날짜 시간 데이터 처리

판다스 라이브러리의 시계열 도구

날짜 시간 데이터 처리

➢ **문자열을 날짜로 변환**

- 날짜와 시간을 나타내는 문자열 벡터를 시계열 데이터로 변환
- to_datetime() - format 매개변수에 날짜와 시간 포맷을 지정
- errors 매개변수 - 오류 처리, coerce 옵션값은 문제가 발생해도 에러를 일으키지 않지만 대신 에러가 난 값을 NaT(누락된 값)으로 설정합니다.

```
import numpy as np
import pandas as pd

date_strings = np.array(['03-04-2005 11:35 PM',
                         '23-05-2010 12:01 AM',
                         '04-09-2009 09:09 PM'])  # 문자열

# Timestamp 객체로 변환
[pd.to_datetime(date, format='%d-%m-%Y %I:%M %p') for date in date_strings]
[pd.to_datetime(date, format="%d-%m-%Y %I:%M %p", errors="ignore")  for date in date_strings]
pd.to_datetime(date_strings)
```

날짜 시간 데이터 처리

➢ 문자열을 날짜로 변환

- 문자열을 datetime으로 바꿀 수 있는 파이썬 도구가 많이 있지만 판다스를 사용하는 것에 맞출 경우 to_datetime으로 변환을 수행
- to_datetime()의 errors매개변수의 기본값은 'raise'로 날짜 포맷에 문제가 있을 때 예외를 발생시킵니다.
- ignore는 예외를 발생시키거나 NaT를 반환하는 대신 원본 문자열을 그대로 반환
- to_datetime함수에 date_strings 리스트를 그대로 전달할 수 있습니다.
- format 매개변수를 지정하지 않아도 가능한 날짜 포맷을 추측하여 변환해 줍니다.

코드	설명	예
%Y	전체 연도	2001
%m	0으로 시작하는 월	04
%d	0으로 시작하는 일	09
%I	0으로 시작하는 시간 (12시간제)	02
%p	AM 또는 PM	AM
%M	0으로 시작하는 분	05
%S	0으로 시작하는 초	09

날짜 시간 데이터 처리

➢ 시간대 데이터 처리

- 시계열 데이터에서 시간대 정보를 추가하거나 변환할 수 있습니다.
- 판다스 객체에는 시간대가 없습니다
- 판다스는 시간대 객체를 만들때 tz 매개변수를 사용하여 시간대를 추가할 수 있습니다.
- 판다스의 Series 객체는 모든 원소에 tz_localize와 tz_convert를 적용합니다.

```
import pandas as pd

pd.Timestamp('2017-05-01 06:00:00', tz='Europe/London')      # datetime을 만듦

date = pd.Timestamp('2017-05-01 06:00:00')                   # datetime을 만듦
date_in_london = date.tz_localize('Europe/London')            # 시간대를 지정
date_in_london                                               # datetime을 확인

date_in_london.tz_convert('Africa/Abidjan')                  # 시간대를 변환
dates = pd.Series(pd.date_range('2/2/2002', periods=3, freq='M'))     # 세 개의 날짜를 만듭니다.
#datetime 속성 dt 을 사용하여 datetime 구성 요소에 액세스 할 수 있습니다.
dates.dt.tz_localize('Africa/Abidjan')                        # 시간대를 지정
```

날짜 시간 데이터 처리

➢ 시간대 데이터 처리

- all_timezone을 임포트하여 전체 시간대 문자열을 확인할 수 있습니다
- 'dateutil/'로 시작하여 datetuil 문자열을 사용할 수 있습니다.
- pytz의 객체를 직접 전달 할 수 있습니다.

```
from pytz import all_timezones
all_timezones[0:2]   # 두 개의 시간대를 확인
dates.dt.tz_localize('dateutil/Aisa/Seoul')

import pytz
tz = pytz.timezone('Asia/Seoul')
dates.dt.tz_localize(tz)
```

날짜 시간 데이터 처리

➢ **날짜와 시간 선택하기**

- 시작과 마지막 날짜를 사용해 불리언 조건을 만들어 날짜 벡터에서 하나 이상의 원소를 선택할 수 있습니다.
- 날짜 열을 데이터프레임의 인덱스로 지정하고 loc를 사용해 슬라이싱할 수 있습니다.

```
import pandas as pd

dataframe = pd.DataFrame()
dataframe['date'] = pd.date_range('1/1/2001', periods=100000, freq='H')      # datetime을 만듭니다.

# 두 datetime 사이의 샘플을 선택합니다.
dataframe[ (dataframe['date'] > '2002-1-1 01:00:00')  &
           (dataframe['date'] <= '2002-1-1 04:00:00') ]

dataframe = dataframe.set_index(dataframe['date'])                    # datetime을 만듭니다.
dataframe.loc[ '2002-1-1 01:00:00':'2002-1-1 04:00:00' ]              # 두 datetime 사이 샘플을 선택
```

복잡한 시계열 데이터를 다루어야 할 경우 날짜 열을 데이터프레임의 인덱스로 지정하고, 간단한 데이터 랭글링이라면 불리언 조건을 사용합니다.

날짜 시간 데이터 처리

- **날짜데이터를 여러 특성으로 분할**
  - 날짜와 시간의 열로부터 년, 월, 일, 시, 분에 해당하는 특성을 만들 수 있습니다.
  - Series.dt의 시간 속성을 사용합니다.

```
import pandas as pd

dataframe = pd.DataFrame()
dataframe['date'] = pd.date_range('1/1/2001', periods=150, freq='W')      #   다섯 개의 날짜를 만듭니다.
# 년, 월, 일, 시, 분에 대한 특성을 만듭니다.
dataframe['year'] = dataframe['date'].dt.year
dataframe['month'] = dataframe['date'].dt.month
dataframe['day'] = dataframe['date'].dt.day
dataframe['hour'] = dataframe['date'].dt.hour
dataframe['minute'] = dataframe['date'].dt.minute

dataframe.head(3)  # 세 개의 행을 확인
```

날짜 시간 데이터 처리

- **날짜 간의 차이 계산**
  - 판다스의 TimeDelta 데이터 타입을 사용하면 두 지점 사이의 시간 변화를 기록한 특성을 계산합니다 (https://bit.ly/2W04StT)

```
import pandas as pd

dataframe = pd.DataFrame()

# 두 datetime 특성을 만듭니다.
dataframe['Arrived'] = [pd.Timestamp('01-01-2017'), pd.Timestamp('01-04-2017')]
dataframe['Left'] = [pd.Timestamp('01-01-2017'), pd.Timestamp('01-06-2017')]

dataframe['Left'] - dataframe['Arrived']                # 특성 사이의 차이를 계산

# 특성 간의 기간을 계산(days출력 삭제)
pd.Series(delta.days for delta in (dataframe['Left'] - dataframe['Arrived']))
```

날짜 시간 데이터 처리

- **요일 인코딩**
  - 날짜 벡터에서 각 날짜의 요일은 판다스 Series.dt의 weekday_name속성을 사용
  - weekday를 사용하여 머신러닝의 특성으로 사용하기 좋도록 수치형 (정수로) 값을 출력 (월요일은 0)

```
import pandas as pd

dates = pd.Series(pd.date_range("2/2/2002", periods=3, freq="M"))  # 시리즈 객체
dates.dt.weekday_name                    # 요일을 확인
dates.dt.weekday                         # 요일을 정수로 확인
```

예) 지난 3년간 일요일에 일어난 전체 판매량을 비교 분석등에 활용

날짜 시간 데이터 처리

➢ 시차 특성

- 판다스의 shift를 사용하여 n기간 만큼 차이가 나는 시차 특성을 만들어 낼 수 있습니다
- 판다스의 shift()를 사용하여 한 행 뒤의 값을 담은 새로운 특성을 만들 수 있습니다.

```
import pandas as pd

dataframe = pd.DataFrame()
# 날짜를 만듭니다.
dataframe["dates"] = pd.date_range("1/1/2001", periods=5, freq="D")
dataframe["stock_price"] = [1.1,2.2,3.3,4.4,5.5]

# 한 행 뒤의 값을 가져옵니다.
dataframe["previous_days_stock_price"] = dataframe["stock_price"].shift(1)
dataframe # 데이터프레임을 확인
```

	dates	stock_price	previous_days_stock_price
0	2001-01-01	1.1	NaN
1	2001-01-02	2.2	1.1
2	2001-01-03	3.3	2.2
3	2001-01-04	4.4	3.3
4	2001-01-05	5.5	4.4

데이터가 일정한 시간 간격으로 생성된 경우, 데이터셋에서 과거 값을 사용하여 예측을 만들어야 합니다.(시차 특성을 사용)
예) 하루 전 주식 가격을 사용해 오늘 가격을 예측

날짜 시간 데이터 처리

- **이동 시간 윈도 사용**
  - 시계열 데이터에서 일정 시간 간격으로 통계를 계산할 수 있습니다.
  - 예] 세 달을 시간 윈도로 이동 평균을 계산 : mean(January, February, March) , mean(February, March, April), mean(March, April, May) 세 달 크기의 시간 윈도가 각 단계마다 윈도의 평균을 계산하면서 샘플 위를 이동합니다.
  - 판다스의 rolling()는 window 매개변수에서 윈도 크기를 지정합니다.
  - 시간 윈도의 max(), mean(), count(), corr()와 같은 통계를 간편하게 계산할 수 있습니다.
  - 판다스의 ewm() - 최근 항목에 높은 가중치를 두지만 전체 기간에 대한 통계를 계산하는 지수 이동 윈도

```
import pandas as pd

# datetime을 만듭니다.
time_index = pd.date_range("01/01/2010", periods=5, freq="M")
# 데이터프레임을 만들고 인덱스를 설정합니다.
dataframe = pd.DataFrame(index=time_index)

dataframe["Stock_Price"] = [1,2,3,4,5]     # 특성을 만듭니다.
dataframe.rolling(window=2).mean()     # 이동 평균을 계산

dataframe.ewm(alpha=0.5).mean()
```

날짜 시간 데이터 처리

- **시계열 데이터에서 누락된 값 처리**
  - 시계열 데이터에서는 보간(interpolation)방법을 사용하여 누락된 값으로 생긴 간격을 채웁니다.
  - 보간은 누락된 값의 양쪽 경계를 잇는 직선이나 곡선을 사용하여 적절한 값을 예측함으로써 비어 있는 간극을 채우는 기법입니다.
  - 시간 간격이 일정하고, 데이터가 노이즈로 인한 변동이 심하지 않고 누락된 값으로 인한 빈 간극이 작을 때 보간 방법이 유용합니다.
  - 두 포인트 사이의 직선이 비선형이라고 가정하면 interpolate의 method 매개변수를 사용해 다른 보간 방법을 지정할 수 있습니다.
  - 누락된 값의 간격이 커서 전체 간격을 보간하는 것이 좋지 않을 때는 limit 매개변수를 사용하여 보간 값의 개수를 제한하고
  - limit_direction 매개변수에서 마지막 데이터로 앞쪽 방향으로 보간할지 그 반대로 할지 지정할 수 있습니다.
  - 누락된 값을 이전에 등장한 마지막 값으로 대체할 수 있다.
  - 누락된 값을 그 이후에 등장한 최초의 값으로 대체할 수 있다

날짜 시간 데이터 처리

- **시계열 데이터에서 누락된 값 처리**

```
import pandas as pd
import numpy as np

time_index = pd.date_range("01/01/2010", periods=5, freq="M")
dataframe = pd.DataFrame(index=time_index)                  # 데이터프레임을 만들고 인덱스를 지정

dataframe["Sales"] = [1.0,2.0,np.nan,np.nan,5.0]            # 누락된 값이 있는 특성을 만듦

dataframe.interpolate()                                     # 누락된 값을 보간
dataframe.ffill()                                           # 앞쪽으로 채우기(Forward-fill)
dataframe.bfill()                                           # 뒤쪽으로 채우기(Back-fill)
dataframe.interpolate(method="quadratic")                   # 비선형의 경우 보간 방법 변경
dataframe.interpolate(limit=1, limit_direction="forward")   #보간 방향 지정
```