GOLDSHTEYNJOSHUA_JULIANSINGKHAM_CS2300_Assignment7

January 4, 2021

1 Pandas Performance

In this notebook we will be exploring the performance differences between different approaches of iterating through a Pandas column. This is based on a post: https://engineering.upside.com/a-beginners-guide-to-optimizing-pandas-code-for-speed-c09ef2c6a4d6

First we will start by loading our data. The data is from Lyft's Go Bike program and inclues every trip from 2017: https://www.lyft.com/bikes/bay-wheels/system-data

[1]:
```
import pandas as pd

df = pd.read_csv('/data/cs2300/examples/2017-fordgobike-tripdata.csv',
                 dtype={"start_station_latitude":float,␣
 ↪"start_station_longitude":float,
                        "end_station_latitude":float, "end_station_longitude":
 ↪float})
```

Next we define a function to calculate distance based on two GPS locations

[2]:
```
import numpy as np

# Define a basic Haversine distance formula
def haversine(lat1, lon1, lat2, lon2):
    MILES = 3959
    lat1, lon1, lat2, lon2 = map(np.deg2rad, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
    total_miles = MILES * c
    return total_miles
```

The slowest approach is to loop through the dataframe using `iloc`

[3]:
```
def haversine_looping(df):
    distance_list = []
    for i in range(0, len(df)):
```

```
        d = haversine(df['start_station_latitude'].iloc[i],␣
↪df['start_station_longitude'].iloc[i],
                          df['end_station_latitude'].iloc[i],␣
↪df['end_station_longitude'].iloc[i])
        distance_list.append(d)
    return distance_list
%time df['distance'] = haversine_looping(df)
```

```
CPU times: user 23.6 s, sys: 196 ms, total: 23.8 s
Wall time: 23.8 s
```

Next, lets try using `iterrows()`

[4]:
```
%%time
haversine_series = []
for index, row in df.iterrows():
    haversine_series.append(haversine(row['start_station_latitude'],␣
↪row['start_station_longitude'],
                                      row['end_station_latitude'],␣
↪row['end_station_longitude']))
df['distance'] = haversine_series
```

```
CPU times: user 49.4 s, sys: 612 ms, total: 50 s
Wall time: 50 s
```

Next, lets use some functional programming! Try using `apply`

[5]:
```
%time df['distance'] = df.apply(lambda row:␣
↪haversine(row['start_station_latitude'], \
                                                                  ␣
↪row['start_station_longitude'], \
                                                                  ␣
↪row['end_station_latitude'], \
                                                                  ␣
↪row['end_station_longitude']), axis=1)
```

```
CPU times: user 19.8 s, sys: 132 ms, total: 19.9 s
Wall time: 19.9 s
```

Lets vectorize!

[6]:
```
%time df['distance'] = haversine(df['start_station_latitude'],␣
↪df['start_station_longitude'], \
                                 df['end_station_latitude'],␣
↪df['end_station_longitude'])
```

```
CPU times: user 126 ms, sys: 176 ms, total: 302 ms
Wall time: 171 ms
```

Lets try numpy vectorize

[7]:
```
%time df['distance'] = haversine(df['start_station_latitude'],␣
↪df['start_station_longitude'], \
                                    df['end_station_latitude'].values,␣
↪df['end_station_longitude'].values)
```

```
CPU times: user 103 ms, sys: 4.03 ms, total: 107 ms
Wall time: 68.2 ms
```

Create a table summarizing the performance results

[8]:
```
#Restarted and ran all before submission, so numbers may be slightly different
dfperf = pd.DataFrame()
dfperf["Type"] = ["iloc","iterrows()","apply","vectorize","numpy vectorize"]
dfperf["CPU"] = [21.5, 44, 18.6, 0.0948, 0.0987]
dfperf["Sys"] = [0.00618, 0.120, 0.132, 0.072, 0.00799]
dfperf["Total"] = [21.5, 44.1, 18.7, 0.167, 0.107]
dfperf["Wall"] = [21.5, 44.1, 18.7, 0.104, 0.0719]
dfperf.head()
```

[8]:

	Type	CPU	Sys	Total	Wall
0	iloc	21.5000	0.00618	21.500	21.5000
1	iterrows()	44.0000	0.12000	44.100	44.1000
2	apply	18.6000	0.13200	18.700	18.7000
3	vectorize	0.0948	0.07200	0.167	0.1040
4	numpy vectorize	0.0987	0.00799	0.107	0.0719