(http://www.pieriandata.com)

Time Methods

Python Datetime Review

Basic Python outside of Pandas contains a datetime library:

In [1]:
```
from datetime import datetime
```

In [2]:
```
# To illustrate the order of arguments
my_year = 2017
my_month = 1
my_day = 2
my_hour = 13
my_minute = 30
my_second = 15
```

In [3]:
```
# January 2nd, 2017
my_date = datetime(my_year,my_month,my_day)
```

In [4]:
```
# Defaults to 0:00
my_date
```

Out[4]:
```
datetime.datetime(2017, 1, 2, 0, 0)
```

In [5]:
```
# January 2nd, 2017 at 13:30:15
my_date_time = datetime(my_year,my_month,my_day,my_hour,my_minute,my_second
```

In [6]:
```
my_date_time
```

Out[6]:
```
datetime.datetime(2017, 1, 2, 13, 30, 15)
```

You can grab any part of the datetime object you want

In [7]:
```
my_date.day
```

Out[7]:
```
2
```

In [8]:
```
my_date_time.hour
```

Out[8]:
```
13
```

Pandas

Converting to datetime

Often when data sets are stored, the time component may be a string. Pandas easily converts strings to datetime objects.

In [9]:
```
import pandas as pd
```

In [10]:
```
myser = pd.Series(['Nov 3, 2000', '2000-01-01', None])
```

In [11]:
```
myser
```

Out[11]:
```
0    Nov 3, 2000
1     2000-01-01
2           None
dtype: object
```

In [12]:
```
myser[0]
```

Out[12]:
```
'Nov 3, 2000'
```

pd.to_datetime()

https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#converting-to-timestamps (https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#converting-to-timestamps)

In [13]:
```
pd.to_datetime(myser)
```

Out[13]:
```
0   2000-11-03
1   2000-01-01
2          NaT
dtype: datetime64[ns]
```

In [14]:
```
pd.to_datetime(myser)[0]
```

Out[14]:
```
Timestamp('2000-11-03 00:00:00')
```

In [15]:
```
obvi_euro_date = '31-12-2000'
```

In [16]:
```
pd.to_datetime(obvi_euro_date)
```

Out[16]:
```
Timestamp('2000-12-31 00:00:00')
```

In [17]:
```
# 10th of Dec OR 12th of October?
# We may need to tell pandas
euro_date = '10-12-2000'
```

In [18]:
```
pd.to_datetime(euro_date)
```

Out[18]:
```
Timestamp('2000-10-12 00:00:00')
```

In [19]:
```
pd.to_datetime(euro_date,dayfirst=True)
```

Out[19]:
```
Timestamp('2000-12-10 00:00:00')
```

Custom Time String Formatting

Sometimes dates can have a non standard format, luckily you can always specify to pandas the format. You should also note this could speed up the conversion, so it may be worth doing even if pandas can parse on its own.

A full table of codes can be found here:
https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes (https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes)

In [20]:
```
style_date = '12--Dec--2000'
```

In [21]:
```
pd.to_datetime(style_date, format='%d--%b--%Y')
```

Out[21]:
```
Timestamp('2000-12-12 00:00:00')
```

In [22]:
```
strange_date = '12th of Dec 2000'
```

In [23]:
```
pd.to_datetime(strange_date)
```

Out[23]:
```
Timestamp('2000-12-12 00:00:00')
```

Data

Retail Sales: Beer, Wine, and Liquor Stores

Units: Millions of Dollars, Not Seasonally Adjusted

Frequency: Monthly

U.S. Census Bureau, Retail Sales: Beer, Wine, and Liquor Stores [MRTSSM4453USN], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/MRTSSM4453USN (https://fred.stlouisfed.org/series/MRTSSM4453USN), July 2, 2020.

In [24]:
```
sales = pd.read_csv('RetailSales_BeerWineLiquor.csv')
```

In [25]:
```
sales
```

Out[25]:

	DATE	MRTSSM4453USN
0	1992-01-01	1509
1	1992-02-01	1541
2	1992-03-01	1597
3	1992-04-01	1675
4	1992-05-01	1822
...	...	...
335	2019-12-01	6630
336	2020-01-01	4388
337	2020-02-01	4533
338	2020-03-01	5562
339	2020-04-01	5207

340 rows × 2 columns

In [26]:
```
sales.iloc[0]['DATE']
```

Out[26]:
```
'1992-01-01'
```

In [27]:
```
type(sales.iloc[0]['DATE'])
```

Out[27]:
```
str
```

In [28]:
```
sales['DATE'] = pd.to_datetime(sales['DATE'])
```

In [29]:
```
sales
```

Out[29]:

	DATE	MRTSSM4453USN
0	1992-01-01	1509
1	1992-02-01	1541
2	1992-03-01	1597
3	1992-04-01	1675
4	1992-05-01	1822
...	...	...
335	2019-12-01	6630
336	2020-01-01	4388
337	2020-02-01	4533
338	2020-03-01	5562
339	2020-04-01	5207

340 rows × 2 columns

In [30]:
```
sales.iloc[0]['DATE']
```

Out[30]:
```
Timestamp('1992-01-01 00:00:00')
```

In [31]:
```
type(sales.iloc[0]['DATE'])
```

Out[31]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

Attempt to Parse Dates Automatically

parse_dates - bool or list of int or names or list of lists or dict, default False The behavior is as follows:

```
boolean. If True -> try parsing the index.

list of int or names. e.g. If [1, 2, 3] -> try parsing columns 1,
2, 3 each as a separate date column.
```

In [32]:
```
# Parse Column at Index 0 as Datetime
sales = pd.read_csv('RetailSales_BeerWineLiquor.csv',parse_dates=[0])
```

In [33]:
```
sales
```

Out[33]:

	DATE	MRTSSM4453USN
0	1992-01-01	1509
1	1992-02-01	1541
2	1992-03-01	1597
3	1992-04-01	1675
4	1992-05-01	1822
...	...	...
335	2019-12-01	6630
336	2020-01-01	4388
337	2020-02-01	4533
338	2020-03-01	5562
339	2020-04-01	5207

340 rows × 2 columns

In [34]:
```
type(sales.iloc[0]['DATE'])
```

Out[34]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

Resample

A common operation with time series data is resampling based on the time series index. Let's see how to use the resample() method. [reference (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.resample.html)]

In [35]:
```
# Our index
sales.index
```

Out[35]:
```
RangeIndex(start=0, stop=340, step=1)
```

In [36]:
```
# Reset DATE to index
```

In [37]:
```
sales = sales.set_index("DATE")
```

In [38]:
```
sales
```

Out[38]:

DATE	MRTSSM4453USN
1992-01-01	1509
1992-02-01	1541
1992-03-01	1597
1992-04-01	1675
1992-05-01	1822
...	...
2019-12-01	6630
2020-01-01	4388
2020-02-01	4533
2020-03-01	5562
2020-04-01	5207

340 rows × 1 columns

When calling `.resample()` you first need to pass in a **rule** parameter, then you need to call some sort of aggregation function.

The **rule** parameter describes the frequency with which to apply the aggregation function (daily, monthly, yearly, etc.)
It is passed in using an "offset alias" - refer to the table below. [reference (http://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases)]

The aggregation function is needed because, due to resampling, we need some sort of mathematical rule to join the rows (mean, sum, count, etc.)

TIME SERIES OFFSET ALIASES

ALIAS	DESCRIPTION
B	business day frequency
C	custom business day frequency (experimental)
D	calendar day frequency
W	weekly frequency
M	month end frequency
SM	semi-month end frequency (15th and end of month)
BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
SMS	semi-month start frequency (1st and 15th)
BMS	business month start frequency
CBMS	custom business month start frequency
Q	quarter end frequency

intentionally left blank

ALIAS	DESCRIPTION
BQ	business quarter endfrequency
QS	quarter start frequency
BQS	business quarter start frequency
A	year end frequency
BA	business year end frequency
AS	year start frequency
BAS	business year start frequency
BH	business hour frequency
H	hourly frequency
T, min	minutely frequency
S	secondly frequency
L, ms	milliseconds
U, us	microseconds
N	nanoseconds

In [39]:
```
# Yearly Means
sales.resample(rule='A').mean()
```

Out[39]:

DATE	MRTSSM4453USN
1992-12-31	1807.250000
1993-12-31	1794.833333
1994-12-31	1841.750000
1995-12-31	1833.916667
1996-12-31	1929.750000
1997-12-31	2006.750000
1998-12-31	2115.166667
1999-12-31	2206.333333
2000-12-31	2375.583333
2001-12-31	2468.416667
2002-12-31	2491.166667
2003-12-31	2539.083333
2004-12-31	2682.416667
2005-12-31	2797.250000
2006-12-31	3001.333333
2007-12-31	3177.333333
2008-12-31	3292.000000
2009-12-31	3353.750000
2010-12-31	3450.083333
2011-12-31	3532.666667
2012-12-31	3697.083333
2013-12-31	3839.666667
2014-12-31	4023.833333
2015-12-31	4212.500000
2016-12-31	4434.416667
2017-12-31	4602.666667
2018-12-31	4830.666667
2019-12-31	4972.750000
2020-12-31	4922.500000

Resampling rule 'A' takes all of the data points in a given year, applies the aggregation function (in this case we calculate the mean), and reports the result as the last day of that year. Note 2020 in this data set was not complete.

.dt Method Calls

Once a column or index is ina datetime format, you can call a variety of methods off of the .dt library inside pandas:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.dt.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.dt.html)

In [44]:
```
sales = sales.reset_index()
```

In [45]:
```
sales
```

Out[45]:

	DATE	MRTSSM4453USN
0	1992-01-01	1509
1	1992-02-01	1541
2	1992-03-01	1597
3	1992-04-01	1675
4	1992-05-01	1822
...	...	...
335	2019-12-01	6630
336	2020-01-01	4388
337	2020-02-01	4533
338	2020-03-01	5562
339	2020-04-01	5207

340 rows × 2 columns

In [46]:
```
help(sales['DATE'].dt)
```

```
Help on DatetimeProperties in module pandas.core.indexes.accessors obje
ct:

class DatetimeProperties(Properties)
 |  Accessor object for datetimelike properties of the Series values.
 |
 |  Examples
 |  --------
 |  >>> s.dt.hour
 |  >>> s.dt.second
 |  >>> s.dt.quarter
 |
 |  Returns a Series indexed like the original Series.
 |  Raises TypeError if the Series does not contain datetimelike value
s.
 |
 |  Method resolution order:
 |      DatetimeProperties
 |      Properties
```

In [48]:
```
sales['DATE'].dt.month
```

Out[48]:
```
0       1
1       2
2       3
3       4
4       5
       ..
335    12
336     1
337     2
338     3
339     4
Name: DATE, Length: 340, dtype: int64
```

In [50]:
```
sales['DATE'].dt.is_leap_year
```

Out[50]:
```
0       True
1       True
2       True
3       True
4       True
       ...
335    False
336     True
337     True
338     True
339     True
Name: DATE, Length: 340, dtype: bool
```