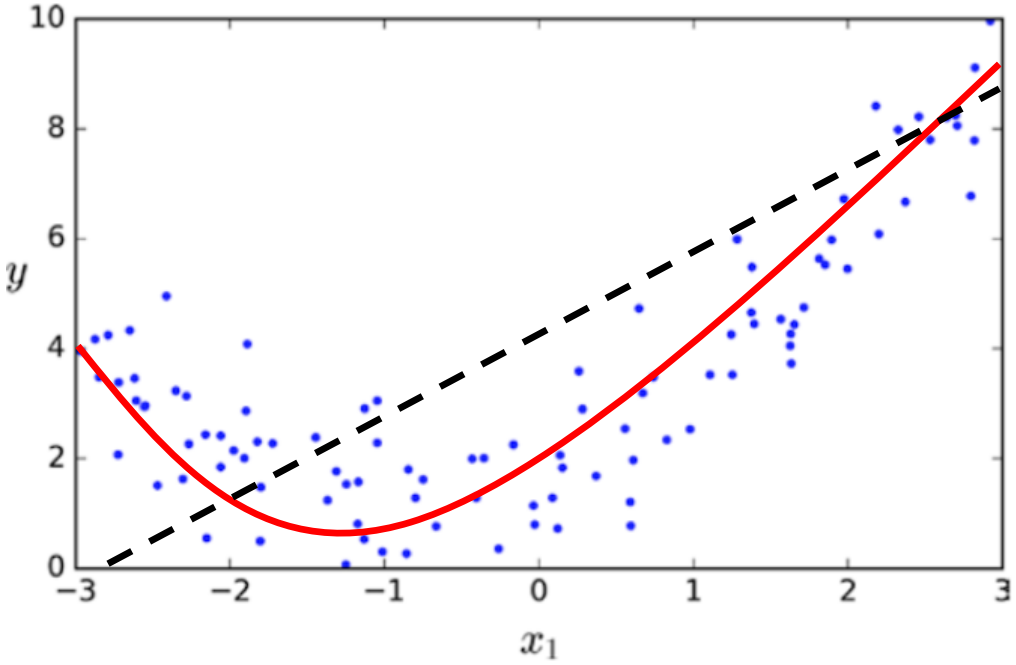
다항 회귀

다항 회귀

- 다항 회귀 (Polynomial Regression)
  - 독립변수와 종속변수들 간의 관계를 다차 다항식으로 표현하는 회귀 방식
  - 예를 들어, 2개의 독립변수 X_1, X_2와 종속변수 Y의 관계를 $Y = w_0 + w_1X_1 + w_2X_2 + w_3X_1X_2 + w_4X_1^2 + w_5X_2^2$ 형태의 다항식으로 표현할 수 있다.



다항 회귀

- 다항 회귀
  - 회귀식의 형태가 독립변수의 선형이 아니기 때문에 비선형 회귀라고 오해할 수 있으나, 다항 회귀는 선형 회귀에 해당 한다.
  - 선형 회귀와 비선형 회귀를 구분하는 기준은 회귀 계수들의 형태가 선형인가 비선형인가의 여부이다. (독립변수의 형태와는 상관 없다.)
  - 보다 엄밀하게, 다항 회귀를 선형화 할 수 있는(linearizable) 회귀라고 명명하기도 한다.

다항 회귀

- 다항 회귀
  - 회귀식의 독립변수들을 각각 새로운 변수로 치환해서 1차 방정식으로 표현할 수 있다.

$$Y = w_0 + w_1X_1 + w_2X_2 + w_3X_1X_2 + w_4X_1^2 + w_5X_2^2$$

$$Y = w_0 + w_1Z_1 + w_2Z_2 + w_3Z_3 + w_4Z_4 + w_5Z_5$$

  - 따라서 다항 회귀는 다중 선형 회귀와 동일한 절차를 통해서 분석을 수행할 수 있다.
  - 단, 비선형 함수를 선형적으로 적용시키는 과정이 선행되어야 한다.

다항 회귀

- 사이킷런에서 다항식 특성으로의 변환 수행

① **preprocessing** 모듈에 있는 **PolynomialFeatures**를 이용하여 특성들을 다항식 형태로 변환하는 객체를 생성한다.

- 매개변수 degree는 변환하려는 다항식의 차수이며, 기본값은 2이다.

```
1  import sklearn.preprocessing as pp
2
3  poly = pp.PolynomialFeatures(degree=2)
```

※ 원본 특성 데이터는 아래와 같이 [[0, 1] [2, 3]] 배열이다.

```
1  import numpy as np
2
3  X = np.arange(4).reshape(2, 2)
```

다항 회귀

- 사이킷런에서 다항식 특성으로의 변환 수행

② 객체에 대해서 **fit_transform** 메소드를 이용하여 변환한다.

- 매개변수는 원본 특성 집합이다.
- 반환 결과는 다항식 형태로 변환된 특성 집합이다.

```
1  X_poly = poly.fit_transform(X)
2  print(X_poly)
```

```
[[1. 0. 1. 0. 0. 1.]
 [1. 2. 3. 4. 6. 9.]]
```

※ fit 및 transform을 각각 순서대로 호출해도 무방하다.

```
1  poly.fit(X)
2  X_poly = poly.transform(X)
3  print(X_poly)
```

```
[[1. 0. 1. 0. 0. 1.]
 [1. 2. 3. 4. 6. 9.]]
```

다항 회귀

- 사이킷런에서 다항식 특성으로의 변환 수행
  - 2차 다항식으로 변환된 결과는 원본 특성에 대해서 0차식, 1차식, 2차식을 순서대로 나열한 것이다.

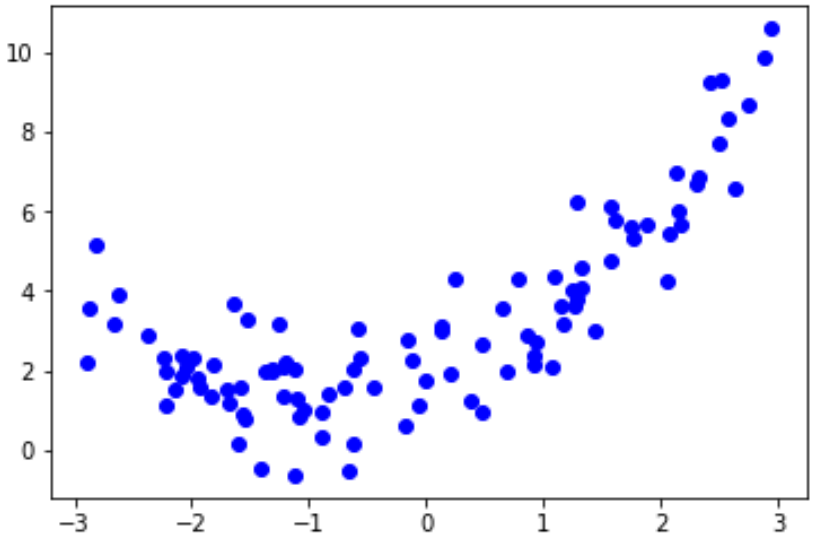
  - 원래의 특성을 2차 다항식으로 변환하는 경우

원본	변환된 결과
[X]	[1, X, X^2]
[X, Y]	[1, X, Y, X^2, XY, Y^2]
[X, Y, Z]	[1, X, Y, Z, X^2, XY, XZ, Y^2, YZ, Z^2]

다항 회귀

- 사이킷런으로 다항 회귀 수행
  - 데이터를 생성하여 확인한다.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  X = 6 * np.random.rand(100, 1) - 3
5  y = 0.5 * X ** 2 + X + 2 + np.random.randn(100, 1)
6
7  plt.scatter(X, y, color="blue")
```



다항 회귀

- 사이킷런으로 다항 회귀 수행
  - 비교를 위해 단순 선형 회귀를 실행한다.

```
1  import sklearn.linear_model as lm
2
3  X_test = np.arange(-3, 3, 0.01)[:, np.newaxis]
4
5  X1_train = X
6  y_train = y
7
8  reg1 = lm.LinearRegression().fit(X1_train, y_train)
9
10 X1_test = X_test
11 y1_pred = reg1.predict(X1_test)
12
13 plt.plot(X_test, y1_pred, color="red", linestyle="--")
```

다항 회귀

- 사이킷런으로 다항 회귀 수행
  - 2차항 형태로 다항 회귀를 실행해 본다.

```
import sklearn.linear_model as lm
import sklearn.preprocessing as pp

X_test = np.arange(-3, 3, 0.01)[:, np.newaxis]

poly = pp.PolynomialFeatures(degree=2)
X2_train = poly.fit_transform(X)
y_train = y

reg2 = lm.LinearRegression().fit(X2_train, y_train)

X2_test = poly.fit_transform(X_test)
y2_pred = reg2.predict(X2_test)

plt.plot(X_test, y2_pred, color="green")
```

다항 회귀

- 사이킷런으로 다항 회귀 수행
  - 7차항 형태로 다항 회귀를 실행해 본다.

```
1  import sklearn.linear_model as lm
2  import sklearn.preprocessing as pp
3
4  X_test = np.arange(-3, 3, 0.01)[:, np.newaxis]
5
6  poly = pp.PolynomialFeatures(degree=7)
7  X7_train = poly.fit_transform(X)
8  y_train = y
9
10 reg7 = lm.LinearRegression().fit(X7_train, y_train)
11
12 X7_test = poly.fit_transform(X_test)
13 y7_pred = reg7.predict(X7_test)
14
15 plt.plot(X_test, y7_pred, color="magenta", linestyle=":")
```

다항 회귀

- 사이킷런으로 다항 회귀 수행
  - 전체적인 결과를 확인한다.