Muhammad Bilal Iqbal

July 3, 2020

1 Working with Dates and Times in Python

You'll probably never have a time machine, but how about a machine for analyzing time? As soon as time enters any analysis, things can get weird. It's easy to get tripped up on day and month boundaries, time zones, daylight saving time, and all sorts of other things that can confuse the unprepared. If you're going to do any kind of analysis involving time, you'll want to use Python to sort it out. Working with data sets on hurricanes and bike trips, we'll cover counting events, figuring out how much time has elapsed between events and plotting data over time. You'll work in both standard Python and in Pandas, and we'll touch on the dateutil library, the only timezone library endorsed by the official Python documentation.

Dates and Calendars

Hurricanes (also known as cyclones or typhoons) hit the U.S. state of Florida several times per year. I will work with date objects in Python, starting with the dates of every hurricane to hit Florida since 1950. I'll handle dates, common date operations, and the right way to format dates to avoid confusion.

Which day of the week?

Hurricane Andrew, which hit Florida on August 24, 1992, was one of the costliest and deadliest hurricanes in US history. Which day of the week did it make landfall?

Importing date from datetime.

Creating a date object for August 24, 1992.

Now I will ask Python what day of the week Hurricane Andrew hit (Python counts days of the week starting from Monday as 0, Tuesday as 1, and so on).

[8]:
```python
# Importing date from datetime
from datetime import date

# Creating a date object
hurricane_andrew = date(1992, 8, 24)

# Which day of the week is the date?
print(hurricane_andrew.weekday())
```

```
0
```

In python datatime library 0 is considered Monday.

[1]:
```
import pandas as pd
import pickle
```

[3]:
```
import pickle
with open('florida_hurricane_dates.pkl', 'rb') as f:
    data = pickle.load(f)
```

[4]:
```
florida_hurricane_dates = data
```

[6]:
```
florida_hurricane_dates
```

[6]:
```
[datetime.date(1988, 8, 4),
 datetime.date(1990, 10, 12),
 datetime.date(2003, 4, 20),
 datetime.date(1971, 9, 1),
 datetime.date(1988, 8, 23),
 datetime.date(1994, 8, 15),
 datetime.date(2002, 8, 4),
 datetime.date(1988, 5, 30),
 datetime.date(2003, 9, 13),
 datetime.date(2009, 8, 21),
 datetime.date(1978, 6, 22),
 datetime.date(1969, 6, 9),
 datetime.date(1976, 6, 11),
 datetime.date(1976, 8, 19),
 datetime.date(1966, 6, 9),
 datetime.date(1968, 7, 5),
 datetime.date(1987, 11, 4),
 datetime.date(1988, 8, 13),
 datetime.date(2007, 12, 13),
 datetime.date(1994, 11, 16),
 datetime.date(2003, 9, 6),
 datetime.date(1971, 8, 13),
 datetime.date(1981, 8, 17),
 datetime.date(1998, 9, 25),
 datetime.date(1968, 9, 26),
 datetime.date(1968, 6, 4),
 datetime.date(1998, 11, 5),
 datetime.date(2008, 8, 18),
 datetime.date(1987, 8, 14),
 datetime.date(1988, 11, 23),
 datetime.date(2010, 9, 29),
 datetime.date(1985, 7, 23),
 datetime.date(2017, 7, 31),
 datetime.date(1955, 8, 21),
 datetime.date(1986, 6, 26),
 datetime.date(1963, 10, 21),
 datetime.date(2011, 10, 28),
 datetime.date(2011, 11, 9),
 datetime.date(1997, 7, 19),
 datetime.date(2007, 6, 2),
 datetime.date(2002, 9, 14),
 datetime.date(1992, 9, 29),
 datetime.date(1971, 10, 13),
 datetime.date(1962, 8, 26),
 datetime.date(1964, 8, 27),
 datetime.date(1984, 9, 27),
 datetime.date(1973, 9, 25),
 datetime.date(1969, 10, 21),
 datetime.date(1994, 7, 3),
 datetime.date(1958, 9, 4),
 datetime.date(1985, 11, 21),
 datetime.date(2011, 9, 3),
 datetime.date(1972, 6, 19),
 datetime.date(1991, 6, 30),
 datetime.date(2004, 8, 12),
 datetime.date(2007, 9, 8),
 datetime.date(1952, 2, 3),
 datetime.date(1965, 9, 30),
 datetime.date(2000, 9, 22),
 datetime.date(2002, 9, 26),
 datetime.date(1950, 9, 5),
 datetime.date(1966, 10, 4),
 datetime.date(1970, 5, 25),
 datetime.date(1979, 9, 24),
 datetime.date(1960, 9, 23),
 datetime.date(2007, 8, 23),
 datetime.date(2009, 8, 16),
 datetime.date(1996, 10, 18),
 datetime.date(2012, 10, 25),
 datetime.date(2011, 8, 25),
 datetime.date(1951, 5, 18),
 datetime.date(1980, 8, 7),
 datetime.date(1979, 9, 3),
 datetime.date(1953, 9, 26),
 datetime.date(1968, 10, 19),
 datetime.date(2009, 11, 9),
 datetime.date(1999, 8, 29),
 datetime.date(2015, 10, 1),
 datetime.date(2008, 9, 2),
 datetime.date(2004, 10, 10),
 datetime.date(2004, 9, 16),
 datetime.date(1992, 8, 24),
 datetime.date(2000, 9, 9),
 datetime.date(1971, 9, 16),
 datetime.date(1996, 9, 2),
 datetime.date(1998, 9, 3),
 datetime.date(1951, 10, 2),
 datetime.date(1979, 9, 12),
 datetime.date(2007, 10, 31),
 datetime.date(1953, 10, 9),
 datetime.date(1952, 8, 30),
 datetime.date(1969, 9, 7),
 datetime.date(2015, 8, 30),
 datetime.date(1959, 10, 8),
 datetime.date(2002, 7, 13),
 datetime.date(1961, 10, 29),
 datetime.date(2007, 5, 9),
 datetime.date(2016, 10, 7),
 datetime.date(1964, 9, 20),
 datetime.date(1979, 7, 11),
 datetime.date(1950, 10, 18),
 datetime.date(2008, 8, 31),
 datetime.date(2012, 8, 25),
 datetime.date(1966, 7, 24),
 datetime.date(2010, 8, 10),
 datetime.date(2005, 8, 25),
 datetime.date(2003, 6, 30),
 datetime.date(1956, 7, 6),
 datetime.date(1974, 9, 8),
 datetime.date(1966, 6, 30),
 datetime.date(2016, 9, 14),
 datetime.date(1968, 6, 18),
 datetime.date(1982, 9, 11),
 datetime.date(1976, 9, 13),
 datetime.date(1975, 7, 29),
 datetime.date(2007, 9, 13),
 datetime.date(1970, 9, 27),
 datetime.date(1969, 10, 2),
 datetime.date(2010, 8, 31),
 datetime.date(1995, 10, 4),
 datetime.date(1969, 8, 29),
 datetime.date(1984, 10, 26),
 datetime.date(1973, 9, 3),
 datetime.date(1976, 5, 23),
 datetime.date(2001, 11, 5),
 datetime.date(2010, 6, 30),
 datetime.date(1985, 10, 10),
 datetime.date(1970, 7, 22),
 datetime.date(1972, 5, 28),
 datetime.date(1982, 6, 18),
 datetime.date(2001, 8, 6),
 datetime.date(1953, 8, 29),
 datetime.date(1965, 9, 8),
 datetime.date(1964, 9, 10),
 datetime.date(1959, 10, 18),
 datetime.date(1957, 6, 8),
 datetime.date(1988, 9, 10),
 datetime.date(2005, 6, 11),
 datetime.date(1953, 6, 6),
 datetime.date(2003, 8, 30),
 datetime.date(2002, 10, 3),
 datetime.date(1968, 8, 10),
 datetime.date(1999, 10, 15),
 datetime.date(2002, 9, 4),
 datetime.date(2001, 6, 12),
 datetime.date(2017, 9, 10),
 datetime.date(2005, 10, 5),
 datetime.date(2005, 7, 10),
 datetime.date(1973, 6, 7),
 datetime.date(1999, 9, 15),
 datetime.date(2005, 9, 20),
 datetime.date(1995, 6, 5),
 datetime.date(2003, 7, 25),
 datetime.date(2004, 9, 13),
 datetime.date(1964, 6, 6),
 datetime.date(1973, 6, 23),
 datetime.date(2005, 9, 12),
 datetime.date(2012, 6, 23),
 datetime.date(1961, 9, 11),
 datetime.date(1990, 5, 25),
 datetime.date(2017, 6, 21),
 datetime.date(1975, 6, 27),
 datetime.date(1959, 6, 18),
 datetime.date(2004, 9, 5),
 datetime.date(1987, 10, 12),
 datetime.date(1995, 7, 27),
 datetime.date(1964, 10, 14),
 datetime.date(1970, 8, 6),
 datetime.date(1969, 10, 1),
 datetime.date(1996, 10, 8),
 datetime.date(1968, 8, 28),
 datetime.date(1956, 10, 15),
 datetime.date(1975, 9, 23),
 datetime.date(1970, 9, 13),
 datetime.date(1975, 10, 16),
 datetime.date(1990, 10, 9),
 datetime.date(2005, 10, 24),
 datetime.date(1950, 8, 31),
 datetime.date(2000, 10, 3),
 datetime.date(2002, 10, 11),
 datetime.date(1983, 8, 28),
 datetime.date(1960, 7, 29),
 datetime.date(1950, 10, 21),
 datetime.date(1995, 8, 2),
 datetime.date(1956, 9, 24),
 datetime.date(2016, 9, 1),
 datetime.date(1993, 6, 1),
 datetime.date(1987, 9, 7),
 datetime.date(2012, 5, 28),
 datetime.date(1995, 8, 23),
 datetime.date(1969, 8, 18),
 datetime.date(2001, 9, 14),
 datetime.date(2000, 8, 23),
 datetime.date(1974, 10, 7),
 datetime.date(1986, 8, 13),
 datetime.date(1977, 8, 27),
 datetime.date(2008, 7, 16),
 datetime.date(1996, 7, 11),
 datetime.date(1988, 9, 4),
 datetime.date(1975, 10, 1),
 datetime.date(2003, 8, 14),
 datetime.date(1957, 9, 8),
 datetime.date(2005, 7, 6),
 datetime.date(1960, 9, 15),
 datetime.date(1974, 9, 27),
 datetime.date(1965, 6, 15),
 datetime.date(1999, 9, 21),
 datetime.date(2004, 8, 13),
 datetime.date(1994, 10, 2),
 datetime.date(1971, 8, 10),
 datetime.date(2008, 7, 22),
 datetime.date(2000, 9, 18),
 datetime.date(1960, 9, 10),
 datetime.date(2006, 6, 13),
 datetime.date(2017, 10, 29),
 datetime.date(1972, 9, 5),
 datetime.date(1964, 10, 5),
 datetime.date(1991, 10, 16),
 datetime.date(1969, 9, 21),
 datetime.date(1998, 9, 20),
 datetime.date(1977, 9, 5),
 datetime.date(1988, 9, 13),
 datetime.date(1974, 6, 25),
 datetime.date(2010, 7, 23),
 datetime.date(2007, 9, 22),
 datetime.date(1984, 9, 9),
 datetime.date(1989, 9, 22),
 datetime.date(1992, 6, 25),
 datetime.date(1971, 8, 29),
 datetime.date(1953, 9, 20),
 datetime.date(1985, 8, 15),
 datetime.date(2016, 6, 6),
 datetime.date(2006, 8, 30),
 datetime.date(1980, 11, 18),
 datetime.date(2011, 7, 18)]
```

How many hurricanes come early?

Here I will work with a list of the hurricanes that made landfall in Florida from 1950 to 2017. There were 235 in total. Checking out the variable florida_hurricane_dates, which has all of these dates.

Atlantic hurricane season officially begins on June 1. How many hurricanes since 1950 have made landfall in Florida before the official start of hurricane season?

[7]:
```
# Counter for how many before June 1
early_hurricanes = 0

# We loop over the dates
for hurricane in florida_hurricane_dates:
  # Check if the month is before June (month number 6)
  if hurricane.month < 6:
    early_hurricanes = early_hurricanes + 1

print(early_hurricanes)
```

```
10
```

Using this same pattern, we could do more complex counting by using .day, .year, .weekday(), and so on.

Subtracting dates

Python date objects let us treat calendar dates as something similar to numbers: we can compare them, sort them, add, and even subtract them. This lets us do math with dates in a way that would be a pain to do by hand.

The 2007 Florida hurricane season was one of the busiest on record, with 8 hurricanes in one year. The first one hit on May 9th, 2007, and the last one hit on December 13th, 2007. How many days elapsed between the first and last hurricane in 2007?

[17]:
```
# Importing date
from datetime import date

# Creating a date object for May 9th, 2007
```

```
start = date(2007, 5, 9)

# Creating a date object for December 13th, 2007
end = date(2007, 12, 13)

# Subtracting the two dates and print the number of days
print((end - start).days)
```

```
218
```

One thing to note: using this technique for historical dates hundreds of years in the past. Our calendar systems have changed over time, and not every date from then would be the same day and month today.

```
[ ]: ###### Counting events per calendar month

     Hurricanes can make landfall in Florida throughout the year. As I've already␣
      ↪discussed, some months are more hurricane-prone than others.

     Using florida_hurricane_dates, let's see how hurricanes in Florida were␣
      ↪distributed across months throughout the year.

     I've created a dictionary called hurricanes_each_month to hold the counts and␣
      ↪set the initial counts to zero. I will loop over the list of hurricanes,␣
      ↪incrementing the correct month in hurricanes_each_month as I go, and then␣
      ↪print the result.
```

```
[18]: # A dictionary to count hurricanes per calendar month
      hurricanes_each_month = {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6:0,
                               7: 0, 8:0, 9:0, 10:0, 11:0, 12:0}

      # Looping over all hurricanes
      for hurricane in florida_hurricane_dates:
        # Pulling out the month
        month = hurricane.month
        # Incrementing the count in your dictionary by one
        hurricanes_each_month[month] += 1

      print(hurricanes_each_month)
```

```
{1: 0, 2: 1, 3: 0, 4: 1, 5: 8, 6: 32, 7: 21, 8: 49, 9: 70, 10: 43, 11: 9, 12: 1}
```

Success! This illustrated a generally useful pattern for working with complex data: creating a dictionary, performing some operation on each element, and storing the results back in the dictionary.

```
[ ]: Putting a list of dates in order
```

```
Much like numbers and strings, date objects in Python can be put in order.␣
↪Earlier dates come before later ones, and so we can sort a list of date␣
↪objects from earliest to latest.

What if our Florida hurricane dates had been scrambled? I will go ahead and␣
↪shuffled them so they're in random order and saved the results as␣
↪dates_scrambled. I will put them back in chronological order, and then print␣
↪the first and last dates from this sorted list.
```

[23]:
```
import datetime
```

[27]:
```
dates_scrambled = [datetime.date(1988, 8, 4),
 datetime.date(1990, 10, 12),
 datetime.date(2003, 4, 20),
 datetime.date(1971, 9, 1),
 datetime.date(1988, 8, 23),
 datetime.date(1994, 8, 15),
 datetime.date(2002, 8, 4),
 datetime.date(1988, 5, 30),
 datetime.date(2003, 9, 13),
 datetime.date(2009, 8, 21),
 datetime.date(1978, 6, 22),
 datetime.date(1969, 6, 9),
 datetime.date(1976, 6, 11),
 datetime.date(1976, 8, 19),
 datetime.date(1966, 6, 9),
 datetime.date(1968, 7, 5),
 datetime.date(1987, 11, 4),
 datetime.date(1988, 8, 13),
 datetime.date(2007, 12, 13),
 datetime.date(1994, 11, 16),
 datetime.date(2003, 9, 6),
 datetime.date(1971, 8, 13),
 datetime.date(1981, 8, 17),
 datetime.date(1998, 9, 25),
 datetime.date(1968, 9, 26),
 datetime.date(1968, 6, 4),
 datetime.date(1998, 11, 5),
 datetime.date(2008, 8, 18),
 datetime.date(1987, 8, 14),
 datetime.date(1988, 11, 23),
 datetime.date(2010, 9, 29),
 datetime.date(1985, 7, 23),
 datetime.date(2017, 7, 31),
 datetime.date(1955, 8, 21),
 datetime.date(1986, 6, 26),
 datetime.date(1963, 10, 21),
datetime.date(2011, 10, 28),
datetime.date(2011, 11, 9),
datetime.date(1997, 7, 19),
datetime.date(2007, 6, 2),
datetime.date(2002, 9, 14),
datetime.date(1992, 9, 29),
datetime.date(1971, 10, 13),
datetime.date(1962, 8, 26),
datetime.date(1964, 8, 27),
datetime.date(1984, 9, 27),
datetime.date(1973, 9, 25),
datetime.date(1969, 10, 21),
datetime.date(1994, 7, 3),
datetime.date(1958, 9, 4),
datetime.date(1985, 11, 21),
datetime.date(2011, 9, 3),
datetime.date(1972, 6, 19),
datetime.date(1991, 6, 30),
datetime.date(2004, 8, 12),
datetime.date(2007, 9, 8),
datetime.date(1952, 2, 3),
datetime.date(1965, 9, 30),
datetime.date(2000, 9, 22),
datetime.date(2002, 9, 26),
datetime.date(1950, 9, 5),
datetime.date(1966, 10, 4),
datetime.date(1970, 5, 25),
datetime.date(1979, 9, 24),
datetime.date(1960, 9, 23),
datetime.date(2007, 8, 23),
datetime.date(2009, 8, 16),
datetime.date(1996, 10, 18),
datetime.date(2012, 10, 25),
datetime.date(2011, 8, 25),
datetime.date(1951, 5, 18),
datetime.date(1980, 8, 7),
datetime.date(1979, 9, 3),
datetime.date(1953, 9, 26),
datetime.date(1968, 10, 19),
datetime.date(2009, 11, 9),
datetime.date(1999, 8, 29),
datetime.date(2015, 10, 1),
datetime.date(2008, 9, 2),
datetime.date(2004, 10, 10),
datetime.date(2004, 9, 16),
datetime.date(1992, 8, 24),
datetime.date(2000, 9, 9),
datetime.date(1971, 9, 16),
datetime.date(1996, 9, 2),
datetime.date(1998, 9, 3),
datetime.date(1951, 10, 2),
datetime.date(1979, 9, 12),
datetime.date(2007, 10, 31),
datetime.date(1953, 10, 9),
datetime.date(1952, 8, 30),
datetime.date(1969, 9, 7),
datetime.date(2015, 8, 30),
datetime.date(1959, 10, 8),
datetime.date(2002, 7, 13),
datetime.date(1961, 10, 29),
datetime.date(2007, 5, 9),
datetime.date(2016, 10, 7),
datetime.date(1964, 9, 20),
datetime.date(1979, 7, 11),
datetime.date(1950, 10, 18),
datetime.date(2008, 8, 31),
datetime.date(2012, 8, 25),
datetime.date(1966, 7, 24),
datetime.date(2010, 8, 10),
datetime.date(2005, 8, 25),
datetime.date(2003, 6, 30),
datetime.date(1956, 7, 6),
datetime.date(1974, 9, 8),
datetime.date(1966, 6, 30),
datetime.date(2016, 9, 14),
datetime.date(1968, 6, 18),
datetime.date(1982, 9, 11),
datetime.date(1976, 9, 13),
datetime.date(1975, 7, 29),
datetime.date(2007, 9, 13),
datetime.date(1970, 9, 27),
datetime.date(1969, 10, 2),
datetime.date(2010, 8, 31),
datetime.date(1995, 10, 4),
datetime.date(1969, 8, 29),
datetime.date(1984, 10, 26),
datetime.date(1973, 9, 3),
datetime.date(1976, 5, 23),
datetime.date(2001, 11, 5),
datetime.date(2010, 6, 30),
datetime.date(1985, 10, 10),
datetime.date(1970, 7, 22),
datetime.date(1972, 5, 28),
datetime.date(1982, 6, 18),
datetime.date(2001, 8, 6),
datetime.date(1953, 8, 29),
datetime.date(1965, 9, 8),
datetime.date(1964, 9, 10),
datetime.date(1959, 10, 18),
datetime.date(1957, 6, 8),
datetime.date(1988, 9, 10),
datetime.date(2005, 6, 11),
datetime.date(1953, 6, 6),
datetime.date(2003, 8, 30),
datetime.date(2002, 10, 3),
datetime.date(1968, 8, 10),
datetime.date(1999, 10, 15),
datetime.date(2002, 9, 4),
datetime.date(2001, 6, 12),
datetime.date(2017, 9, 10),
datetime.date(2005, 10, 5),
datetime.date(2005, 7, 10),
datetime.date(1973, 6, 7),
datetime.date(1999, 9, 15),
datetime.date(2005, 9, 20),
datetime.date(1995, 6, 5),
datetime.date(2003, 7, 25),
datetime.date(2004, 9, 13),
datetime.date(1964, 6, 6),
datetime.date(1973, 6, 23),
datetime.date(2005, 9, 12),
datetime.date(2012, 6, 23),
datetime.date(1961, 9, 11),
datetime.date(1990, 5, 25),
datetime.date(2017, 6, 21),
datetime.date(1975, 6, 27),
datetime.date(1959, 6, 18),
datetime.date(2004, 9, 5),
datetime.date(1987, 10, 12),
datetime.date(1995, 7, 27),
datetime.date(1964, 10, 14),
datetime.date(1970, 8, 6),
datetime.date(1969, 10, 1),
datetime.date(1996, 10, 8),
datetime.date(1968, 8, 28),
datetime.date(1956, 10, 15),
datetime.date(1975, 9, 23),
datetime.date(1970, 9, 13),
datetime.date(1975, 10, 16),
datetime.date(1990, 10, 9),
datetime.date(2005, 10, 24),
datetime.date(1950, 8, 31),
datetime.date(2000, 10, 3),
datetime.date(2002, 10, 11),
datetime.date(1983, 8, 28),
datetime.date(1960, 7, 29),
datetime.date(1950, 10, 21),
datetime.date(1995, 8, 2),
datetime.date(1956, 9, 24),
datetime.date(2016, 9, 1),
datetime.date(1993, 6, 1),
datetime.date(1987, 9, 7),
datetime.date(2012, 5, 28),
datetime.date(1995, 8, 23),
datetime.date(1969, 8, 18),
datetime.date(2001, 9, 14),
datetime.date(2000, 8, 23),
datetime.date(1974, 10, 7),
datetime.date(1986, 8, 13),
datetime.date(1977, 8, 27),
datetime.date(2008, 7, 16),
datetime.date(1996, 7, 11),
datetime.date(1988, 9, 4),
datetime.date(1975, 10, 1),
datetime.date(2003, 8, 14),
datetime.date(1957, 9, 8),
datetime.date(2005, 7, 6),
datetime.date(1960, 9, 15),
datetime.date(1974, 9, 27),
datetime.date(1965, 6, 15),
datetime.date(1999, 9, 21),
datetime.date(2004, 8, 13),
datetime.date(1994, 10, 2),
datetime.date(1971, 8, 10),
datetime.date(2008, 7, 22),
datetime.date(2000, 9, 18),
datetime.date(1960, 9, 10),
datetime.date(2006, 6, 13),
datetime.date(2017, 10, 29),
datetime.date(1972, 9, 5),
datetime.date(1964, 10, 5),
datetime.date(1991, 10, 16),
datetime.date(1969, 9, 21),
datetime.date(1998, 9, 20),
datetime.date(1977, 9, 5),
datetime.date(1988, 9, 13),
datetime.date(1974, 6, 25),
datetime.date(2010, 7, 23),
 datetime.date(2007, 9, 22),
 datetime.date(1984, 9, 9),
 datetime.date(1989, 9, 22),
 datetime.date(1992, 6, 25),
 datetime.date(1971, 8, 29),
 datetime.date(1953, 9, 20),
 datetime.date(1985, 8, 15),
 datetime.date(2016, 6, 6),
 datetime.date(2006, 8, 30),
 datetime.date(1980, 11, 18),
 datetime.date(2011, 7, 18)]
```

[28]:
```
# Printing the first and last scrambled dates
print(dates_scrambled[0])
print(dates_scrambled[-1])

# Putting the dates in order
dates_ordered = sorted(dates_scrambled)

# Printing the first and last ordered dates
print(dates_ordered[0])
print(dates_ordered[-1])
```

```
1988-08-04
2011-07-18
1950-08-31
2017-10-29
```

We can use sorted() on several data types in Python, including sorting lists of numbers, lists of strings, or even lists of lists, which by default are compared on the first element.

Printing dates in a friendly format

Because people may want to see dates in many different formats, Python comes with very flexible functions for turning date objects into strings.

Let's see what event was recorded first in the Florida hurricane data set. Here I will format the earliest date in the florida_hurriance_dates list in two ways so we can decide which one we want to use: either the ISO standard or the typical US style.

[29]:
```
# Assigning the earliest date to first_date
first_date = min(florida_hurricane_dates)

# Converting to ISO and US formats
iso = "Our earliest hurricane date: " + first_date.isoformat()
us = "Our earliest hurricane date: " + first_date.strftime("%m/%d/%Y")

print("ISO: " + iso)
print("US: " + us)
```

```
ISO: Our earliest hurricane date: 1950-08-31
US: Our earliest hurricane date: 08/31/1950
```

When in doubt, we should use the ISO format for dates. ISO dates are unambiguous. And if you sort them 'alphabetically', for example, in filenames, they will be in the correct order.

Representing dates in different ways

date objects in Python have a great number of ways they can be printed out as strings. In some cases, we want to know the date in a clear, language-agnostic format. In other cases, we want something which can fit into a paragraph and flow naturally.

Let's try printing out the same date, August 26, 1992 (the day that Hurricane Andrew made landfall in Florida), in a number of different ways, to practice using the .strftime() method.

A date object called andrew has already been created.

1. Printing andrew in the format 'YYYY-MM'.
2. Printing andrew in the format 'MONTH (YYYY)', where MONTH is the full name (%B).
3. Printing andrew in the format 'YYYY-DDD' where DDD is the day of the year.

Hint - %Y is the 4-digit year and %j is the 3-digit day of the year.

[31]:
```python
# Creating a date object
andrew = date(1992, 8, 26)

# Printing the date in the format 'YYYY-MM'
print(andrew.strftime('%Y-%m'))

# Printing the date in the format 'MONTH (YYYY)'
print(andrew.strftime('%B (%Y)'))


# Printing the date in the format 'YYYY-DDD'
print(andrew.strftime("%Y-%j"))
```

```
1992-08
August (1992)
1992-239
```

We can pick the format that best matches our needs. For example, astronomers usually use the 'day number' out of 366 instead of the month and date, to avoid ambiguities between languages.

Creating datetimes by hand

Often we create datetime objects based on outside data. Sometimes though, we want to create a datetime object from scratch.

I am going to create a few different datetime objects from scratch to get the hang of that process. These come from the bikeshare data set that I'll use throughout the rest of the document.

Creating a datetime for October 1, 2017 at 15:26:26. Printing the results in ISO format.

Creating a datetime for December 31, 2017 at 15:19:13. Printing the results in ISO format.

Creating a new datetime by replacing the year in dt with 1917 (instead of 2017)

[34]:
```
# Importing datetime
from datetime import datetime

# Creating a datetime object
dt = datetime(2017, 10, 1, 15, 26, 26)

# Printing the results in ISO 8601 format
print(dt.isoformat())

# Creating a datetime object
dt = datetime(2017, 12, 31, 15, 19, 13)

# Printing the results in ISO 8601 format
print(dt.isoformat())

# Creating a datetime object
dt = datetime(2017, 12, 31, 15, 19, 13)

# Replacing the year with 1917
dt_old = dt.replace(1917, 12, 31, 15, 19, 13)

# Printing the results in ISO 8601 format
print(dt_old)
```

```
2017-10-01T15:26:26
2017-12-31T15:19:13
1917-12-31 15:19:13
```

Counting events before and after noon

I will be working with a list of all bike trips for one Capital Bikeshare bike, W20529, from October 1, 2017 to December 31, 2017. This list has been loaded as onebike_datetimes.

Each element of the list is a dictionary with two entries: start is a datetime object corresponding to the start of a trip (when a bike is removed from the dock) and end is a datetime object corresponding to the end of a trip (when a bike is put back into a dock).

We can use this data set to understand better how this bike was used. Did more trips start before noon or after noon?

[42]:
```
from datetime import datetime
from datetime import datetime, timedelta
import datetime
onebike_datetimes = [{'end': datetime.datetime(2017, 10, 1, 15, 26, 26),
  'start': datetime.datetime(2017, 10, 1, 15, 23, 25)},
 {'end': datetime.datetime(2017, 10, 1, 17, 49, 59),
  'start': datetime.datetime(2017, 10, 1, 15, 42, 57)},
 {'end': datetime.datetime(2017, 10, 2, 6, 42, 53),
  'start': datetime.datetime(2017, 10, 2, 6, 37, 10)},
 {'end': datetime.datetime(2017, 10, 2, 9, 18, 3),
  'start': datetime.datetime(2017, 10, 2, 8, 56, 45)},
 {'end': datetime.datetime(2017, 10, 2, 18, 45, 5),
  'start': datetime.datetime(2017, 10, 2, 18, 23, 48)},
 {'end': datetime.datetime(2017, 10, 2, 19, 10, 54),
  'start': datetime.datetime(2017, 10, 2, 18, 48, 8)},
 {'end': datetime.datetime(2017, 10, 2, 19, 31, 45),
  'start': datetime.datetime(2017, 10, 2, 19, 18, 10)},
 {'end': datetime.datetime(2017, 10, 2, 19, 46, 37),
  'start': datetime.datetime(2017, 10, 2, 19, 37, 32)},
 {'end': datetime.datetime(2017, 10, 3, 8, 32, 27),
  'start': datetime.datetime(2017, 10, 3, 8, 24, 16)},
 {'end': datetime.datetime(2017, 10, 3, 18, 27, 46),
  'start': datetime.datetime(2017, 10, 3, 18, 17, 7)},
 {'end': datetime.datetime(2017, 10, 3, 19, 52, 8),
  'start': datetime.datetime(2017, 10, 3, 19, 24, 10)},
 {'end': datetime.datetime(2017, 10, 3, 20, 23, 52),
  'start': datetime.datetime(2017, 10, 3, 20, 17, 6)},
 {'end': datetime.datetime(2017, 10, 3, 20, 57, 10),
  'start': datetime.datetime(2017, 10, 3, 20, 45, 21)},
 {'end': datetime.datetime(2017, 10, 4, 7, 13, 31),
  'start': datetime.datetime(2017, 10, 4, 7, 4, 57)},
 {'end': datetime.datetime(2017, 10, 4, 7, 21, 54),
  'start': datetime.datetime(2017, 10, 4, 7, 13, 42)},
 {'end': datetime.datetime(2017, 10, 4, 14, 50),
  'start': datetime.datetime(2017, 10, 4, 14, 22, 12)},
 {'end': datetime.datetime(2017, 10, 4, 15, 44, 49),
  'start': datetime.datetime(2017, 10, 4, 15, 7, 27)},
 {'end': datetime.datetime(2017, 10, 4, 16, 32, 33),
  'start': datetime.datetime(2017, 10, 4, 15, 46, 41)},
 {'end': datetime.datetime(2017, 10, 4, 16, 46, 59),
  'start': datetime.datetime(2017, 10, 4, 16, 34, 44)},
 {'end': datetime.datetime(2017, 10, 4, 17, 31, 36),
  'start': datetime.datetime(2017, 10, 4, 17, 26, 6)},
 {'end': datetime.datetime(2017, 10, 4, 17, 50, 41),
  'start': datetime.datetime(2017, 10, 4, 17, 42, 3)},
 {'end': datetime.datetime(2017, 10, 5, 8, 12, 55),
  'start': datetime.datetime(2017, 10, 5, 7, 49, 2)},
 {'end': datetime.datetime(2017, 10, 5, 8, 29, 45),
  'start': datetime.datetime(2017, 10, 5, 8, 26, 21)},
 {'end': datetime.datetime(2017, 10, 5, 8, 38, 31),
  'start': datetime.datetime(2017, 10, 5, 8, 33, 27)},
 {'end': datetime.datetime(2017, 10, 5, 16, 51, 52),
  'start': datetime.datetime(2017, 10, 5, 16, 35, 35)},
 {'end': datetime.datetime(2017, 10, 5, 18, 16, 50),
  'start': datetime.datetime(2017, 10, 5, 17, 53, 31)},
 {'end': datetime.datetime(2017, 10, 6, 8, 38, 1),
  'start': datetime.datetime(2017, 10, 6, 8, 17, 17)},
 {'end': datetime.datetime(2017, 10, 6, 11, 50, 38),
  'start': datetime.datetime(2017, 10, 6, 11, 39, 40)},
 {'end': datetime.datetime(2017, 10, 6, 13, 13, 14),
  'start': datetime.datetime(2017, 10, 6, 12, 59, 54)},
 {'end': datetime.datetime(2017, 10, 6, 14, 14, 56),
  'start': datetime.datetime(2017, 10, 6, 13, 43, 5)},
 {'end': datetime.datetime(2017, 10, 6, 15, 9, 26),
  'start': datetime.datetime(2017, 10, 6, 14, 28, 15)},
 {'end': datetime.datetime(2017, 10, 6, 16, 12, 34),
  'start': datetime.datetime(2017, 10, 6, 15, 50, 10)},
 {'end': datetime.datetime(2017, 10, 6, 16, 39, 31),
  'start': datetime.datetime(2017, 10, 6, 16, 32, 16)},
 {'end': datetime.datetime(2017, 10, 6, 16, 48, 39),
  'start': datetime.datetime(2017, 10, 6, 16, 44, 8)},
 {'end': datetime.datetime(2017, 10, 6, 17, 9, 3),
  'start': datetime.datetime(2017, 10, 6, 16, 53, 43)},
 {'end': datetime.datetime(2017, 10, 7, 11, 53, 6),
  'start': datetime.datetime(2017, 10, 7, 11, 38, 55)},
 {'end': datetime.datetime(2017, 10, 7, 14, 7, 5),
  'start': datetime.datetime(2017, 10, 7, 14, 3, 36)},
 {'end': datetime.datetime(2017, 10, 7, 14, 27, 36),
  'start': datetime.datetime(2017, 10, 7, 14, 20, 3)},
 {'end': datetime.datetime(2017, 10, 7, 14, 44, 51),
  'start': datetime.datetime(2017, 10, 7, 14, 30, 50)},
 {'end': datetime.datetime(2017, 10, 8, 0, 30, 48),
  'start': datetime.datetime(2017, 10, 8, 0, 28, 26)},
 {'end': datetime.datetime(2017, 10, 8, 11, 33, 24),
  'start': datetime.datetime(2017, 10, 8, 11, 16, 21)},
 {'end': datetime.datetime(2017, 10, 8, 13, 1, 29),
  'start': datetime.datetime(2017, 10, 8, 12, 37, 3)},
 {'end': datetime.datetime(2017, 10, 8, 13, 57, 53),
  'start': datetime.datetime(2017, 10, 8, 13, 30, 37)},
 {'end': datetime.datetime(2017, 10, 8, 15, 7, 19),
  'start': datetime.datetime(2017, 10, 8, 14, 16, 40)},
 {'end': datetime.datetime(2017, 10, 8, 15, 50, 1),
  'start': datetime.datetime(2017, 10, 8, 15, 23, 50)},
 {'end': datetime.datetime(2017, 10, 8, 16, 17, 42),
  'start': datetime.datetime(2017, 10, 8, 15, 54, 12)},
 {'end': datetime.datetime(2017, 10, 8, 16, 35, 18),
  'start': datetime.datetime(2017, 10, 8, 16, 28, 52)},
 {'end': datetime.datetime(2017, 10, 8, 23, 33, 41),
  'start': datetime.datetime(2017, 10, 8, 23, 8, 14)},
 {'end': datetime.datetime(2017, 10, 8, 23, 45, 11),
  'start': datetime.datetime(2017, 10, 8, 23, 34, 49)},
 {'end': datetime.datetime(2017, 10, 9, 0, 10, 57),
  'start': datetime.datetime(2017, 10, 8, 23, 46, 47)},
 {'end': datetime.datetime(2017, 10, 9, 0, 36, 40),
  'start': datetime.datetime(2017, 10, 9, 0, 12, 58)},
 {'end': datetime.datetime(2017, 10, 9, 0, 53, 33),
  'start': datetime.datetime(2017, 10, 9, 0, 37, 2)},
 {'end': datetime.datetime(2017, 10, 9, 1, 48, 13),
  'start': datetime.datetime(2017, 10, 9, 1, 23, 29)},
 {'end': datetime.datetime(2017, 10, 9, 2, 13, 35),
  'start': datetime.datetime(2017, 10, 9, 1, 49, 25)},
 {'end': datetime.datetime(2017, 10, 9, 2, 29, 40),
  'start': datetime.datetime(2017, 10, 9, 2, 14, 11)},
 {'end': datetime.datetime(2017, 10, 9, 13, 13, 25),
  'start': datetime.datetime(2017, 10, 9, 13, 4, 32)},
 {'end': datetime.datetime(2017, 10, 9, 14, 38, 55),
  'start': datetime.datetime(2017, 10, 9, 14, 30, 10)},
 {'end': datetime.datetime(2017, 10, 9, 15, 11, 30),
  'start': datetime.datetime(2017, 10, 9, 15, 6, 47)},
 {'end': datetime.datetime(2017, 10, 9, 16, 45, 38),
  'start': datetime.datetime(2017, 10, 9, 16, 43, 25)},
 {'end': datetime.datetime(2017, 10, 10, 15, 51, 24),
  'start': datetime.datetime(2017, 10, 10, 15, 32, 58)},
 {'end': datetime.datetime(2017, 10, 10, 17, 3, 47),
  'start': datetime.datetime(2017, 10, 10, 16, 47, 55)},
 {'end': datetime.datetime(2017, 10, 10, 18, 0, 18),
  'start': datetime.datetime(2017, 10, 10, 17, 51, 5)},
 {'end': datetime.datetime(2017, 10, 10, 18, 19, 11),
  'start': datetime.datetime(2017, 10, 10, 18, 8, 12)},
 {'end': datetime.datetime(2017, 10, 10, 19, 14, 32),
  'start': datetime.datetime(2017, 10, 10, 19, 9, 35)},
 {'end': datetime.datetime(2017, 10, 10, 19, 23, 8),
  'start': datetime.datetime(2017, 10, 10, 19, 17, 11)},
 {'end': datetime.datetime(2017, 10, 10, 19, 44, 40),
  'start': datetime.datetime(2017, 10, 10, 19, 28, 11)},
 {'end': datetime.datetime(2017, 10, 10, 20, 11, 54),
  'start': datetime.datetime(2017, 10, 10, 19, 55, 35)},
 {'end': datetime.datetime(2017, 10, 10, 22, 33, 23),
  'start': datetime.datetime(2017, 10, 10, 22, 20, 43)},
 {'end': datetime.datetime(2017, 10, 11, 4, 59, 22),
  'start': datetime.datetime(2017, 10, 11, 4, 40, 52)},
 {'end': datetime.datetime(2017, 10, 11, 6, 40, 13),
  'start': datetime.datetime(2017, 10, 11, 6, 28, 58)},
 {'end': datetime.datetime(2017, 10, 11, 17, 1, 14),
  'start': datetime.datetime(2017, 10, 11, 16, 41, 7)},
 {'end': datetime.datetime(2017, 10, 12, 8, 35, 3),
  'start': datetime.datetime(2017, 10, 12, 8, 8, 30)},
 {'end': datetime.datetime(2017, 10, 12, 8, 59, 50),
  'start': datetime.datetime(2017, 10, 12, 8, 47, 2)},
 {'end': datetime.datetime(2017, 10, 12, 13, 37, 45),
  'start': datetime.datetime(2017, 10, 12, 13, 13, 39)},
 {'end': datetime.datetime(2017, 10, 12, 13, 48, 17),
  'start': datetime.datetime(2017, 10, 12, 13, 40, 12)},
 {'end': datetime.datetime(2017, 10, 12, 13, 53, 16),
  'start': datetime.datetime(2017, 10, 12, 13, 49, 56)},
 {'end': datetime.datetime(2017, 10, 12, 14, 39, 57),
  'start': datetime.datetime(2017, 10, 12, 14, 33, 18)},
 {'end': datetime.datetime(2017, 10, 13, 15, 59, 41),
  'start': datetime.datetime(2017, 10, 13, 15, 55, 39)},
 {'end': datetime.datetime(2017, 10, 17, 18, 1, 38),
  'start': datetime.datetime(2017, 10, 17, 17, 58, 48)},
 {'end': datetime.datetime(2017, 10, 19, 20, 29, 15),
  'start': datetime.datetime(2017, 10, 19, 20, 21, 45)},
 {'end': datetime.datetime(2017, 10, 19, 21, 29, 37),
  'start': datetime.datetime(2017, 10, 19, 21, 11, 39)},
 {'end': datetime.datetime(2017, 10, 19, 21, 47, 23),
  'start': datetime.datetime(2017, 10, 19, 21, 30, 1)},
 {'end': datetime.datetime(2017, 10, 19, 21, 57, 7),
  'start': datetime.datetime(2017, 10, 19, 21, 47, 34)},
 {'end': datetime.datetime(2017, 10, 19, 22, 9, 52),
  'start': datetime.datetime(2017, 10, 19, 21, 57, 24)},
 {'end': datetime.datetime(2017, 10, 21, 12, 36, 24),
  'start': datetime.datetime(2017, 10, 21, 12, 24, 9)},
 {'end': datetime.datetime(2017, 10, 21, 12, 42, 13),
  'start': datetime.datetime(2017, 10, 21, 12, 36, 37)},
 {'end': datetime.datetime(2017, 10, 22, 11, 9, 36),
  'start': datetime.datetime(2017, 10, 21, 13, 47, 43)},
 {'end': datetime.datetime(2017, 10, 22, 13, 31, 44),
  'start': datetime.datetime(2017, 10, 22, 13, 28, 53)},
 {'end': datetime.datetime(2017, 10, 22, 13, 56, 33),
  'start': datetime.datetime(2017, 10, 22, 13, 47, 5)},
 {'end': datetime.datetime(2017, 10, 22, 14, 32, 39),
  'start': datetime.datetime(2017, 10, 22, 14, 26, 41)},
 {'end': datetime.datetime(2017, 10, 22, 15, 9, 58),
  'start': datetime.datetime(2017, 10, 22, 14, 54, 41)},
 {'end': datetime.datetime(2017, 10, 22, 16, 51, 40),
  'start': datetime.datetime(2017, 10, 22, 16, 40, 29)},
 {'end': datetime.datetime(2017, 10, 22, 18, 28, 37),
  'start': datetime.datetime(2017, 10, 22, 17, 58, 46)},
 {'end': datetime.datetime(2017, 10, 22, 18, 50, 34),
  'start': datetime.datetime(2017, 10, 22, 18, 45, 16)},
 {'end': datetime.datetime(2017, 10, 22, 19, 11, 10),
  'start': datetime.datetime(2017, 10, 22, 18, 56, 22)},
 {'end': datetime.datetime(2017, 10, 23, 10, 35, 32),
  'start': datetime.datetime(2017, 10, 23, 10, 14, 8)},
 {'end': datetime.datetime(2017, 10, 23, 14, 38, 34),
  'start': datetime.datetime(2017, 10, 23, 11, 29, 36)},
 {'end': datetime.datetime(2017, 10, 23, 15, 32, 58),
  'start': datetime.datetime(2017, 10, 23, 15, 4, 52)},
 {'end': datetime.datetime(2017, 10, 23, 17, 6, 47),
  'start': datetime.datetime(2017, 10, 23, 15, 33, 48)},
 {'end': datetime.datetime(2017, 10, 23, 19, 31, 26),
  'start': datetime.datetime(2017, 10, 23, 17, 13, 16)},
 {'end': datetime.datetime(2017, 10, 23, 20, 25, 53),
  'start': datetime.datetime(2017, 10, 23, 19, 55, 3)},
 {'end': datetime.datetime(2017, 10, 23, 22, 18, 4),
  'start': datetime.datetime(2017, 10, 23, 21, 47, 54)},
 {'end': datetime.datetime(2017, 10, 23, 22, 48, 42),
  'start': datetime.datetime(2017, 10, 23, 22, 34, 12)},
 {'end': datetime.datetime(2017, 10, 24, 7, 2, 17),
  'start': datetime.datetime(2017, 10, 24, 6, 55, 1)},
 {'end': datetime.datetime(2017, 10, 24, 15, 3, 16),
  'start': datetime.datetime(2017, 10, 24, 14, 56, 7)},
 {'end': datetime.datetime(2017, 10, 24, 15, 59, 50),
  'start': datetime.datetime(2017, 10, 24, 15, 51, 36)},
 {'end': datetime.datetime(2017, 10, 24, 16, 55, 9),
  'start': datetime.datetime(2017, 10, 24, 16, 31, 10)},
 {'end': datetime.datetime(2017, 10, 28, 14, 32, 34),
  'start': datetime.datetime(2017, 10, 28, 14, 26, 14)},
 {'end': datetime.datetime(2017, 11, 1, 9, 52, 23),
  'start': datetime.datetime(2017, 11, 1, 9, 41, 54)},
 {'end': datetime.datetime(2017, 11, 1, 20, 32, 13),
  'start': datetime.datetime(2017, 11, 1, 20, 16, 11)},
 {'end': datetime.datetime(2017, 11, 2, 19, 50, 56),
  'start': datetime.datetime(2017, 11, 2, 19, 44, 29)},
 {'end': datetime.datetime(2017, 11, 2, 20, 30, 29),
  'start': datetime.datetime(2017, 11, 2, 20, 14, 37)},
 {'end': datetime.datetime(2017, 11, 2, 21, 38, 57),
  'start': datetime.datetime(2017, 11, 2, 21, 35, 47)},
 {'end': datetime.datetime(2017, 11, 3, 10, 11, 46),
  'start': datetime.datetime(2017, 11, 3, 9, 59, 27)},
 {'end': datetime.datetime(2017, 11, 3, 10, 32, 2),
  'start': datetime.datetime(2017, 11, 3, 10, 13, 22)},
 {'end': datetime.datetime(2017, 11, 3, 10, 50, 34),
  'start': datetime.datetime(2017, 11, 3, 10, 44, 25)},
 {'end': datetime.datetime(2017, 11, 3, 16, 44, 38),
  'start': datetime.datetime(2017, 11, 3, 16, 6, 43)},
 {'end': datetime.datetime(2017, 11, 3, 17, 0, 27),
  'start': datetime.datetime(2017, 11, 3, 16, 45, 54)},
 {'end': datetime.datetime(2017, 11, 3, 17, 35, 5),
  'start': datetime.datetime(2017, 11, 3, 17, 7, 15)},
 {'end': datetime.datetime(2017, 11, 3, 17, 46, 48),
  'start': datetime.datetime(2017, 11, 3, 17, 36, 5)},
 {'end': datetime.datetime(2017, 11, 3, 18, 0, 3),
  'start': datetime.datetime(2017, 11, 3, 17, 50, 31)},
 {'end': datetime.datetime(2017, 11, 3, 19, 45, 51),
  'start': datetime.datetime(2017, 11, 3, 19, 22, 56)},
 {'end': datetime.datetime(2017, 11, 4, 13, 26, 15),
  'start': datetime.datetime(2017, 11, 4, 13, 14, 10)},
 {'end': datetime.datetime(2017, 11, 4, 14, 30, 5),
  'start': datetime.datetime(2017, 11, 4, 14, 18, 37)},
 {'end': datetime.datetime(2017, 11, 4, 15, 3, 20),
  'start': datetime.datetime(2017, 11, 4, 14, 45, 59)},
 {'end': datetime.datetime(2017, 11, 4, 15, 44, 30),
  'start': datetime.datetime(2017, 11, 4, 15, 16, 3)},
 {'end': datetime.datetime(2017, 11, 4, 16, 58, 22),
  'start': datetime.datetime(2017, 11, 4, 16, 37, 46)},
 {'end': datetime.datetime(2017, 11, 4, 17, 34, 50),
  'start': datetime.datetime(2017, 11, 4, 17, 13, 19)},
 {'end': datetime.datetime(2017, 11, 4, 18, 58, 44),
  'start': datetime.datetime(2017, 11, 4, 18, 10, 34)},
 {'end': datetime.datetime(2017, 11, 5, 1, 1, 4),
  'start': datetime.datetime(2017, 11, 5, 1, 56, 50)},
 {'end': datetime.datetime(2017, 11, 5, 8, 53, 46),
  'start': datetime.datetime(2017, 11, 5, 8, 33, 33)},
 {'end': datetime.datetime(2017, 11, 5, 9, 3, 39),
  'start': datetime.datetime(2017, 11, 5, 8, 58, 8)},
 {'end': datetime.datetime(2017, 11, 5, 11, 30, 5),
  'start': datetime.datetime(2017, 11, 5, 11, 5, 8)},
 {'end': datetime.datetime(2017, 11, 6, 8, 59, 5),
  'start': datetime.datetime(2017, 11, 6, 8, 50, 18)},
 {'end': datetime.datetime(2017, 11, 6, 9, 13, 47),
  'start': datetime.datetime(2017, 11, 6, 9, 4, 3)},
 {'end': datetime.datetime(2017, 11, 6, 17, 2, 55),
  'start': datetime.datetime(2017, 11, 6, 16, 19, 36)},
 {'end': datetime.datetime(2017, 11, 6, 17, 34, 6),
  'start': datetime.datetime(2017, 11, 6, 17, 21, 27)},
 {'end': datetime.datetime(2017, 11, 6, 17, 57, 32),
  'start': datetime.datetime(2017, 11, 6, 17, 36, 1)},
 {'end': datetime.datetime(2017, 11, 6, 18, 15, 8),
  'start': datetime.datetime(2017, 11, 6, 17, 59, 52)},
 {'end': datetime.datetime(2017, 11, 6, 18, 21, 17),
  'start': datetime.datetime(2017, 11, 6, 18, 18, 36)},
 {'end': datetime.datetime(2017, 11, 6, 19, 37, 57),
  'start': datetime.datetime(2017, 11, 6, 19, 24, 31)},
 {'end': datetime.datetime(2017, 11, 6, 20, 3, 14),
  'start': datetime.datetime(2017, 11, 6, 19, 49, 16)},
 {'end': datetime.datetime(2017, 11, 7, 8, 1, 32),
  'start': datetime.datetime(2017, 11, 7, 7, 50, 48)},
 {'end': datetime.datetime(2017, 11, 8, 13, 18, 5),
  'start': datetime.datetime(2017, 11, 8, 13, 11, 51)},
 {'end': datetime.datetime(2017, 11, 8, 21, 46, 5),
  'start': datetime.datetime(2017, 11, 8, 21, 34, 47)},
 {'end': datetime.datetime(2017, 11, 8, 22, 4, 47),
  'start': datetime.datetime(2017, 11, 8, 22, 2, 30)},
 {'end': datetime.datetime(2017, 11, 9, 7, 12, 10),
  'start': datetime.datetime(2017, 11, 9, 7, 1, 11)},
 {'end': datetime.datetime(2017, 11, 9, 8, 8, 28),
  'start': datetime.datetime(2017, 11, 9, 8, 2, 2)},
 {'end': datetime.datetime(2017, 11, 9, 8, 32, 24),
  'start': datetime.datetime(2017, 11, 9, 8, 19, 59)},
 {'end': datetime.datetime(2017, 11, 9, 8, 48, 59),
  'start': datetime.datetime(2017, 11, 9, 8, 41, 31)},
 {'end': datetime.datetime(2017, 11, 9, 9, 9, 24),
  'start': datetime.datetime(2017, 11, 9, 9, 0, 6)},
 {'end': datetime.datetime(2017, 11, 9, 9, 24, 25),
  'start': datetime.datetime(2017, 11, 9, 9, 9, 37)},
 {'end': datetime.datetime(2017, 11, 9, 13, 25, 39),
  'start': datetime.datetime(2017, 11, 9, 13, 14, 37)},
 {'end': datetime.datetime(2017, 11, 9, 15, 31, 10),
  'start': datetime.datetime(2017, 11, 9, 15, 20, 7)},
 {'end': datetime.datetime(2017, 11, 9, 18, 53, 10),
  'start': datetime.datetime(2017, 11, 9, 18, 47, 8)},
 {'end': datetime.datetime(2017, 11, 9, 23, 43, 35),
  'start': datetime.datetime(2017, 11, 9, 23, 35, 2)},
 {'end': datetime.datetime(2017, 11, 10, 8, 2, 28),
  'start': datetime.datetime(2017, 11, 10, 7, 51, 33)},
 {'end': datetime.datetime(2017, 11, 10, 8, 42, 9),
  'start': datetime.datetime(2017, 11, 10, 8, 38, 28)},
 {'end': datetime.datetime(2017, 11, 11, 18, 13, 14),
  'start': datetime.datetime(2017, 11, 11, 18, 5, 25)},
 {'end': datetime.datetime(2017, 11, 11, 19, 46, 22),
  'start': datetime.datetime(2017, 11, 11, 19, 39, 12)},
 {'end': datetime.datetime(2017, 11, 11, 21, 16, 31),
  'start': datetime.datetime(2017, 11, 11, 21, 13, 19)},
 {'end': datetime.datetime(2017, 11, 12, 9, 51, 43),
  'start': datetime.datetime(2017, 11, 12, 9, 46, 19)},
 {'end': datetime.datetime(2017, 11, 13, 13, 54, 15),
  'start': datetime.datetime(2017, 11, 13, 13, 33, 42)},
 {'end': datetime.datetime(2017, 11, 14, 8, 55, 52),
  'start': datetime.datetime(2017, 11, 14, 8, 40, 29)},
 {'end': datetime.datetime(2017, 11, 15, 6, 30, 6),
  'start': datetime.datetime(2017, 11, 15, 6, 14, 5)},
 {'end': datetime.datetime(2017, 11, 15, 8, 23, 44),
  'start': datetime.datetime(2017, 11, 15, 8, 14, 59)},
 {'end': datetime.datetime(2017, 11, 15, 10, 33, 41),
  'start': datetime.datetime(2017, 11, 15, 10, 16, 44)},
 {'end': datetime.datetime(2017, 11, 15, 10, 54, 14),
  'start': datetime.datetime(2017, 11, 15, 10, 33, 58)},
 {'end': datetime.datetime(2017, 11, 15, 11, 14, 42),
  'start': datetime.datetime(2017, 11, 15, 11, 2, 15)},
 {'end': datetime.datetime(2017, 11, 16, 9, 38, 49),
  'start': datetime.datetime(2017, 11, 16, 9, 27, 41)},
 {'end': datetime.datetime(2017, 11, 16, 10, 18),
  'start': datetime.datetime(2017, 11, 16, 9, 57, 41)},
 {'end': datetime.datetime(2017, 11, 16, 17, 44, 47),
  'start': datetime.datetime(2017, 11, 16, 17, 25, 5)},
 {'end': datetime.datetime(2017, 11, 17, 16, 36, 56),
  'start': datetime.datetime(2017, 11, 17, 13, 45, 54)},
 {'end': datetime.datetime(2017, 11, 17, 19, 31, 15),
  'start': datetime.datetime(2017, 11, 17, 19, 12, 49)},
 {'end': datetime.datetime(2017, 11, 18, 10, 55, 45),
  'start': datetime.datetime(2017, 11, 18, 10, 49, 6)},
 {'end': datetime.datetime(2017, 11, 18, 11, 44, 16),
  'start': datetime.datetime(2017, 11, 18, 11, 32, 12)},
 {'end': datetime.datetime(2017, 11, 18, 18, 14, 31),
  'start': datetime.datetime(2017, 11, 18, 18, 9, 1)},
 {'end': datetime.datetime(2017, 11, 18, 19, 1, 29),
  'start': datetime.datetime(2017, 11, 18, 18, 53, 10)},
 {'end': datetime.datetime(2017, 11, 19, 14, 31, 49),
  'start': datetime.datetime(2017, 11, 19, 14, 15, 41)},
 {'end': datetime.datetime(2017, 11, 20, 21, 41, 9),
  'start': datetime.datetime(2017, 11, 20, 21, 19, 19)},
 {'end': datetime.datetime(2017, 11, 20, 23, 23, 37),
  'start': datetime.datetime(2017, 11, 20, 22, 39, 48)},
 {'end': datetime.datetime(2017, 11, 21, 17, 51, 32),
  'start': datetime.datetime(2017, 11, 21, 17, 44, 25)},
 {'end': datetime.datetime(2017, 11, 21, 18, 34, 51),
  'start': datetime.datetime(2017, 11, 21, 18, 20, 52)},
 {'end': datetime.datetime(2017, 11, 21, 18, 51, 50),
  'start': datetime.datetime(2017, 11, 21, 18, 47, 32)},
 {'end': datetime.datetime(2017, 11, 21, 19, 14, 33),
  'start': datetime.datetime(2017, 11, 21, 19, 7, 57)},
 {'end': datetime.datetime(2017, 11, 21, 20, 8, 54),
  'start': datetime.datetime(2017, 11, 21, 20, 4, 56)},
 {'end': datetime.datetime(2017, 11, 21, 22, 8, 12),
  'start': datetime.datetime(2017, 11, 21, 21, 55, 47)},
 {'end': datetime.datetime(2017, 11, 23, 23, 57, 56),
  'start': datetime.datetime(2017, 11, 23, 23, 47, 43)},
 {'end': datetime.datetime(2017, 11, 24, 6, 53, 15),
  'start': datetime.datetime(2017, 11, 24, 6, 41, 25)},
 {'end': datetime.datetime(2017, 11, 24, 7, 33, 24),
  'start': datetime.datetime(2017, 11, 24, 6, 58, 56)},
 {'end': datetime.datetime(2017, 11, 26, 12, 41, 36),
  'start': datetime.datetime(2017, 11, 26, 12, 25, 49)},
 {'end': datetime.datetime(2017, 11, 27, 5, 54, 13),
  'start': datetime.datetime(2017, 11, 27, 5, 29, 4)},
 {'end': datetime.datetime(2017, 11, 27, 6, 11, 1),
  'start': datetime.datetime(2017, 11, 27, 6, 6, 47)},
 {'end': datetime.datetime(2017, 11, 27, 6, 55, 39),
  'start': datetime.datetime(2017, 11, 27, 6, 45, 14)},
 {'end': datetime.datetime(2017, 11, 27, 9, 47, 43),
  'start': datetime.datetime(2017, 11, 27, 9, 39, 44)},
 {'end': datetime.datetime(2017, 11, 27, 11, 20, 46),
  'start': datetime.datetime(2017, 11, 27, 11, 9, 18)},
 {'end': datetime.datetime(2017, 11, 27, 11, 35, 44),
  'start': datetime.datetime(2017, 11, 27, 11, 31, 46)},
 {'end': datetime.datetime(2017, 11, 27, 12, 12, 36),
  'start': datetime.datetime(2017, 11, 27, 12, 7, 14)},
 {'end': datetime.datetime(2017, 11, 27, 12, 26, 44),
  'start': datetime.datetime(2017, 11, 27, 12, 21, 40)},
 {'end': datetime.datetime(2017, 11, 27, 17, 36, 7),
  'start': datetime.datetime(2017, 11, 27, 17, 26, 31)},
 {'end': datetime.datetime(2017, 11, 27, 18, 29, 4),
  'start': datetime.datetime(2017, 11, 27, 18, 11, 49)},
 {'end': datetime.datetime(2017, 11, 27, 19, 47, 17),
  'start': datetime.datetime(2017, 11, 27, 19, 36, 16)},
 {'end': datetime.datetime(2017, 11, 27, 20, 17, 33),
  'start': datetime.datetime(2017, 11, 27, 20, 12, 57)},
 {'end': datetime.datetime(2017, 11, 28, 8, 41, 53),
  'start': datetime.datetime(2017, 11, 28, 8, 18, 6)},
 {'end': datetime.datetime(2017, 11, 28, 19, 34, 1),
  'start': datetime.datetime(2017, 11, 28, 19, 17, 23)},
 {'end': datetime.datetime(2017, 11, 28, 19, 46, 24),
  'start': datetime.datetime(2017, 11, 28, 19, 34, 15)},
 {'end': datetime.datetime(2017, 11, 28, 21, 39, 32),
  'start': datetime.datetime(2017, 11, 28, 21, 27, 29)},
 {'end': datetime.datetime(2017, 11, 29, 7, 51, 18),
  'start': datetime.datetime(2017, 11, 29, 7, 47, 38)},
 {'end': datetime.datetime(2017, 11, 29, 9, 53, 44),
  'start': datetime.datetime(2017, 11, 29, 9, 50, 12)},
 {'end': datetime.datetime(2017, 11, 29, 17, 16, 21),
  'start': datetime.datetime(2017, 11, 29, 17, 3, 42)},
 {'end': datetime.datetime(2017, 11, 29, 18, 23, 43),
  'start': datetime.datetime(2017, 11, 29, 18, 19, 15)},
 {'end': datetime.datetime(2017, 12, 1, 17, 10, 12),
  'start': datetime.datetime(2017, 12, 1, 17, 3, 58)},
 {'end': datetime.datetime(2017, 12, 2, 8, 1, 1),
  'start': datetime.datetime(2017, 12, 2, 7, 55, 56)},
 {'end': datetime.datetime(2017, 12, 2, 9, 21, 18),
  'start': datetime.datetime(2017, 12, 2, 9, 16, 14)},
 {'end': datetime.datetime(2017, 12, 2, 19, 53, 18),
  'start': datetime.datetime(2017, 12, 2, 19, 48, 29)},
 {'end': datetime.datetime(2017, 12, 3, 15, 20, 9),
  'start': datetime.datetime(2017, 12, 3, 14, 36, 29)},
 {'end': datetime.datetime(2017, 12, 3, 16, 25, 30),
  'start': datetime.datetime(2017, 12, 3, 16, 4, 2)},
 {'end': datetime.datetime(2017, 12, 3, 16, 43, 58),
  'start': datetime.datetime(2017, 12, 3, 16, 40, 26)},
 {'end': datetime.datetime(2017, 12, 3, 18, 4, 33),
  'start': datetime.datetime(2017, 12, 3, 17, 20, 17)},
 {'end': datetime.datetime(2017, 12, 4, 8, 51),
  'start': datetime.datetime(2017, 12, 4, 8, 34, 24)},
 {'end': datetime.datetime(2017, 12, 4, 17, 53, 57),
  'start': datetime.datetime(2017, 12, 4, 17, 49, 26)},
 {'end': datetime.datetime(2017, 12, 4, 18, 50, 33),
  'start': datetime.datetime(2017, 12, 4, 18, 38, 52)},
 {'end': datetime.datetime(2017, 12, 4, 21, 46, 58),
  'start': datetime.datetime(2017, 12, 4, 21, 39, 20)},
 {'end': datetime.datetime(2017, 12, 4, 21, 56, 17),
  'start': datetime.datetime(2017, 12, 4, 21, 54, 21)},
 {'end': datetime.datetime(2017, 12, 5, 8, 52, 54),
  'start': datetime.datetime(2017, 12, 5, 8, 50, 50)},
 {'end': datetime.datetime(2017, 12, 6, 8, 24, 14),
  'start': datetime.datetime(2017, 12, 6, 8, 19, 38)},
 {'end': datetime.datetime(2017, 12, 6, 18, 28, 11),
  'start': datetime.datetime(2017, 12, 6, 18, 19, 19)},
 {'end': datetime.datetime(2017, 12, 6, 18, 33, 12),
  'start': datetime.datetime(2017, 12, 6, 18, 28, 55)},
 {'end': datetime.datetime(2017, 12, 6, 20, 21, 38),
  'start': datetime.datetime(2017, 12, 6, 20, 3, 29)},
 {'end': datetime.datetime(2017, 12, 6, 20, 39, 57),
  'start': datetime.datetime(2017, 12, 6, 20, 36, 42)},
 {'end': datetime.datetime(2017, 12, 7, 6, 1, 15),
  'start': datetime.datetime(2017, 12, 7, 5, 54, 51)},
 {'end': datetime.datetime(2017, 12, 8, 16, 55, 49),
  'start': datetime.datetime(2017, 12, 8, 16, 47, 18)},
 {'end': datetime.datetime(2017, 12, 8, 19, 29, 12),
  'start': datetime.datetime(2017, 12, 8, 19, 15, 2)},
 {'end': datetime.datetime(2017, 12, 9, 22, 47, 19),
  'start': datetime.datetime(2017, 12, 9, 22, 39, 37)},
 {'end': datetime.datetime(2017, 12, 9, 23, 5, 32),
  'start': datetime.datetime(2017, 12, 9, 23, 0, 10)},
 {'end': datetime.datetime(2017, 12, 10, 0, 56, 2),
  'start': datetime.datetime(2017, 12, 10, 0, 39, 24)},
 {'end': datetime.datetime(2017, 12, 10, 1, 8, 9),
  'start': datetime.datetime(2017, 12, 10, 1, 2, 42)},
 {'end': datetime.datetime(2017, 12, 10, 1, 11, 30),
  'start': datetime.datetime(2017, 12, 10, 1, 8, 57)},
 {'end': datetime.datetime(2017, 12, 10, 13, 51, 41),
  'start': datetime.datetime(2017, 12, 10, 13, 49, 9)},
 {'end': datetime.datetime(2017, 12, 10, 15, 18, 19),
  'start': datetime.datetime(2017, 12, 10, 15, 14, 29)},
 {'end': datetime.datetime(2017, 12, 10, 15, 36, 28),
  'start': datetime.datetime(2017, 12, 10, 15, 31, 7)},
 {'end': datetime.datetime(2017, 12, 10, 16, 30, 31),
  'start': datetime.datetime(2017, 12, 10, 16, 20, 6)},
 {'end': datetime.datetime(2017, 12, 10, 17, 14, 25),
  'start': datetime.datetime(2017, 12, 10, 17, 7, 54)},
 {'end': datetime.datetime(2017, 12, 10, 17, 45, 25),
  'start': datetime.datetime(2017, 12, 10, 17, 23, 47)},
 {'end': datetime.datetime(2017, 12, 11, 6, 34, 4),
  'start': datetime.datetime(2017, 12, 11, 6, 17, 6)},
 {'end': datetime.datetime(2017, 12, 11, 9, 12, 21),
  'start': datetime.datetime(2017, 12, 11, 9, 8, 41)},
 {'end': datetime.datetime(2017, 12, 11, 9, 20, 18),
  'start': datetime.datetime(2017, 12, 11, 9, 15, 41)},
 {'end': datetime.datetime(2017, 12, 12, 8, 59, 34),
  'start': datetime.datetime(2017, 12, 12, 8, 55, 53)},
 {'end': datetime.datetime(2017, 12, 13, 17, 18, 32),
  'start': datetime.datetime(2017, 12, 13, 17, 14, 56)},
 {'end': datetime.datetime(2017, 12, 13, 19, 0, 45),
  'start': datetime.datetime(2017, 12, 13, 18, 52, 16)},
 {'end': datetime.datetime(2017, 12, 14, 9, 11, 6),
  'start': datetime.datetime(2017, 12, 14, 9, 1, 10)},
 {'end': datetime.datetime(2017, 12, 14, 9, 19, 6),
  'start': datetime.datetime(2017, 12, 14, 9, 12, 59)},
 {'end': datetime.datetime(2017, 12, 14, 12, 2),
  'start': datetime.datetime(2017, 12, 14, 11, 54, 33)},
 {'end': datetime.datetime(2017, 12, 14, 14, 44, 40),
  'start': datetime.datetime(2017, 12, 14, 14, 40, 23)},
 {'end': datetime.datetime(2017, 12, 14, 15, 26, 24),
  'start': datetime.datetime(2017, 12, 14, 15, 8, 55)},
 {'end': datetime.datetime(2017, 12, 14, 18, 9, 4),
  'start': datetime.datetime(2017, 12, 14, 17, 46, 17)},
 {'end': datetime.datetime(2017, 12, 15, 9, 23, 45),
  'start': datetime.datetime(2017, 12, 15, 9, 8, 12)},
 {'end': datetime.datetime(2017, 12, 16, 9, 36, 17),
  'start': datetime.datetime(2017, 12, 16, 9, 33, 46)},
 {'end': datetime.datetime(2017, 12, 16, 11, 5, 4),
  'start': datetime.datetime(2017, 12, 16, 11, 2, 31)},
 {'end': datetime.datetime(2017, 12, 17, 10, 32, 3),
  'start': datetime.datetime(2017, 12, 17, 10, 9, 47)},
 {'end': datetime.datetime(2017, 12, 18, 8, 7, 34),
  'start': datetime.datetime(2017, 12, 18, 8, 2, 36)},
 {'end': datetime.datetime(2017, 12, 18, 16, 9, 20),
  'start': datetime.datetime(2017, 12, 18, 16, 3)},
 {'end': datetime.datetime(2017, 12, 18, 16, 53, 12),
  'start': datetime.datetime(2017, 12, 18, 16, 30, 7)},
 {'end': datetime.datetime(2017, 12, 18, 19, 22, 8),
  'start': datetime.datetime(2017, 12, 18, 19, 18, 23)},
 {'end': datetime.datetime(2017, 12, 18, 20, 17, 47),
  'start': datetime.datetime(2017, 12, 18, 20, 14, 46)},
 {'end': datetime.datetime(2017, 12, 19, 19, 23, 49),
  'start': datetime.datetime(2017, 12, 19, 19, 14, 8)},
 {'end': datetime.datetime(2017, 12, 19, 19, 43, 46),
  'start': datetime.datetime(2017, 12, 19, 19, 39, 36)},
 {'end': datetime.datetime(2017, 12, 20, 8, 10, 46),
  'start': datetime.datetime(2017, 12, 20, 8, 5, 14)},
 {'end': datetime.datetime(2017, 12, 20, 8, 29, 50),
  'start': datetime.datetime(2017, 12, 20, 8, 15, 45)},
 {'end': datetime.datetime(2017, 12, 20, 8, 38, 9),
  'start': datetime.datetime(2017, 12, 20, 8, 33, 32)},
 {'end': datetime.datetime(2017, 12, 20, 13, 54, 39),
  'start': datetime.datetime(2017, 12, 20, 13, 43, 36)},
 {'end': datetime.datetime(2017, 12, 20, 19, 6, 54),
  'start': datetime.datetime(2017, 12, 20, 18, 57, 53)},
 {'end': datetime.datetime(2017, 12, 21, 7, 32, 3),
  'start': datetime.datetime(2017, 12, 21, 7, 21, 11)},
 {'end': datetime.datetime(2017, 12, 21, 8, 6, 15),
  'start': datetime.datetime(2017, 12, 21, 8, 1, 58)},
 {'end': datetime.datetime(2017, 12, 21, 13, 33, 49),
  'start': datetime.datetime(2017, 12, 21, 13, 20, 54)},
 {'end': datetime.datetime(2017, 12, 21, 15, 34, 27),
  'start': datetime.datetime(2017, 12, 21, 15, 26, 8)},
 {'end': datetime.datetime(2017, 12, 21, 18, 38, 50),
  'start': datetime.datetime(2017, 12, 21, 18, 9, 46)},
 {'end': datetime.datetime(2017, 12, 22, 16, 21, 46),
  'start': datetime.datetime(2017, 12, 22, 16, 14, 21)},
 {'end': datetime.datetime(2017, 12, 22, 16, 34, 14),
  'start': datetime.datetime(2017, 12, 22, 16, 29, 17)},
 {'end': datetime.datetime(2017, 12, 25, 13, 18, 27),
  'start': datetime.datetime(2017, 12, 25, 12, 49, 51)},
 {'end': datetime.datetime(2017, 12, 25, 14, 20, 50),
  'start': datetime.datetime(2017, 12, 25, 13, 46, 44)},
 {'end': datetime.datetime(2017, 12, 26, 10, 53, 45),
  'start': datetime.datetime(2017, 12, 26, 10, 40, 16)},
 {'end': datetime.datetime(2017, 12, 27, 17, 17, 39),
  'start': datetime.datetime(2017, 12, 27, 16, 56, 12)},
 {'end': datetime.datetime(2017, 12, 29, 6, 12, 30),
  'start': datetime.datetime(2017, 12, 29, 6, 2, 34)},
 {'end': datetime.datetime(2017, 12, 29, 12, 46, 16),
  'start': datetime.datetime(2017, 12, 29, 12, 21, 3)},
 {'end': datetime.datetime(2017, 12, 29, 14, 43, 46),
  'start': datetime.datetime(2017, 12, 29, 14, 32, 55)},
 {'end': datetime.datetime(2017, 12, 29, 15, 18, 51),
  'start': datetime.datetime(2017, 12, 29, 15, 8, 26)},
 {'end': datetime.datetime(2017, 12, 29, 20, 38, 13),
  'start': datetime.datetime(2017, 12, 29, 20, 33, 34)},
 {'end': datetime.datetime(2017, 12, 30, 13, 54, 33),
  'start': datetime.datetime(2017, 12, 30, 13, 51, 3)},
 {'end': datetime.datetime(2017, 12, 30, 15, 19, 13),
  'start': datetime.datetime(2017, 12, 30, 15, 9, 3)}]
```

[43]:
```
# Creating dictionary to hold results
trip_counts = {'AM': 0, 'PM': 0}

# Looping over all trips
for trip in onebike_datetimes:
  # Checking to see if the trip starts before noon
  if trip['start'].hour < 12:
    # Incrementing the counter for before noon
    trip_counts['AM'] += 1
  else:
    # Incrementing the counter for after noon
    trip_counts['PM'] += 1

print(trip_counts)
```

```
{'AM': 94, 'PM': 196}
```

It looks like this bike is used about twice as much after noon than it is before noon. One obvious follow up would be to see which hours the bike is most likely to be taken out for a ride.

Turning strings into datetimes

When we download data from the Internet, dates and times usually come as strings. Often the first step is to turn those strings into datetime objects.

I will practice this transformation.

Reference %Y 4 digit year (0000-9999)

%m 2 digit month (1-12)

%d 2 digit day (1-31)

%H 2 digit hour (0-23)

%M 2 digit minute (0-59)

%S 2 digit second (0-59)

[45]:
```python
# Determine the format needed to convert s to datetime and assign it to fmt.
# Convert the string s to datetime using fmt.

# Import the datetime class
from datetime import datetime

# Starting string, in YYYY-MM-DD HH:MM:SS format
s = '2017-02-03 00:00:01'

# Write a format string to parse s
fmt = '%Y-%m-%d %H:%M:%S'

# Create a datetime object d
d = datetime.strptime(s, fmt)

# Print d
print(d)
```

```
2017-02-03 00:00:01
```

[46]:
```python
# Determine the format needed to convert s to datetime and assign it to fmt.
# Converting the string s to datetime using fmt.

# Importing the datetime class
from datetime import datetime

# Starting string, in YYYY-MM-DD format
s = '2030-10-15'

# Writing a format string to parse s
fmt = '%Y-%m-%d'

# Creating a datetime object d
d = datetime.strptime(s, fmt)

# Printing d
print(d)
```

```
2030-10-15 00:00:00
```

[47]:
```python
# Determine the format needed to convert s to datetime and assign it to fmt.
# Converting the string s to datetime using fmt.

# Importing the datetime class
from datetime import datetime

# Starting string, in MM/DD/YYYY HH:MM:SS format
```

```
s = '12/15/1986 08:00:00'

# Writing a format string to parse s
fmt = '%m/%d/%Y %H:%M:%S'

# Creating a datetime object d
d = datetime.strptime(s, fmt)

# Printing d
print(d)
```

```
1986-12-15 08:00:00
```

Now we can parse dates in most common formats. Unfortunately, Python does not have the ability to parse non-zero-padded dates and times out of the box (such as 1/2/2018). If needed, we can use other string methods to create zero-padded strings suitable for strptime()

[48]:
```
onebike_datetime_strings = [('2017-10-01 15:23:25', '2017-10-01 15:26:26'),
 ('2017-10-01 15:42:57', '2017-10-01 17:49:59'),
 ('2017-10-02 06:37:10', '2017-10-02 06:42:53'),
 ('2017-10-02 08:56:45', '2017-10-02 09:18:03'),
 ('2017-10-02 18:23:48', '2017-10-02 18:45:05'),
 ('2017-10-02 18:48:08', '2017-10-02 19:10:54'),
 ('2017-10-02 19:18:10', '2017-10-02 19:31:45'),
 ('2017-10-02 19:37:32', '2017-10-02 19:46:37'),
 ('2017-10-03 08:24:16', '2017-10-03 08:32:27'),
 ('2017-10-03 18:17:07', '2017-10-03 18:27:46'),
 ('2017-10-03 19:24:10', '2017-10-03 19:52:08'),
 ('2017-10-03 20:17:06', '2017-10-03 20:23:52'),
 ('2017-10-03 20:45:21', '2017-10-03 20:57:10'),
 ('2017-10-04 07:04:57', '2017-10-04 07:13:31'),
 ('2017-10-04 07:13:42', '2017-10-04 07:21:54'),
 ('2017-10-04 14:22:12', '2017-10-04 14:50:00'),
 ('2017-10-04 15:07:27', '2017-10-04 15:44:49'),
 ('2017-10-04 15:46:41', '2017-10-04 16:32:33'),
 ('2017-10-04 16:34:44', '2017-10-04 16:46:59'),
 ('2017-10-04 17:26:06', '2017-10-04 17:31:36'),
 ('2017-10-04 17:42:03', '2017-10-04 17:50:41'),
 ('2017-10-05 07:49:02', '2017-10-05 08:12:55'),
 ('2017-10-05 08:26:21', '2017-10-05 08:29:45'),
 ('2017-10-05 08:33:27', '2017-10-05 08:38:31'),
 ('2017-10-05 16:35:35', '2017-10-05 16:51:52'),
 ('2017-10-05 17:53:31', '2017-10-05 18:16:50'),
 ('2017-10-06 08:17:17', '2017-10-06 08:38:01'),
 ('2017-10-06 11:39:40', '2017-10-06 11:50:38'),
 ('2017-10-06 12:59:54', '2017-10-06 13:13:14'),
 ('2017-10-06 13:43:05', '2017-10-06 14:14:56'),
 ('2017-10-06 14:28:15', '2017-10-06 15:09:26'),
 ('2017-10-06 15:50:10', '2017-10-06 16:12:34'),
 ('2017-10-06 16:32:16', '2017-10-06 16:39:31'),
 ('2017-10-06 16:44:08', '2017-10-06 16:48:39'),
 ('2017-10-06 16:53:43', '2017-10-06 17:09:03'),
 ('2017-10-07 11:38:55', '2017-10-07 11:53:06'),
 ('2017-10-07 14:03:36', '2017-10-07 14:07:05'),
 ('2017-10-07 14:20:03', '2017-10-07 14:27:36'),
 ('2017-10-07 14:30:50', '2017-10-07 14:44:51'),
 ('2017-10-08 00:28:26', '2017-10-08 00:30:48'),
 ('2017-10-08 11:16:21', '2017-10-08 11:33:24'),
 ('2017-10-08 12:37:03', '2017-10-08 13:01:29'),
 ('2017-10-08 13:30:37', '2017-10-08 13:57:53'),
 ('2017-10-08 14:16:40', '2017-10-08 15:07:19'),
 ('2017-10-08 15:23:50', '2017-10-08 15:50:01'),
 ('2017-10-08 15:54:12', '2017-10-08 16:17:42'),
 ('2017-10-08 16:28:52', '2017-10-08 16:35:18'),
 ('2017-10-08 23:08:14', '2017-10-08 23:33:41'),
 ('2017-10-08 23:34:49', '2017-10-08 23:45:11'),
 ('2017-10-08 23:46:47', '2017-10-09 00:10:57'),
 ('2017-10-09 00:12:58', '2017-10-09 00:36:40'),
 ('2017-10-09 00:37:02', '2017-10-09 00:53:33'),
 ('2017-10-09 01:23:29', '2017-10-09 01:48:13'),
 ('2017-10-09 01:49:25', '2017-10-09 02:13:35'),
 ('2017-10-09 02:14:11', '2017-10-09 02:29:40'),
 ('2017-10-09 13:04:32', '2017-10-09 13:13:25'),
 ('2017-10-09 14:30:10', '2017-10-09 14:38:55'),
 ('2017-10-09 15:06:47', '2017-10-09 15:11:30'),
 ('2017-10-09 16:43:25', '2017-10-09 16:45:38'),
 ('2017-10-10 15:32:58', '2017-10-10 15:51:24'),
 ('2017-10-10 16:47:55', '2017-10-10 17:03:47'),
 ('2017-10-10 17:51:05', '2017-10-10 18:00:18'),
 ('2017-10-10 18:08:12', '2017-10-10 18:19:11'),
 ('2017-10-10 19:09:35', '2017-10-10 19:14:32'),
 ('2017-10-10 19:17:11', '2017-10-10 19:23:08'),
 ('2017-10-10 19:28:11', '2017-10-10 19:44:40'),
 ('2017-10-10 19:55:35', '2017-10-10 20:11:54'),
 ('2017-10-10 22:20:43', '2017-10-10 22:33:23'),
 ('2017-10-11 04:40:52', '2017-10-11 04:59:22'),
 ('2017-10-11 06:28:58', '2017-10-11 06:40:13'),
 ('2017-10-11 16:41:07', '2017-10-11 17:01:14'),
 ('2017-10-12 08:08:30', '2017-10-12 08:35:03'),
 ('2017-10-12 08:47:02', '2017-10-12 08:59:50'),
 ('2017-10-12 13:13:39', '2017-10-12 13:37:45'),
 ('2017-10-12 13:40:12', '2017-10-12 13:48:17'),
 ('2017-10-12 13:49:56', '2017-10-12 13:53:16'),
 ('2017-10-12 14:33:18', '2017-10-12 14:39:57'),
 ('2017-10-13 15:55:39', '2017-10-13 15:59:41'),
 ('2017-10-17 17:58:48', '2017-10-17 18:01:38'),
 ('2017-10-19 20:21:45', '2017-10-19 20:29:15'),
 ('2017-10-19 21:11:39', '2017-10-19 21:29:37'),
 ('2017-10-19 21:30:01', '2017-10-19 21:47:23'),
 ('2017-10-19 21:47:34', '2017-10-19 21:57:07'),
 ('2017-10-19 21:57:24', '2017-10-19 22:09:52'),
 ('2017-10-21 12:24:09', '2017-10-21 12:36:24'),
 ('2017-10-21 12:36:37', '2017-10-21 12:42:13'),
 ('2017-10-21 13:47:43', '2017-10-22 11:09:36'),
 ('2017-10-22 13:28:53', '2017-10-22 13:31:44'),
 ('2017-10-22 13:47:05', '2017-10-22 13:56:33'),
 ('2017-10-22 14:26:41', '2017-10-22 14:32:39'),
 ('2017-10-22 14:54:41', '2017-10-22 15:09:58'),
 ('2017-10-22 16:40:29', '2017-10-22 16:51:40'),
 ('2017-10-22 17:58:46', '2017-10-22 18:28:37'),
 ('2017-10-22 18:45:16', '2017-10-22 18:50:34'),
 ('2017-10-22 18:56:22', '2017-10-22 19:11:10'),
 ('2017-10-23 10:14:08', '2017-10-23 10:35:32'),
 ('2017-10-23 11:29:36', '2017-10-23 14:38:34'),
 ('2017-10-23 15:04:52', '2017-10-23 15:32:58'),
 ('2017-10-23 15:33:48', '2017-10-23 17:06:47'),
 ('2017-10-23 17:13:16', '2017-10-23 19:31:26'),
 ('2017-10-23 19:55:03', '2017-10-23 20:25:53'),
 ('2017-10-23 21:47:54', '2017-10-23 22:18:04'),
 ('2017-10-23 22:34:12', '2017-10-23 22:48:42'),
 ('2017-10-24 06:55:01', '2017-10-24 07:02:17'),
 ('2017-10-24 14:56:07', '2017-10-24 15:03:16'),
 ('2017-10-24 15:51:36', '2017-10-24 15:59:50'),
 ('2017-10-24 16:31:10', '2017-10-24 16:55:09'),
 ('2017-10-28 14:26:14', '2017-10-28 14:32:34'),
 ('2017-11-01 09:41:54', '2017-11-01 09:52:23'),
 ('2017-11-01 20:16:11', '2017-11-01 20:32:13'),
 ('2017-11-02 19:44:29', '2017-11-02 19:50:56'),
 ('2017-11-02 20:14:37', '2017-11-02 20:30:29'),
 ('2017-11-02 21:35:47', '2017-11-02 21:38:57'),
 ('2017-11-03 09:59:27', '2017-11-03 10:11:46'),
 ('2017-11-03 10:13:22', '2017-11-03 10:32:02'),
 ('2017-11-03 10:44:25', '2017-11-03 10:50:34'),
 ('2017-11-03 16:06:43', '2017-11-03 16:44:38'),
 ('2017-11-03 16:45:54', '2017-11-03 17:00:27'),
 ('2017-11-03 17:07:15', '2017-11-03 17:35:05'),
 ('2017-11-03 17:36:05', '2017-11-03 17:46:48'),
 ('2017-11-03 17:50:31', '2017-11-03 18:00:03'),
 ('2017-11-03 19:22:56', '2017-11-03 19:45:51'),
 ('2017-11-04 13:14:10', '2017-11-04 13:26:15'),
 ('2017-11-04 14:18:37', '2017-11-04 14:30:05'),
 ('2017-11-04 14:45:59', '2017-11-04 15:03:20'),
 ('2017-11-04 15:16:03', '2017-11-04 15:44:30'),
 ('2017-11-04 16:37:46', '2017-11-04 16:58:22'),
 ('2017-11-04 17:13:19', '2017-11-04 17:34:50'),
 ('2017-11-04 18:10:34', '2017-11-04 18:58:44'),
 ('2017-11-05 01:56:50', '2017-11-05 01:01:04'),
 ('2017-11-05 08:33:33', '2017-11-05 08:53:46'),
 ('2017-11-05 08:58:08', '2017-11-05 09:03:39'),
 ('2017-11-05 11:05:08', '2017-11-05 11:30:05'),
 ('2017-11-06 08:50:18', '2017-11-06 08:59:05'),
 ('2017-11-06 09:04:03', '2017-11-06 09:13:47'),
 ('2017-11-06 16:19:36', '2017-11-06 17:02:55'),
 ('2017-11-06 17:21:27', '2017-11-06 17:34:06'),
 ('2017-11-06 17:36:01', '2017-11-06 17:57:32'),
 ('2017-11-06 17:59:52', '2017-11-06 18:15:08'),
 ('2017-11-06 18:18:36', '2017-11-06 18:21:17'),
 ('2017-11-06 19:24:31', '2017-11-06 19:37:57'),
 ('2017-11-06 19:49:16', '2017-11-06 20:03:14'),
 ('2017-11-07 07:50:48', '2017-11-07 08:01:32'),
 ('2017-11-08 13:11:51', '2017-11-08 13:18:05'),
 ('2017-11-08 21:34:47', '2017-11-08 21:46:05'),
 ('2017-11-08 22:02:30', '2017-11-08 22:04:47'),
 ('2017-11-09 07:01:11', '2017-11-09 07:12:10'),
 ('2017-11-09 08:02:02', '2017-11-09 08:08:28'),
 ('2017-11-09 08:19:59', '2017-11-09 08:32:24'),
 ('2017-11-09 08:41:31', '2017-11-09 08:48:59'),
 ('2017-11-09 09:00:06', '2017-11-09 09:09:24'),
 ('2017-11-09 09:09:37', '2017-11-09 09:24:25'),
 ('2017-11-09 13:14:37', '2017-11-09 13:25:39'),
 ('2017-11-09 15:20:07', '2017-11-09 15:31:10'),
 ('2017-11-09 18:47:08', '2017-11-09 18:53:10'),
 ('2017-11-09 23:35:02', '2017-11-09 23:43:35'),
 ('2017-11-10 07:51:33', '2017-11-10 08:02:28'),
 ('2017-11-10 08:38:28', '2017-11-10 08:42:09'),
 ('2017-11-11 18:05:25', '2017-11-11 18:13:14'),
 ('2017-11-11 19:39:12', '2017-11-11 19:46:22'),
 ('2017-11-11 21:13:19', '2017-11-11 21:16:31'),
 ('2017-11-12 09:46:19', '2017-11-12 09:51:43'),
 ('2017-11-13 13:33:42', '2017-11-13 13:54:15'),
 ('2017-11-14 08:40:29', '2017-11-14 08:55:52'),
 ('2017-11-15 06:14:05', '2017-11-15 06:30:06'),
 ('2017-11-15 08:14:59', '2017-11-15 08:23:44'),
 ('2017-11-15 10:16:44', '2017-11-15 10:33:41'),
 ('2017-11-15 10:33:58', '2017-11-15 10:54:14'),
 ('2017-11-15 11:02:15', '2017-11-15 11:14:42'),
 ('2017-11-16 09:27:41', '2017-11-16 09:38:49'),
 ('2017-11-16 09:57:41', '2017-11-16 10:18:00'),
 ('2017-11-16 17:25:05', '2017-11-16 17:44:47'),
 ('2017-11-17 13:45:54', '2017-11-17 16:36:56'),
 ('2017-11-17 19:12:49', '2017-11-17 19:31:15'),
 ('2017-11-18 10:49:06', '2017-11-18 10:55:45'),
 ('2017-11-18 11:32:12', '2017-11-18 11:44:16'),
 ('2017-11-18 18:09:01', '2017-11-18 18:14:31'),
 ('2017-11-18 18:53:10', '2017-11-18 19:01:29'),
 ('2017-11-19 14:15:41', '2017-11-19 14:31:49'),
 ('2017-11-20 21:19:19', '2017-11-20 21:41:09'),
 ('2017-11-20 22:39:48', '2017-11-20 23:23:37'),
 ('2017-11-21 17:44:25', '2017-11-21 17:51:32'),
 ('2017-11-21 18:20:52', '2017-11-21 18:34:51'),
 ('2017-11-21 18:47:32', '2017-11-21 18:51:50'),
 ('2017-11-21 19:07:57', '2017-11-21 19:14:33'),
 ('2017-11-21 20:04:56', '2017-11-21 20:08:54'),
 ('2017-11-21 21:55:47', '2017-11-21 22:08:12'),
 ('2017-11-23 23:47:43', '2017-11-23 23:57:56'),
 ('2017-11-24 06:41:25', '2017-11-24 06:53:15'),
 ('2017-11-24 06:58:56', '2017-11-24 07:33:24'),
 ('2017-11-26 12:25:49', '2017-11-26 12:41:36'),
 ('2017-11-27 05:29:04', '2017-11-27 05:54:13'),
 ('2017-11-27 06:06:47', '2017-11-27 06:11:01'),
 ('2017-11-27 06:45:14', '2017-11-27 06:55:39'),
 ('2017-11-27 09:39:44', '2017-11-27 09:47:43'),
 ('2017-11-27 11:09:18', '2017-11-27 11:20:46'),
 ('2017-11-27 11:31:46', '2017-11-27 11:35:44'),
 ('2017-11-27 12:07:14', '2017-11-27 12:12:36'),
 ('2017-11-27 12:21:40', '2017-11-27 12:26:44'),
 ('2017-11-27 17:26:31', '2017-11-27 17:36:07'),
 ('2017-11-27 18:11:49', '2017-11-27 18:29:04'),
 ('2017-11-27 19:36:16', '2017-11-27 19:47:17'),
 ('2017-11-27 20:12:57', '2017-11-27 20:17:33'),
 ('2017-11-28 08:18:06', '2017-11-28 08:41:53'),
 ('2017-11-28 19:17:23', '2017-11-28 19:34:01'),
 ('2017-11-28 19:34:15', '2017-11-28 19:46:24'),
 ('2017-11-28 21:27:29', '2017-11-28 21:39:32'),
 ('2017-11-29 07:47:38', '2017-11-29 07:51:18'),
 ('2017-11-29 09:50:12', '2017-11-29 09:53:44'),
 ('2017-11-29 17:03:42', '2017-11-29 17:16:21'),
 ('2017-11-29 18:19:15', '2017-11-29 18:23:43'),
 ('2017-12-01 17:03:58', '2017-12-01 17:10:12'),
 ('2017-12-02 07:55:56', '2017-12-02 08:01:01'),
 ('2017-12-02 09:16:14', '2017-12-02 09:21:18'),
 ('2017-12-02 19:48:29', '2017-12-02 19:53:18'),
 ('2017-12-03 14:36:29', '2017-12-03 15:20:09'),
 ('2017-12-03 16:04:02', '2017-12-03 16:25:30'),
 ('2017-12-03 16:40:26', '2017-12-03 16:43:58'),
 ('2017-12-03 17:20:17', '2017-12-03 18:04:33'),
 ('2017-12-04 08:34:24', '2017-12-04 08:51:00'),
 ('2017-12-04 17:49:26', '2017-12-04 17:53:57'),
 ('2017-12-04 18:38:52', '2017-12-04 18:50:33'),
 ('2017-12-04 21:39:20', '2017-12-04 21:46:58'),
 ('2017-12-04 21:54:21', '2017-12-04 21:56:17'),
 ('2017-12-05 08:50:50', '2017-12-05 08:52:54'),
 ('2017-12-06 08:19:38', '2017-12-06 08:24:14'),
 ('2017-12-06 18:19:19', '2017-12-06 18:28:11'),
 ('2017-12-06 18:28:55', '2017-12-06 18:33:12'),
 ('2017-12-06 20:03:29', '2017-12-06 20:21:38'),
 ('2017-12-06 20:36:42', '2017-12-06 20:39:57'),
 ('2017-12-07 05:54:51', '2017-12-07 06:01:15'),
 ('2017-12-08 16:47:18', '2017-12-08 16:55:49'),
 ('2017-12-08 19:15:02', '2017-12-08 19:29:12'),
 ('2017-12-09 22:39:37', '2017-12-09 22:47:19'),
 ('2017-12-09 23:00:10', '2017-12-09 23:05:32'),
 ('2017-12-10 00:39:24', '2017-12-10 00:56:02'),
 ('2017-12-10 01:02:42', '2017-12-10 01:08:09'),
 ('2017-12-10 01:08:57', '2017-12-10 01:11:30'),
 ('2017-12-10 13:49:09', '2017-12-10 13:51:41'),
 ('2017-12-10 15:14:29', '2017-12-10 15:18:19'),
 ('2017-12-10 15:31:07', '2017-12-10 15:36:28'),
 ('2017-12-10 16:20:06', '2017-12-10 16:30:31'),
 ('2017-12-10 17:07:54', '2017-12-10 17:14:25'),
 ('2017-12-10 17:23:47', '2017-12-10 17:45:25'),
 ('2017-12-11 06:17:06', '2017-12-11 06:34:04'),
 ('2017-12-11 09:08:41', '2017-12-11 09:12:21'),
 ('2017-12-11 09:15:41', '2017-12-11 09:20:18'),
 ('2017-12-12 08:55:53', '2017-12-12 08:59:34'),
 ('2017-12-13 17:14:56', '2017-12-13 17:18:32'),
 ('2017-12-13 18:52:16', '2017-12-13 19:00:45'),
 ('2017-12-14 09:01:10', '2017-12-14 09:11:06'),
 ('2017-12-14 09:12:59', '2017-12-14 09:19:06'),
 ('2017-12-14 11:54:33', '2017-12-14 12:02:00'),
 ('2017-12-14 14:40:23', '2017-12-14 14:44:40'),
 ('2017-12-14 15:08:55', '2017-12-14 15:26:24'),
 ('2017-12-14 17:46:17', '2017-12-14 18:09:04'),
 ('2017-12-15 09:08:12', '2017-12-15 09:23:45'),
 ('2017-12-16 09:33:46', '2017-12-16 09:36:17'),
 ('2017-12-16 11:02:31', '2017-12-16 11:05:04'),
 ('2017-12-17 10:09:47', '2017-12-17 10:32:03'),
 ('2017-12-18 08:02:36', '2017-12-18 08:07:34'),
 ('2017-12-18 16:03:00', '2017-12-18 16:09:20'),
 ('2017-12-18 16:30:07', '2017-12-18 16:53:12'),
 ('2017-12-18 19:18:23', '2017-12-18 19:22:08'),
 ('2017-12-18 20:14:46', '2017-12-18 20:17:47'),
 ('2017-12-19 19:14:08', '2017-12-19 19:23:49'),
 ('2017-12-19 19:39:36', '2017-12-19 19:43:46'),
 ('2017-12-20 08:05:14', '2017-12-20 08:10:46'),
 ('2017-12-20 08:15:45', '2017-12-20 08:29:50'),
 ('2017-12-20 08:33:32', '2017-12-20 08:38:09'),
 ('2017-12-20 13:43:36', '2017-12-20 13:54:39'),
 ('2017-12-20 18:57:53', '2017-12-20 19:06:54'),
 ('2017-12-21 07:21:11', '2017-12-21 07:32:03'),
 ('2017-12-21 08:01:58', '2017-12-21 08:06:15'),
 ('2017-12-21 13:20:54', '2017-12-21 13:33:49'),
 ('2017-12-21 15:26:08', '2017-12-21 15:34:27'),
 ('2017-12-21 18:09:46', '2017-12-21 18:38:50'),
 ('2017-12-22 16:14:21', '2017-12-22 16:21:46'),
 ('2017-12-22 16:29:17', '2017-12-22 16:34:14'),
 ('2017-12-25 12:49:51', '2017-12-25 13:18:27'),
 ('2017-12-25 13:46:44', '2017-12-25 14:20:50'),
 ('2017-12-26 10:40:16', '2017-12-26 10:53:45'),
 ('2017-12-27 16:56:12', '2017-12-27 17:17:39'),
 ('2017-12-29 06:02:34', '2017-12-29 06:12:30'),
 ('2017-12-29 12:21:03', '2017-12-29 12:46:16'),
 ('2017-12-29 14:32:55', '2017-12-29 14:43:46'),
 ('2017-12-29 15:08:26', '2017-12-29 15:18:51'),
 ('2017-12-29 20:33:34', '2017-12-29 20:38:13'),
 ('2017-12-30 13:51:03', '2017-12-30 13:54:33'),
 ('2017-12-30 15:09:03', '2017-12-30 15:19:13')]
```

Parsing pairs of strings as datetimes

Up until now, I've been working with a pre-processed list of datetimes for W20529's trips. Here I am going to go one step back in the data cleaning pipeline and work with the strings that the data started as.

Exploring onebike_datetime_strings in the IPython shell to determine the correct format.

Reference

%Y 4 digit year (0000-9999)

%m 2 digit month (1-12)

%d 2 digit day (1-31)

%H 2 digit hour (0-23)

%M 2 digit minute (0-59)

%S 2 digit second (0-59)

[49]:
```
# Writing down the format string
fmt = "%Y-%m-%d %H:%M:%S"

# Initializing a list for holding the pairs of datetime objects
onebike_datetimes = []
```

```
# Looping over all trips
for (start, end) in onebike_datetime_strings:
  trip = {'start': datetime.strptime(start, fmt),
          'end': datetime.strptime(end, fmt)}

  # Appending the trip
  onebike_datetimes.append(trip)
```

[50]:
```
trip
```

[50]:
```
{'start': datetime.datetime(2017, 12, 30, 15, 9, 3),
 'end': datetime.datetime(2017, 12, 30, 15, 19, 13)}
```

Now I know how to process lists of strings into a more useful structure. If you haven't come across this approach before, many complex data cleaning tasks follow this same format: start with a list, process each element, and add the processed data to a new list.

Recreating ISO format with strftime()

Above I used strftime() to create strings from date objects. Now that we know about datetime objects, let's practice doing something similar.

Re-create the .isoformat() method, using .strftime(), and print the first trip start in the data set.

Reference %Y 4 digit year (0000-9999) %m 2 digit month (1-12) %d 2 digit day (1-31) %H 2 digit hour (0-23) %M 2 digit minute (0-59) %S 2 digit second (0-59)

[51]:
```
# Importing datetime
from datetime import datetime

# Pulling out the start of the first trip
first_start = onebike_datetimes[0]['start']

# Formatting to feed to strftime()
fmt = "%Y-%m-%dT%H:%M:%S"

# Printing out date with .isoformat(), then with .strftime() to compare
print(first_start.isoformat())
print(first_start.strftime(fmt))
```

```
2017-10-01T15:23:25
2017-10-01T15:23:25
```

There are a wide variety of time formats we can create with strftime(), depending on the needs. However, if you don't know exactly what you need, .isoformat() is a perfectly fine place to start.

Unix timestamps

Datetimes are sometimes stored as Unix timestamps: the number of seconds since January 1, 1970. This is especially common with computer infrastructure, like the log files that websites keep when they get visitors.

[52]:
```python
# Importing datetime
from datetime import datetime

# Starting timestamps
timestamps = [1514665153, 1514664543]

# Datetime objects
dts = []

# Looping
for ts in timestamps:
  dts.append(datetime.fromtimestamp(ts))

# Printing results
print(dts)
```

```
[datetime.datetime(2017, 12, 30, 15, 19, 13), datetime.datetime(2017, 12, 30,
15, 9, 3)]
```

The largest number that some older computers can hold in one variable is 2147483648, which as a Unix timestamp is in January 2038. On that day, many computers which haven't been upgraded will fail. Hopefully, none of them are running anything critical!

Turning pairs of datetimes into durations

When working with timestamps, we often want to know how much time has elapsed between events. Thankfully, we can use datetime arithmetic to ask Python to do the heavy lifting for us so we don't need to worry about day, month, or year boundaries. Let's calculate the number of seconds that the bike was out of the dock for each trip.

Continuing our work from a previous coding exercise, the bike trip data has been loaded as the list onebike_datetimes. Each element of the list consists of two datetime objects, corresponding to the start and end of a trip, respectively.

Within the loop:

Using arithmetic on the start and end elements to find the length of the trip

Saving the results to trip_duration.

Calculating trip_length_seconds from trip_duration

[53]:
```python
# Initialize a list for all the trip durations
onebike_durations = []

for trip in onebike_datetimes:
  # Create a timedelta object corresponding to the length of the trip
  trip_duration = trip['end'] - trip['start']

  # Getting the total elapsed seconds in trip_duration
```

```
    trip_length_seconds = trip_duration.total_seconds()

    # Appending the results to our list
    onebike_durations.append(trip_length_seconds)
```

Remember that timedelta objects are represented in Python as a number of days and seconds of elapsed time. Be careful not to use .seconds on a timedelta object, since you'll just get the number of seconds without the days!

```
[55]: trip_length_seconds
```

```
[55]: 610.0
```

```
[58]: onebike_durations
```

```
[58]: [181.0,
       7622.0,
       343.0,
       1278.0,
       1277.0,
       1366.0,
       815.0,
       545.0,
       491.0,
       639.0,
       1678.0,
       406.0,
       709.0,
       514.0,
       492.0,
       1668.0,
       2242.0,
       2752.0,
       735.0,
       330.0,
       518.0,
       1433.0,
       204.0,
       304.0,
       977.0,
       1399.0,
       1244.0,
       658.0,
       800.0,
       1911.0,
       2471.0,
       1344.0,
       435.0,
       271.0,
       920.0,
       851.0,
       209.0,
       453.0,
       841.0,
       142.0,
       1023.0,
       1466.0,
       1636.0,
       3039.0,
       1571.0,
       1410.0,
       386.0,
       1527.0,
       622.0,
       1450.0,
       1422.0,
       991.0,
       1484.0,
       1450.0,
       929.0,
       533.0,
       525.0,
       283.0,
       133.0,
       1106.0,
       952.0,
       553.0,
       659.0,
       297.0,
       357.0,
       989.0,
       979.0,
       760.0,
       1110.0,
       675.0,
       1207.0,
       1593.0,
       768.0,
       1446.0,
       485.0,
       200.0,
       399.0,
       242.0,
       170.0,
       450.0,
       1078.0,
       1042.0,
       573.0,
       748.0,
       735.0,
       336.0,
       76913.0,
       171.0,
       568.0,
       358.0,
       917.0,
       671.0,
       1791.0,
       318.0,
       888.0,
       1284.0,
       11338.0,
       1686.0,
       5579.0,
       8290.0,
       1850.0,
       1810.0,
       870.0,
       436.0,
       429.0,
       494.0,
       1439.0,
       380.0,
       629.0,
       962.0,
       387.0,
       952.0,
       190.0,
       739.0,
       1120.0,
       369.0,
       2275.0,
       873.0,
       1670.0,
       643.0,
       572.0,
       1375.0,
       725.0,
       688.0,
       1041.0,
       1707.0,
       1236.0,
       1291.0,
       2890.0,
       -3346.0,
       1213.0,
       331.0,
       1497.0,
       527.0,
       584.0,
       2599.0,
       759.0,
       1291.0,
       916.0,
       161.0,
       806.0,
       838.0,
       644.0,
       374.0,
       678.0,
       137.0,
       659.0,
       386.0,
       745.0,
       448.0,
       558.0,
       888.0,
       662.0,
       663.0,
       362.0,
       513.0,
       655.0,
       221.0,
       469.0,
       430.0,
       192.0,
       324.0,
       1233.0,
       923.0,
       961.0,
       525.0,
       1017.0,
       1216.0,
       747.0,
       668.0,
       1219.0,
       1182.0,
       10262.0,
       1106.0,
       399.0,
       724.0,
       330.0,
       499.0,
       968.0,
       1310.0,
       2629.0,
       427.0,
       839.0,
       258.0,
       396.0,
       238.0,
       745.0,
       613.0,
       710.0,
       2068.0,
       947.0,
       1509.0,
       254.0,
       625.0,
       479.0,
       688.0,
       238.0,
       322.0,
       304.0,
       576.0,
       1035.0,
       661.0,
       276.0,
       1427.0,
       998.0,
       729.0,
       723.0,
       220.0,
       212.0,
       759.0,
       268.0,
       374.0,
       305.0,
       304.0,
       289.0,
       2620.0,
       1288.0,
       212.0,
       2656.0,
       996.0,
       271.0,
       701.0,
       458.0,
       116.0,
       124.0,
       276.0,
       532.0,
       257.0,
       1089.0,
       195.0,
       384.0,
       511.0,
       850.0,
       462.0,
       322.0,
       998.0,
       327.0,
       153.0,
       152.0,
       230.0,
       321.0,
       625.0,
       391.0,
       1298.0,
       1018.0,
       220.0,
       277.0,
       221.0,
       216.0,
       509.0,
       596.0,
       367.0,
       447.0,
       257.0,
       1049.0,
       1367.0,
       933.0,
       151.0,
       153.0,
       1336.0,
       298.0,
       380.0,
       1385.0,
       225.0,
       181.0,
       581.0,
       250.0,
       332.0,
       845.0,
       277.0,
       663.0,
       541.0,
       652.0,
       257.0,
       775.0,
       499.0,
       1744.0,
       445.0,
       297.0,
       1716.0,
       2046.0,
       809.0,
       1287.0,
       596.0,
       1513.0,
       651.0,
       625.0,
       279.0,
       210.0,
       610.0]
```

Average trip time

W20529 took 291 trips in our data set. How long were the trips on average? We can use the built-in Python functions sum() and len() to make this calculation.

Based on the above coding exercise, the data has been loaded as onebike_durations. Each entry is a number of seconds that the bike was out of the dock.

```
[59]: # What was the total duration of all trips?
total_elapsed_time = sum(onebike_durations)

# What was the total number of trips?
number_of_trips = len(onebike_durations)

# Dividing the total duration by the number of trips
print(total_elapsed_time / number_of_trips)
```

```
1178.9310344827586
```

Great work, and not remotely average! For the average to be a helpful summary of the data, we need for all of our durations to be reasonable numbers, and not a few that are way too big, way too small, or even malformed. For example, if there is anything fishy happening in the data, and our trip ended before it started, we'd have a negative trip length.

The long and the short of why time is hard

Out of 291 trips taken by W20529, how long was the longest? How short was the shortest? Does anything look fishy?

[60]:
```python
# Calculating shortest and longest trips
shortest_trip = min(onebike_durations)
longest_trip = max(onebike_durations)

# Printing out the results
print("The shortest trip was " + str(shortest_trip) + " seconds")
print("The longest trip was " + str(longest_trip) + " seconds")
```

```
The shortest trip was -3346.0 seconds
The longest trip was 76913.0 seconds
```

Weird huh?! For at least one trip, the bike returned before it left. Why could that be? Here's a hint: it happened in early November, around 2AM local time. What happens to clocks around that time each year? By the end of the next coding, we'll have all the tools we need to deal with this situation!

Creating timezone aware datetimes

I will practice setting timezones manually.

- Importing timezone.
- Setting the tzinfo to UTC, without using timedelta.

[61]:
```python
# Importing datetime, timezone
from datetime import datetime, timezone

# October 1, 2017 at 15:26:26, UTC
dt = datetime(2017, 10, 1, 15, 26, 26, tzinfo=timezone.utc)

# Printing results
print(dt.isoformat())
```

```
2017-10-01T15:26:26+00:00
```

[62]:
```python
# Importing datetime, timedelta, timezone
from datetime import datetime, timedelta, timezone

# Creating a timezone for Pacific Standard Time, or UTC-8
pst = timezone(timedelta(hours=-8))

# October 1, 2017 at 15:26:26, UTC-8
dt = datetime(2017, 10, 1, 15, 26, 26, tzinfo= pst)

# Printing results
print(dt.isoformat())
```

```
2017-10-01T15:26:26-08:00
```

Setting tz to be a timezone set for UTC+11.

Setting dt's timezone to be tz.

[63]:
```python
# Importing datetime, timedelta, timezone
from datetime import datetime, timedelta, timezone

# Creating a timezone for Australian Eastern Daylight Time, or UTC+11
aedt = timezone(timedelta(hours=11))

# October 1, 2017 at 15:26:26, UTC+11
dt = datetime(2017, 10, 1, 15, 26, 26, tzinfo=aedt)

# Printing results
print(dt.isoformat())
```

```
2017-10-01T15:26:26+11:00
```

Did you know that Russia and France are tied for the most number of time zones, with 12 each? The French mainland only has one timezone, but because France has so many overseas dependencies they really add up!

Setting timezones

Now that I have the hang of setting timezones one at a time, let's look at setting them for the first ten trips that W20529 took. timezone and timedelta have already been imported. Making the change using .replace()

[64]:
```python
# Creating a timezone object corresponding to UTC-4
edt = timezone(timedelta(hours=-4))

# Looping over trips, updating the start and end datetimes to be in UTC-4
for trip in onebike_datetimes[:10]:
  # Updating trip['start'] and trip['end']
  trip['start'] = trip['start'].replace(tzinfo = edt)
  trip['end'] = trip['end'].replace(tzinfo = edt)
```

Awesome! Did you know that despite being over 2,500 miles (4,200 km) wide (about as wide as the continential United States or the European Union) China has only one official timezone? There's a second, unofficial timezone, too. It is used by much of the Uyghurs population in the Xinjiang province in the far west of China.

What time did the bike leave in UTC?

Having set the timezone for the first ten rides that W20529 took, let's see what time the bike left in UTC.

Within the for loop, setting dt to be the trip['start'] but moved to UTC. Using timezone.utc as a convenient shortcut for UTC.

[65]:
```python
# Looping over the trips
for trip in onebike_datetimes[:10]:
  # Pulling out the start and set it to UTC
  dt = trip['start'].astimezone(timezone.utc)

  # Printing the start time in UTC
  print('Original:', trip['start'], '| UTC:', dt.isoformat())
```

```
Original: 2017-10-01 15:23:25-04:00 | UTC: 2017-10-01T19:23:25+00:00
Original: 2017-10-01 15:42:57-04:00 | UTC: 2017-10-01T19:42:57+00:00
Original: 2017-10-02 06:37:10-04:00 | UTC: 2017-10-02T10:37:10+00:00
Original: 2017-10-02 08:56:45-04:00 | UTC: 2017-10-02T12:56:45+00:00
Original: 2017-10-02 18:23:48-04:00 | UTC: 2017-10-02T22:23:48+00:00
Original: 2017-10-02 18:48:08-04:00 | UTC: 2017-10-02T22:48:08+00:00
Original: 2017-10-02 19:18:10-04:00 | UTC: 2017-10-02T23:18:10+00:00
Original: 2017-10-02 19:37:32-04:00 | UTC: 2017-10-02T23:37:32+00:00
Original: 2017-10-03 08:24:16-04:00 | UTC: 2017-10-03T12:24:16+00:00
Original: 2017-10-03 18:17:07-04:00 | UTC: 2017-10-03T22:17:07+00:00
```

Did you know that there is no official time zone at the North or South pole? Since all the lines of longitude meet each other, it's up to each traveler (or research station) to decide what time they want to use.

Putting the bike trips into the right time zone

Instead of setting the timezones for W20529 by hand, let's assign them to their IANA timezone: 'America/New_York'. Since we know their political jurisdiction, we don't need to look up their UTC offset. Python will do that for us.

[68]:
```python
# Importing tz
from dateutil import tz

# Creating a timezone object for Eastern Time
et = tz.gettz('America/New_York')

# Looping over trips, updating the datetimes to be in Eastern Time
for trip in onebike_datetimes[:10]:
  # Updating trip['start'] and trip['end']
  trip['start'] = trip['start'].replace(tzinfo = et)
  trip['end'] = trip['end'].replace(tzinfo = et)
```

Great! Time zone rules actually change quite frequently. IANA time zone data gets updated every 3-4 months, as different jurisdictions make changes to their laws about time or as more historical information about timezones are uncovered. tz is smart enough to use the date in your datetime to determine which rules to use historically.

What time did the bike leave? (Global edition)

When you need to move a datetime from one timezone into another, use .astimezone() and tz. Often you will be moving things into UTC, but for fun let's try moving things from 'America/New_York' into a few different time zones.

[69]:
```
# Creating the timezone object
uk = tz.gettz('Europe/London')

# Pulling out the start of the first trip
local = onebike_datetimes[0]['start']

# What time was it in the UK?
notlocal = local.astimezone(uk)

# Printing them out and see the difference
print(local.isoformat())
print(notlocal.isoformat())
```

```
2017-10-01T15:23:25-04:00
2017-10-01T20:23:25+01:00
```

Set ist to be the timezone for India: 'Asia/Kolkata'.

Change local to be in the ist timezone and assign it to notlocal.

[70]:
```
# Createing the timezone object
ist = tz.gettz('Asia/Kolkata')

# Pulling out the start of the first trip
local = onebike_datetimes[0]['start']

# What time was it in India?
notlocal = local.astimezone(ist)

# Printing them out and see the difference
print(local.isoformat())
print(notlocal.isoformat())
```

```
2017-10-01T15:23:25-04:00
2017-10-02T00:53:25+05:30
```

Setting sm to be the timezone for Samoa: 'Pacific/Apia'.

Changing local to be in the sm timezone and assign it to notlocal.

[71]:
```
# Creating the timezone object
sm = tz.gettz('Pacific/Apia')

# Pulling out the start of the first trip
local = onebike_datetimes[0]['start']

# What time was it in Samoa?
notlocal = local.astimezone(sm)
```

```
# Printing them out and see the difference
print(local.isoformat())
print(notlocal.isoformat())
```

```
2017-10-01T15:23:25-04:00
2017-10-02T09:23:25+14:00
```

Did you notice the time offset for this one? It's at UTC+14! Samoa used to be UTC-10, but in 2011 it changed to the other side of the International Date Line to better match New Zealand, its closest trading partner. However, they wanted to keep the clocks the same, so the UTC offset shifted from -10 to +14, since 24-10 is 14. Timezones... not simple!

How many hours elapsed around daylight saving?

Since our bike data takes place in the fall, we'll have to do something else to learn about the start of daylight savings time.

Let's look at March 12, 2017, in the Eastern United States, when Daylight Saving kicked in at 2 AM.

If you create a datetime for midnight that night, and add 6 hours to it, how much time will have elapsed?

[72]:
```
# Importing datetime, timedelta, tz, timezone
from datetime import datetime, timedelta, timezone
from dateutil import tz

# Starting on March 12, 2017, midnight, then add 6 hours
start = datetime(2017, 3, 12, tzinfo = tz.gettz('America/New_York'))
end = start + timedelta(hours=6)
print(start.isoformat() + " to " + end.isoformat())
```

```
2017-03-12T00:00:00-05:00 to 2017-03-12T06:00:00-04:00
```

You added 6 hours, and got 6 AM, despite the fact that the clocks springing forward means only 5 hours would have actually elapsed!

Calculate the time between start and end. How much time does Python think has elapsed?

[73]:
```
# Importing datetime, timedelta, tz, timezone
from datetime import datetime, timedelta, timezone
from dateutil import tz

# Starting on March 12, 2017, midnight, then add 6 hours
start = datetime(2017, 3, 12, tzinfo = tz.gettz('America/New_York'))
end = start + timedelta(hours=6)
print(start.isoformat() + " to " + end.isoformat())

# How many hours have elapsed?
print((end - start).total_seconds()/(60*60))
```

```
2017-03-12T00:00:00-05:00 to 2017-03-12T06:00:00-04:00
6.0
```

Move your datetime objects into UTC and calculate the elapsed time again.

Once you're in UTC, what result do you get?

```
[ ]: # Importing datetime, timedelta, tz, timezone
     from datetime import datetime, timedelta, timezone
     from dateutil import tz

     # Starting on March 12, 2017, midnight, then add 6 hours
     start = datetime(2017, 3, 12, tzinfo = tz.gettz('America/New_York'))
     end = start + timedelta(hours=6)
     print(start.isoformat() + " to " + end.isoformat())

     # How many hours have elapsed?
     print((end - start).total_seconds()/(60*60))

     # What if we move to UTC?
     print((end.astimezone(timezone.utc) - start.astimezone(timezone.utc))\
           .total_seconds()/(60*60))
```

When we compare times in local time zones, everything gets converted into clock time. Remember if you want to get absolute time differences, always move to UTC!

March 29, throughout a decade

Daylight Saving rules are complicated: they're different in different places, they change over time, and they usually start on a Sunday (and so they move around the calendar).

For example, in the United Kingdom, Daylight Saving begins on the last Sunday in March. Let's look at the UTC offset for March 29, at midnight, for the years 2000 to 2010.

```
[ ]: # Importing datetime and tz
     from datetime import datetime
     from dateutil import tz

     # Creating starting date
     dt = datetime(2000, 3, 29, tzinfo = tz.gettz('Europe/London'))

     # Looping over the dates, replacing the year, and print the ISO timestamp
     for y in range(2000, 2011):
       print(dt.replace(year=y).isoformat())
```

As you can see, the rules for Daylight Saving are not trivial. When in doubt, always use tz instead of hand-rolling timezones, so it will catch the Daylight Saving rules (and rule changes!) for you.

```
[ ]: Finding ambiguous datetimes
```

```
Above we saw something anomalous in our bike trip duration data. Let's see if␣
↪we can identify what the problem might be.

The data has is loaded as onebike_datetimes, and tz has already been imported␣
↪from dateutil.
```

[ ]:
```
# Loop over trips
for trip in onebike_datetimes:
  # Rides with ambiguous start
  if tz.datetime_ambiguous(trip['start']):
    print("Ambiguous start at " + str(trip['start']))
  # Rides with ambiguous end
  if tz.datetime_ambiguous(trip['end']):
    print("Ambiguous end at " + str(trip['end']))
```

Cleaning daylight saving data with fold

As we've just discovered, there is a ride in our data set which is being messed up by a Daylight Savings shift. Let's clean up the data set so we actually have a correct minimum ride length. We can use the fact that we know the end of the ride happened after the beginning to fix up the duration messed up by the shift out of Daylight Savings.

Since Python does not handle tz.enfold() when doing arithmetic, we must put our datetime objects into UTC, where ambiguities have been resolved.

onebike_datetimes is already loaded and in the right timezone. tz and timezone have been imported. Use tz.UTC for your timezone.

Hint

Use trip['start'] to retrieve the start time of the trip.

Remember that .replace() sets the tzinfo of a datetime, but .astimezone() will change the date and time to match the new timezone.

[ ]:
```
trip_durations = []
for trip in onebike_datetimes:
  # When the start is later than the end, set the fold to be 1
  if trip['start'] > trip['end']:
    trip['end'] = tz.enfold(trip['end'])
  # Converting to UTC
  start = trip['start'].astimezone(tz.UTC)
  end = trip['end'].astimezone(tz.UTC)

  # Subtracting the difference
  trip_length_seconds = (end-start).total_seconds()
  trip_durations.append(trip_length_seconds)

# Taking the shortest trip duration
print("Shortest trip: " + str(min(trip_durations)))
```

Good work! Now you know how to handle some pretty gnarly edge cases in datetime data. To give a sense for how tricky these things are: we actually still don't know how long the rides are which only started or ended in our ambiguous hour but not both. If you're collecting data, store it in UTC or with a fixed UTC offset!

Loading a csv file in Pandas

The capital_onebike.csv file covers the October, November and December rides of the Capital Bikeshare bike W20529.

[129]:
```
# Importing pandas
import pandas as pd

# Loading CSV into the rides variable
rides = pd.read_csv('bike_share.csv',
                    parse_dates = ['Start date', 'End date'])

# Printing the initial (0th) row
print(rides.iloc[0])
```

```
Start date                         2017-10-01 15:23:00
End date                           2017-10-01 15:26:00
Start station number                             31038
Start station                     Glebe Rd & 11th St N
End station number                               31036
End station                George Mason Dr & Wilson Blvd
Bike number                                     W20529
Member type                                     Member
Name: 0, dtype: object
```

[ ]:
```
Making timedelta columns

Earlier I wrote a loop to subtract datetime objects and determined how long our␣
 ↪sample bike had been out of the docks. Now I'll do the same thing with␣
 ↪Pandas.
```

[115]:
```
# Subtracting the start date from the end date
ride_durations = rides['End date'] - rides['Start date']

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

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

```
0     180.0
1    7620.0
2     300.0
3    1320.0
4    1320.0
```

```
Name: Duration, dtype: float64
```

Great! Because Pandas supports method chaining, I could also perform this operation in one line: rides[‘Duration’] = (rides[‘End date’] - rides[‘Start date’]).dt.total_seconds()

How many joyrides?

Suppose you have a theory that some people 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.

[116]:
```
# Creating joyrides
joyrides = (rides['Start station'] == rides['End station'])

# 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()))
```

```
6 rides were joyrides
The median duration overall was 660.00 seconds
The median duration for joyrides was 2640.00 seconds
```

[117]:
```
rides.head(2)
```

[117]:
```
           Start date            End date  Start station number  \
0 2017-10-01 15:23:00 2017-10-01 15:26:00                 31038
1 2017-10-01 15:42:00 2017-10-01 17:49:00                 31036

                   Start station  End station number  \
0            Glebe Rd & 11th St N               31036
1  George Mason Dr & Wilson Blvd               31036

                     End station Bike number Member type  Duration
0  George Mason Dr & Wilson Blvd      W20529      Member     180.0
1  George Mason Dr & Wilson Blvd      W20529      Casual    7620.0
```

Great work! Pandas makes analyses like these concise to write and reason about. Writing this as a for loop would have been more complex.

It’s getting cold outside, W20529

Washington, D.C. has mild weather overall, but the average high temperature in October (68ºF / 20ºC) is certainly higher than the average high temperature in December (47ºF / 8ºC). People also travel more in December, and they work fewer days so they commute less.

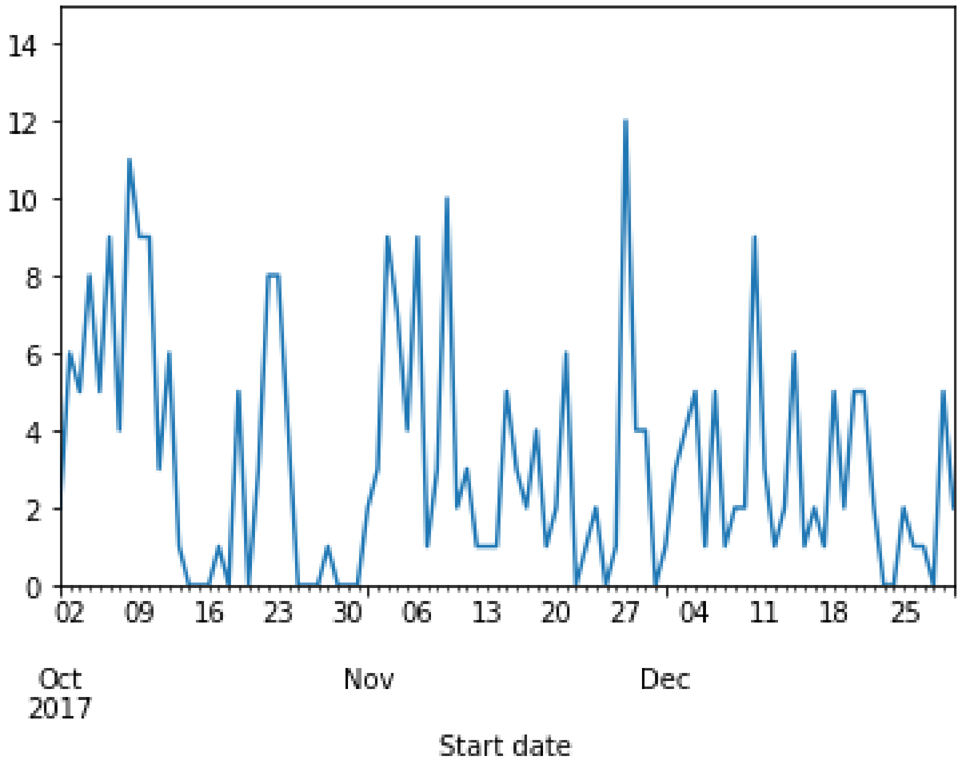
How might the weather or the season have affected the length of bike trips?

How might the weather or the season have affected the length of bike trips?

[119]:
```python
# Importing matplotlib
import matplotlib.pyplot as plt

# Resampling rides to daily, take the size, plot the results
rides.resample('D', on = 'Start date')\
  .size()\
  .plot(ylim = [0, 15])

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

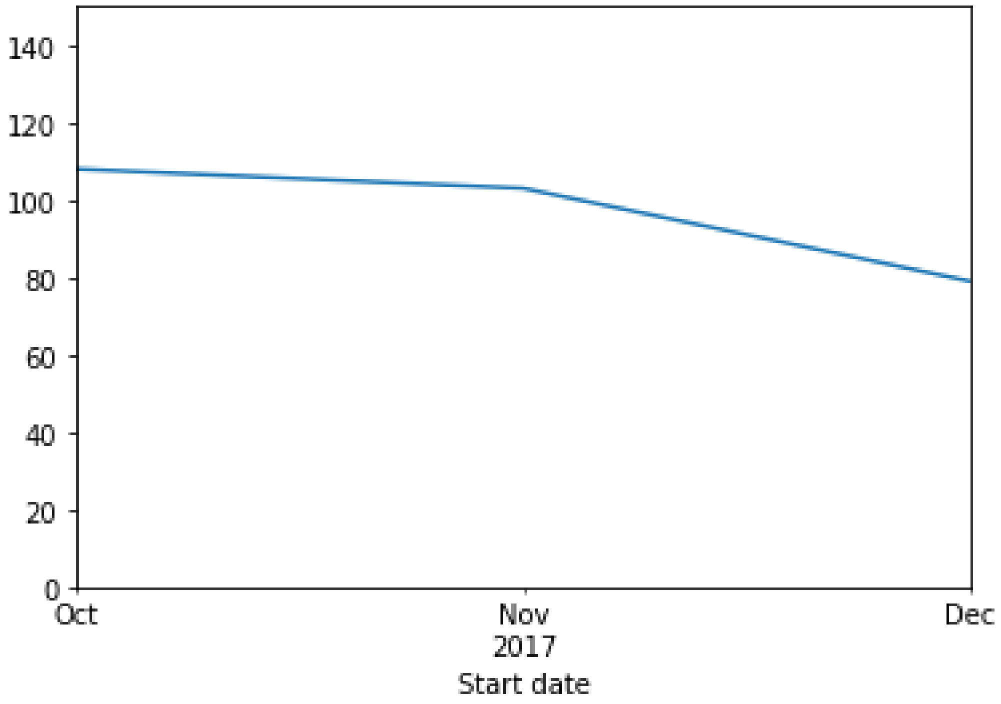


Since the daily time series is so noisy for this one bike, changing the resampling to be monthly.

```
[120]: # Importing matplotlib
import matplotlib.pyplot as plt

# Resampling rides to monthly, take the size, plot the results
rides.resample('M', on = 'Start date')\
  .size()\
  .plot(ylim = [0, 150])

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



As you can see, the pattern is clearer at the monthly level: there were fewer rides in November, and then fewer still in December, possibly because the temperature got colder.

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.

Do members and casual riders drop off at the same rate over October to December, or does one drop off faster than the other?

I'm going to use the Pandas method .value_counts(), which returns the number of instances of each value in a Series. In this case, the counts of "Member" or "Casual".

```
[ ]: # Resampling rides to be monthly on the basis of Start date
     monthly_rides = rides.resample('M', on = 'Start date')['Member type']

     # Taking the ratio of the .value_counts() over the total number of rides
     print(monthly_rides.value_counts() / monthly_rides.size())
```

Nice! Note that by default, .resample() labels Monthly resampling with the last day in the month and not the first. It certainly looks like the fraction of Casual riders went down as the number of rides dropped. With a little more digging, you could figure out if keeping Member rides only would be enough to stabilize the usage numbers throughout the fall.

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 month, and by Membership type?

```
[122]: # Groupping rides by member type, and resample to the month
       grouped = rides.groupby('Member type')\
         .resample('M', on = 'Start date')

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

```
Member type  Start date
Casual       2017-10-31    1620.0
             2017-11-30    1170.0
             2017-12-31     840.0
Member       2017-10-31     660.0
             2017-11-30     660.0
             2017-12-31     420.0
Name: Duration, dtype: float64
```

Nice! It looks like casual riders consistently took longer rides, but that both groups took shorter rides as the months went by. Note that, by combining grouping and resampling, you can answer a lot of questions about nearly any data set that includes time as a feature. Keep in mind that you can also group by more than one column at once.

Timezones in Pandas

Earlier I assigned a timezone to each datetime in a list. Now with Pandas I can do that with a single method call.

(Note that, just as before, the data set actually includes some ambiguous datetimes on account of daylight saving; for now, we'll tell Pandas to not even try on those ones. Figuring them out would require more work.)

[123]:
```
# Localizing the Start date column to America/New_York
rides['Start date'] = rides['Start date'].dt.tz_localize('America/New_York',␣
 ↪ambiguous = 'NaT')

# Printing first value
print(rides['Start date'].iloc[0])
```

```
2017-10-01 15:23:00-04:00
```

Now switching the Start date column to the timezone 'Europe/London' using the .dt.tz_convert() method.

[125]:
```
# Converting the Start date column to Europe/London
rides['Start date'] = rides['Start date'].dt.tz_convert('Europe/London')

# Printing the new value
print(rides['Start date'].iloc[0])
```

```
2017-10-01 20:23:00+01:00
```

Nicely done! dt.tzconvert() converts to a new timezone, whereas dt.tzlocalize() sets a timezone in the first place. You now know how to deal with datetimes in Pandas.

How long per weekday?

Pandas has a number of datetime-related attributes within the .dt accessor. Many of them are ones you've encountered before, like .dt.month. Others are convenient and save time compared to standard Python, like .dt.weekday_name.

[126]:
```
# Adding a column for the weekday of the start of the ride
rides['Ride start weekday'] = rides['Start date'].dt.weekday_name

# Printing the median trip time per weekday
print(rides.groupby('Ride start weekday')['Duration'].median())
```

```
Ride start weekday
Friday       690.0
Monday       930.0
Saturday     600.0
Sunday       600.0
Thursday     660.0
Tuesday      660.0
Wednesday    660.0
Name: Duration, dtype: float64
```

Well done! There are .dt attributes for all of the common things you might want to pull out of a datetime, such as the day, month, year, hour, and so on, and also some additional convenience ones, such as quarter and week of the year out of 52.

How long between rides?

Here I will take advantage of Pandas indexing to do something interesting. How much time elapsed between rides?

Calculating the difference in the Start date of the current row and the End date of the previous row and assign it to rides['Time since'].

Converting rides['Time since'] to seconds to make it easier to work with.

Resampling rides to be in monthly buckets according to the Start date.

Dividing the average by (60*60) to get the number of hours on average that W20529 waited in the dock before being picked up again.

[130]:
```python
# Shifting the index of the end date up one; now subract it from the start date
rides['Time since'] = rides['Start date'] - (rides['End date'].shift(1))

# Moveing from a timedelta to a number of seconds, which is easier to work with
rides['Time since'] = rides['Time since'].dt.total_seconds()

# Resampling to the month
monthly = rides.resample('M', on = 'Start date')

# Printing the average hours between rides each month
print(monthly['Time since'].mean()/(60*60))
```

```
Start date
2017-10-31    5.520405
2017-11-30    7.256149
2017-12-31    9.201688
Freq: M, Name: Time since, dtype: float64
```

Great job! As you can see, there are a huge number of Pandas tricks that let you express complex logic in just a few lines, assuming you understand how the indexes actually work. If you haven't taken it yet, have you considered taking Pandas Foundations? In addition to lots of other useful Pandas information, it covers working with time series (like stock prices) in Pandas. Time series have many overlapping techniques with datetime data.

2 Thanks for your attention.

[ ]:
```python

```