3.2 statsmodels의 전처리 기능

StatsModels 패키지는 통계분석과 관련된 R의 기능을 파이썬으로 옮겨오기 위한 패키지이다. R에는 데이터프레임과 문자열 기호를 이용하여 회귀모형을 정의하는 방법이 존재한다. StatsModels 패키지도 이러한 R 스타일 모형 정의 방법을 지원한다. 이러한 지원을 가능하게 하는 것은 patsy라는 패키지 덕분이다. 여기에서는 patsy 패키지의 간단한 사용법과 이를 이용하여 StatsModels에서 회귀모형을 정의하는 방법을 설명한다.

patsy 패키지 소개

patsy 패키지는 회귀분석 전처리를 위한 패키지로 데이터프레임을 가공하여 인코딩, 변환 등을 쉽게 해주는 기능을 제공한다.

In [1]:

```
from patsy import *
```

patsy 패키지의 `dmatrix` 라는 명령을 사용하면 실험 설계 행렬(experiment design matrix)을 간단히 만들수 있다. `dmatrix` 에 다음과 같이 모형 정의 문자열 `formula` 와 원 데이터를 담은 데이터프레임 `data` 을 입력하면 `formula` 에서 지정한 대로 변환된 데이터 `data_transformed` 를 출력한다.

```
data_transformed = dmatrix(formula, data)
```

patsy 패키지가 제공하는 `demo_data` 명령으로 다음과 같이 예제 데이터 `x1` , `x2` , `y` 를 만들자.

In [2]:

```
df = pd.DataFrame(demo_data("x1", "x2", "y"))
df
```

Out[2]:

	x1	x2	y
0	1.764052	-0.977278	0.144044
1	0.400157	0.950088	1.454274
2	0.978738	-0.151357	0.761038
3	2.240893	-0.103219	0.121675
4	1.867558	0.410599	0.443863

`dmatrix` 의 첫번째 기능은 자동 상수항 결합 기능이다. 대상이 되는 데이터에 자동으로 Intecept라는 이름의 데이터 열을 추가한다.

다음 예제에서 스타일 문자열은 단순히 `"x1"` 이다. 스타일 문자열은 데이터와 연산자로 이루어지는데 데이터는 변수명 혹은 데이터프레임 열 이름으로 지정한다. 변수명으로 지정하는 경우에는 현재의 이름 공간(name space)에서 변수를 찾고 데이터프레임 열 이름을 지정하는 경우에는 `data` 라는 인수에 데이터프레임을 넣어주어야 한다.

In [3]:

```
dmatrix("x1", df)
```

Out[3]:

```
DesignMatrix with shape (5, 2)
  Intercept       x1
          1  1.76405
          1  0.40016
          1  0.97874
          1  2.24089
          1  1.86756
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
```

R-style formula 연산자

모형정의 연산자 `formula` 에 복수의 데이터를 지정하는 경우에는 다음과 같은 연산자를 포함해야 한다.

기호	설명
1, 0	바이어스(bias, intercept) 추가 및 제거
+	설명 변수 추가
-	설명 변수 제거
:	상호작용(interaction)
*	a*b = a + b + a:b
/	a/b = a + a:b

상수항을 제외하고자 하는 경우에는 `- 1` 또는 `+ 0` 을 써주어야 한다.

In [4]:

```
dmatrix("x1 - 1", df)
```

Out[4]:

```
DesignMatrix with shape (5, 1)
       x1
  1.76405
  0.40016
  0.97874
  2.24089
  1.86756
  Terms:
    'x1' (column 0)
```

In [5]:

```
dmatrix("x1 + 0", df)
```

Out[5]:

```
DesignMatrix with shape (5, 1)
       x1
  1.76405
  0.40016
  0.97874
  2.24089
  1.86756
  Terms:
    'x1' (column 0)
```

데이터를 추가하는 경우에는 + 연산자를 사용한다.

In [6]:

```
dmatrix("x1 + x2", df)
```

Out[6]:

```
DesignMatrix with shape (5, 3)
  Intercept       x1        x2
          1  1.76405  -0.97728
          1  0.40016   0.95009
          1  0.97874  -0.15136
          1  2.24089  -0.10322
          1  1.86756   0.41060
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'x2' (column 2)
```

마찬가지로 -1 또는 +0 이 있으면 상수항이 없어진다.

In [7]:

```
dmatrix("x1 + x2 - 1", df)
```

Out[7]:

```
DesignMatrix with shape (5, 2)
       x1        x2
  1.76405  -0.97728
  0.40016   0.95009
  0.97874  -0.15136
  2.24089  -0.10322
  1.86756   0.41060
  Terms:
    'x1' (column 0)
    'x2' (column 1)
```

두 변수의 곱을 새로운 변수로 추가하려면 상호작용(interaction) 연산자 : 를 사용한다.

In [8]:

```
dmatrix("x1 + x2 + x1:x2", df)
```

Out[8]:

```
DesignMatrix with shape (5, 4)
  Intercept       x1        x2     x1:x2
          1  1.76405  -0.97728  -1.72397
          1  0.40016   0.95009   0.38018
          1  0.97874  -0.15136  -0.14814
          1  2.24089  -0.10322  -0.23130
          1  1.86756   0.41060   0.76682
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'x2' (column 2)
    'x1:x2' (column 3)
```

위 식은 다음과 같이 * 연산자로 간단하게 나타낼 수도 있다.

In [9]:

```
dmatrix("x1 * x2", df)
```

Out[9]:

```
DesignMatrix with shape (5, 4)
  Intercept       x1        x2     x1:x2
          1  1.76405  -0.97728  -1.72397
          1  0.40016   0.95009   0.38018
          1  0.97874  -0.15136  -0.14814
          1  2.24089  -0.10322  -0.23130
          1  1.86756   0.41060   0.76682
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'x2' (column 2)
    'x1:x2' (column 3)
```

/ 연산자는 두번째 데이터를 빼고 출력한다.

In [10]:

```
dmatrix("x1 / x2", df)
```

Out[10]:

```
DesignMatrix with shape (5, 3)
  Intercept       x1     x1:x2
          1  1.76405  -1.72397
          1  0.40016   0.38018
          1  0.97874  -0.14814
          1  2.24089  -0.23130
          1  1.86756   0.76682
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'x1:x2' (column 2)
```

수학 변환

dmatrix 에서는 일반적인 수학 변환(transform)도 가능하다. numpy 함수 뿐 아니라 사용자 정의 함수도 사용할 수 있다.

In [11]:

```
dmatrix("x1 + np.log(np.abs(x2))", df)
```

Out[11]:

```
DesignMatrix with shape (5, 3)
  Intercept       x1  np.log(np.abs(x2))
          1  1.76405            -0.02298
          1  0.40016            -0.05120
          1  0.97874            -1.88811
          1  2.24089            -2.27090
          1  1.86756            -0.89014
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'np.log(np.abs(x2))' (column 2)
```

In [12]:

```
def doubleit(x):
    return 2 * x

dmatrix("doubleit(x1)", df)
```

Out[12]:

```
DesignMatrix with shape (5, 2)
  Intercept  doubleit(x1)
          1       3.52810
          1       0.80031
          1       1.95748
          1       4.48179
          1       3.73512
  Terms:
    'Intercept' (column 0)
    'doubleit(x1)' (column 1)
```

상태 보존 변환

patsy의 가장 강력한 기능 중의 하나는 상태 보존 변환(stateful trasform)이 가능하다는 점이다. 예를 들어 다음 변환 함수는 평균을 계산하여 빼주거나 및 표준편차를 계산하여 나누어주는데 이 때 계산한 평균과 표준편차를 내부에 상태변수로 저장한다.

- center(x) : 평균 제거
- standardize(x) : 평균 제거 및 표준편차로 스케일링
- scale(x) : standardize(x) 과 같음

예를 들어 x1 데이터의 평균을 제거하는 변환은 다음과 같다.

In [13]:

```
dm = dmatrix("center(x1)", df)
dm
```

Out[13]:

```
DesignMatrix with shape (5, 2)
  Intercept  center(x1)
          1     0.31377
          1    -1.05012
          1    -0.47154
          1     0.79061
          1     0.41728
  Terms:
    'Intercept' (column 0)
    'center(x1)' (column 1)
```

이 변환 연산은 다음과 같이 x1 데이터에서 x1의 평균을 빼는 것이다.

In [14]:

```
df.x1 - np.mean(df.x1)
```

Out[14]:

```
0     0.313773
1    -1.050123
2    -0.471542
3     0.790613
4     0.417278
Name: x1, dtype: float64
```

그런데 이 때 사용한 평균값은 design_info 라는 속성에 상태변수(state variable)로서 저장된다.

In [15]:

```
type(dm.design_info)
```

Out[15]:

```
patsy.design_info.DesignInfo
```

이 값을 상태변수로 저장하는 이유는 다음과 같다.

어떤 데이터 X_{train}을 학습용 데이터로 사용하여 예측모형으로 만든다고 하자. 이 때 학습성능을 좋게 하기 위해 X_{train}에서 X_{train}의 평균 $\bar{X}_{train}$(예를 들어 100)을 뺀 평균 제거 데이터 $X_{train} - 100$를 원래의 데이터 대신 학습용 데이터로 사용하여 모형을 만드는 경우가 있다. 이를 전처리 단계라고 한다.

학습이 끝난 후 이 모형을 사용하여 실제 예측을 하자. 새로운 검증용 데이터 X_{test}를 이 모형에 넣으려면 모형을 학습할 때 사용한 것과 같은 전처리를 해야 한다. 즉, X_{test}에서 X_{train}의 평균인 100을 뺀 $X_{test} - 100$을 입력으로 넣어서 출력을 계산해야 한다. 이 때 X_{test}의 평균이 아니라 X_{train}의 평균을 사용한다는 점에 주의한다. 이렇게 하기 위해서는 전처리 과정에서 계산한 X_{train}의 평균값 100을 기억하고 있어야 한다.

patsy 패키지에서는 center 변환을 했을 때 사용한 평균값을 내부에 저장하고 있기 때문에 이러한 일을 할 수 있다. 예를 들어 다음처럼 검증용의 새로운 데이터가 있을 때,

In [16]:

```
df_new = df.copy()
df_new["x1"] = df_new["x1"] * 10
df_new
```

Out[16]:

	x1	x2	y
0	17.640523	-0.977278	0.144044
1	4.001572	0.950088	1.454274
2	9.787380	-0.151357	0.761038
3	22.408932	-0.103219	0.121675
4	18.675580	0.410599	0.443863

build_design_matrices 명령을 사용하면 이미 저장된 x1의 평균을 이용하여 같은 변환을 한다.

In [17]:

```
build_design_matrices([dm.design_info], df_new)
```

Out[17]:

```
[DesignMatrix with shape (5, 2)
   Intercept  center(x1)
           1    16.19024
           1     2.55129
           1     8.33710
           1    20.95865
           1    17.22530
   Terms:
     'Intercept' (column 0)
     'center(x1)' (column 1)]
```

이 값은 다음 계산 결과와 같다.

In [18]:

```
df_new.x1 - np.mean(df.x1)
```

Out[18]:

```
0     16.190244
1      2.551292
2      8.337100
3     20.958652
4     17.225300
Name: x1, dtype: float64
```

평균값을 다시 새롭게 구해서 계산한 것과 다르다는 것을 알 수있다.

In [19]:

```
dmatrix("center(x1)", df_new)
```

Out[19]:

```
DesignMatrix with shape (5, 2)
  Intercept  center(x1)
          1     3.13773
          1   -10.50123
          1    -4.71542
          1     7.90613
          1     4.17278
  Terms:
    'Intercept' (column 0)
    'center(x1)' (column 1)
```

In [20]:

```
df_new.x1 - np.mean(df_new.x1)
```

Out[20]:

```
0     3.137726
1   -10.501225
2    -4.715418
3     7.906135
4     4.172782
Name: x1, dtype: float64
```

변수 보호

함수를 사용한 변수 변환 이외에도 모형 정의 문자열 자체내에 연산기호를 넣어 연산한 값을 만드는 것도 가능하다. 이 때에는 모형정의 연산자와 혼동되지 않도록 `I()` 연산자를 추가해야 한다.

In [21]:

```
dmatrix("I(x1 + x2)", df)
```

Out[21]:

```
DesignMatrix with shape (5, 2)
  Intercept  I(x1 + x2)
          1     0.78677
          1     1.35025
          1     0.82738
          1     2.13767
          1     2.27816
  Terms:
    'Intercept' (column 0)
    'I(x1 + x2)' (column 1)
```

이 값을 다음 식과 비교하면 `I()` 의 기능을 확실히 알 수 있다.

In [22]:

```
dmatrix("x1 + x2", df)
```

Out[22]:

```
DesignMatrix with shape (5, 3)
  Intercept       x1        x2
          1  1.76405  -0.97728
          1  0.40016   0.95009
          1  0.97874  -0.15136
          1  2.24089  -0.10322
          1  1.86756   0.41060
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'x2' (column 2)
```

다항회귀

I() 연산자를 활용하면 다항회귀(polynomial regression)도 할 수 있다.

In [23]:

```
dmatrix("x1 + I(x1*x1) + I(x1**3) + I(x1**4)", df)
```

Out[23]:

```
DesignMatrix with shape (5, 5)
  Intercept       x1  I(x1 * x1)  I(x1 ** 3)  I(x1 ** 4)
          1  1.76405     3.11188     5.48952     9.68380
          1  0.40016     0.16013     0.06408     0.02564
          1  0.97874     0.95793     0.93756     0.91763
          1  2.24089     5.02160    11.25287    25.21649
          1  1.86756     3.48777     6.51362    12.16456
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'I(x1 * x1)' (column 2)
    'I(x1 ** 3)' (column 3)
    'I(x1 ** 4)' (column 4)
```

OLS.from_formula 메서드

선형회귀분석을 위한 OLS 클래스에는 모형 정의 문자열을 사용할 수 있는 from_formula 라는 클래스 메서드가 있다. 이 메서드를 쓰면 사용자가 데이터 행렬을 직접 정의하지 않고 모형 정의 문자열만으로 선형회귀모형을 만드는 것이 가능하다.

선형 회귀모형을 formula로 정의할 때는 ~ 연산자를 사용한다. ~ 연산자의 왼쪽에는 종속 변수, 오른쪽에는 독립 변수를 넣어서 정의한다. 예를 들어 다음과 같은 데이터가 있을 때,

In [24]:

```
np.random.seed(0)
x1 = np.random.rand(20) + 10
x2 = np.random.rand(20) * 10
y = x1 + 2 * x2 + np.random.randn(20)
df4 = pd.DataFrame(np.array([x1, x2, y]).T, columns=["x1", "x2", "y"])
```

다음 두가지 방법으로 만든 모형은 동일하다.

In [25]:

```
# 직접 데이터 행렬을 만드는 경우
dfy = df4.iloc[:, -1]
dfX = sm.add_constant(df4.iloc[:, :-1])
model1 = sm.OLS(dfy, dfX)

print(model1.fit().summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.967
Model:                            OLS   Adj. R-squared:                  0.963
Method:                 Least Squares   F-statistic:                     246.8
Date:                Mon, 17 Jun 2019   Prob (F-statistic):           2.75e-13
Time:                        15:59:58   Log-Likelihood:                -29.000
No. Observations:                  20   AIC:                             64.00
Df Residuals:                      17   BIC:                             66.99
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.4226     10.140      0.140      0.890     -19.971      22.816
x1             0.8114      0.977      0.831      0.418      -1.249       2.872
x2             2.0367      0.100     20.305      0.000       1.825       2.248
==============================================================================
Omnibus:                        1.081   Durbin-Watson:                   1.896
Prob(Omnibus):                  0.583   Jarque-Bera (JB):                0.949
Skew:                           0.470   Prob(JB):                        0.622
Kurtosis:                       2.494   Cond. No.                         494.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```

In [26]:

```
# 모형 정의 문자열을 사용하는 경우
model2 = sm.OLS.from_formula("y ~ x1 + x2", data=df4)

print(model2.fit().summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.967
Model:                            OLS   Adj. R-squared:                  0.963
Method:                 Least Squares   F-statistic:                     246.8
Date:                Mon, 17 Jun 2019   Prob (F-statistic):           2.75e-13
Time:                        15:59:58   Log-Likelihood:                -29.000
No. Observations:                  20   AIC:                             64.00
Df Residuals:                      17   BIC:                             66.99
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.4226     10.140      0.140      0.890     -19.971      22.816
x1             0.8114      0.977      0.831      0.418      -1.249       2.872
x2             2.0367      0.100     20.305      0.000       1.825       2.248
==============================================================================
Omnibus:                        1.081   Durbin-Watson:                   1.896
Prob(Omnibus):                  0.583   Jarque-Bera (JB):                0.949
Skew:                           0.470   Prob(JB):                        0.622
Kurtosis:                       2.494   Cond. No.                         494.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```