assignment07_FoxAndrea

May 2, 2021

Name: Andrea Fox
Date: April 29, 2021
Course: DSC650 - T301 Big Data
Assignment 07

[105]:

```python
#load libraries
import os
import json
from pathlib import Path
import gzip
import hashlib
import shutil
import pandas as pd
import pygeohash
import s3fs
```

[106]:

```python
endpoint_url='https://storage.budsc.midwest-datascience.com'

#set directories
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)

#Pulled from assignment03
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
```

0.1 7.1.a

Start by loading the dataset from the previous assignment using Pandas's read_parquet method. I

[107]:

```
#Worked with Jolene on this who took some samples from winter term slack channel
#create function to flatten record
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

#create function for flatten dataset
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])

#create dataframe and column key
df = create_flattened_dataset()
df['key'] = df['src_airport_iata'].astype(str) + df['dst_airport_iata'].
 ↪astype(str) + df['airline_iata'].astype(str)
```

[108]:

```
#Create 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')
    )
```

[109]:

```
#create new key kv_key
partition_dict = {}
for key in partitions:
    if key[0] == key[1]:
        kv_key = key[0]
    else:
        kv_key = key[0] + '-' + key[1]
    partition_dict[key] = kv_key

#wanted to make sure it looked correct
partition_dict
```

[109]:

```
{('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'}
```

[110]:

```
#create functon to get_key
def get_key(s_key):
    for key, value in partition_dict.items():
        if s_key[0] == key[0] or s_key[0] == key[1]:
            return value
    return ' '

#add kv_key column
df['kv_key'] = df['key'].apply(get_key)

#tested to make sure it worked
df.to_csv('test', sep = ',') #downloaded it and opened in excel and looked␣
 ↪accurate
```

[111]:

```
#use to_parquet method with partition_cols = ['kv_key'] to save partitioned␣
 ↪dataset
df.to_parquet(os.getcwd() + '/results/kv.parquet', partition_cols = ['kv_key'])
```

0.2 7.1 b

Next, we are going to partition the dataset again, but this time we will partition by the hash

We will partition the data using the first character of the hexadecimal hash. As such, there a

[112]:
```
#load libraries for part b
import hashlib
```

[113]:
```
#create SHA256 hash of the input key and return hexadecimal string rep of hash
def hash_key(key):
    m = hashlib.sha256()
    m.update(str(key).encode('utf-8'))
    return m.hexdigest()
```

[114]:
```
#create hashed and hash_key column. Found an old example for hashed and hash_key
df['key'] = df['src_airport_iata'].astype(str) + df['dst_airport_iata'].
 ↪astype(str) + df['airline_iata'].astype(str)
df['hashed'] = df.apply(lambda x: hash_key(x.key), axis=1)
df['hash_key'] = df['hashed'].str[:1]
```

I did this several times and had to remove the hash.parquet a couple of times because I had fo

[115]:
```
#created csv to test it worked
df.to_csv('hash_test1', sep = ',')
```

[116]:
```
#using the to_parquet again but changing partition_cols = hash_key instead of␣
 ↪kv_key like previous section
df.to_parquet(os.getcwd() + '/results/hash.parquet', partition_cols =␣
 ↪['hash_key'])
```

0.3 7.1 c

Finally, we will simulate multiple geographically distributed data centers. For this example, w

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 yo

[117]:
```
! pip install geolib
```

```
Requirement already satisfied: geolib in /opt/conda/lib/python3.8/site-packages
```

```
(1.0.7)
Requirement already satisfied: future in /opt/conda/lib/python3.8/site-packages
(from geolib) (0.18.2)
```

[118]:

```
#load libraries needed for c
import pandas as pd
import numpy as np
import sklearn.neighbors
from geolib import geohash
```

[119]:

```
df['src_airport_geohash'] = df.apply(
    lambda row: pygeohash.encode(row.src_airport_latitude, row.
 ↪src_airport_longitude), axis=1
)
def determine_location(src_airport_geohash):
    locations = dict(
        central = pygeohash.encode(41.1544433, -96.0422378),
        east = pygeohash.encode(39.08344, -77.6497145),
        west = pygeohash.encode(45.5945645, -121.1786823)
    )
    #Got this from Corinne
    distances = []
    for location, geohash in locations.items():
        hav = pygeohash.geohash_haversine_distance(src_airport_geohash, geohash)
        distances.append(tuple((hav, location)))


    distances.sort()
    return distances[0][1]
df['location'] = df['src_airport_geohash'].apply(determine_location)

#Create csv to verify it looks accurate
df.to_csv('geo_test', sep = ',')
```

[122]:

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

0.4 7.1 d

Create a Python function that takes as input a list of keys and the number of partitions and re

[72]:

```
#Used some code from github as reference as well as this website https://www.
 ↪geeksforgeeks.org/partition-problem-dp-18/
def balance_partitions (keys, num_partitions):
    vals = sorted(set(keys))
    num_vals = len(vals)
    partition_counts = (num_vals / num_partitions)+1
    partitions  = []
    x = 1
    y = 1
    for i in range(num_vals):
        key_val ={}
        if x <= partition_counts:
            key_val[vals[i]] = y
            x = x + 1
        else:
            x = 1
            y = y + 1
            key_val[vals[i]] = y
            x = x + 1
        partitions.append(key_val)
    return partitions

#create list of keys
keys = ['duck', 'chicken', 'pig', 'rabbit', 'horse', 'cow', 'donkey', 'cat',␣
 ↪'dog', 'goose', 'mouse']
#set number of partitions
num_partitions = 3

#create partitions and then print
partitions  = balance_partitions(keys, num_partitions)
print(partitions)
```

```
[{'cat': 1}, {'chicken': 1}, {'cow': 1}, {'dog': 1}, {'donkey': 2}, {'duck': 2},
{'goose': 2}, {'horse': 2}, {'mouse': 3}, {'pig': 3}, {'rabbit': 3}]
```

[73]:

```
#change number of paritions
num_partitions = 2

#create partitions and then print
partitions = balance_partitions(keys, num_partitions)
print(partitions)
```

```
[{'cat': 1}, {'chicken': 1}, {'cow': 1}, {'dog': 1}, {'donkey': 1}, {'duck': 1},
{'goose': 2}, {'horse': 2}, {'mouse': 2}, {'pig': 2}, {'rabbit': 2}]
```