5.1-Working with Date and Times in Pandas

KeytoDataScience.com

Dates and Times in Pandas

To conclude this session, we'll apply everything we've learned about working with dates and times in standard Python to working with dates and times in Pandas. With additional information about each bike ride, such as what station it started and stopped at and whether or not the rider had a yearly membership, we'll be able to dig much more deeply into the bike trip data. In this notebook, we'll cover powerful Pandas operations, such as grouping and plotting results by time.

Table of Contents

💡💡💡💡💡 **Let's take an interesting example to understand Datetime Concept** 💡💡💡💡💡

---

1 Reading datetime data in Pandas

Loading a csv file in Pandas

The 202006-capitalbikeshare-tripdata.csv file covers the June'20 rides of the Capital Bikeshare.

In [1]:
```
# Import pandas
import pandas as pd

# Load CSV into the rides variable
rides = pd.read_csv('capitalbikeshare.csv')
```

```
# Print the dataset
rides.head()
```

Out[1]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_stat
0	16A86B048B01DE6F	docked_bike	2020-06-10 20:20:36	2020-06-10 20:27:28	Edgewood Rec Center	642.0	Rhode
1	661EA082175DB7E9	docked_bike	2020-06-23 06:31:23	2020-06-23 06:58:35	Wisconsin Ave & O St NW	289.0	Edge
2	15C659A06C439B74	docked_bike	2020-06-06 11:49:29	2020-06-06 11:49:38	19th & K St NW	269.0	19th &
3	59AD75CFBF96DEC1	docked_bike	2020-06-06 11:52:51	2020-06-06 12:21:49	19th & K St NW	269.0	10th &
4	06F6881BEFCFC106	docked_bike	2020-06-07 15:25:45	2020-06-07 17:15:28	37th & O St NW / Georgetown University	152.0	Ge Harbo

In [2]:

```
# Print the dataset info
print(rides.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213995 entries, 0 to 213994
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   ride_id             213995 non-null  object 
 1   rideable_type       213995 non-null  object 
 2   started_at          213995 non-null  object 
 3   ended_at            213995 non-null  object 
 4   start_station_name  213943 non-null  object 
 5   start_station_id    213943 non-null  float64
 6   end_station_name    212884 non-null  object 
 7   end_station_id      212884 non-null  float64
 8   start_lat           213995 non-null  float64
 9   start_lng           213995 non-null  float64
 10  end_lat             212936 non-null  float64
 11  end_lng             212936 non-null  float64
 12  member_casual       213995 non-null  object 
dtypes: float64(6), object(7)
memory usage: 21.2+ MB
None
```

📌 As you can see above `started_at and ended_at` columns are treated **as objects** and **not as datetimes**

To convert these columns as datetime format we can use any of the **2 methods:**

Method 1: We will be using `parse_dates` to convert them into **datetime format**.

In [3]:
```
# Load CSV into the rides variable
rides = pd.read_csv('capitalbikeshare.csv', parse_dates = ['started_at', 'ended_at'])

# Print the dataset
rides.head()
```

Out[3]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_stat
0	16A86B048B01DE6F	docked_bike	2020-06-10 20:20:36	2020-06-10 20:27:28	Edgewood Rec Center	642.0	Rhode
1	661EA082175DB7E9	docked_bike	2020-06-23 06:31:23	2020-06-23 06:58:35	Wisconsin Ave & O St NW	289.0	Edge
2	15C659A06C439B74	docked_bike	2020-06-06 11:49:29	2020-06-06 11:49:38	19th & K St NW	269.0	19th &
3	59AD75CFBF96DEC1	docked_bike	2020-06-06 11:52:51	2020-06-06 12:21:49	19th & K St NW	269.0	10th &
4	06F6881BEFCFC106	docked_bike	2020-06-07 15:25:45	2020-06-07 17:15:28	37th & O St NW / Georgetown University	152.0	Ge Harbo

In [4]:
```
# Print the dataset info
print(rides.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213995 entries, 0 to 213994
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             213995 non-null  object
 1   rideable_type       213995 non-null  object
 2   started_at          213995 non-null  datetime64[ns]
 3   ended_at            213995 non-null  datetime64[ns]
 4   start_station_name  213943 non-null  object
 5   start_station_id    213943 non-null  float64
 6   end_station_name    212884 non-null  object
 7   end_station_id      212884 non-null  float64
 8   start_lat           213995 non-null  float64
 9   start_lng           213995 non-null  float64
 10  end_lat             212936 non-null  float64
 11  end_lng             212936 non-null  float64
 12  member_casual       213995 non-null  object
dtypes: datetime64[ns](2), float64(6), object(5)
memory usage: 21.2+ MB
None
```

Now we can verify that **`started_at and ended_at`** columns are stored as **datetime**

💬**OR**💬

Method 2: We can manually parse the format of the datetime column using **`pd.to_datetime`**

In [5]:
```
# Load CSV into the rides variable
rides = pd.read_csv('capitalbikeshare.csv')

rides['started_at'] = pd.to_datetime(rides['started_at'], format='%Y-%m-%d %H:%M:%S')
rides['ended_at'] = pd.to_datetime(rides['ended_at'], format='%Y-%m-%d %H:%M:%S')

rides.head()
```

Out[5]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_stat
0	16A86B048B01DE6F	docked_bike	2020-06-10 20:20:36	2020-06-10 20:27:28	Edgewood Rec Center	642.0	Rhode
1	661EA082175DB7E9	docked_bike	2020-06-23 06:31:23	2020-06-23 06:58:35	Wisconsin Ave & O St NW	289.0	Edge
2	15C659A06C439B74	docked_bike	2020-06-06 11:49:29	2020-06-06 11:49:38	19th & K St NW	269.0	19th &
3	59AD75CFBF96DEC1	docked_bike	2020-06-06 11:52:51	2020-06-06 12:21:49	19th & K St NW	269.0	10th &
4	06F6881BEFCFC106	docked_bike	2020-06-07 15:25:45	2020-06-07 17:15:28	37th & O St NW / Georgetown University	152.0	Ge Harbo

In [6]:
```
print(rides.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213995 entries, 0 to 213994
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype         
---  ------              --------------   -----         
 0   ride_id             213995 non-null  object        
 1   rideable_type       213995 non-null  object        
 2   started_at          213995 non-null  datetime64[ns]
 3   ended_at            213995 non-null  datetime64[ns]
 4   start_station_name  213943 non-null  object        
 5   start_station_id    213943 non-null  float64       
 6   end_station_name    212884 non-null  object        
 7   end_station_id      212884 non-null  float64       
 8   start_lat           213995 non-null  float64       
 9   start_lng           213995 non-null  float64       
 10  end_lat             212936 non-null  float64       
 11  end_lng             212936 non-null  float64       
 12  member_casual       213995 non-null  object        
dtypes: datetime64[ns](2), float64(6), object(5)
memory usage: 21.2+ MB
None
```

↥ back to top

2 Operations of datetime columns

📌 **Q.** We want to determine how long the bikes had been out of the docks. We'll use Pandas to subtract datetime objects.

In [7]:
```
# Subtract the start date from the end date
ride_durations = rides['ended_at'] - rides['started_at']
```

In [8]:
```
ride_durations
```

Out[8]:
```
0        0 days 00:06:52
1        0 days 00:27:12
2        0 days 00:00:09
3        0 days 00:28:58
4        0 days 01:49:43
               ...
213990   0 days 00:52:46
213991   0 days 00:09:43
213992   0 days 00:16:17
213993   0 days 00:23:21
213994   0 days 01:07:42
Length: 213995, dtype: timedelta64[ns]
```

In [9]:
```
import datetime as dt

# Convert the results to seconds
rides['Duration'] = ride_durations.dt.total_seconds()

print(rides['Duration'].head())
```

```
0     412.0
1    1632.0
2       9.0
3    1738.0
4    6583.0
Name: Duration, dtype: float64
```

↥ back to top

3 Summarizing datetime (with examples)

3.1 Q. How many joyrides?

Suppose you have a theory that some people do not take bike ride to go to any other destination point and park there. They just take long bike rides before putting their bike back in the same dock. Let's call these rides `"joyrides"` .

You only have data on one bike, so while you can't draw any bigger conclusions, it's certainly worth a look.

Are there many joyrides? How long were they in our data set? Use the median instead of the mean, because we know there are some very long trips in our data set that might skew the answer, and the median is less sensitive to outliers.

In [10]:
```
# Create joyrides
joyrides = (rides['start_station_name'] == rides['end_station_name'])

# Total number of joyrides
print("{} rides were joyrides".format(joyrides.sum()))

# Median of all rides
print("The median duration overall was {:.2f} seconds"\
      .format(rides['Duration'].median()))

# Median of joyrides
print("The median duration for joyrides was {:.2f} seconds"\
      .format(rides[joyrides]['Duration'].median()))
```

```
32966 rides were joyrides
The median duration overall was 1074.00 seconds
The median duration for joyrides was 2227.00 seconds
```

3.2 Q. Find the daily number of rides

Method1: Using Groupby

Fetch and aggregate data at Day Level, then count rides.

Read more on Group by

In [11]:
```
rides.groupby(rides.started_at.dt.date).agg(
                            daily_rides = ('ride_id','count')
                                 )
```

Out[11]:

started_at	daily_rides
2020-06-01	5012
2020-06-02	5385
2020-06-03	6512
2020-06-04	5163
2020-06-05	5044
2020-06-06	15475
2020-06-07	10632
2020-06-08	7018
2020-06-09	6754
2020-06-10	5410
2020-06-11	4699
2020-06-12	7423
2020-06-13	12097
2020-06-14	10332

started_at	daily_rides
2020-06-15	6041
2020-06-16	6203
2020-06-17	2910
2020-06-18	5902
2020-06-19	7259
2020-06-20	6304
2020-06-21	9036
2020-06-22	4958
2020-06-23	6189
2020-06-24	7399
2020-06-25	6069
2020-06-26	8095
2020-06-27	9057
2020-06-28	8589
2020-06-29	6364
2020-06-30	6664

Method2: Using Resample

In [12]:
```
# alternate method to achieve the same
# Read more at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.resample.
rides.resample('D', on = 'started_at')['ride_id'].count()
```

Out[12]:
```
started_at
2020-06-01     5012
2020-06-02     5385
2020-06-03     6512
2020-06-04     5163
2020-06-05     5044
2020-06-06    15475
2020-06-07    10632
2020-06-08     7018
2020-06-09     6754
2020-06-10     5410
2020-06-11     4699
2020-06-12     7423
2020-06-13    12097
2020-06-14    10332
2020-06-15     6041
2020-06-16     6203
2020-06-17     2910
2020-06-18     5902
2020-06-19     7259
```

```
2020-06-20     6304
2020-06-21     9036
2020-06-22     4958
2020-06-23     6189
2020-06-24     7399
2020-06-25     6069
2020-06-26     8095
2020-06-27     9057
2020-06-28     8589
2020-06-29     6364
2020-06-30     6664
Freq: D, Name: ride_id, dtype: int64
```

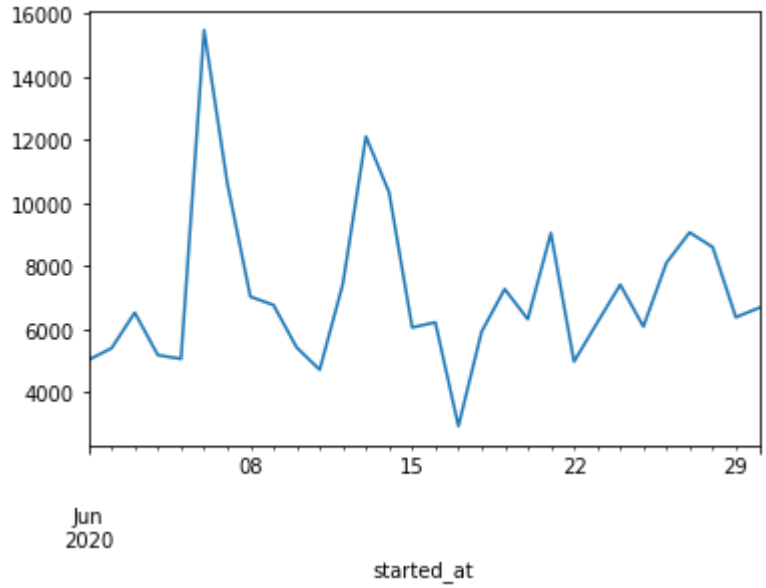
Plotting the data to easily understand the output

In [13]:

```
# Import matplotlib
import matplotlib.pyplot as plt

# Resample rides to daily, take the count, plot the results
rides.resample('D', on = 'started_at')['ride_id'].count().plot()

# Show the results
plt.show()
```



3.3 Q. Find Members vs Casual riders over time

Riders can either be "Members", meaning they pay yearly for the ability to take a bike at any time, or "Casual", meaning they pay at the kiosk attached to the bike dock.

In [14]:

```
# Resample rides to be daily on the basis of Start date
daily_rides = rides.resample('D', on='started_at')['member_casual']

# Take the ratio of the .value_counts() over the total number of rides
daily_rides.value_counts()/ daily_rides.count()
```

Out[14]:

```
started_at  member_casual
2020-06-01  member          0.506584
            casual          0.493416
```

```
2020-06-02  casual          0.528505
            member          0.471495
2020-06-03  casual          0.560504
            member          0.439496
2020-06-04  casual          0.522371
            member          0.477629
2020-06-05  casual          0.500397
            member          0.499603
2020-06-06  casual          0.671276
            member          0.328724
2020-06-07  casual          0.631678
            member          0.368322
2020-06-08  casual          0.571388
            member          0.428612
2020-06-09  casual          0.568552
            member          0.431448
2020-06-10  member          0.520333
            casual          0.479667
2020-06-11  member          0.530326
            casual          0.469674
2020-06-12  casual          0.511788
            member          0.488212
2020-06-13  casual          0.619492
            member          0.380508
2020-06-14  casual          0.614305
            member          0.385695
2020-06-15  casual          0.504387
            member          0.495613
2020-06-16  member          0.522005
            casual          0.477995
2020-06-17  member          0.592096
            casual          0.407904
2020-06-18  member          0.536767
            casual          0.463233
2020-06-19  casual          0.529274
            member          0.470726
2020-06-20  casual          0.539499
            member          0.460501
2020-06-21  casual          0.578575
            member          0.421425
2020-06-22  member          0.553247
            casual          0.446753
2020-06-23  member          0.536112
            casual          0.463888
2020-06-24  member          0.513988
            casual          0.486012
2020-06-25  member          0.528753
            casual          0.471247
2020-06-26  casual          0.512168
            member          0.487832
2020-06-27  casual          0.588826
            member          0.411174
2020-06-28  casual          0.576086
            member          0.423914
2020-06-29  member          0.520113
            casual          0.479887
2020-06-30  member          0.512155
            casual          0.487845
Name: member_casual, dtype: float64
```

3.4 Combining groupby() and resample()

A very powerful method in Pandas is `.groupby()`.

Whereas `.resample()` groups rows by some time or date information, `.groupby()` groups rows based on the values in one or more columns.

- For example, `rides.groupby('Member type').size()` would tell us how many rides there were by member type in our entire DataFrame.

`.resample()` can be called after `.groupby()`.

- For example, how long was the median ride by day, and by Membership type?

Q. How long was the median ride by day, and by Membership type?

In [15]:
```
# Group rides by member type, and resample to the daily
grouped = rides.groupby('member_casual').resample('D', on='started_at')

# Print the median duration for each group
grouped['Duration'].median()
```

Out[15]:
```
member_casual  started_at
casual         2020-06-01    1474.0
               2020-06-02    1442.5
               2020-06-03    1472.5
               2020-06-04    1335.0
               2020-06-05    1291.5
               2020-06-06    1574.0
               2020-06-07    1715.0
               2020-06-08    1594.0
               2020-06-09    1499.0
               2020-06-10    1404.0
               2020-06-11    1329.0
               2020-06-12    1385.0
               2020-06-13    1666.0
               2020-06-14    1665.0
               2020-06-15    1447.0
               2020-06-16    1376.0
               2020-06-17    1026.0
               2020-06-18    1341.0
               2020-06-19    1313.5
               2020-06-20    1449.0
               2020-06-21    1579.5
               2020-06-22    1245.0
               2020-06-23    1336.0
               2020-06-24    1340.0
               2020-06-25    1232.0
               2020-06-26    1306.0
               2020-06-27    1498.0
               2020-06-28    1497.5
               2020-06-29    1249.0
               2020-06-30    1274.0
member         2020-06-01     802.0
               2020-06-02     814.0
               2020-06-03     825.0
               2020-06-04     763.0
```

```
2020-06-05     745.0
2020-06-06     892.0
2020-06-07     894.0
2020-06-08     784.5
2020-06-09     779.5
2020-06-10     735.0
2020-06-11     742.5
2020-06-12     770.0
2020-06-13     886.0
2020-06-14     865.0
2020-06-15     757.0
2020-06-16     744.5
2020-06-17     658.0
2020-06-18     739.0
2020-06-19     746.0
2020-06-20     787.0
2020-06-21     849.5
2020-06-22     658.0
2020-06-23     731.0
2020-06-24     719.0
2020-06-25     667.0
2020-06-26     727.0
2020-06-27     773.5
2020-06-28     763.0
2020-06-29     673.0
2020-06-30     685.0
Name: Duration, dtype: float64
```

↥ back to top

4 Additional datetime methods in Pandas

4.1 Find median ride duration of weekdays?

Pandas has a number of datetime-related attributes within the **`.dt`** accessor.

These can be easily used, like `.dt.month` .

Others are convenient and save time compared to standard Python, like `.dt.day_name()` .

Read more regarding `.dt` accessor

In [16]:

```python
# Add a column for the weekday of the start of the ride
rides['ride_start_weekday'] = rides['started_at'].dt.day_name()

# Print the median trip time per weekday
print(rides.groupby('ride_start_weekday')['Duration'].median())
```

```
ride_start_weekday
Friday        998.0
Monday        993.0
Saturday     1238.0
Sunday       1243.0
Thursday      937.0
Tuesday       996.0
Wednesday     977.0
Name: Duration, dtype: float64
```

↥ back to top

Great Job!
