Assignment 3

September 19, 2021

1 Assignment 3

Import libraries and define common helper functions

[4]:
```
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 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()]


    return records
```

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

[5]:
```
records = read_jsonl_data()

records[0:1]
```

[5]:
```
[{'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']}]
```

[8]:
```
!python -m pip install -U genson
import genson
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))
```

```
Collecting genson
  Using cached genson-1.2.2-py2.py3-none-any.whl
Installing collected packages: genson
Successfully installed genson-1.2.2
{
  "$schema": "http://json-schema.org/schema#",
  "anyOf": [
    {
      "type": "object"
    },
    {
      "type": "array",
      "items": {
        "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": [
      "active",
      "airline_id",
      "alias",
      "callsign",
      "country",
      "iata",
      "icao",
      "name"
    ]
  },
  "src_airport": {
    "anyOf": [
      {
        "type": "null"
      },
      {
        "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",
        "altitude",
        "city",
        "country",
        "dst",
        "iata",
        "icao",
        "latitude",
        "longitude",
        "name",
        "source",
        "timezone",
        "type",
        "tz_id"
      ]
    }
  ]
},
"dst_airport": {
  "anyOf": [
    {
```

```
  "type": "null"
},
{
  "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",
                  "altitude",
                  "city",
                  "country",
                  "dst",
                  "iata",
                  "icao",
                  "latitude",
                  "longitude",
                  "name",
                  "source",
                  "timezone",
                  "type",
                  "tz_id"
                ]
              }
            ]
          },
          "codeshare": {
            "type": "boolean"
          },
          "equipment": {
            "type": "array",
            "items": {
              "type": "string"
            }
          }
        },
        "required": [
          "airline",
          "codeshare",
          "dst_airport",
          "equipment",
          "src_airport"
        ]
      }
    }
  ]
}
```

1.1 3.1

1.1.1 3.1.a JSON Schema

[9]:
```
def validate_jsonl_data(records):
    schema_path = schema_dir.joinpath('routes-schema.json')
    with open(schema_path) as f:
        _schema = json.load(f)

    print(_schema)

    validation_csv_path = results_dir.joinpath('validation-results.csv')
    with open(validation_csv_path, 'w') as f:
        for i, record in enumerate(records):
            try:
                ## TODO: Validate record
                jsonschema.validate(record, _schema)
                ##pass
            except ValidationError as e:
                ## Print message if invalid record
                detail = e.message
                print(detail)
                f.write(str(e.path))
                f.write(str(e.instance))
                f.write(str(detail))
                return detail

validate_jsonl_data(records)
```

```
{'$schema': 'http://json-schema.org/draft-04/schema#', 'type': 'object',
'properties': {'airline': {'type': 'object', 'properties': {'active': {'type':
'boolean'}, 'airline_id': {'type': 'integer'}, 'alias': {'type': 'string'},
'callsign': {'type': 'string'}, 'country': {'type': 'string'}, 'iata': {'type':
'string'}, 'icao': {'type': 'string'}, 'name': {'type': 'string'}}, 'required':
['active', 'airline_id', 'alias', 'callsign', 'country', 'iata', 'icao',
'name']}, 'codeshare': {'type': 'boolean'}, 'dst_airport': {'type': ['object',
'null'], 'properties': {'airport_id': {'type': 'integer'}, 'altitude': {'type':
'integer'}, 'city': {'type': 'string'}, 'country': {'type': 'string'}, 'dst':
{'type': 'string'}, 'iata': {'type': 'string'}, 'icao': {'type': 'string'},
'latitude': {'type': 'number'}, 'longitude': {'type': 'number'}, 'name':
{'type': 'string'}, 'source': {'type': 'string'}, 'timezone': {'type':
'number'}, 'type': {'type': 'string'}, 'tz_id': {'type': 'string'}}, 'required':
['airport_id', 'altitude', 'city', 'country', 'dst', 'iata', 'icao', 'latitude',
'longitude', 'name', 'source', 'timezone', 'type', 'tz_id']}, 'equipment':
{'type': 'array', 'items': [{'type': 'string'}]}, 'src_airport': {'type':
['object', 'null'], 'properties': {'airport_id': {'type': 'integer'},
'altitude': {'type': 'integer'}, 'city': {'type': 'string'}, 'country': {'type':
'string'}, 'dst': {'type': 'string'}, 'iata': {'type': 'string'}, 'icao':
```

{'type': 'string'}, 'latitude': {'type': 'number'}, 'longitude': {'type': 'number'}, 'name': {'type': 'string'}, 'source': {'type': 'string'}, 'timezone': {'type': 'number'}, 'type': {'type': 'string'}, 'tz_id': {'type': 'string'}}, 'required': ['airport_id', 'altitude', 'city', 'country', 'dst', 'iata', 'icao', 'latitude', 'longitude', 'name', 'source', 'timezone', 'type', 'tz_id']}}, 'required': ['airline', 'codeshare', 'dst_airport', 'equipment', 'src_airport']}

1.1.2 3.1.b Avro

[10]:
```
def create_avro_dataset(records):
    schema_path = schema_dir.joinpath('routes.avsc')

    data_path = results_dir.joinpath('routes.avro')
    print (data_path)
    ## TODO: Use fastavro to create Avro dataset
    with open(schema_path) as fo:
        schema = json.loads(fo.read())

    parsed_schema = fastavro.parse_schema(schema)

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


create_avro_dataset(records)
```

/home/jovyan/dsc650/dsc650/assignments/assignment03/results/routes.avro

1.1.3 3.1.c Parquet

[11]:
```
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 s3.open(src_data_path, 'rb') as f_gz:
        with gzip.open(f_gz, 'rb') as f:
            ## TODO: Use Apache Arrow to create Parquet table and save the␣
 ↪dataset
            table = read_json(f)
            print(table)
            pq.write_table(table, parquet_output_path, compression='none')
```

```
create_parquet_dataset()
```

```
pyarrow.Table
airline: struct<airline_id: int64, name: string, alias: string, iata: string,
icao: string, callsign: string, country: string, active: bool>
  child 0, airline_id: int64
  child 1, name: string
  child 2, alias: string
  child 3, iata: string
  child 4, icao: string
  child 5, callsign: string
  child 6, country: string
  child 7, active: bool
src_airport: struct<airport_id: int64, name: string, city: string, country:
string, iata: string, icao: string, latitude: double, longitude: double,
altitude: int64, timezone: double, dst: string, tz_id: string, type: string,
source: string>
  child 0, airport_id: int64
  child 1, name: string
  child 2, city: string
  child 3, country: string
  child 4, iata: string
  child 5, icao: string
  child 6, latitude: double
  child 7, longitude: double
  child 8, altitude: int64
  child 9, timezone: double
  child 10, dst: string
  child 11, tz_id: string
  child 12, type: string
  child 13, source: string
dst_airport: struct<airport_id: int64, name: string, city: string, country:
string, iata: string, icao: string, latitude: double, longitude: double,
altitude: int64, timezone: double, dst: string, tz_id: string, type: string,
source: string>
  child 0, airport_id: int64
  child 1, name: string
  child 2, city: string
  child 3, country: string
  child 4, iata: string
  child 5, icao: string
  child 6, latitude: double
  child 7, longitude: double
  child 8, altitude: int64
  child 9, timezone: double
```

```
  child 10, dst: string
  child 11, tz_id: string
  child 12, type: string
  child 13, source: string
codeshare: bool
equipment: list<item: string>
  child 0, item: string
```

1.1.4 3.1.d Protocol Buffers

[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()
    ## 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('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')
    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
        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 = _airport_to_proto_obj(record.get('dst_airport', {}))
        if dst_airport:
            route.dst_airport.CopyFrom(dst_airport)

        if record.get('codeshare'):
            route.codeshare = record.get('codeshare')
```

```
        else:
            route.codeshare = False

        if record.get('stops'):
            route.stops = record.get('stops')

        equipment = record.get('equipment')

        if len(equipment) > 1:
            for i, v in enumerate(equipment):
                route.equipment.append(v)
        else:
            equipment = record.get('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)
```

1.2 3.2

1.2.1 3.2.a Simple Geohash Index

[13]:
```
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:
        src_airport = record.get('src_airport', {})
        if src_airport:
            latitude = src_airport.get('latitude')
            longitude = src_airport.get('longitude')
            if latitude and longitude:
                hashes.append(pygeohash.encode(latitude, longitude))

    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)
```

1.2.2 3.2.b Simple Search Feature

[41]:
```
def airport_search(latitude, longitude):
    ## TODO: Create simple search to return nearest airport
    h = pygeohash.encode(latitude,longitude)
    #print(h) #9z7f174u17zb
    v_dist = 0
    v_name = ''
    for i,record in enumerate(records):
        src_airport = record.get('src_airport', {})
        if src_airport:
            lat = src_airport.get('latitude')
            long = src_airport.get('longitude')
            ap_name = src_airport.get('name')
            if lat and long:
                h1 = pygeohash.encode(lat,long)
                #print(h1)

                dist_m = pygeohash.geohash_approximate_distance(h,h1)
                dist_km = dist_m/1000 # convert meter to kilometers

                if i==0:
                    v_dist = dist_km
                    print (dist_km, "km")
                else:
                    if v_dist > dist_km:
                        v_dist = dist_km
                        v_name = ap_name
```

```
    print(v_name) # airport name
```

[42]:
```
airport_search(41.1499988, -95.91779)
```

```
20000.0 km
Eppley Airfield
```

2 3.1 e

[1]:
```
import os
routes_avro_size = os.path.getsize("/home/jovyan/dsc650/dsc650/assignments/
↪assignment03/results/routes.avro")
print (routes_avro_size, "bytes")
```

```
19646227 bytes
```

[56]:
```
routes_parquet_size = os.path.getsize("/home/jovyan/dsc650/dsc650/assignments/
↪assignment03/results/routes.parquet")
print (routes_parquet_size, "bytes")
```

```
2327907 bytes
```

[57]:
```
routes_snappy_size = os.path.getsize("/home/jovyan/dsc650/dsc650/assignments/
↪assignment03/results/routes.pb.snappy")
print (routes_snappy_size, "bytes")
```

```
3705406 bytes
```

[58]:
```
routes_pb_size = os.path.getsize("/home/jovyan/dsc650/dsc650/assignments/
↪assignment03/results/routes.pb")
print (routes_pb_size, "bytes")
```

```
22270594 bytes
```

[59]:
```
routes_JSONSCHEMA_size = os.path.getsize("/home/jovyan/dsc650/dsc650/
↪assignments/assignment03/schemas/routes-schema.json")
print (routes_JSONSCHEMA_size, "bytes")
```

```
3461 bytes
```

[60]:
```
routes_JSONSCHEMA_gzsize = os.path.getsize("/home/jovyan/dsc650/data/processed/
↪openflights/routes.jsonl.gz")
print (routes_JSONSCHEMA_gzsize, "bytes")
```

```
3327145 bytes
```

[ ]:
```

```