DSC650-T301 Big Data (2235-1)

4/28/2023

Joshua Greenert

7.1.a

In [26]:
```
import pandas as pd

# use the read_parquet method
routes_df = pd.read_parquet("data/routes.parquet", engine='pyarrow')

# Normalize the data
routes_df = pd.json_normalize(routes_df["Flights"])

routes_df.head(5)
```

Out[26]:

	codeshare	equipment	airline.active	airline.airline_id	airline.alias	airline.callsign	airline.country	airline.iata	a
0	False	[CR2]	True	410	ANA All Nippon Airways	AEROCONDOR	Portugal	2B	
1	False	[CR2]	True	410	ANA All Nippon Airways	AEROCONDOR	Portugal	2B	
2	False	[CR2]	True	410	ANA All Nippon Airways	AEROCONDOR	Portugal	2B	
3	False	[CR2]	True	410	ANA All Nippon Airways	AEROCONDOR	Portugal	2B	
4	False	[CR2]	True	410	ANA All Nippon Airways	AEROCONDOR	Portugal	2B	

5 rows × 40 columns

In [27]:
```
# Create a method to use with apply
def key_gen(row):
    return str(row['src_airport.iata']) + str(row['dst_airport.iata']) + str(row['airlin
```

In [28]:
```
# Apply the method
routes_df["key"] = routes_df.apply(key_gen, axis=1)

# show the head of the file.
routes_df[['src_airport.iata', 'dst_airport.iata', 'airline.iata', 'key']].head(5)
```

Out[28]:

	src_airport.iata	dst_airport.iata	airline.iata	key
0	ASF	KZN	2B	ASFKZN2B
1	ASF	MRV	2B	ASFMRV2B

2	CEK	KZN	2B	CEKKZN2B
3	CEK	OVB	2B	CEKOVB2B
4	DME	KZN	2B	DMEKZN2B

In [29]:
```
# Set the 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 [30]:
```
# Define a function to set the partitions
def kv_key_gen(row):
    for tuple in partitions:
        first_char = row["key"][:1]
        if first_char >= tuple[0] and first_char <= tuple[1]:
            if tuple[0] == tuple[1]:
                return tuple[0]
            else:
                return tuple[0] + '-' + tuple[1]
```

In [31]:
```
# Use the defined method
routes_df["kv_key"] = routes_df.apply(kv_key_gen, axis=1)

# Show the head of the dataframe.
routes_df[['src_airport.iata', 'dst_airport.iata', 'airline.iata', 'key', 'kv_key']].hea
```

Out[31]:

	src_airport.iata	dst_airport.iata	airline.iata	key	kv_key
0	ASF	KZN	2B	ASFKZN2B	A
1	ASF	MRV	2B	ASFMRV2B	A
2	CEK	KZN	2B	CEKKZN2B	C-D
3	CEK	OVB	2B	CEKOVB2B	C-D
4	DME	KZN	2B	DMEKZN2B	C-D

In [32]:
```
# Set the folders for the partitions.
routes_df.to_parquet('results/kv/', partition_cols=['kv_key'])
```

7.1.b

In [33]:
```
import hashlib

def hash_key(key):
    m = hashlib.sha256()
    m.update(str(key).encode('utf-8'))
    return m.hexdigest()
```

In [34]:
```
# Create a new column called hashed
routes_df["hashed"] = routes_df.apply(lambda x: hash_key(x["key"]), axis=1)

# Show the head with the updated column
routes_df[['src_airport.iata', 'dst_airport.iata', 'airline.iata', 'key', 'hashed']].hea
```

Out[34]:

	src_airport.iata	dst_airport.iata	airline.iata	key	hashed

0	ASF	KZN	2B	ASFKZN2B	9eea5dd88177f8d835b2bb9cb27fb01268122b635b241a...
1	ASF	MRV	2B	ASFMRV2B	161143856af25bd4475f62c80c19f68936a139f653c1d3...
2	CEK	KZN	2B	CEKKZN2B	39aa99e6ae2757341bede9584473906ef1089e30820c90...
3	CEK	OVB	2B	CEKOVB2B	143b3389bce68eea3a13ac26a9c76c1fa583ec2bd26ea8...
4	DME	KZN	2B	DMEKZN2B	e4ec7b234cd26c4afd736cd49d1d02e4ec5f294f14533a...

In [35]:
```
# Create the hash key and save it to the results folder.
routes_df["hash_key"] = routes_df.apply(lambda x: x["hashed"][0].upper(), axis=1)

routes_df.to_parquet('results/hash/', partition_cols=['hash_key'])
```

7.1.c

In [44]:
```
import pygeohash as ph
import os

routes_df['src_airport_geohash'] = routes_df.apply(
    lambda row: ph.encode(row['src_airport.latitude'], row['src_airport.longitude']), ax
)
def determine_location(src_airport_geohash):
    locations = dict(
        central = ph.encode(41.1544433, -96.0422378),
        ## TODO: add west and east
        west = ph.encode(45.5945645, -121.1786823),
        east = ph.encode(39.08344, -77.6497145)
    )

    distances = [(loc, ph.geohash_haversine_distance(src_airport_geohash, geo)) for loc,

    distances.sort(key=lambda x: x[1])
    return distances[0][0]


routes_df['location'] = routes_df['src_airport_geohash'].apply(determine_location)

# Create the "results/geo" directory if it doesn't exist
os.makedirs('results/geo', exist_ok=True)

# Group the DataFrame by the 'location' column
grouped_routes_df = routes_df.groupby('location')

# Save each group in a separate Parquet file under its corresponding region directory
for location, group_df in grouped_routes_df:
    location_path = os.path.join('results/geo', f'location={location}')
    os.makedirs(location_path, exist_ok=True)
    group_df.to_parquet(os.path.join(location_path, f'location={location}.parquet'))
```

7.1.d

In [45]:
```
def balance_partitions(keys, num_partitions):
    partition_size = (len(keys) + num_partitions - 1) // num_partitions
    partitions = [keys[i:i + partition_size] for i in range(0, len(keys), partition_size
    partitions = [sorted(partition) for partition in partitions]
    return partitions
```

In [46]:
```
# Use the new method to balance the partitions.
balanced_partitions = balance_partitions(list(routes_df.key), 30)
```

```python
# show the size of all the partitions
for partition in balanced_partitions:
    print(len(partition))
```

```
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2256
2238
```