Assignment 07

Name: Anjani Bonda

Date: 04-28-2023

Assignment 7.1 a

In [29]:
```
# Load required libraries
import os
import json
from pathlib import Path
import gzip
import hashlib
import shutil
import pandas as pd
import pygeohash
import s3fs
import uuid
import math
```

Load routes dataset

In [30]:
```
endpoint_url='https://storage.budsc.midwest-datascience.com'
curr_dir = Path(os.getcwd()).absolute()
result_dir = curr_dir.joinpath('results')

if result_dir.exists():
    shutil.rmtree(result_dir)
result_dir.mkdir(parents=True, exist_ok=True)
```

In [31]:
```
## read_jsonl_data function to process the json file

def read_jsonl_data():
    s3 = s3fs.S3FileSystem(
        anon=True,
        client_kwargs={
            'endpoint_url': endpoint_url
        }
    )
    src_path = '/home/jovyan/git_akb/dsc650/data/processed/openflights/routes.j
    with s3.open(src_data_path, 'rb') as f_gz:
        with gzip.open(f_gz, 'rb') as f:
            recs = [json.loads(line) for line in f.readlines()]
    return recs

def read_jsonl_data_local():
    '''Create function to read file from local'''
    src_path = '/home/jovyan/git_akb/dsc650/data/processed/openflights/routes.j
    with open(src_path, 'rb') as f_gz:
        with gzip.open(f_gz, 'rb') as f:
            recs = [json.loads(line) for line in f.readlines()]
    return recs
```

In [32]:
```
## Flattening the dataset
def flatten_record(record):
    flat_record = dict()
    for key, value in record.items():
        if key in ['airline', 'src_airport', 'dst_airport']:
            if isinstance(value, dict):
                for child_key, child_value in value.items():
                    flat_key = '{}_{}'.format(key, child_key)
                    flat_record[flat_key] = child_value
        else:
            flat_record[key] = value
    return flat_record

def create_flattened_dataset():
    recs = read_jsonl_data_local()
    parquet_path = result_dir.joinpath('routes-flattened.parquet')
    return pd.DataFrame.from_records([flatten_record(record) for record in recs
```

In [33]:
```
## Create df and the key field
df = create_flattened_dataset()
df['key'] = df['src_airport_iata'].astype(str) + df['dst_airport_iata'].astype(
```

In [34]:
```
## Check sample records from dataframe
df.head()
```

Out[34]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	

5 rows × 39 columns

In [35]:
```
## set Partitions
partitions = (
        ('A', 'A'), ('B', 'B'), ('C', 'D'), ('E', 'F'),
        ('G', 'H'), ('I', 'J'), ('K', 'L'), ('M', 'M'),
        ('N', 'N'), ('O', 'P'), ('Q', 'R'), ('S', 'T'),
        ('U', 'U'), ('V', 'V'), ('W', 'X'), ('Y', 'Z')
        )
```

From above, ('A', 'A') refers that the folder contain all of the flight routes whose key starts with A. The results/kv directory contains below folders.

In [36]:

```
# kv
# ├── kv_key=A
# ├── kv_key=B
# ├── kv_key=C-D
# ├── kv_key=E-F
# ├── kv_key=G-H
# ├── kv_key=I-J
# ├── kv_key=K-L
# ├── kv_key=M
# ├── kv_key=N
# ├── kv_key=O-P
# ├── kv_key=Q-R
# ├── kv_key=S-T
# ├── kv_key=U
# ├── kv_key=V
# ├── kv_key=W-X
# └── kv_key=Y-Z
```

In [37]:

```
# define dictionary of partitions and kv_keys
partition_dict = {}
for i in partitions:
    if i[0] == i[1]:
        partition_dict[i] = i[0]
    else:
        partition_dict[i] = i[0] + '-' + i[1]
```

In [38]:

```
## Print partition_dict
partition_dict
```

Out[38]:

```
{('A', 'A'): 'A',
 ('B', 'B'): 'B',
 ('C', 'D'): 'C-D',
 ('E', 'F'): 'E-F',
 ('G', 'H'): 'G-H',
 ('I', 'J'): 'I-J',
 ('K', 'L'): 'K-L',
 ('M', 'M'): 'M',
 ('N', 'N'): 'N',
 ('O', 'P'): 'O-P',
 ('Q', 'R'): 'Q-R',
 ('S', 'T'): 'S-T',
 ('U', 'U'): 'U',
 ('V', 'V'): 'V',
 ('W', 'X'): 'W-X',
 ('Y', 'Z'): 'Y-Z'}
```

In [39]:

```
# Create kv_key from key
def create_kv_key(data_key):
    for key, val in partition_dict.items():
        if data_key[0] == key[0] or data_key[0] == key[1]:
            return val
    return None
```

In [40]:

```
# Add this new column to the existing dataframe
df['kv_key'] = df['key'].apply(create_kv_key)
```

In [42]:

```
## Check sample records from dataframe
df.head()
```

Out[42]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	

5 rows × 40 columns

In [43]:

```
## Check key and kv_key from dataframe for quick comparison
df[['key', 'kv_key']]
```

Out[43]:

	key	kv_key
0	AERKZN2B	A
1	ASFKZN2B	A
2	ASFMRV2B	A
3	CEKKZN2B	C-D
4	CEKOVB2B	C-D
...	...	...
67658	WYAADLZL	W-X
67659	DMEFRUZM	C-D
67660	FRUDMEZM	E-F
67661	FRUOSSZM	E-F
67662	OSSFRUZM	O-P

67663 rows × 2 columns

In [44]:

```
# Lets save the dataframe in parquet format with partition_col as kv_key
try:
    df.to_parquet(result_dir.joinpath('kv'), partition_cols=['kv_key'])
except:
    print("Failure in parquet format conversion")
else:
    print("Successful parquet format conversion")
```

```
Successful parquet format conversion
```

Assignment 7.1.b

In [45]:

```
# Load hashlib library
import hashlib
```

In [46]:

```
# Define Hash key function with utf-8 encoding
def hash_key(key):
    m = hashlib.sha256()
    m.update(str(key).encode('utf-8'))
    return m.hexdigest()
```

We will partition the data using the first character of the hexadecimal hash. As such, there are 16 possible partitions. Create a new column called hashed that is a hashed value of the key column. Next, create a partitioned dataset based on the first character of the hashed key and save the results to results/hash. The directory should contain the following folders.

In [47]:

```
# hash
# ├── hash_key=0
# ├── hash_key=1
# ├── hash_key=2
# ├── hash_key=3
# ├── hash_key=4
# ├── hash_key=5
# ├── hash_key=6
# ├── hash_key=7
# ├── hash_key=8
# ├── hash_key=9
# ├── hash_key=A
# ├── hash_key=B
# ├── hash_key=C
# ├── hash_key=D
# ├── hash_key=E
```

In [48]:

```
# Add 'hashed' column to the dataframe as suggested
df['hashed'] = df['key'].apply(hash_key)
```

In [49]:

```
# Now, Add hash_key paritioning column to dataframe
df['hash_key'] = df['hashed'].str[0]
```

In [51]:

```
## showing few records from dataframe
df.head()
```

Out[51]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	

5 rows × 42 columns

In [53]:

```
# Convert to parquet format with partition column as hask_key
try:
    df.to_parquet(result_dir.joinpath('hash'), partition_cols=['hash_key'])
except:
    print("Failure in parquet format conversion")
else:
    print("Successful parquet format conversion")
```

```
Successful parquet format conversion
```

Assignment 7.1 c

Assume that you have an application that provides routes for each of the source airports and you want to store routes in the data center closest to the source airport. The output folders should look as follows.

In [54]:

```
# geo
# ├── location=central
# ├── location=east
# └── location=west
```

In [55]:

```
# Check columns on dataframe
df.columns
```

Out[55]:
```
Index(['airline_airline_id', 'airline_name', 'airline_alias', 'airline_iata',
       'airline_icao', 'airline_callsign', 'airline_country', 'airline_activ
e',
       'src_airport_airport_id', 'src_airport_name', 'src_airport_city',
       'src_airport_country', 'src_airport_iata', 'src_airport_icao',
       'src_airport_latitude', 'src_airport_longitude', 'src_airport_altitud
e',
       'src_airport_timezone', 'src_airport_dst', 'src_airport_tz_id',
       'src_airport_type', 'src_airport_source', 'dst_airport_airport_id',
       'dst_airport_name', 'dst_airport_city', 'dst_airport_country',
       'dst_airport_iata', 'dst_airport_icao', 'dst_airport_latitude',
       'dst_airport_longitude', 'dst_airport_altitude', 'dst_airport_timezon
e',
       'dst_airport_dst', 'dst_airport_tz_id', 'dst_airport_type',
       'dst_airport_source', 'codeshare', 'equipment', 'key', 'kv_key',
       'hashed', 'hash_key'],
      dtype='object')
```

In [56]:
```python
## Lets define a function and new column to calculate source airport geographic

geo_val = lambda x: pygeohash.encode(x.src_airport_latitude, x.src_airport_long

df['geo_hash'] = df.apply(geo_val, axis=1)
```

In [57]:
```python
## Check sample records in dataframe for new column 'geo_hash'
df.head()
```

Out[57]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	

5 rows × 43 columns

In [58]:
```python
## Set the datacenter/location values per given values

data_center = dict(
    west = pygeohash.encode(45.5945645, -121.1786823),
    central = pygeohash.encode(41.1544433, -96.0422378),
    east = pygeohash.encode(39.08344, -77.6497145)
)
data_center
```

Out[58]:

```
{'west': 'c21g6s0rs4c7', 'central': '9z7dnebnj8kb', 'east': 'dqby34cjw922'}
```

In [61]:

```python
## Define function to identify datacenter close to the source airport
def closest_datacenter_loc(geo_hash):

    distance= {}

    for key, val in data_center.items():
        distance[key] = pygeohash.geohash_haversine_distance(val, geo_hash)
    closest_datacenter = sorted(distance.items(), key=lambda x: x[1])[0][0]
    return closest_datacenter
```

In [62]:

```python
# Add location column for the datacenter closest to source airport
df['location'] = df['geo_hash'].apply(closest_datacenter_loc)
```

In [63]:

```python
## Check sample recs for location values
df.head()
```

Out[63]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	

5 rows × 44 columns

In [64]:

```python
## Check distinct location values
df['location'].unique()
```

Out[64]:

```
array(['east', 'west', 'central'], dtype=object)
```

In [65]:

```python
# Convert the df to parquet format using partition columns as location
try:
    df.to_parquet(result_dir.joinpath('geo'), partition_cols=['location'])
except:
    print("Failure in parquet format conversion")
else:
    print("Successful parquet format conversion")
```

```
Successful parquet format conversion
```

Assignment 7.1 d

In [66]:
```
## Load itertools/islice library
from itertools import islice
```

In [67]:
```
## Define function to create balance partitions
def balance_partitions(keys, num_partitions):

    arr_size = round(len(keys)/num_partitions)
    arr_range = iter(keys)
    partitions_iters = iter(lambda: tuple(islice(arr_range, arr_size)), ())
    partitions = [sorted(part) for part in partitions_iters]

    return partitions
```

In [68]:
```
## Check sample records in dataframe
df.head()
```

Out[68]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	

5 rows × 44 columns

In [72]:
```
## Check values for airline_icao column
df.airline_icao.unique()
```

```
Out[72]: array(['ARD', 'CRG', 'FOF', 'VBW', 'GLG', 'OAW', 'NTJ', 'nan', 'GAP',
                'CGN', 'WE1', '3FF', 'AYZ', 'WZP', 'JSA', 'ISK', 'DAK', 'KW1',
                'TNM', 'GAI', 'CEY', 'CSC', 'BTQ', 'ASD', 'GZP', 'UBD', 'AAS',
                'DSM', 'ANT', 'IBS', 'LOC', 'BHP', 'GWI', 'RBU', 'ICL', 'SSV',
                'FFV', 'CEB', 'SIB', 'AUL', 'JRB', 'RBY', 'MPE', 'VVC', 'IGO',
                'ISR', 'SGG', 'DRU', 'RAC', 'SOV', 'MSE', 'JJA', 'AWU', 'FAB',
                'SFJ', 'ERR', 'M1F', 'GFY', 'ZTF', 'BTV', 'BRB', 'BCC', 'BRG',
                'CGP', 'MXL', 'PCO', 'OHY', 'TIB', 'AAW', 'ACP', 'THS', 'FLG',
                'KAP', 'N78', 'PBA', 'NSE', 'MLD', 'VC9', 'JAI', 'AL2', 'AEE',
                'SWD', 'RLA', 'BGL', 'TGZ', 'AAL', 'BER', 'ABJ', 'ACA', 'AZU',
                'MDA', 'AFR', 'DAH', 'AIC', 'AXM', 'AMX', 'ADH', 'ARG', 'ASA',
                'RAM', 'AVA', 'AWM', 'FIN', 'AZA', 'BRU', 'FLT', 'JBU', 'UIA',
                'BGD', 'BAW', 'SBS', 'SKY', 'BEE', 'RSR', 'BBC', 'RBA', 'LBT',
                'PDC', 'PIC', 'BOT', 'EVA', 'BTI', 'BUU', 'BPA', 'BWA', 'ABL',
                'WDY', 'CCA', 'ABD', 'NTW', 'TOK', 'CAL', 'CMP', 'YCP', 'CUB',
                'CPA', 'CYP', 'CSN', 'RGG', 'DAO', 'ILN', 'XAX', 'GAO', 'NOK',
                'CFG', 'SRQ', 'DSY', 'DAL', 'DTA', 'VSV', 'DTR', 'NAX', 'FVM',
                'OTJ', 'RBG', 'ESF', 'GTA', 'CEL', 'EFA', 'JAA', 'EIN', 'UAE',
                'ANK', 'DLA', 'LHN', 'IRC', 'TAE', 'ETH', 'EEA', 'EWG', 'ETD',
                'FLM', 'FI5', 'BBO', 'FFT', 'LZB', 'AIQ', 'WCP', 'AFG', 'ICE',
                'FJI', 'TRS', 'CSH', 'ATM', 'RYR', 'STU', 'SDM', 'IBX', 'FFM',
                'FDB', 'CIX', 'AAY', 'AEF', 'GOW', 'ABY', 'GIA', 'TNA', 'GBA',
                'LIX', 'GRL', 'BSY', 'AUR', 'UPA', 'ARF', 'GBK', 'RAR', 'SKU',
                'HAL', 'ADO', 'HLF', 'NLY', 'TWB', 'VNP', 'SEY', 'HNX', 'DKH',
                'HMY', 'CUA', 'CHH', 'TRA', 'FHE', 'CRK', 'UZB', 'SOZ', 'Z9H',
                'MXI', 'HRM', 'AEY', 'IBE', 'ITK', 'SOL', 'ISS', 'ILW', 'IOS',
                'IRA', 'AXB', 'IYE', 'AIZ', 'AHY', 'PLR', 'HCW', 'BVT', 'JZR',
                'BON', 'JBA', 'JAS', 'MNO', 'VIS', 'TAM', 'JAL', 'XLA', 'ADR',
                'JST', 'KOR', 'LNI', 'ASL', 'BLS', 'APW', 'AXZ', 'ELO', 'ORG',
                'SQH', 'BZL', 'HDA', 'DRK', 'KZR', 'KAL', 'KKK', 'KLM', 'AMC',
                'GCR', 'AER', 'KQA', 'CWK', 'PEN', 'KAC', 'CAY', 'KSY', 'MAI',
                'LAN', 'VLO', 'NDC', 'LGL', 'DLH', 'LIA', 'JNA', 'LAM', 'LAA',
                'LOT', 'LRC', 'EXS', 'LTU', 'LBC', 'NMI', 'SWR', 'ELY', 'TUS',
                '1QA', 'KEN', 'AJT', 'MSI', 'MDG', 'MEA', 'CXA', 'MAS', 'SLK',
                'LPR', 'MAU', 'GOE', 'CAW', 'AUH', 'OME', 'MSR', 'CES', 'MYD',
                'MWA', 'OMS', 'AVN', 'ANA', 'NHG', 'NKS', 'SAI', 'NIA', 'GEG',
                'IBB', 'JTA', 'AMU', 'FXI', 'ANZ', 'ONE', 'OAL', 'ASZ', 'EFY',
                'MXD', 'OLA', 'CSA', 'MGL', 'RON', 'TFL', 'AUA', 'CTN', 'ELL',
                'OEA', 'OAE', 'AAR', 'BXS', 'NDN', 'PGT', 'POE', 'AEL', 'BKP',
                'PDT', 'SPM', 'PIA', 'PLI', 'TOS', 'CHB', 'LOO', 'PAL', 'AUI',
                'PNR', 'PRF', 'ANG', 'SLM', 'LAP', 'JGN', 'QER', 'MLA', 'CDP',
                'JJP', 'PEC', 'NAK', 'GFG', 'NX1', 'QFA', 'EAV', 'FLZ', 'RLN',
                'ARR', 'QTR', 'TVS', 'UGX', 'LAO', 'AWQ', 'ORB', 'SYL', 'OCA',
                'RNA', 'FLI', 'VRN', 'MDL', 'RJA', 'RFJ', 'ROT', 'KMF', 'RPO',
                'RSH', 'BBR', 'RZO', 'TCF', 'SBI', 'SAA', 'ACI', 'CDG', 'SUD',
                'SEU', 'SEJ', 'SHA', 'SIH', 'SJY', 'SAS', 'DAT', 'SAT', 'SIA',
                'CRL', 'GMI', 'AFL', 'SVA', 'NMB', 'TJA', 'SCX', 'EZE', 'TUA',
                'TJT', 'TAT', 'TCX', 'LUR', 'LIL', 'SCW', 'THA', 'THY', 'ANO',
                'THT', 'TVF', 'TOM', 'TAP', 'TGW', 'TSC', 'TAR', 'VEX', 'FWI',
                'IWD', 'SCO', 'EZY', 'SVR', 'UAL', 'HER', 'NAS', 'TUI', 'LMU',
                'ALK', 'AZW', 'TSO', 'HKE', 'BHS', 'USA', 'REU', 'AEA', 'UYC',
                'VCV', 'KRP', 'REK', 'VOE', 'VOZ', 'VLU', 'HVN', 'VKH', 'TCV',
                'VIR', 'VTA', 'TAO', 'VRD', 'FOS', 'VJA', 'WSS', 'WER', 'IRM',
                'WZZ', 'JAB', 'RWD', 'WIF', 'WEB', 'SA1', 'SWA', 'MKU', 'WJA',
                'WAU', 'BMI', 'BCY', 'OMA', 'HLX', 'CHF', 'CCM', 'LNE', 'SXS',
                'VKJ', 'KNE', 'VOI', 'AWA', '1CH', 'MRS', 'IRK', 'YCC', 'SKV',
                'MGX', 'TYS', 'CUD', 'AA1', 'SMJ', 'OOM', 'ZTT', 'AZN', 'BUB',
                'ESR', 'CSZ', 'AAF', 'GLA', 'RXA', 'IWA'], dtype=object)
```

In [73]:

```
## Lets use airline_icao column to create keys
airln_icao_keys = df.airline_icao.sample(20).to_list()
airln_icao_keys
```

Out[73]:

```
['AEE',
 'M1F',
 'TOK',
 'PIA',
 'WSS',
 'KAP',
 'USA',
 'UZB',
 'NAX',
 'FOS',
 'GMI',
 'AZU',
 'SQH',
 'DAL',
 'FOS',
 'JSA',
 'CES',
 'IBE',
 'CSZ',
 'CHB']
```

In [75]:

```
## Lets create 4 partitions using earlier function: balance_partitions
airln_icao_partitions = balance_partitions(airln_icao_keys, 4)
airln_icao_partitions
```

Out[75]:

```
[['AEE', 'M1F', 'PIA', 'TOK', 'WSS'],
 ['FOS', 'KAP', 'NAX', 'USA', 'UZB'],
 ['AZU', 'DAL', 'FOS', 'GMI', 'SQH'],
 ['CES', 'CHB', 'CSZ', 'IBE', 'JSA']]
```