13. 파이썬 모델링 라이브러리

이 책은 파이썬을 활용한 데이터 분석에 필요한 기본 프로그래밍 실력을 키우는 데 초점을 맞추었다.

데이터 분석가와 과학자들은 정제하고 준비하는 데 너무 많은 시간을 쓰고 있으며 이 책에서도 관련 기법을 습득하는 데 많은 지면을 할애했다.

모델을 개발하는 데 어떤 라이브러리를 사용할지는 어떤 애플리케이션을 적용하느냐에 따라 달라진다. 많은 통계 문제는 최소제곱회귀 같은 단순한 기법으로 해결할 수 있으며 어떤 문제는 고급 머신러닝 방식으로 해결할 수 있다. 다행히도 파이썬은 이런 분석 기법들을 구현할 수 있는 언어 중 하나가 되었다.

이 장에서는 모델 피팅 및 스코어링과 pandas를 이용한 데이터 정제 작업 사이를 오가는 와중에 도움이 될만한 pandas의 기능을 살펴보겠다. 그리고 유명한 모델링 도구인 statmodels와 scikit-learn을 간단히 소개한다. 이 두 프로젝트는 그 자체로도 책 한권이 필요할 정도의 방대한 프로젝트이므로 전체를 다루기보다는 두 프로젝트의 온라인 문서와 데이터 과학, 통계, 그리고 머신러닝을 다루는 다른 파이썬 서적을 소개하는 것으로 대신하려 한다.

13.1 pandas와 모델 코드의 인터페이스

모델 개발의 일반적인 흐름은 데이터를 불러오고 정제하는 과정은 pandas를 이용하고 그 후 모델 개발을 위해 모델링 라이브러리로 넘어간다. 모델을 개발하는 과정에서 중요한 단계는 특징을 선택하고 추출하는 **피쳐 엔지니어링**인데 원시 데이터셋으로부터 모델링에서 유용할 수 있는 정보를 추출하는 변환이나 분석 과정을 일컫는다. 이 책에서 살펴본 데이터 요약이나 GroupBy 도구들이 피쳐 엔지니어링 과정에서 자주 사용된다.

'좋은' 피쳐 엔지니어링에 대한 자세한 내용은 이 책의 범위를 벗어나므로, pandas를 이용한 데이터 조작과 모델링 사이를 편리하게 오갈 수 있는 방법을 설명하겠다.

pandas와 다른 분석 라이브러리는 주로 NumPy 배열을 사용해서 연계할 수 있다. DataFrame을 NumPy 배열로 변환하려면 .values 속성을 이용한다.

```
In [75]: data = pd.DataFrame({
    ...:                       'x0':[1,2,3,4,5],
    ...:                       'x1':[0.01,-0.01,0.25,-4.10,0.00],
    ...:                       'y1':[-1.5,0.,3.6,1.3,-2.]})

In [76]: data
Out[76]:
   x0    x1   y1
0   1  0.01 -1.5
1   2 -0.01  0.0
2   3  0.25  3.6
3   4 -4.10  1.3
4   5  0.00 -2.0

In [77]: data.columns
Out[77]: Index(['x0', 'x1', 'y1'], dtype='object')

In [78]: data.values
Out[78]:
array([[ 1.  ,  0.01, -1.5 ],
       [ 2.  , -0.01,  0.  ],
       [ 3.  ,  0.25,  3.6 ],
```

```
       [ 4.  , -4.1 ,  1.3 ],
       [ 5.  ,  0.  , -2.  ]])
```

다시 DataFrame 으로 되돌리려면 앞서 공부했던 것 처럼 2차원 ndarray 와 필요하다면 컬럼 이름을 이름 리스트로 넘겨서 생성할 수 있다.

```
In [79]: df2 = pd.DataFrame(data.values, columns=['one','two','three'])
    ...:

In [80]: df2
Out[80]:
   one   two  three
0  1.0  0.01   -1.5
1  2.0 -0.01    0.0
2  3.0  0.25    3.6
3  4.0 -4.10    1.3
4  5.0  0.00   -2.0
```

Note_values 속성은 데이터가 한 가지 타입(예를 들면 모두 숫자형) 으로 이뤄졌다는 가정 하에 사용된다. 만약 데이터 속성이 한 가지가 아니라면 파이썬 객체의 ndarray가 반환될 것이다.

```
In [81]: df3 = data.copy()

In [82]: df3['strings'] = ['a','b','c','d','e']

In [83]: df3
Out[83]:
   x0    x1   y1 strings
0   1  0.01 -1.5       a
1   2 -0.01  0.0       b
2   3  0.25  3.6       c
3   4 -4.10  1.3       d
4   5  0.00 -2.0       e

In [84]: df3.values
Out[84]:
array([[1, 0.01, -1.5, 'a'],
       [2, -0.01, 0.0, 'b'],
       [3, 0.25, 3.6, 'c'],
       [4, -4.1, 1.3, 'd'],
       [5, 0.0, -2.0, 'e']], dtype=object)
```

어떤 모델은 전체 컬럼 중 일부만 사용하고 싶은 경우도 있을 것이다. 이 경우 loc를 이용해서 values 속성에 접근하기 바란다.

```
In [86]: data.loc[:,model_cols].values
Out[86]:
array([[ 1.  ,  0.01],
       [ 2.  , -0.01],
       [ 3.  ,  0.25],
       [ 4.  , -4.1 ],
       [ 5.  ,  0.  ]])
```

어떤 라이브러리는 pandas를 직접 지원하기도 하는데 위에서 설명한 과정을 자동으로 처리해준다. DataFrame에서 NumpPy 배열로 변환하고 모델 인자의 이름을 출력 테이블이나 Series의 컬럼으로 추가한다. 아니라면 이런 메타 데이터 관리를 수동으로 직접해야 한다.

12장에서 pandas의 Categorical형과 pandas.get_dummies 함수를 살펴봤다. 예제 데이터셋에 숫자가 아닌 컬럼이 있다고 가정하자.

```
In [87]: data['category'] = pd.Categorical(['a','b','a','a','b'],
    ...:                                   categories= ['a','b'])

In [88]: data
Out[88]:
   x0    x1   y1 category
0   1  0.01 -1.5        a
1   2 -0.01  0.0        b
2   3  0.25  3.6        a
3   4 -4.10  1.3        a
4   5  0.00 -2.0        b
```

만일 'category' 컬럼을 더미값으로 치환하고 싶다면 더미값을 생성하고 'category' 컬럼을 삭제한 다음 결과와 합쳐야 한다.

```
In [89]: dummies = pd.get_dummies(data.category, prefix='category')

In [90]: data_with_dummy = data.drop('category',axis = 1).join(dummies)
    ...:

In [91]: data_with_dummy
Out[91]:
   x0    x1   y1  category_a  category_b
0   1  0.01 -1.5           1           0
1   2 -0.01  0.0           0           1
2   3  0.25  3.6           1           0
3   4 -4.10  1.3           1           0
4   5  0.00 -2.0           0           1
```

특정 통계 모델을 더미값으로 피팅하는 기법도 있다. 단순히 숫자형 컬럼만 가지고 있는게 아니라면 다음 절에서 살펴 볼 Patsy를 사용하는 편이 더 단순하고 에러를 일으킬 가능성도 줄여준다.

12.3 Patsy를 이용해서 모델 생성하기

Patsy(팻시)는 통계 모델(특히 선형 모델)을 위한 파이썬 라이브러리이며 R이나 S 통계 프로그래밍 언어에서 사용하는 수식 문법과 비슷한 형식의 문자열 기반 '수식 문법'을 제공한다.

Patsy는 통계 모델에서 선형 모델을 잘 지원하므로 이해를 돕기 위해 주요 기능 중 일부만 살펴보자. Patsy의 수식 문법은 다음과 같은 특수한 형태의 문자열이다.

```
y1 ~ x0 + x1
```

a + b 문법은 a와 b를 더하라는 것이 아닌, 모델을 위해 생성된 **배열을 설계** 하는 용법이다. patsy.dmatrices 함수는 수식 문자열과 데이터셋(DataFrame 또는 배열의 사전) 을 함께 받아 선형 모델을 위한 설계 배열을 만들어낸다.

```
In [97]: data = pd.DataFrame({
```

```
    ...:                     'x0':[1,2,3,4,5],
    ...:                     'x1':[0.01,-0.01,0.25,-4.10,0.00],
    ...:                     'y1':[-1.5,0.,3.6,1.3,-2.]})

In [93]: data
Out[93]:
   x0    x1   y1
0   1  0.01 -1.5
1   2 -0.01  0.0
2   3  0.25  3.6
3   4 -4.10  1.3
4   5  0.00 -2.0

In [98]: y,X = patsy.dmatrices('y1 ~ x0 + x1',data)
```

dmatrices 함수를 실행하면 다음과 같은 결과를 얻을 수 있다.

```
In [101]: y
Out[101]:
DesignMatrix with shape (5, 1)
    y1
  -1.5
   0.0
   3.6
   1.3
  -2.0
  Terms:
    'y1' (column 0)

In [102]: X
Out[102]:
DesignMatrix with shape (5, 3)
  Intercept  x0     x1
          1   1   0.01
          1   2  -0.01
          1   3   0.25
          1   4  -4.10
          1   5   0.00
  Terms:
    'Intercept' (column 0)
    'x0' (column 1)
    'x1' (column 2)
```

Patsy의 DesignMatrix 인스턴스는 몇 가지 추가 데이터가 포함된 NumPy ndarray로 볼 수 있다.

```
In [103]: np.asarray(y)
Out[103]:
array([[-1.5],
       [ 0. ],
       [ 3.6],
       [ 1.3],
       [-2. ]])

In [105]: np.asarray(X)
Out[105]:
array([[ 1.  ,  1.  ,  0.01],
```

```
       [ 1.  ,  2.  , -0.01],
       [ 1.  ,  3.  ,  0.25],
       [ 1.  ,  4.  , -4.1 ],
       [ 1.  ,  5.  ,  0.  ]])
```

여기서 Intercept는 최소자승회귀와 같은 선형 모델을 위한 표현이다. 모델에 0을 더해서 intercept(절편)를 제거할 수 있다.

```
In [109]: patsy.dmatrices('y1 ~ x0 + x1 + 0 ',data)[1]
Out[109]:
DesignMatrix with shape (5, 2)
  x0     x1
   1   0.01
   2  -0.01
   3   0.25
   4  -4.10
   5   0.00
  Terms:
    'x0' (column 0)
    'x1' (column 1)
```

Patsy 객체는 최소자승회귀분석을 위해 numpy.linalg.lstsq 같은 알고리즘에 바로 넘길 수도 있다.

```
In [110]: coef, resid, _, _ = np.linalg.lstsq(X,y)
```

모델 메타데이터는 design_info 속성을 통해 얻을 수 있는데 예를 들면 모델의 컬럼명을 피팅된 항에 맞춰 Series를 만들어 낼 수도 있다.

```
In [111]: coef
Out[111]:
array([[ 0.31290976],
       [-0.07910564],
       [-0.26546384]])

In [114]: coef = pd.Series(coef.squeeze(), index = X.design_info.column
     ...: _names)

In [115]: coef
Out[115]:
Intercept    0.312910
x0          -0.079106
x1          -0.265464
dtype: float64
```

13.2.1 Patsy 용법으로 데이터 변환하기

파이썬 코드를 Patsy 용법과 섞어서 사용할 수도 있는데, Patsy 문법을 해석하는 과정에서 해당 함수를 찾아 실행해준다.

```
In [116]: y, X = patsy.dmatrices('y1~x0 + np.log(np.abs(x1)+1)',data)

In [117]: X
Out[117]:
DesignMatrix with shape (5, 3)
  Intercept  x0  np.log(np.abs(x1) + 1)
```

```
          1   1                0.00995
          1   2                0.00995
          1   3                0.22314
          1   4                1.62924
          1   5                0.00000
  Terms:
    'Intercept' (column 0)
    'x0' (column 1)
    'np.log(np.abs(x1) + 1)' (column 2)
```

자주 쓰이는 변수 변환으로는 표준화 (평균 0, 분산 1)와 센터링(평균값을 뺌)이 있는데 Pasty에는 이런 목적을 위한 내장 함수가 있다.

```
In [118]: y, X = patsy.dmatrices('y1~standardize(x0)+center(x1)',data)

In [119]: X
Out[119]:
DesignMatrix with shape (5, 3)
  Intercept  standardize(x0)  center(x1)
          1         -1.41421        0.78
          1         -0.70711        0.76
          1          0.00000        1.02
          1          0.70711       -3.33
          1          1.41421        0.77
  Terms:
    'Intercept' (column 0)
    'standardize(x0)' (column 1)
    'center(x1)' (column 2)
```

모델링 과정에서 모델을 어떤 데이터셋에 피팅하고 그 다음에 다른 모델에 기반하여 평가해야 하는 경우가 있다. 이는 홀드 아웃 이러간 신규 데이터가 나중에 관측되는 경우다. 센터링이나 표준화 같은 변환을 적용하는 경우 새로운 데이터에 기반하여 예측하기 위한 용도로 모델을 사용한다면 주의해야 한다. 이에 대한 상태를 가지는(stateful) 변환이라고 하는데 새로운 데이터셋을 변경하기 위해 원본 데이터의 표준편차나 평균 같은 통계를 사용해야 하기 때문이다.

patsy.build_design_matrices 함수는 입력으로 사용되는 원본 데이터셋에서 저장한 정보를 사용해서 출력 데이터를 만들어내는 변환에 적용할 수 있는 함수다.

```
In [122]: new_X = patsy.build_design_matrices([X.design_info],new_data)
     ...:

In [123]: new_X
Out[123]:
[DesignMatrix with shape (4, 3)
   Intercept  standardize(x0)  center(x1)
           1          2.12132        3.87
           1          2.82843        0.27
           1          3.53553        0.77
           1          4.24264        3.07
   Terms:
     'Intercept' (column 0)
     'standardize(x0)' (column 1)
     'center(x1)' (column 2)]
```

Patsy 문법에 더하기 기호(+)는 덧셈이 아니므로 데이터셋에서 이름으로 컬럼을 추가하고 싶다면 I라는 특수한 함수로 둘러싸야 한다.

```
In [124]: y, X = patsy.dmatrices('y~I(x0+x1)',data)

In [125]: X
Out[125]:
DesignMatrix with shape (5, 2)
  Intercept  I(x0 + x1)
          1        1.01
          1        1.99
          1        3.25
          1       -0.10
          1        5.00
  Terms:
    'Intercept' (column 0)
    'I(x0 + x1)' (column 1)
```

Patsy는 pasty.builtins 모듈 내에 여러 가지 변환을 위한 내장함수들을 제공한다.

13.2.2 범주형 데이터와 Pasty

비산술 데이터는 여러 가지 형태의 모델 설계 배열로 변환될 수 있다.

Patsy에서 비산술 용법을 사용하면 기본적으로 더미 변수로 변환된다. 만약 Intercept가 존재한다면 공선성을 피하기 위해 레벨 중 하나는 남겨둔다.

```
In [127]: data = pd.DataFrame({
     ...:         'key1':['a','a','b','b','a','b','a','b'],
     ...:         'key2':[0,1,0,1,0,1,0,0],
     ...:         'v1':[1,2,3,4,5,6,7,8],
     ...:         'v2':[-1,0,2.5,-0.5,4.0,-1.2,0.2,-1.7]})

In [128]: y, X = patsy.dmatrices('v2~key1',data)

In [129]: X
Out[129]:
DesignMatrix with shape (8, 2)
  Intercept  key1[T.b]
          1          0
          1          0
          1          1
          1          1
          1          0
          1          1
          1          0
          1          1
  Terms:
    'Intercept' (column 0)
    'key1' (column 1)
```

모델에서 intercept를 생략하면 각 범주값의 컬럼은 모델 설계 배열에 포함된다.

```
In [130]: y, X = patsy.dmatrices('v2 ~ key1 + 0',data)

In [131]: X
```

```
Out[131]:
DesignMatrix with shape (8, 2)
  key1[a]  key1[b]
        1        0
        1        0
        0        1
        0        1
        1        0
        0        1
        1        0
        0        1
  Terms:
    'key1' (columns 0:2)
```

산술 컬럼은 C 함수를 이용해서 범주형으로 해석할 수 있다.

```
In [132]: y, X = patsy.dmatrices('v2 ~ C(key2)',data)

In [133]: X
Out[133]:
DesignMatrix with shape (8, 2)
  Intercept  C(key2)[T.1]
          1             0
          1             1
          1             0
          1             1
          1             0
          1             1
          1             0
          1             0
  Terms:
    'Intercept' (column 0)
    'C(key2)' (column 1)
```

모델에서 여러 범주형 항을 사용한다면 ANOVA(분산분석) 모델에서처럼 key1:key2 같은 용법을 사용할 수 있게 되므로 더 복잡해진다.

```
In [134]: data
Out[134]:
  key1  key2  v1   v2
0    a     0   1 -1.0
1    a     1   2  0.0
2    b     0   3  2.5
3    b     1   4 -0.5
4    a     0   5  4.0
5    b     1   6 -1.2
6    a     0   7  0.2
7    b     0   8 -1.7

In [135]: data['key2'] = data['key2'].map({0:'zero',1:'one'})

In [136]: data
Out[136]:
  key1  key2  v1   v2
0    a  zero   1 -1.0
1    a   one   2  0.0
```

```
2    b  zero   3  2.5
3    b   one   4 -0.5
4    a  zero   5  4.0
5    b   one   6 -1.2
6    a  zero   7  0.2
7    b  zero   8 -1.7

In [137]: y, X = patsy.dmatrices('v2 ~ key1+key2', data)

In [138]: X
Out[138]:
DesignMatrix with shape (8, 3)
  Intercept  key1[T.b]  key2[T.zero]
          1          0             1
          1          0             0
          1          1             1
          1          1             0
          1          0             1
          1          1             0
          1          0             1
          1          1             1
  Terms:
    'Intercept' (column 0)
    'key1' (column 1)
    'key2' (column 2)

In [139]: y, X = patsy.dmatrices('v2 ~ key1+key2+key1:key2', data)

In [140]: X
Out[140]:
DesignMatrix with shape (8, 4)
  Intercept  key1[T.b]  key2[T.zero]  key1[T.b]:key2[T.zero]
          1          0             1                       0
          1          0             0                       0
          1          1             1                       1
          1          1             0                       0
          1          0             1                       0
          1          1             0                       0
          1          0             1                       0
          1          1             1                       1
  Terms:
    'Intercept' (column 0)
    'key1' (column 1)
    'key2' (column 2)
    'key1:key2' (column 3)
```

Patsy는 특정 순서에 따라 데이터를 변환하는 방법을 포함하여 범주형 데이터를 변환하는 여러 가지 방법을 제공한다.

13.3 statsmodels 소개

statsmodels는 다양한 종류의 통계 모델 피팅, 통계 테스트 수행 그리고 데이터 탐색과 시각화를 위한 파이썬 라이브러리다. statsmodels는 좀 더 '전통적인' 빈도주의적 통계 메서드를 포함하고 있다. 베이지안 메서드나 머신러닝 모델은 다른 라이브러리에서 찾을 수 있다.

statsmodels는 다음과 같은 모델을 포함한다.

- **선형 모델, 일반 선형 모델, 로버스트 선형 모델**
- **선형 복합효과(Linear Mixed Effects, LME) 모델**
- **아노바(Anova) 메서드**
- **시계열 처리 및 상태 공간 모델**
- **일반적률추정법(Generalized Method of Moments, GMM)**

이제 statsmodels의 몇 가지 기본 도구를 사용해보고 Patsy와 pandas의 DataFrame 객체와 모델링 인터페이스를 어떻게 사용하는지 살펴보자.

13.3.1 선형 모델 예측하기

statsmodels에는 아주 기본적인 선형회귀 모델(예를 들어 최소제곱(Ordinary Least Squares, OLS)) 부터 좀 더 복잡한 선형회귀 모델(예를 들어 반복재가중 최소제곱(Iteratively Reweighted Least Squares, IRLS))까지 존재한다.

statsmodels의 선형 모델은 두 가지 주요한 인터페이스를 가지는데, 배열 기반과 용법 기반이다. 이 인터페이스는 API 모듈을 임포트하여 사용할 수 있다.

```
In [141]: import statsmodels.api as sm

In [142]: import statsmodels.formula.api as smf
```

어떻게 사용하는지 알아보기 위해 랜덤 데이터에서 선형 모델을 생성해보자.

```
In [143]: def dnorm(mean, variance, size = 1):
     ...:     if isinstance(size, int):
     ...:         size = size,
     ...:     return mean + np.sqrt(variance) + np.random.randn(*size)
```

여기서는 알려진 인자는 beta를 이용해서 진짜 모델을 작성했다. dnorm은 특정 평균과 분산을 가지는 정규분포 데이터를 생성하기 위한 도움 함수다. 이제 다음과 같은 데이터셋을 가지게 됐다.

```
In [152]: #동일한 난수 발생을 위해 시드값 직접 지정
     ...: np.random.seed(12345)
     ...:
     ...: N = 100
     ...: X = np.c_[dnorm(0,0.4,size = N),
     ...:               dnorm(0,0.6,size = N),
     ...:               dnorm(0,0.2,size = N )]
     ...: eps = dnorm(0,0.1,size = N)
     ...: beta = [0.1,0.3,0.5]
     ...:
     ...: y = np.dot(X, beta) + eps
     ...:
```

```
In [155]: X[:5]
Out[155]:
array([[ 0.42774787, -0.79106062,  1.5746947 ],
       [ 1.11139887,  0.21205648, -0.12114985],
       [ 0.11301682,  0.74193253,  0.75657576],
       [ 0.07672523, -0.15440953, -0.13017188],
       [ 2.5982361 ,  0.29202402, -0.72142048]])

In [158]: y[:5]
Out[158]: array([ 2.65626531, -0.97983647,
                 0.55015556, -0.13332918,  0.68301081])
```

선형 모델은 이전에 Patsy에서 봤던 것처럼 일반적으로 intercept와 함께 피팅된다. sm.add_constant 함수는 intercept 컬럼을 기존 행렬에 더할 수 있다.

```
In [159]: X_model = sm.add_constant(X)

In [160]: X_model[:5]
Out[160]:
array([[ 1.        ,  0.42774787, -0.79106062,  1.5746947 ],
       [ 1.        ,  1.11139887,  0.21205648, -0.12114985],
       [ 1.        ,  0.11301682,  0.74193253,  0.75657576],
       [ 1.        ,  0.07672523, -0.15440953, -0.13017188],
       [ 1.        ,  2.5982361 ,  0.29202402, -0.72142048]])
```

sm.OLS 클래스는 최소자승 선형회귀에 피팅할 수 있다.

```
In [161]: model = sm.OLS(y,X)
```

모델의 fit 메서드는 예측 모델 인자와 다른 분석 정보를 포함하는 회귀 결과 객체를 반환한다.

```
In [163]: results = model.fit()

In [164]: results.params
Out[164]: array([0.39004368, 0.29561838, 0.5841267 ])
```

results의 summary 메서드를 호출하여 해당 모델의 자세한 분석 결과를 출력하도록 할 수 있다.

```
In [165]: print(results.summary())
                                 OLS Regression Results
==============================================================================
======
Dep. Variable:                      y   R-squared (uncentered):
0.437
Model:                            OLS   Adj. R-squared (uncentered):
0.420
Method:                 Least Squares   F-statistic:
25.14
Date:                Wed, 17 Jun 2020   Prob (F-statistic):
4.05e-12
Time:                        12:33:34   Log-Likelihood:
-153.12
No. Observations:                 100   AIC:
312.2
```

```
Df Residuals:                       97   BIC:
320.0
Df Model:                            3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.3900      0.098      3.969      0.000       0.195       0.585
x2             0.2956      0.098      3.005      0.003       0.100       0.491
x3             0.5841      0.118      4.968      0.000       0.351       0.818
==============================================================================
Omnibus:                        1.984   Durbin-Watson:                   2.181
Prob(Omnibus):                  0.371   Jarque-Bera (JB):                1.805
Skew:                           0.328   Prob(JB):                        0.405
Kurtosis:                       2.943   Cond. No.                         1.52
==============================================================================

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

지금까지는 인자의 이름을 x1, x2 등으로 지었다. 그대신 모든 모델 인자가 하나의 DataFrame에 들어 있다고 가정해보자.

```
In [167]: data = pd.DataFrame(X, columns = ['col0','col1','col2'])

In [168]: data['y'] = y

In [169]: data[:5]
Out[169]:
       col0      col1      col2         y
0  0.427748 -0.791061  1.574695  2.656265
1  1.111399  0.212056 -0.121150 -0.979836
2  0.113017  0.741933  0.756576  0.550156
3  0.076725 -0.154410 -0.130172 -0.133329
4  2.598236  0.292024 -0.721420  0.683011
```

이제 statsmodels 의 API와 Patsy의 문자열 용법을 사용할 수 있다.

```
In [177]: results = smf.ols('y ~ col0 + col1+ col2',data = data).fit()

In [178]: results.params
Out[178]:
Intercept    0.459295
col0         0.252297
col1         0.115862
col2         0.520942
dtype: float64

In [179]: results.tvalues
Out[179]:
Intercept    2.882883
col0         2.377438
col1         1.020530
col2         4.510690
dtype: float64
```

statsmodels 에서 반환하는 결과 Series가 DataFrame의 컬럼 이름을 사용하고 있는 것을 알 수 있다. 또한 pandas 객체를 이용해서 수식을 사용하는 경우에는 add_constant를 호출할 필요가 없다.

주어진 새로운 샘플 데이터를 통해 예측 모델 인자에 전달한 예측값을 계산할 수 있다.

```
In [180]: results.predict(data[:5])
Out[180]:
0    1.295885
1    0.701156
2    0.967903
3    0.392951
4    0.772840
dtype: float64
```

statsmodels 에는 선형 모델 결과에 대한 분석, 진단 그리고 시각화를 위한 많은 추가적인 도구가 포함되어 있다. 최소제곱뿐만 아니라 다른 선형 모델에 대한 것들도 포함하고 있다.

13.3.2 시계열 처리 예측

statsmodels에 포함된 또 다른 모델 클래스로는 시계열분석을 위한 모델이 있다. 시계열분석을 위한 모델에는 자동회귀 처리, 칼만 필터링과 같은 다른 상태 공간 모델 그리고 다변 자동회귀 모델 등이 있다.

자동회귀 구조와 노이즈를 이용해서 시계열 데이터를 시뮬레이션 해보자.

```
In [181]: init_x = 4
     ...:
     ...: import random
     ...: values = [init_x, init_x]
     ...: N= 1000
     ...:
     ...: b0 = 0.8
     ...: b1 = -0.4
     ...: noise = dnorm(0,0.1,N)
     ...: for i in range(N):
     ...:     new_x = values[-1] * b0 + values[-2] * b1 + noise[i]
     ...:     values.append(new_x)
     ...:
```

이 데이터는 인자가 0.8과 -0.4인 AR(2) 구조 (두 개의 지연)다. AR 모델을 피팅할 때는 포함시켜야 할 지연항을 얼마나 두어야 하는지 알지 못하므로 적당히 큰 값으로 모델을 피팅한다.

```
In [182]: MAXLAGS = 5

In [183]: model = sm.tsa.AR(values)

In [184]: results = model.fit(MAXLAGS)
```

결과에서 예측된 인자는 intercept를 가지고 그 다음에 두 지연에 대한 예측치를 갖는다.

```
In [185]: results.params
Out[185]:
array([ 0.30290074,  0.78501265, -0.4086365 , -0.01340513,  0.01965317,
        0.00653243])
```

13.4 scikit-learn 소개

scilit=learn은 가장 널리 쓰이는 범용 파이썬 머신러닝 툴킷이다. scikit-learn은 표준적인 지도 학습과 비지도 학습 메서드를 포함하고 있으며 모델 선택, 평가, 데이터 변형, 데이터 적재, 모델 유지 및 기타 작업을 위한 도구들을 제공한다.

온라인에는 실제 문제를 해결하기 위해 scikit-learn과 텐서플로를 적용하는 방법, 머신러닝을 공부할 수 있는 다양한 자료가 존재한다. 이 절에서는 scikit=learn API 스타일을 간략하게 살펴보겠다.

이 책을 쓰는 시점에 scikit=learn 은 pandas와 통합 기능을 제공하지 않으며 일부 서드파티 패키지는 아직 개발이 진행중이다. 하지만 pandas는 모델 피팅 이전 단계에서 데이터셋을 전달하는 데 굉장히 유용하다.

```
In [187]: train = pd.read_csv('datasets/titanic/train.csv')

In [188]: test = pd.read_csv('datasets/titanic/test.csv')

In [189]: train[:4]
Out[189]:
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
```

statsmodels 나 scikit-learn 라이브러리는 일반적으로 누락된 데이터를 처리하지 못하므로 데이터셋에 빠진 값이 있는지 살펴본다.

```
In [190]: train.isnull().sum()
Out[190]:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

In [191]: test.isnull().sum()
Out[191]:
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
```

```
Embarked         0
dtype: int64
```

이와 같은 통계 및 머신러닝 예제에서는 데이터에 기술된 특징에 기반하여 특정 승객이 생존할 것인지 예측하는 것이 일반적인 과제인데, 학습 데이터셋에 모델을 피팅하고 나서 테스트 데이터셋으로 검증하는 식이다.

나이를 기반으로 생존 여부를 예측하고자 하지만 누락 데이터가 존재한다. 결측치를 보완하기 위한 여러 가지 방법이 존재하지만 여기서는 간단히 학습 데이터셋의 중간값을 채워 넣는 것으로 처리하자.

```
In [194]: impute_value = train['Age'].median()

In [195]: train['Age'] = train['Age'].fillna(impute_value)

In [196]: test['Age'] = train['Age'].fillna(impute_value)
```

이제 모델을 명세해야 한다. IsFemale 컬럼을 추가해서 'Sex' 컬럼을 인코딩한다.

```
In [198]: train['IsFemale'] = (train['Sex']=='female').astype(int)

In [199]: test['IsFemale'] = (test['Sex']=='female').astype(int)
```

몇 가지 모델 변수를 선언하고 NumpPy 배열을 생성한다.

```
In [204]: predictors = ['Pclass','IsFemale', 'Age']

In [205]: X_train = train[predictors].values

In [206]: X_test = test[predictors].values

In [207]: y_train = train['Survived'].values

In [208]: X_train[:5]
Out[208]:
array([[ 3.,  0., 22.],
       [ 1.,  1., 38.],
       [ 3.,  1., 26.],
       [ 1.,  1., 35.],
       [ 3.,  0., 35.]])

In [209]: y_train[:5]
Out[209]: array([0, 1, 1, 1, 0], dtype=int64)
```

지금 만든 모델이 좋은 모델이라거나 추출한 특징들이 공학적으로 제대로 선택된 것이라고 주장하지는 않겠다. scikit=learn의 LogisticRegression 모델을 이용해서 model 인스턴스를 생성하자.

```
In [210]: from sklearn.linear_model import LogisticRegression

In [211]: model = LogisticRegression()
```

statsmodels와 유사하게 model의 fit 메서드를 이용하여 이 모델을 학습 데이터에 피팅할 수 있다.

```
In [212]: model.fit(X_train, y_train)
Out[212]:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

model.predict를 이용해서 테스트 데이터셋으로 예측을 해볼 수 있다.

```
In [213]: y_predict = model.predict(X_test)

In [214]: y_predict[:10]
Out[214]: array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0], dtype=int64)
```

테스트 데이터셋의 실제 생존 여부 값을 가지고 있다면 정확도나 기타 오류율을 계산해볼 수 있을 것이다.

```
In [216]: (y_true == y_predict).mean()
```

실제로는 복잡하고 다양한 단계를 거쳐 모델 학습을 진행한다. 많은 모델은 조절할 수 있는 인자를 가지고 학습 데이터에 오버피팅되는 것을 피할 수 있도록 **교차검증** 같은 기법을 활용하기도 한다. 이는 새로운 데이터에 대해 좀 더 견고하거나 예측 성능을 높여주기도 한다.

교차검증은 학습 데이터를 분할하여 예측을 위한 샘플로 활용하는 방식으로 작동한다. 평균제곱오차 같은 모델 정확도 점수에 기반하여 모델 이잔에 대한 그리드 검색을 수행한다. 로지스틱 회귀 같은 모델에서는 교차검증을 내장한 추정 클래스를 제공하기도 한다. 예를 들어, LogisticRegressionCV 클래스는 모델 정규화 인자 C에 대한 그리드 검색을 얼마나 정밀하게 수행할 것인지, 나타내는 인자와 함께 사용할 수 있다.

```
In [217]: from sklearn.linear_model import LogisticRegressionCV

In [218]: model_cv = LogisticRegressionCV(10)

In [219]: model_cv.fit(X_train, y_train)
Out[219]:
LogisticRegressionCV(Cs=10, class_weight=None, cv=None, dual=False,
                     fit_intercept=True, intercept_scaling=1.0, l1_ratios=None,
                     max_iter=100, multi_class='auto', n_jobs=None,
                     penalty='l2', random_state=None, refit=True, scoring=None,
                     solver='lbfgs', tol=0.0001, verbose=0)
```

직접 교차검증을 수행하려면 데이터를 분할하는 과정을 도와주는 cross_val_score 함수를 이용하면 된다. 예를 들어 학습 데이터를 겹치지 않는 4개의 그룹으로 나누려멈ㄴ 아래와 같이 하면 된다.

```
In [220]: from sklearn.model_selection import cross_val_score

In [221]: model = LogisticRegression(C=10)

In [222]: scores = cross_val_score(model, X_train, y_train, cv=4)

In [223]: scores
Out[223]: array([0.77578475, 0.79820628, 0.77578475, 0.78828829])
```

기본 스코어링은 모델에 의존적이지만 명시적으로 스코러링 함수를 선택하는 것도 가능하다. 교차검증된 모델은 학습에 시간이 오래 걸리지만 더 나은 성능을 보여주기도 한다.