VRANJAN_Week7

May 2, 2021

[1]:
```
# Course DSC 650 - Data Mining
# Name - Vikas Ranjan
# Assignment - Assignment 7.1 - Dataset Partitioning
```

[2]:
```
import pandas as pd
from pathlib import Path
import gzip
import os
import json
import hashlib
import shutil
import pygeohash
import s3fs
```

1 A

[3]:
```
# Set the URL and directory path
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)

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 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()
    parquet_path = results_dir.joinpath('routes-flattened.parquet')
    return pd.DataFrame.from_records([flatten_record(record) for record in␣
 ↪records])
```

[4]:
```
# Create the dataset
df = create_flattened_dataset()
df['key'] = df['src_airport_iata'].astype(str) + df['dst_airport_iata'].
 ↪astype(str) + df['airline_iata'].astype(str)
```

[5]:
```
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')
    )
```

[6]:
```
# Remove NaN values from the dataset
df = df[df['src_airport_iata'].isna() == False]
```

[7]:
```
# Set kv-key equal to the first letter
df['kv_key'] = df['key'].str[0]

# Assign a value fromt he partitions list of tuples
df['kv_key'] = df['kv_key'].apply(lambda x: [str('-'.join(partition)) for␣
 ↪partition in partitions if (str(x) >= partition[0]) & (str(x) <=␣
 ↪partition[1])])

df['kv_key'] = [''.join(partition) for partition in df['kv_key']]

# Replace the partitions that have the same start and end with a single letter
```

```
df['kv_key'] = [partition[0] if partition[0] == partition[2] else partition for␣
↪partition in df['kv_key']]
```

[8]:
```
df.to_parquet(
    path='results/kv',
    partition_cols=['kv_key']
)
```

2 B

[9]:
```
# Define Hash key
def hash_key(key):
    m = hashlib.sha256()
    m.update(str(key).encode('utf-8'))
    return m.hexdigest()
```

[10]:
```
df['hashed'] = df['key'].apply(lambda x: hash_key(x))
df['hash_key'] = df['hashed'].str[0]
```

[11]:
```
df.head()
```

[11]:

	airline_airline_id	airline_name	airline_alias	airline_iata	\
0	410	Aerocondor	ANA All Nippon Airways	2B	
1	410	Aerocondor	ANA All Nippon Airways	2B	
2	410	Aerocondor	ANA All Nippon Airways	2B	
3	410	Aerocondor	ANA All Nippon Airways	2B	
4	410	Aerocondor	ANA All Nippon Airways	2B	

	airline_icao	airline_callsign	airline_country	airline_active	\
0	ARD	AEROCONDOR	Portugal	True	
1	ARD	AEROCONDOR	Portugal	True	
2	ARD	AEROCONDOR	Portugal	True	
3	ARD	AEROCONDOR	Portugal	True	
4	ARD	AEROCONDOR	Portugal	True	

	src_airport_airport_id	src_airport_name	…	dst_airport_dst	\
0	2965.0	Sochi International Airport	…	N	
1	2966.0	Astrakhan Airport	…	N	
2	2966.0	Astrakhan Airport	…	N	
3	2968.0	Chelyabinsk Balandino Airport	…	N	
4	2968.0	Chelyabinsk Balandino Airport	…	N	

	dst_airport_tz_id	dst_airport_type	dst_airport_source	codeshare	equipment	\
0	Europe/Moscow	airport	OurAirports	False	[CR2]	
1	Europe/Moscow	airport	OurAirports	False	[CR2]	
2	Europe/Moscow	airport	OurAirports	False	[CR2]	

```
3     Europe/Moscow          airport        OurAirports      False      [CR2]
4   Asia/Krasnoyarsk         airport        OurAirports      False      [CR2]

         key  kv_key                                              hashed  \
0   AERKZN2B       A  652cdec02010381f175efe499e070c8cbaac1522bac59a…
1   ASFKZN2B       A  9eea5dd88177f8d835b2bb9cb27fb01268122b635b241a…
2   ASFMRV2B       A  161143856af25bd4475f62c80c19f68936a139f653c1d3…
3   CEKKZN2B     C-D  39aa99e6ae2757341bede9584473906ef1089e30820c90…
4   CEKOVB2B     C-D  143b3389bce68eea3a13ac26a9c76c1fa583ec2bd26ea8…

  hash_key
0        6
1        9
2        1
3        3
4        1

[5 rows x 42 columns]
```

[12]:
```
df.to_parquet(
    path='results/hash',
    partition_cols=['hash_key']
 )
```

3 C

[13]:
```
# Get hash for datacenters
datacenters = {}

datacenters['west'] = pygeohash.encode(45.5945645, -121.1786823)
datacenters['central'] = pygeohash.encode(41.1544433, -96.0422378)
datacenters['east'] = pygeohash.encode(39.08344, -77.6497145)

print(datacenters)
```

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

[14]:
```
# Cycle through the datacenter dictionary to assign the closest
def closest_datacenter(latitude, longitude):
    geohash = pygeohash.encode(latitude, longitude)
    dist_dict = {}
    closest_datacenter = ''
    last_distance = None
    for key, value in datacenters.items():
        dist = pygeohash.geohash_approximate_distance(str(geohash), str(value))
        dist_dict[key] = dist
```

```
        if (last_distance == None) or (dist < last_distance):
            closest_datacenter = key
            last_distance = dist

    return closest_datacenter
```

[15]:
```
df['datacenter'] = df[['src_airport_latitude', 'src_airport_longitude']].
↪apply(lambda x: closest_datacenter(x[0], x[1]), axis=1)
```

[16]:
```
df.to_parquet(
    path='results/geo',
    partition_cols=['datacenter']
)
```

4 D

[17]:
```
# Define
def balance_partitions(keys, num_partitions):
    partitions = []

    #get the ideal number of records in each partition
    partition_size = len(keys) / num_partitions

    #get the count of records for each key
    key_grp_cnts = []
    for key in set(keys):
        occurences = keys.count(key)
        key_grp_cnts.append(tuple([key, occurences]))

    key_grp_cnts.sort(key=lambda v: v[0].lower())

    total = 0
    partition_list = []
    #loop through the group counts until you exceed partition_size
    for grp in key_grp_cnts:

        #if the total is 0, then this is the first key in the group
        if total == 0:
            min_grp = grp[0]
            last_group = grp[0]

       #if the incremented total exceeds the ideal partition size, then this␣
↪key is the max group and reset the total
        if  (total + grp[1]) > partition_size:
            max_grp = last_group
            partition_list.append(tuple([min_grp, max_grp]))
```

```
            last_group = grp[0]
            total=0
        else:
            last_group = grp[0]
            total += grp[1]

    #add last partition
    partition_list.append(tuple([min_grp, last_group]))

    return partition_list
```

[18]:
```
# Start by using a series from the df above as the list of keys
keys = list(df['airline_name'])
num_partitions=10
```

[19]:
```
# Show the results
print(balance_partitions(keys, num_partitions))
```

```
[('40-Mile Air', 'Air Foyle'), ('Air Greenland', 'Amaszonas'), ('Amerijet
International', 'China Eastern Airlines'), ('China SSS', 'Eurowings'), ('Excel
Airways', 'Jet Airways'), ('JetBlue Airways', 'Omni Air International'), ('Onur
Air', 'Shaheen Air International'), ('Shanghai Airlines', 'TransAsia Airways'),
('Transavia Holland', 'UTair-Express'), ('Valuair', 'Zoom Airlines')]
```

[ ]:
```

```