1.2 파이썬 패키지와 데이터

이 절에서는 앞으로 회귀분석과 시계열분석을 하는데 사용할 두가지 파이썬 패키지를 소개한다.

- statsmodels 패키지 (스탯츠모델즈 패키지라고 읽는다.)
- scikit-learn 패키지 (싸이킷-런 패키지라고 읽는다.)

statsmodels 패키지

statsmodels 패키지는

- 검정 및 추정(test and estimation)
- 회귀분석(regression analysis)
- 시계열분석(time-series analysis)

등의 다양한 통계분석 기능을 제공하는 파이썬 패키지다. 특히 회귀분석의 경우 R 스타일의 모형 기술을 가능하게 하는 patsy 패키지를 포함하고 있어 기존에 R에서만 가능했던 회귀분석과 시계열분석 방법론을 그대로 파이썬에서 이용할 수 있게 되었다.

statsmodels 패키지를 사용할 때는 다음처럼 api 서브패키지를 임포트하여 사용한다.

```
import statsmodels.api as sm
```

statsmodels에서 제공하는 데이터

statsmodels 패키지의 개발 목표 중 하나는 기존에 R을 사용하여 통계 분석 및 시계열 분석을 하던 사용자가 파이썬에서 동일한 분석을 할 수 있도록 하는 것이다. 따라서 R에서 제공하던 명령어 뿐만 아니라 데이터셋도 동일하게 제공하기 위해 Rdatasets이라는 프로젝트를 통해 R에서 사용하던 1000개 이상의 표준 데이터셋을 사용할 수 있도록 지원한다. 자세한 사항은 다음 프로젝트 홈페이지에서 확인할 수 있다.

- https://github.com/vincentarelbundock/Rdatasets (https://github.com/vincentarelbundock/Rdatasets)

다음은 위 프로젝트에서 제공하는 데이터셋의 목록이다.

- http://vincentarelbundock.github.io/Rdatasets/datasets.html (http://vincentarelbundock.github.io/Rdatasets/datasets.html)

이 목록에 있는 데이터를 가져오려면 우선 "Package"이름과 "Item"을 알아낸 후 다음에 설명하는 `get_rdataset` 명령을 이용한다.

```
get_rdataset(item, [package="datasets"])
```

`item` 과 `package` 인수로 해당 데이터의 "Package"이름과 "Item"을 넣는다. "Package"이름이 `datasets` 인 경우에는 생략할 수 있다. 이 함수는 인터넷에서 데이터를 다운로드 받으므로 인터넷에 연결되어 있어야 한다. 이렇게 받은 데이터는 다음과 같은 속성을 가지고 있다.

- `package` : 데이터를 제공하는 R 패키지 이름
- `title` : 데이터 이름 문자열
- `data` : 데이터를 담고 있는 데이터프레임

- __doc__ : 데이터에 대한 설명 문자열. 이 설명은 R 패키지의 내용을 그대로 가져온 것이므로 예제 코드가 R로 되어 있어 파이썬에서는 사용할 수 없다.

get_rdataset 명령으로 받을 수 있는 몇가지 예제 데이터를 소개한다.

예제 1

datasets 패키지의 Titanic 데이터는 타이타닉호의 탑승자들에 대한 데이터이다.

In [1]:

```
data = sm.datasets.get_rdataset("Titanic", package="datasets")

df = data.data
df.tail()
```

Out[1]:

	Class	Sex	Age	Survived	Freq
27	Crew	Male	Adult	Yes	192
28	1st	Female	Adult	Yes	140
29	2nd	Female	Adult	Yes	80
30	3rd	Female	Adult	Yes	76
31	Crew	Female	Adult	Yes	20

다음은 데이터에 포함된 설명의 일부이다.

In [2]:

```
print(data.__doc__[:1005])
```

```
+---------+----------------+
| Titanic | R Documentation |
+---------+----------------+

Survival of passengers on the Titanic
-------------------------------------

Description
~~~~~~~~~~~

This data set provides information on the fate of passengers on the
fatal maiden voyage of the ocean liner  'Titanic', summarized according
to economic status (class), sex, age and survival.

Usage
~~~~~

::

   Titanic

Format
~~~~~~

A 4-dimensional array resulting from cross-tabulating 2201 observations
on 4 variables. The variables and their levels are as follows:

+----+----------+--------------------+
| No | Name     | Levels             |
+----+----------+--------------------+
| 1  | Class    | 1st, 2nd, 3rd, Crew |
+----+----------+--------------------+
| 2  | Sex      | Male, Female       |
+----+----------+--------------------+
| 3  | Age      | Child, Adult       |
+----+----------+--------------------+
| 4  | Survived | No, Yes            |
+----+----------+--------------------+
```

예제 2

MASS 패키지의 deaths 데이터는 1974년부터 1979년까지의 영국의 호흡기 질환 사망자 수를 나타내는 시계열 데이터이다. 이 시계열 데이터에서는 시간이 1년을 1.0으로, 1개월을 1/12로 하는 값(year-fraciton)으로 인코딩되어 있다.

In [3]:

```
data = sm.datasets.get_rdataset("deaths", "MASS")

df = data.data
df.tail()
```

Out[3]:

	time	value
67	1979.583333	1354
68	1979.666667	1333
69	1979.750000	1492
70	1979.833333	1781
71	1979.916667	1915

1개월을 1/12로 하는 값(year-fraciton)을 파이썬의 `datatime` 포맷으로 바꾸려면 다음과 같은 함수를 사용해야 한다.

In [4]:

```
def yearfraction2datetime(yearfraction, startyear=0):
    import datetime
    import dateutil
    year = int(yearfraction) + startyear
    month = int(round(12 * (yearfraction - year)))
    delta = dateutil.relativedelta.relativedelta(months=month)
    date = datetime.datetime(year, 1, 1) + delta
    return date

df["datetime"] = df.time.map(yearfraction2datetime)
df.tail()
```

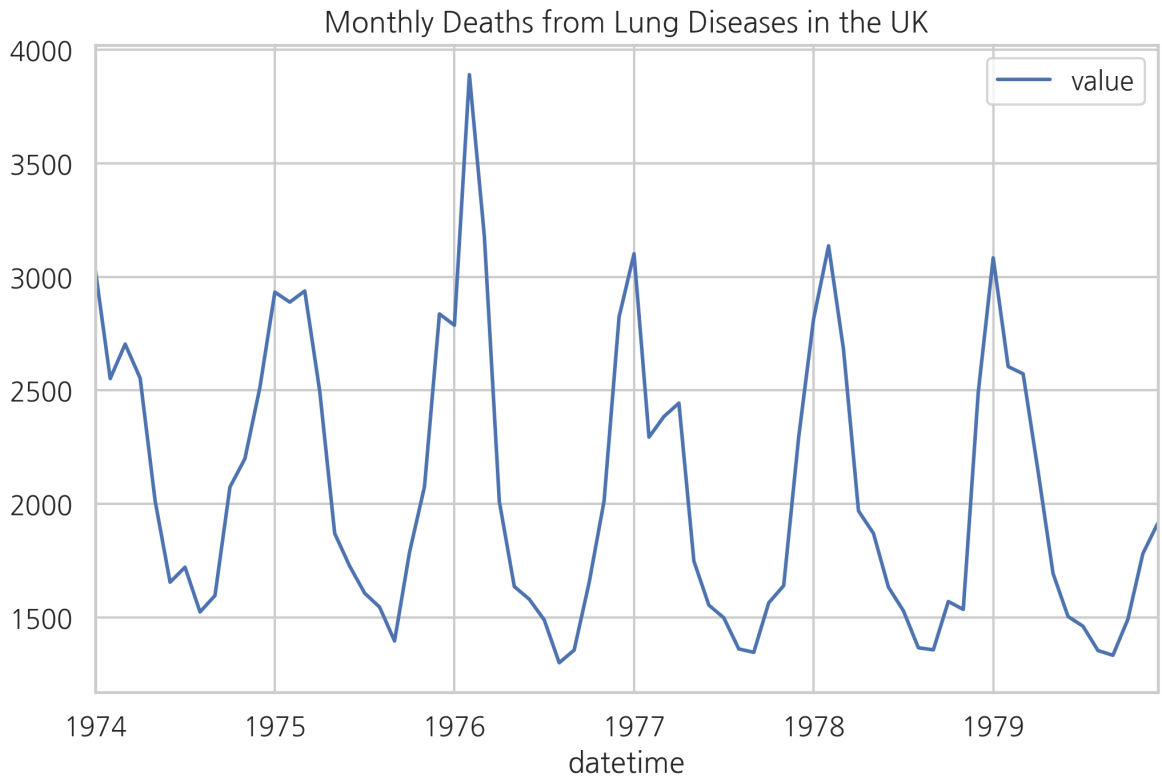
Out[4]:

	time	value	datetime
67	1979.583333	1354	1979-08-01
68	1979.666667	1333	1979-09-01
69	1979.750000	1492	1979-10-01
70	1979.833333	1781	1979-11-01
71	1979.916667	1915	1979-12-01

이 데이터는 계절에 따른 패턴 즉, 계절성(seasonality)를 보이는 시계열이다.

In [5]:

```
df.plot(x="datetime", y="value")
plt.title(data.title)
plt.show()
```



scikit-learn 패키지

scikit-learn 패키지는 머신러닝 교육을 위한 파이썬 패키지다. scikit-learn 패키지의 장점은 다양한 머신러닝 모형 즉, 알고리즘을 하나의 패키지에서 모두 제공하고 있다는 점이다. 다음은 scikit-learn 패키지에서 제공하는 머신러닝 모형의 목록이다. 이 목록은 대표적인 것들만을 나열한 것이며 지속적으로 모형들이 추가되고 있다.

scikit-learn 패키지의 임포트 이름은 `sklearn` 이다. 이 책에서는 `sk` 라는 축양형 이름(alias)으로 임포트하여 사용한다.

```
import sklearn as sk
```

scikit-learn에서 제공하는 데이터

sklearn.datasets 서브패키지는 다양한 예제 데이터셋을 제공한다. 데이터를 불러오는 명령들은 크게 다음과 같이 세가지 계열의 명령으로 나눌 수 있다.

- `load` 계열 명령: scikit-learn 설치 패키지에 같이 포함된 데이터를 가져오는 명령

- fetch 계열 명령: 인터넷에서 다운로드할 수 있는 데이터를 가져오는 명령
- make 계열 명령: 무작위로 가상의 데이터를 생성하는 명령

load 계열의 명령들은 설치 패키지에 처음부터 저장되어 있어서 별로도 다운로드 받지 않아도 바로 쓸 수 있는 데이터를 제공한다.

- load_boston : 회귀 분석용 보스턴 집값
- load_diabetes : 회귀 분석용 당뇨병 자료
- load_linnerud : 회귀 분석용 linnerud 자료
- load_iris : 분류용 붓꽃(iris) 자료
- load_digits : 분류용 숫자(digit) 필기 이미지 자료
- load_wine : 분류용 포도주(wine) 등급 자료
- load_breast_cancer : 분류용 유방암(breast cancer) 진단 자료

fetch 계열의 명령들은 데이터의 크기가 커서 패키지에 처음부터 저장되어 있지 않고 인터넷에서 다운로드 받아 홈 디렉토리 아래의 scikit_learn_data 라는 서브 디렉토리에 저장한 후 추후 불러들이는 데이터들이다. 따라서 최초 사용시에 인터넷에 연결되어 있지 않으면 사용할 수 없다.

- fetch_california_housing : : 회귀분석용 캘리포니아 집값 자료
- fetch_covtype : 회귀분석용 토지 조사 자료
- fetch_20newsgroups : 뉴스 그룹 텍스트 자료
- fetch_olivetti_faces : 얼굴 이미지 자료
- fetch_lfw_people : 유명인 얼굴 이미지 자료
- fetch_lfw_pairs : 유명인 얼굴 이미지 자료
- fetch_rcv1 : 로이터 뉴스 말뭉치
- fetch_kddcup99 : Kddcup 99 Tcp dump 자료

경우에 따라서는 모형을 시험하기 위해 원하는 특성을 가진 가상의 데이터가 필요할 수 있다. make 계열 명령은 이러한 가상 데이터를 생성하는 역할을 한다.

- make_regression : 회귀 분석용 가상 데이터 생성
- make_classification : 분류용 가상 데이터 생성
- make_blobs : 클러스터링용 가상 데이터 생성

이 외에도 다양한 가상 데이터 생성 명령이 있다.

scikit-learn에서 제공하는 데이터셋은 Bunch 라는 클래스 객체 형식으로 생성된다. 이 클래스 객체는 다음과 같은 속성을 가진다.

- data : (필수) 독립 변수 ndarray 배열
- target : (필수) 종속 변수 ndarray 배열
- feature_names : (옵션) 독립 변수 이름 리스트
- target_names : (옵션) 종속 변수 이름 리스트
- DESCR : (옵션) 자료에 대한 설명

예제 3

scikit-learn에서 제공하는 load_boston 명령으로 받을 수 있는 보스턴 주택 가격 데이터는 다음과 같은 데이터이다.

- 타겟 데이터
  - 1978년 보스턴 주택 가격
  - 506개 타운의 주택 가격 중앙값 (단위 1,000 달러)

- 특징 데이터
  - CRIM : 범죄율
  - INDUS : 비소매상업지역 면적 비율
  - NOX : 일산화질소 농도
  - RM : 주택당 방 수
  - LSTAT : 인구 중 하위 계층 비율
  - B : 인구 중 흑인 비율
  - PTRATIO : 학생/교사 비율
  - ZN : 25,000 평방피트를 초과 거주지역 비율
  - CHAS : 찰스강의 경계에 위치한 경우는 1, 아니면 0
  - AGE : 1940년 이전에 건축된 주택의 비율
  - RAD : 방사형 고속도로까지의 거리
  - DIS : 직업센터의 거리
  - TAX : 재산세율

In [6]:

```
from sklearn.datasets import load_boston

boston = load_boston()
print(boston.DESCR[:1200])
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribut
e 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherw
ise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner
```

데이터는 후속 작업을 위해 Pandas 데이터프레임 형태로 만들어야 한다. 여기에서는 특징 행렬을 `dfX` 로, 종속변수 벡터를 `dfy` 로 만들었다.

In [7]:

```
dfX = pd.DataFrame(boston.data, columns=boston.feature_names)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
```

statsmodels에서 작업할 때는 `concat` 명령을 사용하여 특징 행렬과 종속 변수를 `df` 라는 하나의 데이터프레임으로 묶어두면 편리하다.

In [8]:

```
df = pd.concat([dfX, dfy], axis=1)
df.tail()
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	

가상 데이터

때로는 회귀분석 결과를 검증하기 위해 가상의 데이터가 필요한 경우가 있다. 이 때는 `make_regression()` 명령을 사용한다. 사용법은 다음과 같다.

```
X, y = make_regression(n_samples, n_features, bias, noise, random_state)
```

또는

```
X, y, w = make_regression(... coef=True)
```

- `n_samples` : 정수 (옵션, 디폴트 100)
  - 표본 데이터의 갯수 N
- `n_features` : 정수 (옵션, 디폴트 100)
  - 독립 변수(feature)의 수(차원) M
- `bias` : 실수 (옵션, 디폴트 0.0)
  - y 절편
- `noise` : 실수 (옵션, 디폴트 0.0)
  - 출력 즉, 종속 변수에 더해지는 잡음 ϵ의 표준편차
- `random_state` : 정수 (옵션, 디폴트 None)
  - 난수 발생용 시드값
- `coef` : 불리언 (옵션, 디폴트 False)
  - True 이면 선형 모형의 계수도 출력

출력은 다음과 같다.

- `X` : [`n_samples` , `n_features`] 형상의 2차원 배열
  - 독립 변수의 표본 데이터 행렬 X

- y : [n_samples] 형상의 1차원 배열
  - 종속 변수의 표본 데이터 벡터 y
- coef : [n_features] 형상의 1차원 배열 또는 [n_features , n_targets] 형상의 2차원 배열 (옵션)
  - 선형 모형의 계수 벡터 w, 입력 인수 coef 가 True 인 경우에만 출력됨

make_regression() 명령은 내부적으로 다음 과정을 거쳐 가상의 데이터를 만든다.

1. 독립변수 데이터 행렬 X 를 무작위로 만든다.
2. 종속변수와 독립변수를 연결하는 가중치 벡터 w 를 무작위로 만든다.
3. X 와 w 를 내적하고 y절편 b 값을 더하여 독립변수와 완전선형인 종속변수 벡터 y_0 를 만든다.
4. 기댓값이 0이고 표준편차가 noise 인 정규분포를 이용하여 잡음 epsilon 를 만든다.
5. 독립변수와 완전선형인 종속변수 벡터 y_0 에 잡음 epsilon 을 더해서 종속변수 데이터 y를 만든다.

$$y = w^T x + b + \epsilon$$

예제 4

다음은 독립 변수가 1개이고 noise 인수값이 0이므로 잡음이 없는 경우의 가상데이터다.

In [9]:

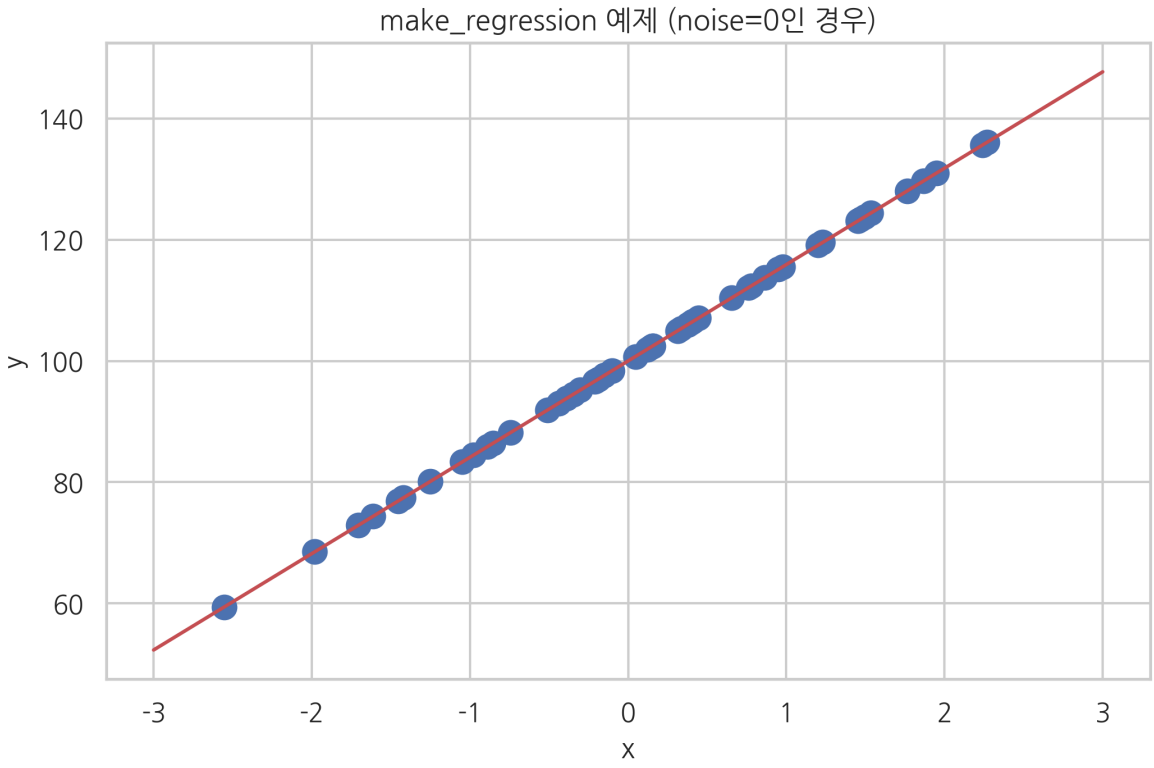
```
from sklearn.datasets import make_regression

X, y, w = make_regression(
    n_samples=50, n_features=1, bias=100, noise=0, coef=True, random_state=0
)

print("w:₩n", w)

xx = np.linspace(-3, 3, 100)
y0 = w * xx + 100
plt.plot(xx, y0, "r-")
plt.scatter(X, y, s=100)
plt.xlabel("x")
plt.ylabel("y")
plt.title("make_regression 예제 (noise=0인 경우)")
plt.show()
```

```
w:
 15.896958364551972
```



위 코드를 실행하여 나온 선형 모형은 다음과 같다.

$$y = 15.897x + 100$$

예제 5

`noise` 인수를 증가시키면 잡음의 표준편차가 증가하고 `bias` 인수를 증가시키면 y 절편 b가 증가한다.

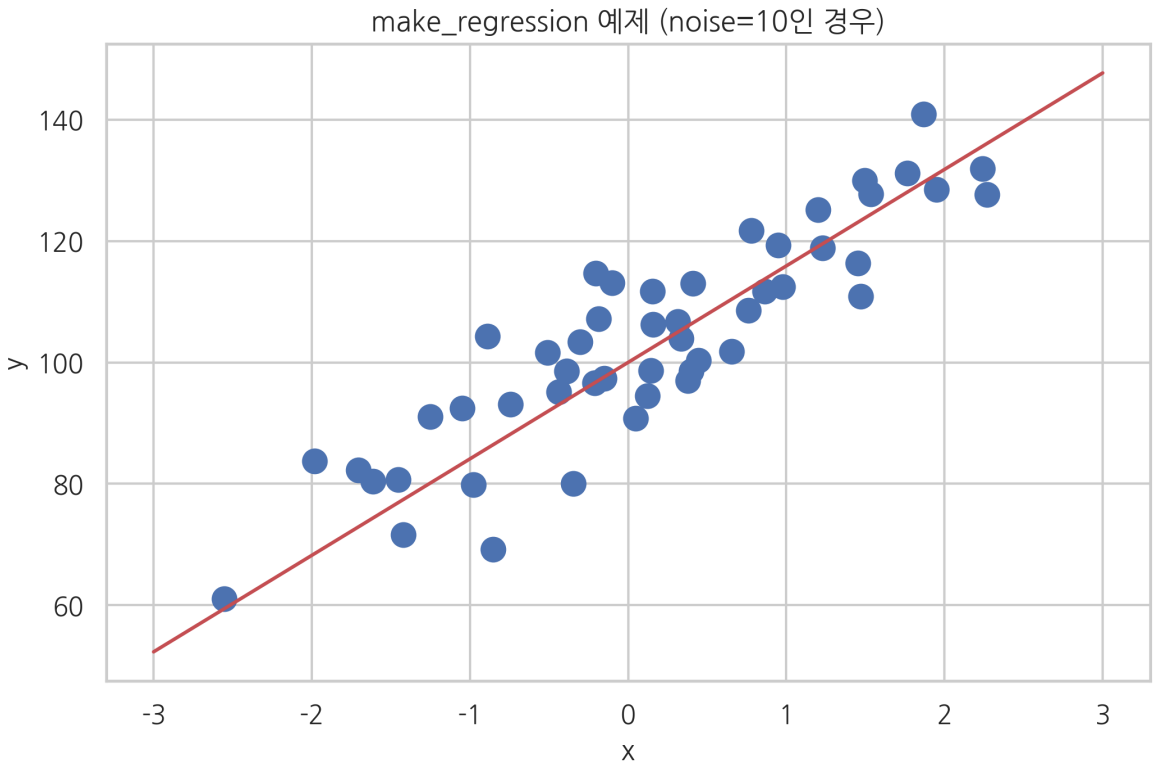
In [10]:

```
X, y, w = make_regression(
    n_samples=50, n_features=1, bias=100, noise=10, coef=True, random_state=0
)

print("w:₩n", w)

xx = np.linspace(-3, 3, 100)
y0 = w * xx + 100
plt.plot(xx, y0, "r-")
plt.scatter(X, y, s=100)
plt.xlabel("x")
plt.ylabel("y")
plt.title("make_regression 예제 (noise=10인 경우)")
plt.show()
```

w:
 15.896958364551972



연습 문제 1.2.1

(1) make_regression 과 같은 기능을 하는 함수 make_regression2 를 만들어라. 단 make_regression2 는 coef=True, n_features=1 라고 가정한다. 즉 항상 가중치 계수를 반환하고 1차원 독립 변수만 생성할 수 있다. 따라서 make_regression2 는 다음과 같은 인수만 가진다.

- n_samples
- bias
- noise

- random_state

따라서 함수 사용법은 다음과 같아야 한다.

```
X, y, w = make_regression2(n_samples, bias, noise, random_state)
```

(2) make_regression2 함수에 coef 인수를 추가하여 make_regression3 함수를 만들어라.
make_regression3 함수는 가중치를 스스로 생성하지 않고 coef 인수로 받은 가중치 계수 배열 값을 그대로 사용하며 가중치 계수를 반환하지 않는다. 독립 변수의 차원은 coef 인수로 받은 가중치 계수 배열의 길이로 결정된다. 따라서 함수 사용법은 다음과 같아야 한다.

```
X, y = make_regression3(n_samples, coef, bias, noise, random_state)
```

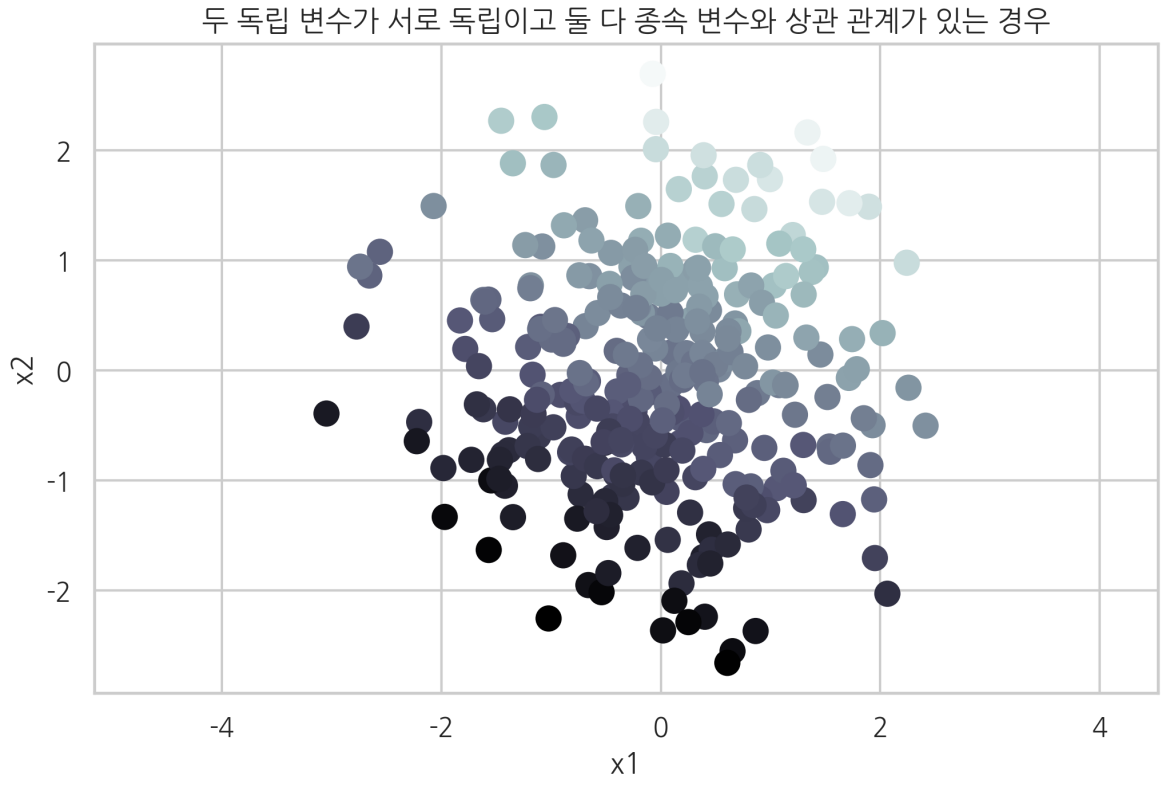
예제 6

n_features 즉, 독립 변수가 2개인 표본 데이터를 생성하여 스캐터 플롯을 그리면 다음과 같다. 종속 변수 값은 점의 명암으로 표시하였다.

In [11]:

```
X, y, w = make_regression(
    n_samples=300, n_features=2, noise=10, coef=True, random_state=0
)

plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap=mpl.cm.bone)
plt.xlabel("x1")
plt.ylabel("x2")
plt.axis("equal")
plt.title("두 독립 변수가 서로 독립이고 둘 다 종속 변수와 상관 관계가 있는 경우")
plt.show()
```



make_regression 명령은 위에서 설명한 인수 이외에도 다음과 같은 인수를 가질 수 있다.

- n_informative : 정수 (옵션, 디폴트 10)
  - 독립 변수(feature) 중 실제로 종속 변수와 상관 관계가 있는 독립 변수의 수(차원)
- effective_rank : 정수 또는 None (옵션, 디폴트 None)
  - 독립 변수(feature) 중 서로 독립인 독립 변수의 수. 만약 None이면 모두 독립
- tail_strength : 0부터 1사이의 실수 (옵션, 디폴트 0.5)
  - effective_rank 가 None이 아닌 경우 독립 변수간의 상관관계를 결정하는 변수. 0.5면 독립 변수간의 상관관계가 없다.

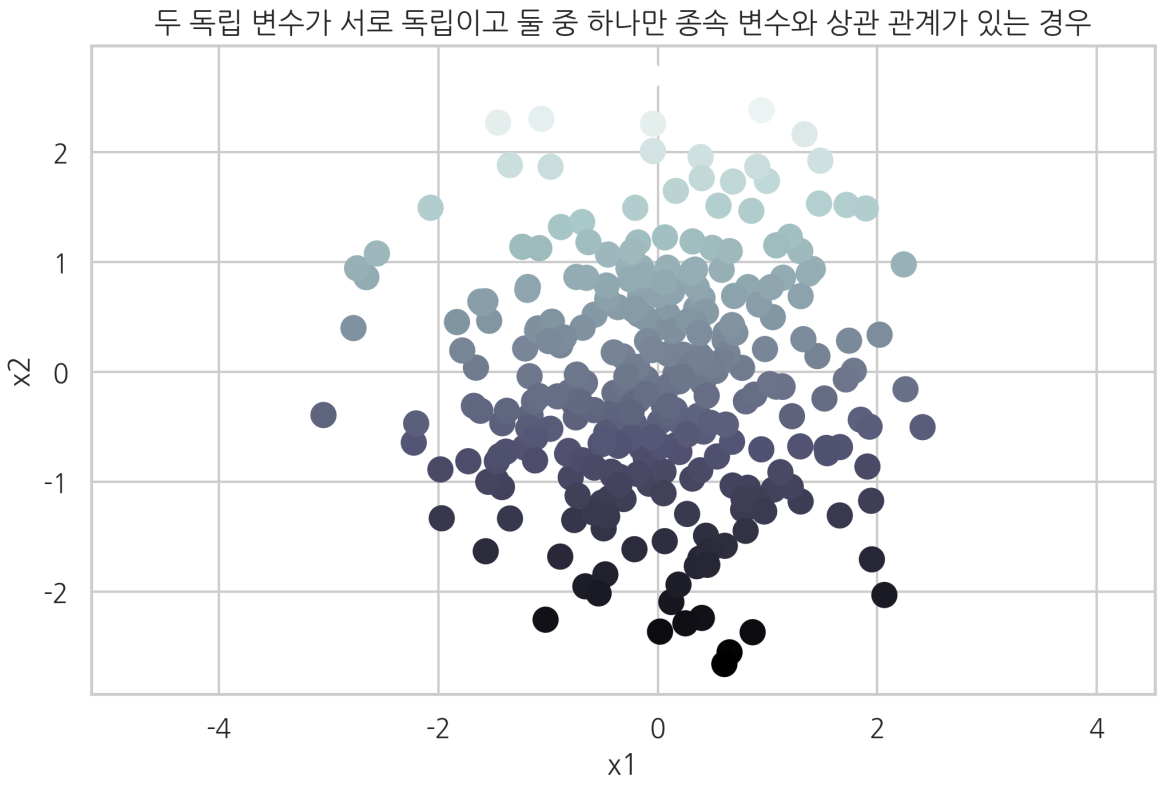
예제 7

2차원의 독립 변수 중 실제로 종속 변수에 영향을 미치는 독립 변수는 하나 뿐이라면 다음처럼 n_informative=1 로 설정한다.

In [12]:

```
X, y, w = make_regression(
    n_samples=300, n_features=2, n_informative=1, noise=0, coef=True, random_state=0
)

plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap=mpl.cm.bone)
plt.xlabel("x1")
plt.ylabel("x2")
plt.axis("equal")
plt.title("두 독립 변수가 서로 독립이고 둘 중 하나만 종속 변수와 상관 관계가 있는 경우")
plt.show()
```



만약 두 독립 변수가 서로 독립이 아니고 상관관계를 가지는 경우에는 tail_strength 인수를 0에 가까운 작은 값으로 설정한다. 이 기능은 다중 공선성(multicolinearity)을 가지는 데이터를 시뮬레이션할 때 유용하다.

In [13]:

```
X, y, w = make_regression(
    n_samples=300, n_features=2, effective_rank=1, noise=0, coef=True, random_state=0,
    tail_strength=0
)

plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap=mpl.cm.bone)
plt.xlabel("x1")
plt.ylabel("x2")
plt.axis("equal")
plt.title("두 독립 변수가 독립이 아닌 경우")
plt.show()
```