assignment07

February 7, 2022

[1]:
```
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
```

[2]:
```
endpoint_url = 'https://storage.budsc.midwest-datascience.com'
```

[14]:
```
current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')
```

[15]:
```
if results_dir.exists():
    shutil.rmtree(results_dir)
results_dir.mkdir(parents=True, exist_ok=True)
kv_dir = results_dir.joinpath('kv')
kv_dir.mkdir(parents=True, exist_ok=True)
```

[16]:
```
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
```

[17]:
```
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
```

[18]:
```
def create_flatten_dataset():
    records = read_jsonl_data()
    return pd.DataFrame.from_records([flatten_record(record) for record in␣
 ↪records])
```

[19]:
```
df = create_flatten_dataset()
df['key'] = df['src_airport_iata'].astype(str) + df['dst_airport_iata'].
 ↪astype(str) + df['airline_iata'].astype(str)
df['key_init'] = df['key'].astype(str).str[0]
```

[20]:
```
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')
)
```

7.a

[21]:
```
def kv_key(row):
    for letter in partitions:
        if row['key_init'] in letter:
            if letter[0] == letter[1]:
                # return (print(letter[0]))
                part = letter[0]
            else:
                # return (print(f'{letter[0]}-{letter[1]}'))
                part = f'{letter[0]}-{letter[1]}'

            return part

df['kv_key'] = df.apply(lambda row: kv_key(row), axis=1)
```

```
# df
```

[22]:
```
import pyarrow as pa
import pyarrow.parquet as pq

table = pa.Table.from_pandas(df)
```

[23]:
```
pq.write_to_dataset(
    table,
    root_path=f'{kv_dir}',
    partition_cols=['kv_key']
)
```

[24]:
```
parquet_path = results_dir.joinpath('routes-flattened.parquet')
# print(parquet_path)
pq.write_table(table,f'{parquet_path}')
```

7.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.

[25]:
```
import hashlib

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

[26]:
```
df[['key','key_init','kv_key']]
```

[26]:

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

[67663 rows x 3 columns]

[27]:
```
df['hashed'] = df.apply(lambda row: hash_key(row['key']), axis=1)
```

[28]:
```
df[['key','key_init','kv_key','hashed']]
```

[28]:

	key	key_init	kv_key	hashed
0	AERKZN2B	A	A	652cdec02010381f175efe499e070c8cbaac1522bac59a…
1	ASFKZN2B	A	A	9eea5dd88177f8d835b2bb9cb27fb01268122b635b241a…
2	ASFMRV2B	A	A	161143856af25bd4475f62c80c19f68936a139f653c1d3…
3	CEKKZN2B	C	C-D	39aa99e6ae2757341bede9584473906ef1089e30820c90…
4	CEKOVB2B	C	C-D	143b3389bce68eea3a13ac26a9c76c1fa583ec2bd26ea8…
…	…	…	…	…
67658	WYAADLZL	W	W-X	F31527be84c36208c05cac57dfac8a46b48a87dda151f8…
67659	DMEFRUZM	D	C-D	880fc35ca283ad034c90becc4e331b72ee894b9eb69f76…
67660	FRUDMEZM	F	E-F	E976939986fbf947bb9318018cef717c0b34dff91e5e67…
67661	FRUOSSZM	F	E-F	8b0c0b835a58a4250e020d51ec2a896e4ef3f5c3543b8e…
67662	OSSFRUZM	O	O-P	629f14f3fb6f94ebd1522d33a3c50675942e3148d028b4…

[67663 rows x 4 columns]

[29]:
```
df['hash_key'] = df['hashed'].astype(str).str[0]
df[['key','key_init','kv_key','hashed','hash_key']]
```

[29]:

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

```
0       652cdec02010381f175efe499e070c8cbaac1522bac59a…        6
1       9eea5dd88177f8d835b2bb9cb27fb01268122b635b241a…        9
2       161143856af25bd4475f62c80c19f68936a139f653c1d3…        1
3       39aa99e6ae2757341bede9584473906ef1089e30820c90…        3
4       143b3389bce68eea3a13ac26a9c76c1fa583ec2bd26ea8…        1
…                                                     …       …
67658   F31527be84c36208c05cac57dfac8a46b48a87dda151f8…        F
67659   880fc35ca283ad034c90becc4e331b72ee894b9eb69f76…        8
67660   E976939986fbf947bb9318018cef717c0b34dff91e5e67…        E
67661   8b0c0b835a58a4250e020d51ec2a896e4ef3f5c3543b8e…        8
67662   629f14f3fb6f94ebd1522d33a3c50675942e3148d028b4…        6

[67663 rows x 5 columns]
```

[30]:
```
hash_dir = results_dir.joinpath('hash')
hash_dir.mkdir(parents=True, exist_ok=True)
```

[31]:
```
table = pa.Table.from_pandas(df)

pq.write_to_dataset(
    table,
    root_path=f'{hash_dir}',
    partition_cols=['hash_key']
)
```

7.1.c

In the next few cells doing some EDA on the dataset and was planning to take in the US only airport locations.

[32]:
```
# df.info()
```

[33]:
```
# df.src_airport_country.value_counts()
```

[34]:
```
#  total number of missing values in each column
# df.isna().sum()
```

[35]:
```
# us_airports = df[df['src_airport_country'] == 'United States']
# us_airports.src_airport_city.value_counts()
```

[36]:
```
# us_airports.info()
```

[37]:
```
#␣
 ↪us_airports[['src_airport_name','src_airport_city','src_airport_country','src_airport_latit
```

[38]:
```
# def airport_search(latitude, longitude):
#     geohash = pygeohash.encode(latitude, longitude)
```

```
#     dist_dict = {}
#     for record in records:
#         dist = pygeohash.geohash_approximate_distance(str(geohash), str(record.
 ↪get('geohash')))
#         dist_dict[dist] = record.get('src_airport')
#
#         print(list(sorted(dist_dict.items()))[0][1]['name'])
#         pass

# airport_search(41.1499988, -95.91779)
```

[39]:
```
# Adding a new column to the dataframe keeping all the rows for the geohash

df['src_airport_geohash'] = df.apply( lambda row: pygeohash.encode(row.
 ↪src_airport_latitude, row.src_airport_longitude), axis=1)
```

[40]:
```
df[['src_airport_name','src_airport_city','src_airport_country','src_airport_latitude','src_ai
```

[40]:

	src_airport_name	src_airport_city	src_airport_country	\
0	Sochi International Airport	Sochi	Russia	
1	Astrakhan Airport	Astrakhan	Russia	
2	Astrakhan Airport	Astrakhan	Russia	
3	Chelyabinsk Balandino Airport	Chelyabinsk	Russia	
4	Chelyabinsk Balandino Airport	Chelyabinsk	Russia	
…	…	…	…	
67658	Whyalla Airport	Whyalla	Australia	
67659	Domodedovo International Airport	Moscow	Russia	
67660	Manas International Airport	Bishkek	Kyrgyzstan	
67661	Manas International Airport	Bishkek	Kyrgyzstan	
67662	Osh Airport	Osh	Kyrgyzstan	

	src_airport_latitude	src_airport_longitude	src_airport_geohash
0	43.449902	39.956600	szsrjjzd02b3
1	46.283298	48.006302	v04pk3t5gbjj
2	46.283298	48.006302	v04pk3t5gbjj
3	55.305801	61.503300	v3gdxs17du83
4	55.305801	61.503300	v3gdxs17du83
…	…	…	…
67658	-33.058899	137.514008	r41gcjy9uwef
67659	55.408798	37.906300	ucfgnwfe8u9e
67660	43.061298	74.477600	txsuyz0fjzgd
67661	43.061298	74.477600	txsuyz0fjzgd
67662	40.609001	72.793297	tx5z02wkwf2p

[67663 rows x 6 columns]

[41]:
```
# Creating a function to determine the airport location based on geohash key
def determine_location(src_airport_geohash):
    locations = dict(
        central=pygeohash.encode(41.1544433, -96.0422378),
        west=pygeohash.encode(45.5945645, -121.1786823),
        east=pygeohash.encode(39.08344, -77.6497145)
    )

    distances = []
    for key in locations:
        distance = pygeohash.
↪geohash_haversine_distance(src_airport_geohash,locations[key])
        distances.append([distance,key])

    distances.sort()
    return distances[0][1]
```

[42]:
```
df['location'] = df['src_airport_geohash'].apply(determine_location)
```

[43]:
```
df[['src_airport_name','src_airport_city','src_airport_country','src_airport_geohash','locatio
```

[43]:

	src_airport_name	src_airport_city	src_airport_country	\
0	Sochi International Airport	Sochi	Russia	
1	Astrakhan Airport	Astrakhan	Russia	
2	Astrakhan Airport	Astrakhan	Russia	
3	Chelyabinsk Balandino Airport	Chelyabinsk	Russia	
4	Chelyabinsk Balandino Airport	Chelyabinsk	Russia	
…	…	…	…	
67658	Whyalla Airport	Whyalla	Australia	
67659	Domodedovo International Airport	Moscow	Russia	
67660	Manas International Airport	Bishkek	Kyrgyzstan	
67661	Manas International Airport	Bishkek	Kyrgyzstan	
67662	Osh Airport	Osh	Kyrgyzstan	

	src_airport_geohash	location
0	szsrjjzd02b3	east
1	v04pk3t5gbjj	east
2	v04pk3t5gbjj	east
3	v3gdxs17du83	west
4	v3gdxs17du83	west
…	…	…
67658	r41gcjy9uwef	west
67659	ucfgnwfe8u9e	east
67660	txsuyz0fjzgd	west
67661	txsuyz0fjzgd	west
67662	tx5z02wkwf2p	west

```
[67663 rows x 5 columns]
```

[44]:
```
df.to_parquet('results/geo', partition_cols=['location'])
```

[ ]:
```
# determine_location('szsrjjzd02b3')
```

[29]:
```
# distances = pygeohash.
 ↪geohash_haversine_distance('szsrjjzd02b3','9z7dnebnj8kb')
# distances
```

[29]:
```
9628959.589672396
```

[45]:
```
# locations = dict(
#     central=pygeohash.encode(41.1544433, -96.0422378),
#    west=pygeohash.encode(45.5945645, -121.1786823),
#    east=pygeohash.encode(39.08344, -77.6497145)
# )
```

[46]:
```
# locations
```

[47]:
```
# distances = []
# for key in locations:
#    distance = pygeohash.
 ↪geohash_haversine_distance('szsrjjzd02b3',locations[key])
#    distances.append([distance,key])
    # print(key, '->', locations[key])

# distances
```

[48]:
```
# distances.sort()
# distances[0][1]
```

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.

[62]:
```
import numpy as np

def balance_partitions(keys, num_partitions):
    partitions = []

    partitions.append([np.array_split(keys, num_partitions)])
    return partitions
```

[64]:
```
keys = ['k1','k2','k3','k4','k5','k6','k7','k8','k9','k10']
num_partitions = 4

balance_partitions(keys,num_partitions)
```

[64]:
```
[[[array(['k1', 'k2', 'k3'], dtype='<U3'),
   array(['k4', 'k5', 'k6'], dtype='<U3'),
   array(['k7', 'k8'], dtype='<U3'),
   array(['k9', 'k10'], dtype='<U3')]]]
```

[65]:
```
keys = ['k1','k2','k3','k4','k5','k6','k7','k8','k9','k10']
num_partitions = 3

balance_partitions(keys,num_partitions)
```

[65]:
```
[[[array(['k1', 'k2', 'k3', 'k4'], dtype='<U3'),
   array(['k5', 'k6', 'k7'], dtype='<U3'),
   array(['k8', 'k9', 'k10'], dtype='<U3')]]]
```

[66]:
```
keys = ['k1','k2','k3','k4','k5','k6','k7','k8','k9','k10']
num_partitions = 5

balance_partitions(keys,num_partitions)
```

[66]:
```
[[[array(['k1', 'k2'], dtype='<U3'),
   array(['k3', 'k4'], dtype='<U3'),
   array(['k5', 'k6'], dtype='<U3'),
   array(['k7', 'k8'], dtype='<U3'),
   array(['k9', 'k10'], dtype='<U3')]]]
```