
    if airline.get('alias'):
        obj.name = airline.get('alias')
    if airline.get('iata'):
        obj.name = airline.get('iata')
```

```
    if airline.get('icao'):
        obj.name = airline.get('icao')
    if airline.get('callsign'):
        obj.name = airline.get('callsign')
    if airline.get('country'):
        obj.name = airline.get('country')

    obj.active = airline.get('active') # boolean

    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 Datas

        # copy 'airline' data
        airline = _airline_to_proto_obj(record.get('airline'))
        if airline:
            route.airline.CopyFrom(airline)

        # copy 'src_airport' data
        src_airport = _airport_to_proto_obj(record.get('src_airport'))
        if src_airport:
            route.src_airport.CopyFrom(src_airport)

        # copy 'dst_airport' data
        dst_airport = _airport_to_proto_obj(record.get('dst_airport'))
        if dst_airport:
            route.dst_airport.CopyFrom(dst_airport)

        # get 'codeshare' data
        route.codeshare = record.get('codeshare')

        # get 'equipment' and iterate through for multiple
        equipment = record.get('equipment')
        for equip in equipment:
              route.equipment.append(equip)

        # add generated route to route db
        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)
```

3.2

3.2.a Simple Geohash Index

In [ ]:

```
def create_hash_dirs(records):
    geoindex_dir = results_dir.joinpath('geoindex')
    geoindex_dir.mkdir(exist_ok=True, parents=True)
    hashes = []
    ## TODO: Create hash index
    # Create hash index
    # iterate over records
    for record in records:
        # get source airport info
        src_airport = record.get('src_airport', {})

        # if airport_data there
        if src_airport:
            lat = src_airport.get('latitude')
            lon = src_airport.get('longitude')
            if lat and lon:
              hashes.append(pygeohash.encode(lat, lon))

        hashes.sort()

        three_char = sorted(list(set([entry[:3] for entry in hashes])))

        hash_index = {value: [] for value in three_char}

        # iterate through records and add to appropriate hash index
        for record in records:
            geohash = record.get('geohash')
            if geohash:
              hash_index[geohash[:3]].append(record)

        # for index items create folder/subfolder directories by short l
        for key, values in hash_index.items():
            output_dir = geoindex_dir.joinpath(str(key[:1])).joinpath(s
            output_dir.mkdir(exist_ok = True, parents = True)
            output_path = output_dir.joinpath(f"{key}.jsonl.gz")

            # save record to appropriate subfolder/file
            with gzip.open(output_path, 'w') as f:
                json_output = '\n'.join([json.dumps(value) for value in
                f.write(json_output.encode('utf-8'))

create_hash_dirs(records)
```

3.2.b Simple Search Feature

In [28]:

```
def airport_search(latitude, longitude):
    ## TODO: Create simple search to return nearest airport
    # get input lat and lon
    input_hash = pygeohash.encode(latitude, longitude)

    # initialize variables
    distance = 0
    name = ''

    # Iterate through records
    for idx, record in enumerate(records):
        # get 'src_airport' info
        src_airport = record.get('src_airport', {})

        # if src_airport not empty
        if src_airport:
            # get latitude, longitude, and airport name
            lat = src_airport.get('latitude')
            lon = src_airport.get('longitude')
            airport_name = src_airport.get('name')

            # if lat and lon not empty
            if lat and lon:
                # encode lat and lon
                airport_hash = pygeohash.encode(lat, lon)

                # get distance between input hash and airport hash
                dist_n = pygeohash.geohash_approximate_distance(input_h

                # If its the first record save distance to dist
                # else check if calc distance > distance and save smalle
                if idx == 0:
                    distance = dist_n
                else:
                    if distance > dist_n:
                        distance = dist_n
                        name = airport_name

    print(f"Closest airport is {name}")

airport_search(41.1499988, -95.91779)
```

```
Closest airport is Eppley Airfield
```

In [29]:

```
routes_avro_size = os.path.getsize("results/routes.avro")
print (f"Avro uncompressed: {round(routes_avro_size/1024, 2)} KB")
```

```
Avro uncompressed: 19185.77 KB
```

In [30]:
```
routes_parquet_size = os.path.getsize("results/routes.parquet")
print (f"Parquet uncompressed: {round(routes_parquet_size/1024, 2)} KB"
```

```
Parquet uncompressed: 1929.17 KB
```

In [31]:
```
routes_snappy_size = os.path.getsize("results/routes.pb.snappy")
print (f"PB Snappy uncompressed: {round(routes_snappy_size/1024, 2)} KB
```

```
PB Snappy uncompressed: 3395.79 KB
```

In [32]:
```
routes_pb_size = os.path.getsize("results/routes.pb")
print (f"PB uncompressed: {round(routes_pb_size/1024, 2)} KB")
```

```
PB uncompressed: 18765.11 KB
```

In [33]:
```
routes_JSONSCHEMA_size = os.path.getsize("schemas/routes-schema.json")
print (f"JSON Schema: {round(routes_JSONSCHEMA_size/1024, 2)} KB")
```

```
JSON Schema: 0.0 KB
```

In [36]:
```
routes_JSONSCHEMA_gzsize = os.path.getsize("../../../data/processed/ope
print (f"JSON Schema Zipped: {round(routes_JSONSCHEMA_gzsize/1024, 2)} 
```

```
JSON Schema Zipped: 3249.17 KB
```

In [ ]: