Assignment 07

Name: Kesav Adithya Venkidusamy
Course: DSC650 - Big Data
Instructor: Amirfarrokh Iranitalab

Assignment 7.1

In this part of the assignment, you will partition a dataset using different strategies. You will use the routes.parquet dataset you created in a previous assignment. For this dataset, the key for each route will be the three-letter source airport code concatenated with the three-letter destination airport code and the two-letter airline. For instance, a route from Omaha Eppley Airfield (OMA) to Denver International Airport (DEN) on American Airlines (AA) has a key of OMADENAA.

Assignment 7.1.a

Start by loading the dataset from the previous assignment using Pandas's read_parquet method. Next, add the concatenated key then using Panda's apply method to create a new column called key. For this part of the example, we will create 16 partitions so that we can compare it to the partitions we create from hashed keys in the next part of the assignment. The partitions are determined by the first letter of the composite key using the following partitions.

In [2]:
```
# Load all the 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 the route dataset

In [3]:
```
endpoint_url='https://storage.budsc.midwest-datascience.com'
current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')

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

In [7]:
```
## Function to process json file
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

def read_jsonl_data_local():
    '''Creating a function to read the file from local'''
    src_data_path = '/home/jovyan/dsc650/data/processed/openflights/routes.jsonl.gz'
    with 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
```

In [8]:
```
## Function to flatten 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():
    records = read_jsonl_data_local()
    parquet_path = results_dir.joinpath('routes-flattened.parquet')
    return pd.DataFrame.from_records([flatten_record(record) for record in records])
```

In [9]:

```
## Create dataframe and key field

df = create_flattened_dataset()
df['key'] = df['src_airport_iata'].astype(str) + df['dst_airport_iata'].astype(str) + df['airline_iata'].astype(str)
```

In [10]:

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

Out[10]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_country	airline_active	src_airport_airport_id	src_a
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2965.0	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala

5 rows × 39 columns

For this part of the example, we will create 16 partitions so that we can compare it to the partitions we create from hashed keys in the next part of the assignment. The partitions are determined by the first letter of the composite key using the following partitions.

In [11]:
```
## Defining 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')
        )
```

In this case ('A', 'A') means the folder should contain all of the routes whose composite key starts with A. Similarly, ('E', 'F') should contain routes whose composite key starts with E or F.

The results/kv directory should contain the following folders.

In [12]:
```
# 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
```

An easy way to create this directory structure is to create a new key called kv_key from the key column and use the to_parquet() method with partition_cols=['kv_key'] to save a partitioned dataset.

In [13]:
```
# Set up 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 [14]:
```
## Printing the values of partition_dict
partition_dict
```

Out[14]:
```
{('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 [15]:
```
# Generate kv_key from key
def kv_key_gen(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 [16]:
```
# Add kv_key column to df
df['kv_key'] = df['key'].apply(kv_key_gen)
```

In [17]:
```
## Showing sample records from dataframe
df.head()
```

Out[17]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_country	airline_active	src_airport_airport_id	src_a

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_country	airline_active	src_airport_airport_id	src_a
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2965.0	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala

5 rows × 40 columns

In [22]:
```
## Showing key and key-value from the dataframe
df[['key', 'kv_key']]
```

Out[22]:

	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

	key	kv_key
67660	FRUDMEZM	E-F
67661	FRUOSSZM	E-F
67662	OSSFRUZM	O-P

67663 rows × 2 columns

In [24]:
```
# Saving the dataframe in parquet format using kv_keys
try:
    df.to_parquet(results_dir.joinpath('kv'), partition_cols=['kv_key'])
except:
    print("The dataframe write operation has been failed")
else:
    print("The dataframe write operation to create partition is successful")
```

```
The dataframe write operation to create partition is successful
```

In [ ]:
```

```

Assignment 7.1.b

Next, we are going to partition the dataset again, but this time we will partition by the hash value of the key. The following is a function that will create a SHA256 hash of the input key and return a hexadecimal string representation of the hash.

In [25]:
```
import hashlib

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 [26]:
```
# 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 [27]:
```
# Add hash column to df
df['hashed'] = df['key'].apply(hash_key)
```

In [28]:
```
# Add hash key column to df for partitioning
df['hash_key'] = df['hashed'].str[0]
```

In [29]:
```
## showing few records from dataframe
df.head()
```

Out[29]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_country	airline_active	src_airport_airport_id	src_a
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2965.0	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_country	airline_active	src_airport_airport_id	src_a
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala

5 rows × 42 columns

In [31]:
```
# Partition dataset using hk_hash
try:
    df.to_parquet(results_dir.joinpath('hash'), partition_cols=['hash_key'])
except:
    print("The dataframe write operation to create hash partitions has been failed")
else:
    print("The dataframe write operation to create hash partitions is successful")
```

```
The dataframe write operation to create hash partitions is successful
```

In [ ]:
```

```

Assignment 7.1.c

Finally, we will simulate multiple geographically distributed data centers. For this example, we will assume we have three data centers located in the western, central, and eastern United States. Google lists the locations of their data centers and we will use the following locations for our three data centers.

West

- The Dalles, Oregon
- Latitude: 45.5945645
- Longitude: -121.1786823

Central

- Papillion, NE
- Latitude: 41.1544433
- Longitude: -96.0422378

East

- Loudoun County, Virginia
- Latitude: 39.08344
- Longitude: -77.6497145

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 [32]:
```
# geo
# ├── location=central
# ├── location=east
# └── location=west
```

In [35]:
```
df.columns
```

Out[35]:
```
Index(['airline_airline_id', 'airline_name', 'airline_alias', 'airline_iata',
       'airline_icao', 'airline_callsign', 'airline_country', 'airline_active',
       '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_altitude',
       '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_timezone',
       'dst_airport_dst', 'dst_airport_tz_id', 'dst_airport_type',
       'dst_airport_source', 'codeshare', 'equipment', 'key', 'kv_key',
       'hashed', 'hash_key'],
      dtype='object')
```

In [36]:
```
## Create a new column to calculare sourrce airport geo value
func = lambda x: pygeohash.encode(x.src_airport_latitude, x.src_airport_longitude)
df['geohash'] = df.apply(func, axis=1)
```

In [37]:
```

```

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

Out[37]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_country	airline_active	src_airport_airport_id	src_a
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2965.0	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala

5 rows × 43 columns

In [38]:

```
## Displaying new column values for few records
df['geohash'].head()
```

Out[38]:

```
0    szsrjjzd02b3
1    v04pk3t5gbjj
2    v04pk3t5gbjj
3    v3gdxs17du83
4    v3gdxs17du83
Name: geohash, dtype: object
```

In [40]:

```
## Defining the datacenters
# Get geohash location info for data centers
data_centers = dict(
    west = pygeohash.encode(45.5945645, -121.1786823),
    central = pygeohash.encode(41.1544433, -96.0422378),
    east = pygeohash.encode(39.08344, -77.6497145)
```

```
)
data_centers
```

Out[40]:
```
{'west': 'c21g6s0rs4c7', 'central': '9z7dnebnj8kb', 'east': 'dqby34cjw922'}
```

In [41]:
```
## Create a function to get closest datacenters from source airport
def get_dc_location(geohash):

    distance_dict= {}

    for key, val in data_centers.items():
        distance_dict[key] = pygeohash.geohash_haversine_distance(val, geohash)
    closest = sorted(distance_dict.items(), key=lambda x: x[1])[0][0]
    return closest
```

In [42]:
```
# Add column for closest data center
df['location'] = df['geohash'].apply(get_dc_location)
```

In [43]:
```
## Printing few records from dataframe
df.head()
```

Out[43]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_country	airline_active	src_airport_airport_id	src_
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2965.0	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala

5 rows × 44 columns

In [44]:
```
## Printing unique locations
df['location'].unique()
```

Out[44]:
```
array(['east', 'west', 'central'], dtype=object)
```

In [46]:
```
# Partition dataset using closest data center location
try:
    df.to_parquet(results_dir.joinpath('geo'), partition_cols=['location'])
except:
    print("The dataframe write operation to create geo partition has been failed")
else:
    print("The dataframe write operation to create geo partition is successful")
```

```
The dataframe write operation to create geo partition is successful
```

Assignment 7.1.d

Create a Python function that takes as input a list of keys and the number of partitions and returns a list of keys sorted into the specified number of partitions. The partitions should be roughly equal in size. Furthermore, the partitions should have the property that each partition contains all the keys between the least key in the partition and the greatest key in the partition. In other words, the partitions should be ordered.

In [53]:
```
## We will use itertools library to divide the given list into equal number of sub lists
from itertools import islice
```

In [59]:
```
## Function to create balance partitions
## Reference: https://www.geeksforgeeks.org/break-list-chunks-size-n-python/

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 [60]:
```
## Showing few sample records from dataframe
df.head()
```

Out[60]:

	airline_airline_id	airline_name	airline_alias	airline_iata	airline_icao	airline_callsign	airline_country	airline_active	src_airport_airport_id	src_a
0	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2965.0	
1	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
2	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2966.0	Astr
3	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala
4	410	Aerocondor	ANA All Nippon Airways	2B	ARD	AEROCONDOR	Portugal	True	2968.0	Bala

5 rows × 44 columns

In [64]:
```
## We will use airline_iata field present in the dataframe to create keys

airlines = df.airline_iata.sample(50).to_list()
airlines
```

Out[64]:
```
['AD',
 'EY',
 'IB',
 'TO',
 'GA',
 'CZ',
 'AY',
```

```
'KE',
'MF',
'KL',
'TS',
'G8',
'WN',
'EY',
'U6',
'TY',
'5J',
'W6',
'AZ',
'FL',
'DY',
'VA',
'S4',
'FR',
'CX',
'F9',
'KM',
'BA',
'S2',
'AA',
'CZ',
'AF',
'DY',
'RO',
'FR',
'EY',
'KL',
'AD',
'TG',
'B6',
'TP',
'B6',
'QF',
'AF',
'AR',
'VY',
'GS',
'HX',
'3U',
'TK']
```

In [65]:

```
## Create 5 partitions by calling balance_partitions
partitions = balance_partitions(airlines, 5)
partitions
```

Out[65]:
```
[['AD', 'AY', 'CZ', 'EY', 'GA', 'IB', 'KE', 'KL', 'MF', 'TO'],
 ['5J', 'AZ', 'EY', 'FL', 'G8', 'TS', 'TY', 'U6', 'W6', 'WN'],
 ['AA', 'BA', 'CX', 'DY', 'F9', 'FR', 'KM', 'S2', 'S4', 'VA'],
 ['AD', 'AF', 'B6', 'CZ', 'DY', 'EY', 'FR', 'KL', 'RO', 'TG'],
 ['3U', 'AF', 'AR', 'B6', 'GS', 'HX', 'QF', 'TK', 'TP', 'VY']]
```

In [66]:
```
## Create 3 partitions by calling balance_partitions
partitions = balance_partitions(airlines, 10)
partitions
```

Out[66]:
```
[['AD', 'EY', 'GA', 'IB', 'TO'],
 ['AY', 'CZ', 'KE', 'KL', 'MF'],
 ['EY', 'G8', 'TS', 'U6', 'WN'],
 ['5J', 'AZ', 'FL', 'TY', 'W6'],
 ['CX', 'DY', 'FR', 'S4', 'VA'],
 ['AA', 'BA', 'F9', 'KM', 'S2'],
 ['AF', 'CZ', 'DY', 'FR', 'RO'],
 ['AD', 'B6', 'EY', 'KL', 'TG'],
 ['AF', 'AR', 'B6', 'QF', 'TP'],
 ['3U', 'GS', 'HX', 'TK', 'VY']]
```

In [68]:
```
## Create 12 partitions by calling balance_partitions
partitions = balance_partitions(airlines, 12)
partitions
```

Out[68]:
```
[['AD', 'EY', 'IB', 'TO'],
 ['AY', 'CZ', 'GA', 'KE'],
 ['G8', 'KL', 'MF', 'TS'],
 ['EY', 'TY', 'U6', 'WN'],
 ['5J', 'AZ', 'FL', 'W6'],
 ['DY', 'FR', 'S4', 'VA'],
 ['BA', 'CX', 'F9', 'KM'],
 ['AA', 'AF', 'CZ', 'S2'],
 ['DY', 'EY', 'FR', 'RO'],
 ['AD', 'B6', 'KL', 'TG'],
 ['AF', 'B6', 'QF', 'TP'],
 ['AR', 'GS', 'HX', 'VY'],
 ['3U', 'TK']]
```

In [ ]:
```

```