날짜 모듈

컴퓨터의 시간 표현법

❖Time Stamp: 1970년 1월 1일 자정을 기준으로 지나온 시간을 초 또는 1/1000초 단위의 정수로 가지는 것으로 보통 epoch time이라고 합니다.

❖UTC(Universal Time Coordinated): 1972년 부터 시행된 국제 표준시로 세슘 원자의 진동수에 의거한 초의 길이

❖GMT(Greenwich Mean Time): 런던 그리니치 천문대의 자오선상에서의 평균 태양시

❖지방 표준시(LST, Local Standard Time): UTC를 기준으로 경도 15도마다 1시간 차이가 발생하는 시간

time 모듈

❖struct_time 시퀀스 객체(구조체)

- ✓ Tm_year, tm_mon, tm_mday, tm_hour, tm_min, tm_sec, tm_wday, tm_yday, tm_isdst

❖time.time(): 1970년 1월 1일 자정 이후로 누적된 초를 float으로 반환

❖time.sleep(secs): 현재 동작 중인 스레드를 주어진 초 만큼 정지

❖time.gmtime([secs]): 입력된 초를 반환해서 struct_time 시퀀스 객체로 반환하는데 파라미터를 대입하지 않으면 현재 시간

❖time.localtime([secs]): 지방 표준시 기준으로 struct_time 시퀀스 객체 반환

time 모듈

```
import time
time.sleep(10)
print(time.time()) #1970년 1월 1일 부터 누적된 시간
print(time.gmtime()) #UTC 시간
print(time.localtime()) #현재 지방시
```

날짜 데이터

❖파이썬 표준 라이브러리의 datetime 패키지

- ✓ 서브 클래스
  - datetime 클래스 : 날짜+시간 복합 정보
  - date 클래스: 날짜 정보
  - time 클래스: 시간 정보
  - timedelta 클래스: 시간 차이 정보
  - tzinfo 클래스: 시간대 정보
- ✓ date 클래스
  - datetime.date(년, 월, 일)
  - 클래스 메서드
    - ✓ Today()

날짜 데이터

- datetime.datetime 클래스
  - ✓ 날짜와 시간 정보를 가지는 클래스
  - ✓ 생성자: datetime(year, month, day[, hour[,minute[,second[,microsecond[,tzinfo]]]]])
  - ✓ 클래스 메서드
    - today(), now([tzinfo]): 현재 날짜 및 시각으로 생성
    - Strptime(date_string, format): 문자열 -> datetime
    - fromtimestamp(timestamp): timestamp -> datetime
    - fromordinal(ordinal): 1년1월1일 부터의 누적된 날짜 -> datetime
    - Combine(date, time): date + time -> datetime

날짜 데이터

- datetime.datetime 클래스
  - ✓ 속성: 읽기 전용
    - year, month, day, hour, minute, second, microsecond
    - tzinfo
  - ✓ 인스턴스 메서드
    - weekday(): 요일 {0:월, 1:화, 2:수, 3:목, 4:금, 5:토, 6:일} , 1부터 시작하는 isoweekday()
    - Strftime(format): datetime -> 문자열
    - toordinal(): datetime -> proleptic Gregorian ordinal (엑셀 날짜)
    - Date(): datetime -> date
    - Time(): datetime -> time
  - ✓ 문자열 변환
    - parsing

    문자열 -> datetime : strptime(date_string, format)
    - formatting

    datetime -> 문자열: strftime(format)

날짜 데이터

- datetime.datetime 클래스
  - ✓ 포맷 기호
    - %Y: 네자리 년도
    - %y: 두자리 년도
    - %m: 두자리 월
    - %d: 두자리 일
    - %H: 24시간
    - %I: 12시간
    - %M: 분, zero-padded decimal
    - %S: 초, zero-padded decimal

날짜 데이터

```
import datetime
dt = datetime.datetime.now()
print("현재시간:", dt)
print(dt.year, dt.month, dt.day, dt.hour, dt.minute, dt.second, dt.microsecond, dt.tzinfo)
print("요일:",dt.weekday())
dt1 = datetime.datetime.strptime("2015-12-31 11:32", "%Y-%m-%d %H:%M")
print(dt1)
s = dt1.strftime('%Y %m %d %H %M %S')
print(s)
```

날짜 데이터

❖ datetime, date, time 변환

✓ datetime -> date 또는 time

date() 메소드 또는 time() 메소드

✓ date + time -> datetime

combine() 메소드

❖ timedelta와 날짜시간 연산

✓ 빼기

datetime - datetime => timedelta

✓ 더하기

datetime + timedelta => datetime

날짜 데이터

- timedelta 클래스
  - ✓ days: 일수
  - ✓ seconds: 초. (0 ~ 86399)
  - ✓ microseconds: 마이크로초 (0 and 999999)
  - ✓ total_seconds(): 모든 속성을 초단위로 모아서 변환

- time 패키지
  - ✓ platform C library 기반의 low 레벨 시간처리
  - ✓ sleep 함수: 일정 시간 동안 프로세스 정지

날짜 데이터

- dateutil 패키지
  - ✓ parse() 함수: 많이 사용되는 날짜 시간 관련 문자열 형식을 스스로 판단

날짜 데이터

```
import datetime
dt = datetime.datetime.now()
print(dt.date(), dt.time())
d = datetime.date(2015, 12, 31)
print(d)
t = datetime.time(11, 31, 29)
print(datetime.datetime.combine(d, t))

dt1 = datetime.datetime(2016, 2, 19, 14, 11, 9)
dt2 = datetime.datetime(2016, 1, 2, 13, 16, 23)
td = dt1 - dt2
print(td)
print(td.days, td.seconds, td.microseconds)
```

날짜 데이터

```
print("start...")

import time
time.sleep(10)
print("finish!")

from dateutil.parser import parse
print(parse('2016-04-16'))
```

날짜 데이터

- pandas 시계열 자료
  - ✓ 일반적인 테이블 형태의 자료는 임의의 값을 인덱스로 가질 수 있지만 시계열 자료는 DatetimeIndex라는 타임스탬프를 가집니다.
- DatatimeIndex
  - ✓ DatetimeIndex는 특정한 순간에 기록된 타임스탬프(timestamp) 형식의 시계열 자료를 다루기 위한 인덱스
  - ✓ 타임스탬프 인덱스는 반드시 일정한 간격으로 자료가 있어야 한다는 조건은 없습니다.
  - ✓ DatetimeIndex 타입의 인덱스의 생성
    - pd.to_datetime 함수
    - pd.date_range 함수

날짜 데이터

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np

date_str = ["2017, 1, 1", "2017, 1, 4", "2017, 1, 5", "2017, 1, 6"]
idx = pd.to_datetime(date_str)
print(idx)

np.random.seed(0)
s = pd.Series(np.random.randn(4), index=idx)
print(s)
```

날짜 데이터

- 가장 기본적인 시계열의 종류는 파이썬 문자열이나 datetime 객체로 표현되는 타임스탬프로 색인된 Series
- index가 DateTime인 Series는 TimeSeries 자료형이 됩니다.
- TimeSeries는 인덱싱하는 방법은 Series와 동일
- 날짜를 문자열 형식으로 인덱싱 하는 것이 가능
- 년도나 월 까지만 입력해서 인덱싱 하는 것도 가능
- 슬라이싱은 기존의 Series와 동일한 방법으로 가능
- truncate 함수를 이용해서 분할이 가능한데 이 때 옵션으로는 before 와 after 사용 가능
- 날짜가 중복된 경우에 기술 통계를 수행해야 한다면 groupby(level=0)을 대입해서 그룹화 한 후 기술 통계를 수행

날짜 데이터

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
from datetime import datetime
dates = [datetime(2017, 1, 1), datetime(2017, 1, 4), datetime(2017, 1, 5),
         datetime(2017, 1, 6), datetime(2017,1,8), datetime(2017,2,10)]
ts = Series(np.random.randn(6), index=dates)
print(ts[ts.index[2]]) #3번째 데이터 출력
print(ts['1/1/2017']) #문자열로 인덱싱해서 조회 가능
print(ts['20170104']) #문자열로 인덱싱해서 조회 가능
print(ts['2017-01']) #월 단위 인덱싱
print(ts['2017-01-04':'2017-01-08']) #슬라이싱
print(ts.truncate(before='2017-01-08')) #슬라이싱
print(ts.truncate(after='2017-01-08')) #슬라이싱
```

날짜 데이터

- resample()을 이용하면 누락된 데이터를 삽입하거나 삭제 할 수 있습니다.
- date_range
  - ✓ 시작일(start)과 종료일(end) 또는 시작일과 개수(periods)을 입력하면 범위 내의 날짜 및 시간 인덱스 생성
  - ✓ freq 인수로 빈도 지정 가능
    - B: buseness day
    - MS:월의 시작
    - M: 월의 마지막 날짜
    - H,T,S: 매시간, 분, 초
    - BM, BMS: 일을 하는 월의 마지막과 시작날짜
    - W-MON: 요일
    - WOM-1MON: 월의 첫번째 월요일

날짜 데이터

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
from datetime import datetime

print(pd.date_range("2017-4-1", "2017-4-30"))
print(pd.date_range(start="2017-4-1", periods=30))
#buseness day
print(pd.date_range("2017-4-1", "2017-4-30", freq="B"))
#월의 시작
print(pd.date_range("2017-4-1", "2017-12-31", freq="MS"))
#월의 종료
print(pd.date_range("2017-4-1", "2017-12-31", freq="M"))
ts = pd.Series(np.random.randn(4), index=pd.date_range("2000-1-1", periods=4, freq="M"))
```

날짜 데이터

- shift(이동할 인덱스): 데이터를 인덱스 만큼 이동
  - freq에 데이터를 대입하면 그 만큼 날짜 이동

```
print(ts)
print(ts.shift(1))
print(ts.shift(-1))
print(ts.shift(1, freq="M"))
```

날짜 데이터

- Resampling
  - ✓ 시계열의 빈도를 변환하는 과정
  - ✓ 상위 빈도의 데이터를 하위 빈도로 집계하는 것을 다운샘플링이라고 하고 반대의 과정을 업샘플링이라고 합니다.
  - ✓ resample(freq,how,fill_method, closed, label, kind)
    - freq:리샘플링 빈도('M', '5min', Second(15)
    - how: 집계값으로 기본은 mean으로 first, last, median, max, min 등
    - fill_method: 업 샘플링을 할 때 데이터를 채우는 방법으로 기본은 None 이고 ffill 또는 bfill 설정 가능
    - limit: 채우는 개수
    - closed: 다운 샘플링을 할 때 왼쪽과 오른쪽 어느 쪽을 포함시킬 지 설정
    - label: 다운 샘플링을 할 때 왼쪽과 오른쪽 어느 쪽 라벨을 사용할 것인지 설정

날짜 데이터

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
from datetime import datetime

ran = pd.date_range('1/1/2017', periods=30, freq='T')
ts = Series(np.arange(30), index = ran)
print(ts)
#10분 단위 합계 구하기
print(ts.resample('10T').sum())
#10분 단위 합계 구하기
print(ts.resample('10T', label='left').sum())
```

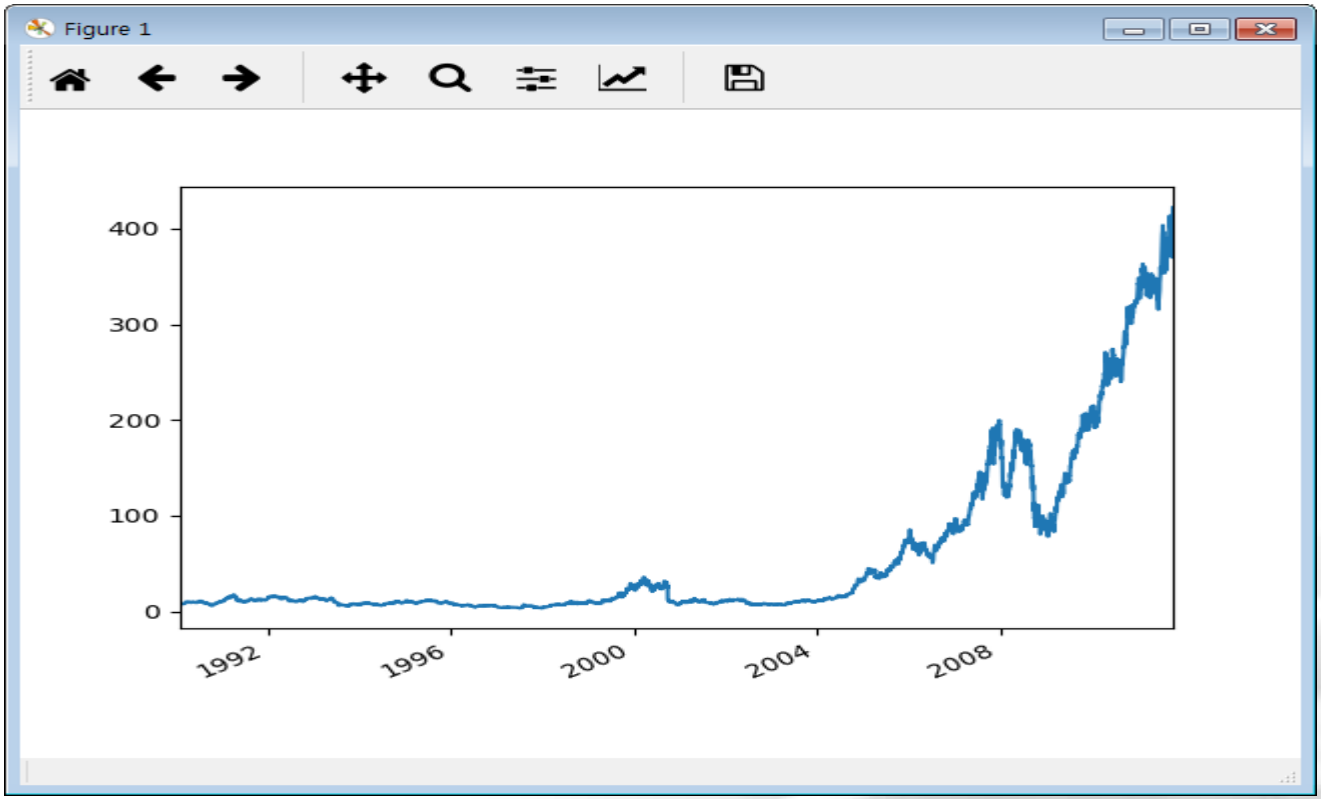
날짜 데이터

```
ran = pd.date_range('1/1/2017', periods=150, freq='D')
ts = Series(np.arange(150), index = ran)
def mo(x):
    return x.month

# 월별 합계 구하기 - resample 이용
print(ts.resample('M').sum())
# 그룹화 이용
print(ts.groupby(mo).sum())
```

날짜 데이터

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
from datetime import datetime

ran = pd.date_range('1/1/2017', periods=2, freq='M')
ts = Series(np.arange(2), index = ran)
ts = ts.resample('W').bfill().replace(np.nan, 0)
print(ts)
```

시계열 그래프



시계열 그래프

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt

close_px_all = pd.read_csv('stock_px.csv', parse_dates=True, index_col=0)
#close_px_all['AAPL'].plot()
#close_px_all['AAPL'].ix['2009'].plot()
#close_px_all['AAPL'].ix['01-2009':'06-2009'].plot()
#pd.rolling_mean(close_px_all.AAPL, 30).plot()
#pd.rolling_std(close_px_all.AAPL, 30).plot()
plt.show()
```