Assignment 3

Import libraries and define common helper functions

In [56]:
```
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)

# Correct the function to use the github resource.
def read_jsonl_data():
    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

In [57]:
```
records = read_jsonl_data()
```

In [58]:
```
# Show the data shape.
records[0]
```

Out[58]:
```
{'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 [59]:

```python
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 = "validation.txt"

    with open(validation_csv_path, 'w') as f:
        for i, record in enumerate(records):
            try:
                # Use the validate method through jsonschema
                jsonschema.validate(record, schema)
                pass
            except ValidationError as e:
                # Write the failed record.
                f.write(f"Record {i} Failed: {str(e)}\n")
                pass

    # Add the records to the routes-schema.json file.
    schema = genson.Schema()

    for record in records:
        schema.add_object(record)

    # Convert to dict to ensure map-like structure
    json_schema = schema.to_dict()

    with open(schema_path, 'w') as f:
        json.dump(json_schema, f, indent=2)

validate_jsonl_data(records)
```

3.1.b Avro

In [60]:
```
def create_avro_dataset(records):
    schema_path = schema_dir.joinpath('routes.avsc')
    data_path = results_dir.joinpath('routes.avro')

    # Load the Avro schema
    with open(schema_path, 'r') as f:
        parsed_schema = json.load(f)

    # Solution suggested was to remove all none values. Confirmed worked but still not w
    with open(data_path, 'wb') as out:
        fastavro.writer(out, parsed_schema, records)


create_avro_dataset(records)
```

3.1.c Parquet

In [68]:
```
def create_parquet_dataset():
    src_data_path = '../../../data/processed/openflights/routes.jsonl.gz'
    parquet_output_path = results_dir.joinpath('routes.parquet')

    # Deleted s3 stuff since it doesn't work.
    with open(src_data_path, 'rb') as f_gz:
        with gzip.open(f_gz, 'rb') as f:

            # Loop over the lines and save the records, then batch them to the table.
            record_batches = []

            for line in f:
                record = json.loads(line)
                record_batch = pa.record_batch([pa.array([json.loads(line) for line in f
                record_batches.append(record_batch)

            # Collect all the record batches into the table.
            table = pa.Table.from_batches(record_batches)

    # Write the Parquet table to a file
    pq.write_table(table, str(parquet_output_path))

create_parquet_dataset()
```

3.1.d Protocol Buffers

In [83]:
```
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 not airport.get('airport_id'):
        return None

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

    # set all values to the object if they are present in the record.
    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')

    return obj


def _airline_to_proto_obj(airline):
    obj = routes_pb2.Airline()
    if not airline.get('name'):
        return None
    if not airline.get('airline_id'):
        return None

    # Professor provided lines start here:
    obj.airline_id = airline.get('airline_id')
    obj.name = airline.get('name')
    if airline.get('alias'):
        obj.alias = airline.get('alias')
    # End of professor's code.

    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')

    # check if airline is found or set, set to false if not.
    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()

        # Start of professor's code:
        airline = _airline_to_proto_obj(record.get('airline', {}))
        if airline:
            route.airline.CopyFrom(airline)
        src_airport = _airport_to_proto_obj(record.get('src_airport', {}))
        # End professor's code.

        # set the remaining pieces.
```

```
        if record.get('dst_airport'):
            route.dst_airport.CopyFrom(_airport_to_proto_obj(record["dst_airport"]))
        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)
```

3.1.e Output Sizes

In [85]:
```
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 = "validation.txt"
parquet_file = "results/routes.parquet"
output_file = "results/comparison.csv"

# Add the entries for each set of objects.
entries = []

entries.append({"format" : "avro file","uncompressed" : get_file_size(avro_file),"gzip"
```

```
entries.append({"format" : "protocol buffer file","uncompressed" : get_file_size(pb_file
entries.append({"format" : "json Schema file","uncompressed" : get_file_size(json_file),
entries.append({"format" : "parquet file","uncompressed" : get_file_size(parquet_file),"
sizes_df = pd.DataFrame(entries)

# Push the sizes to the csv
sizes_df.to_csv(output_file, sep=',')

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

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

3.2

3.2.a Simple Geohash Index

In [86]:
```
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:
                # use pygeohash.encode() to assign geohashes to the records and complete
                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 [89]:
```
def airport_search(latitude, longitude):

    # Set a location source.
    location_temp = pygeohash.encode(latitude, longitude, precision=5)

    # Set the records file again from the beginning (just in case)
    records = read_jsonl_data()

    # Set a new record and distance.
    short_distance = 20000000
    short_record = {}
```

```
    # Loop over the records to find the nearest one based on lat and long.
    for record in records:
        if record.get('src_airport'):
            temp_loc = pygeohash.encode(record["src_airport"]["latitude"], record["src_a
            if pygeohash.geohash_approximate_distance(location_temp, temp_loc, check_val
                short_distance = pygeohash.geohash_approximate_distance(location_temp, t
                short_record = record

    # Print a friendly message to the user.
    print(f"The nearest airport is {short_record['src_airport']['name']} with a distance

# Test the airport search function.
airport_search(41.1499988, -95.91779)
```

```
The nearest airport is Eppley Airfield with a distance of 19.55 Km
```