

---

빅데이터 분석처리 과정  
회귀 분석

## 4. 다항회귀

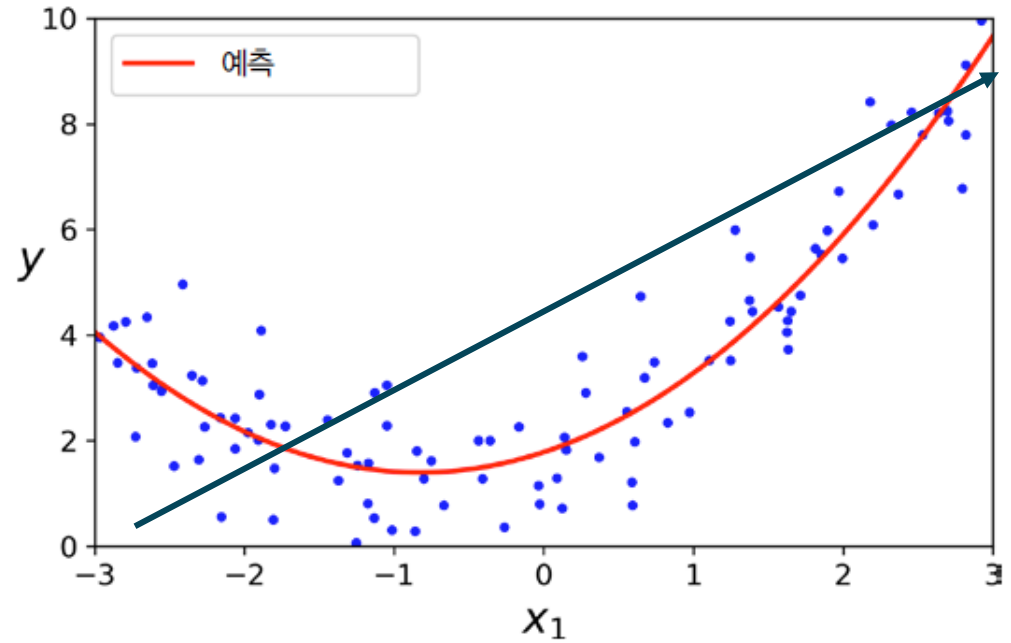
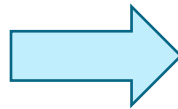
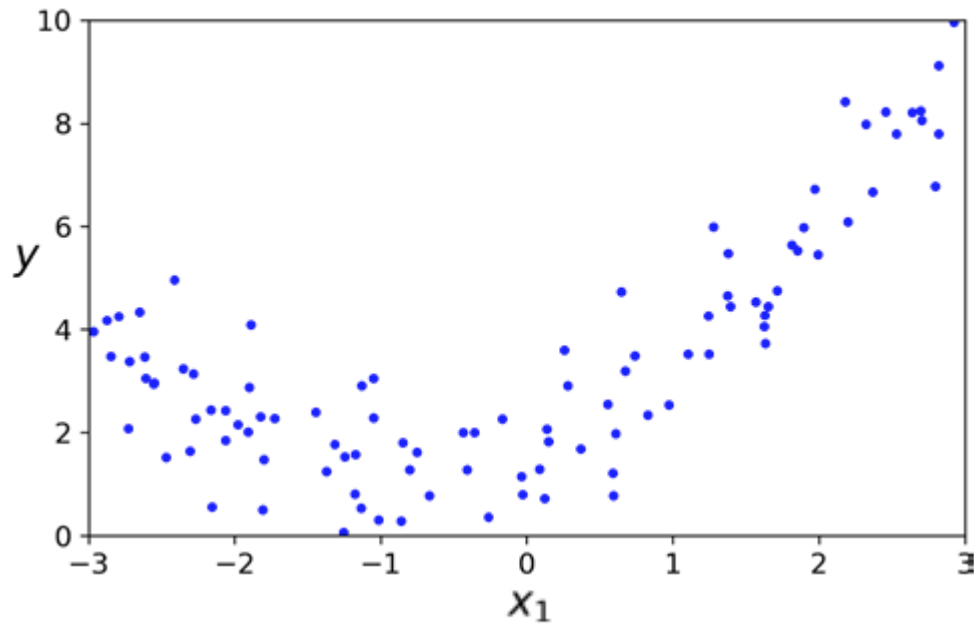
---

## 4. 다항회귀

### Polynomial Regression

- 다항회귀의 개념
- 다항회귀 모델 구현
- 모델 성능 평가

## 다항회귀(Polynomial Regression):



## 본 수업의 목표

---

1. 다항회귀의 개념을 이해하고 구현할 수 있다.
2. 파이프라인의 기능과 효과를 이해하고 구현할 수 있다
3. 데이터를 설명하는 좋은 모델의 기준을 설정할 수 있다

- 다항회귀(Polynomial Regression):

- 독립변수가 단항식이 아닌 2차, 3차 등으로 표현되는 것
- 데이터가 단순한 직선이 아닌 복잡한 형태인 경우, 산점도 상의 관측값을 통과하는 추세선을 그렸을 때 <n-1>개의 굴절이 관찰되면 이를 n차 다항식으로 모델링함
- 독립변수와 종속 변수의 관계를 다차 다항식으로 표현

$$y = \omega_0 + \omega_1 x + \omega_2 x^2 + \omega_3 x^3 + \cdots \omega_n x^n$$

- 다중 선형회귀(여러 개의 독립변수)의 특별한 형태: 회귀식의 독립변수들을 각각 새로운 변수로 치환

$$y = \omega_0 + \omega_1 X_1 + \omega_2 X_2 + \cdots + \omega_n X_n$$

- 특성을 변환(차수의 변환)한 후 다중 선형회귀와 같은 방식으로 비선형 관계를 모델링함
- 차수가 증가하면 곡선 모델(**비선형모델**)이 될 때 다항회귀라고 함
- **장점** : 두 입력변수 사이의 관계성을 설명해서 데이터를 잘 설명한다.

# 다항회귀 종류

---

- 단일 속성 다항회귀 VS 다중속성 다항회귀

- 단일속성 다항회귀: 독립 변수가 1개인 경우, 한 개의 독립변수로 한 개의 종속변수 예측

$$y = \omega_0 + \omega_1 x + \omega_2 x^2 + \omega_3 x^3 + \cdots \omega_n x^n$$

- 다중속성 다항회귀: 독립 변수 여러 개인 경우, 여러 개의 독립변수로 한 개의 종속변수 예측

$$f(X, Y) = \omega_0 + \omega_1 X + \omega_2 Y + \omega_3 XY + \omega_4 X^2 + \omega_5 Y^2$$

- 사이킷런에서 다항식 특성으로 변환: 차수<sup>degree</sup> VS. 속성의 개수  
2차 다항식으로 변환된 결과는 원본 특성에 대하여 0차식, 1차식, 2차식을 순서대로 나열

속성의 개수	degree = 2	degree = 3
1개[X]	[1, X, X <sup>2</sup> ]	[1, X, X <sup>2</sup> , X <sup>3</sup> ]
2개[X, Y]	[1, X, Y, X <sup>2</sup> , XY, Y <sup>2</sup> ]	[1, X, Y, X <sup>2</sup> , Y <sup>2</sup> , XY, X <sup>3</sup> , Y <sup>3</sup> , X <sup>2</sup> Y, XY <sup>2</sup> ]
3개[X, Y, Z]	[1, X, Y, Z, X <sup>2</sup> , XY, XZ, Y <sup>2</sup> , YZ, Z <sup>2</sup> ]	[1, X, Y, Z, X <sup>2</sup> , Y <sup>2</sup> , Z <sup>2</sup> , XY, XZ, XY, X <sup>3</sup> , Y <sup>3</sup> , Z <sup>3</sup> , X <sup>2</sup> Y, X <sup>2</sup> Z, Y <sup>2</sup> X, Y <sup>2</sup> Z, Z <sup>2</sup> X, Z <sup>2</sup> Y, XYZ]

차수가 높아질 수록 새로운 속성이 기하급수적으로 늘어남- 계산 시간이 오래걸리고 과적합<sup>Overfitting</sup> 문제가 발생  
특성: n개, 차수: d 인 경우 ->  $\frac{(n+d)!}{d!n!}$  특성 수로 늘어남

## 사이킷런의 변환기<sup>transformer</sup>

---

변환기<sup>transformer</sup>: 특성을 만들거나 전처리하기 위한 다양한 클래스를 제공

다항회귀를 위한 변환기:

```
from sklearn.preprocessing import PolynomialFeatures
```

사이킷런 변환기의 메소드

`fit()`: 새롭게 만들 특성의 조합(기준)을 찾음

`transform()`: 실제로 데이터 변환

`fit_transform()`



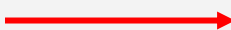
## 다항회귀 Polynomial Regression > 머신러닝 구현

- 사이킷런에서 다항식 특성으로 변환하는 방법
- 예1) 원본 특성 데이터: 1개인 경우

```
# 특성이 1개인 경우 - [2] 변환 연습
# 2차식으로 변환
import numpy as np
X = np.array([2])

# 사이킷런 입력 형태인 2차원 데이터로 변환
X = X.reshape(-1, 1)
print(X)
```

- preprocessing 모듈의 PolynomialFeatures를 이용하여 특성을 조합하여 다항식 형태로 변환하는 객체 생성

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)  degree: 변환하려는 다항식의 차수, 기본값은 2
poly.fit(X)
X_poly = poly.transform(X)
print(X_poly)

# 어떤 항(특성)이 곱해지는지 표시
poly.get_feature_names_out()
```

## 다항회귀 Polynomial Regression > 머신러닝 구현

- 사이킷런에서 다항식 특성으로 변환하는 방법
- 예 2) 원본 특성 데이터: 2개인 경우

```
import numpy as np
```

```
X = np.arange(4).reshape(2,2)  
print(X)
```

```
[[0 1]  
 [2 3]]
```

- preprocessing 모듈의 PolynomialFeatures를 이용하여 특성을 조합하여 다항식 형태로 변환하는 객체 생성

```
from sklearn.preprocessing import PolynomialFeatures  
  
poly = PolynomialFeatures(degree=2)
```

degree: 변환하려는 다항식의 차수, 기본값은 2

## 다항회귀 Polynomial Regression > 머신러닝 구현

- 사이킷런에서 다항식 특성으로 변환하는 방법

2) 객체에 대하여 fit(), transform() 메소드를 이용하여 특성 변환

```
poly.fit(X)
X_poly = poly.transform(X)
print(X_poly)
```

← 매개변수: 원본 특성 데이터

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

← 반환 결과: 다항식 형태로 변환된 특성 데이터

2-1) fit\_transform() 메소드를 이용하여도 특성 변환 가능

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

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

```
# 어떤 항(특성)이 곱해지는지 알고 싶을 때
poly.get_feature_names_out()
```

```
['1', 'x0', 'x1', 'x0^2', 'x0 x1', 'x1^2']
```

- 사이킷런에서 다항식 특성으로 변환: 차수<sup>degree</sup> VS. 속성의 개수  
2차 다항식으로 변환된 결과는 원본 특성에 대하여 0차식, 1차식, 2차식을 순서대로 나열

속성의 개수	degree = 2	degree = 3
1개[X]	[1, X, X <sup>2</sup> ]	[1, X, X <sup>2</sup> , X <sup>3</sup> ]
2개[X, Y]	[1, X, Y, X <sup>2</sup> , XY, Y <sup>2</sup> ]	[1, X, Y, X <sup>2</sup> , Y <sup>2</sup> , XY, X <sup>3</sup> , Y <sup>3</sup> , X <sup>2</sup> Y, XY <sup>2</sup> ]
3개[X, Y, Z]	[1, X, Y, Z, X <sup>2</sup> , XY, XZ, Y <sup>2</sup> , YZ, Z <sup>2</sup> ]	[1, X, Y, Z, X <sup>2</sup> , Y <sup>2</sup> , Z <sup>2</sup> , XY, XZ, XY, X <sup>3</sup> , Y <sup>3</sup> , Z <sup>3</sup> , X <sup>2</sup> Y, X <sup>2</sup> Z, Y <sup>2</sup> X, Y <sup>2</sup> Z, Z <sup>2</sup> X, Z <sup>2</sup> Y, XYZ]

차수가 높아질 수록 새로운 속성이 기하급수적으로 늘어남- 계산 시간이 오래걸리고 과적합<sup>Overfitting</sup> 문제가 발생  
특성: n개, 차수: d 인 경우 ->  $\frac{(n+d)!}{d!n!}$  특성 수로 늘어남

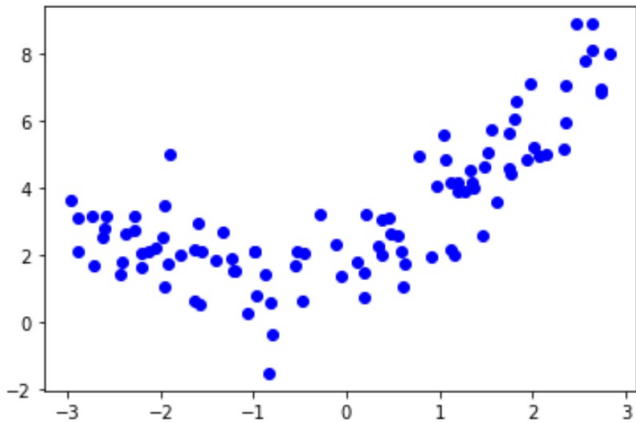
# 선형 회귀 유형 정리

선형회귀	단일	다중
단항	$y = w_0 + w_1x$	$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n$
다항	$y = w_0 + w_1x + w_2x^2 + w_3x^3 + \dots + w_nx^n$	$y(x_1, x_2, \dots, x_n) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + \dots$

✓ 손실함수, 경사 하강법 모두 동일

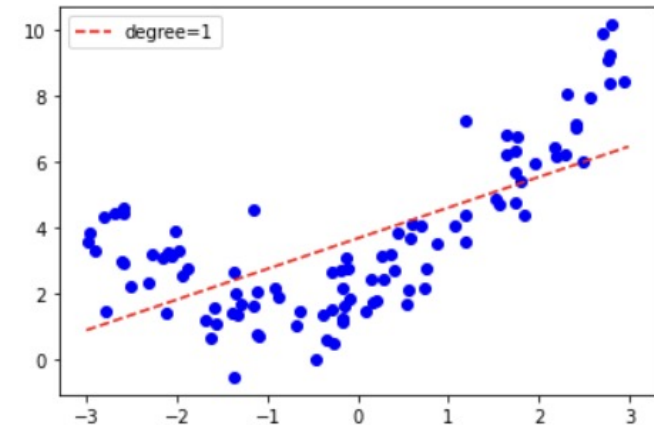
## 1. 데이터 준비: 데이터 생성 및 탐색

```
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')
8 plt.show()
```



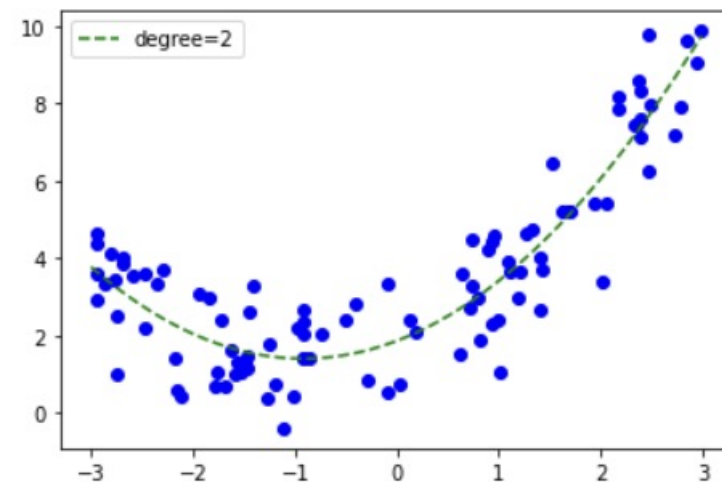
## 2. 비교를 위한 단순 선형 회귀 수행

```
1 from sklearn.linear_model import LinearRegression
2
3 X1_train = X
4 y_train = y
5
6 reg1 = LinearRegression().fit(X1_train, y_train)
7
8 xx = np.arange(-3, 3, 0.01)[: , np.newaxis]
9 yy = reg1.predict(xx)
10 plt.plot(xx, yy, color='r', linestyle='--', label='degree=1')
11 plt.scatter(X, y, color='b')
12 plt.legend()
13
14 plt.show()
```



### 3. 2차항 형태의 다항회귀 수행

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.preprocessing import PolynomialFeatures
3
4 poly = PolynomialFeatures(degree=2)
5
6 X2_train = poly.fit_transform(X)
7 y_train = y
8
9 reg2 = LinearRegression().fit(X2_train, y_train)
10
11 xx = np.arange(-3, 3, 0.01)
12 yy = reg2.predict(poly.transform(xx[:, np.newaxis]))
13
14 plt.plot(xx, yy, color='g', linestyle='--', label='degree=2')
15 plt.scatter(X, y, color='b')
16 plt.legend()
17 plt.show()
```



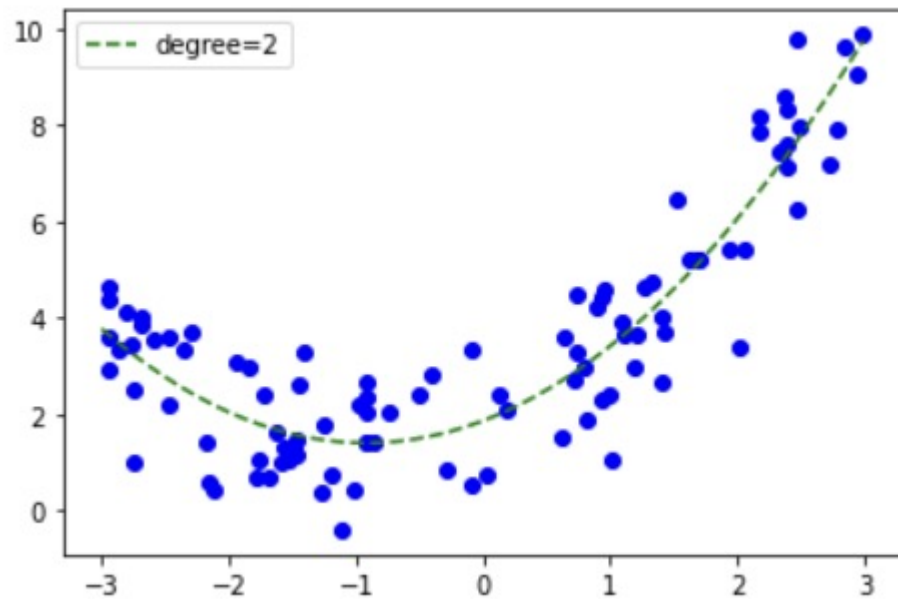


- 학습한 모델의 계수 확인

```
1 # 구해진 2차식의 계수 확인
```

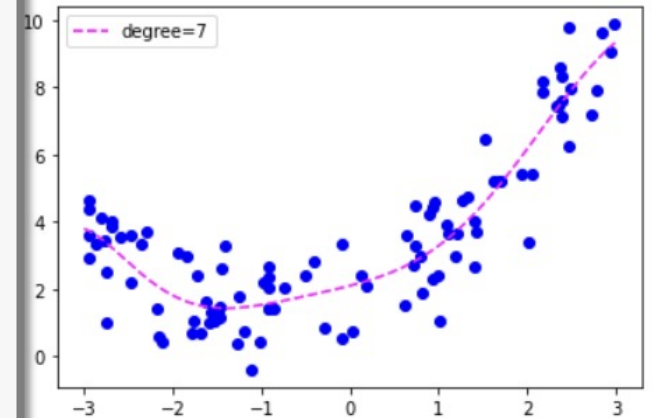
```
1 reg2.intercept_, reg2.coef_
```

```
(array([1.85820109]), array([[0.          , 1.00377512, 0.54676586]]))
```



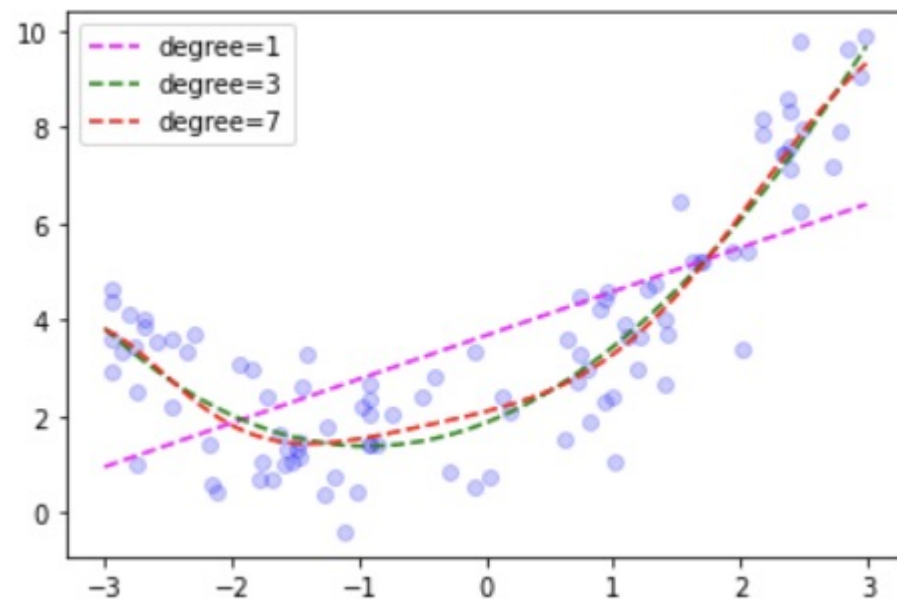
## 4. 7차항 형태의 다항회귀 수행

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.preprocessing import PolynomialFeatures
3
4 poly = PolynomialFeatures(degree=7)
5
6 X7_train = poly.fit_transform(X)
7 y_train = y
8
9 reg7 = LinearRegression().fit(X7_train, y_train)
10
11 xx = np.arange(-3, 3, 0.01)
12 yy = reg7.predict(poly.transform(xx[:, np.newaxis]))
13
14 plt.plot(xx, yy, color='magenta', linestyle='--', label='degree=7')
15 plt.scatter(X, y, color='b')
16 plt.legend()
17 plt.show()
```



## 5. 여러 형태의 시각화 그래프 비교

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.preprocessing import PolynomialFeatures
3
4 degree = [1, 3, 7]
5 colors = ['red', 'green', 'magenta']
6
7 plt.scatter(X, y, color='b', alpha=0.2)
8 for i, d in enumerate(degree):
9     poly = PolynomialFeatures(degree=d)
10    X_train = poly.fit_transform(X)
11    y_train = y
12
13    reg = LinearRegression().fit(X_train, y_train)
14    xx = np.arange(-3, 3, 0.01)
15    yy = reg.predict(poly.transform(xx[:, np.newaxis]))
16
17    plt.plot(xx, yy, color=colors[i], linestyle='--', \
18             label='degree={}'.format(d))
19    plt.legend()
20 plt.show()
```



# Pipeline

---

- 기능: 대부분의 데이터가 학습에 적용할 수 있는 이상적인 상태가 아니므로 sklearn.pipeline 모듈을 사용해서 **데이터를 변환하는 과정을 단순화**할 수 있다.
- 데이터의 사전 처리와 모델링 연결시켜 단일 객체를 만들 수 있다.
- 교차 검증 및 기타 모델 클래스 선택에 도움이 된다.
- 변환기(Transformer)와 추정기(Estimator)를 연결하여 모델에 대한 이해를 높일 수 있다.
- 데이터 준비 단계를 자동화할수록 더 많은 조합을 자동으로 시동해볼 수 있고 최상의 조합을 찾을 가능성을 높이며 시간을 절약할 수 있다.

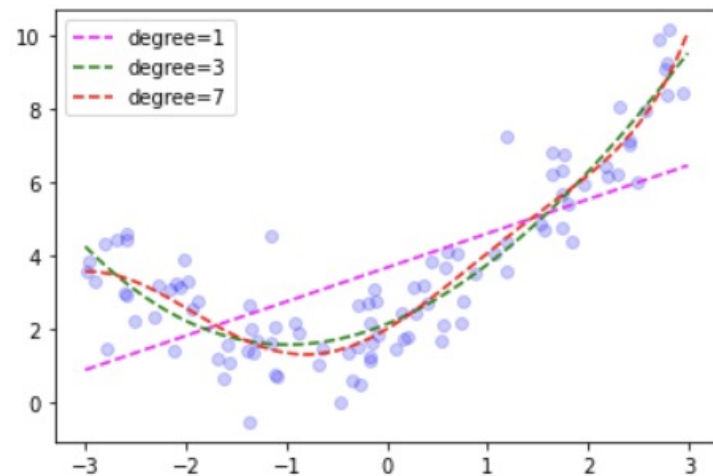
## pipeline을 활용한 다항회귀

---

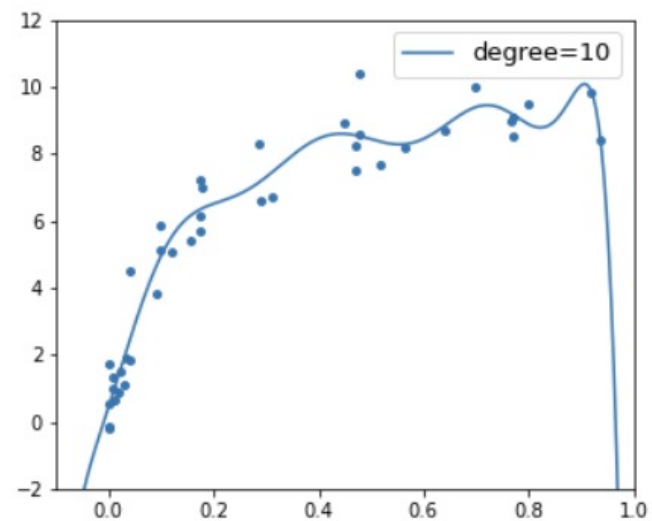
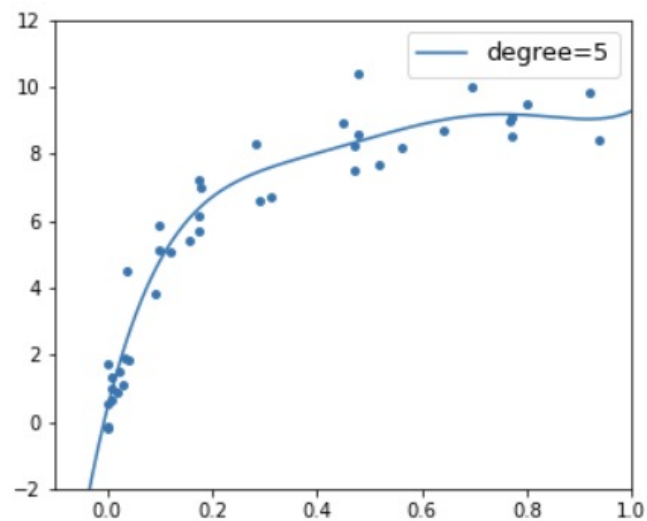
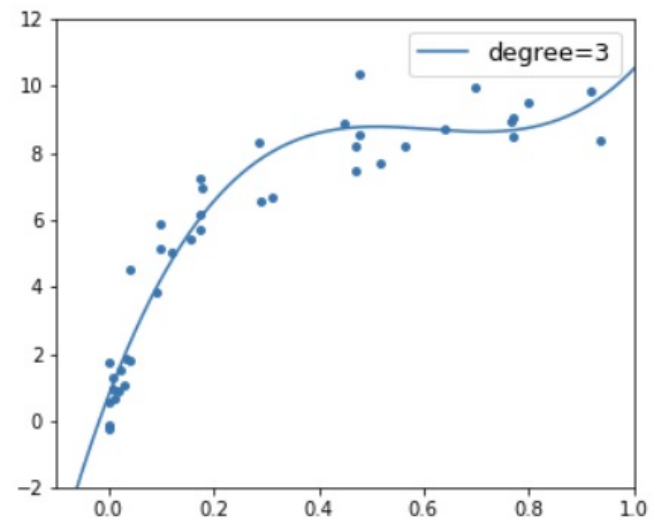
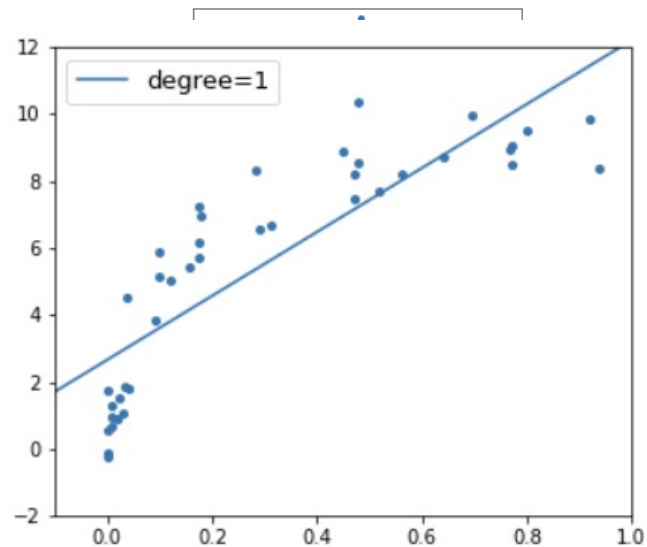
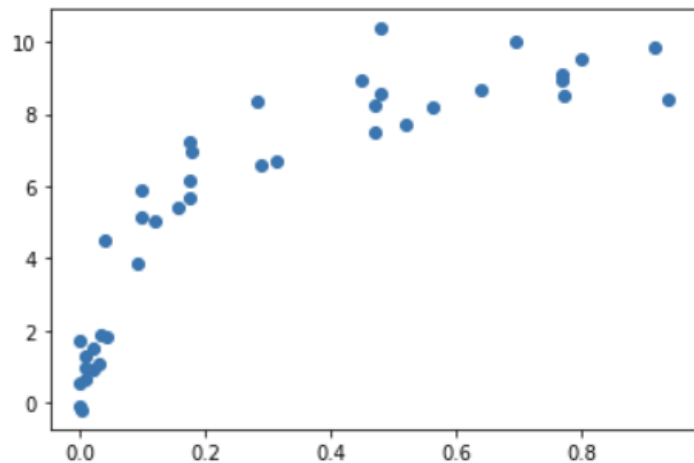
```
1 # 1) degree=2인 모델
2 from sklearn.pipeline import make_pipeline
3
4 reg = make_pipeline(PolynomialFeatures(degree=2),\
5                     LinearRegression()).fit(X, y)
6 xx = np.arange(-3, 3, 0.01)[: , np.newaxis]
7 yy = reg.predict(xx)
8
9 plt.plot(xx, yy, color='r', linestyle='--', label='degree=2')
10 plt.scatter(X, y, color='b', alpha=0.2)
11 plt.legend()
12 plt.show()
```

## pipeline을 활용한 다항회귀

```
1 # 2) degree = [1, 3, 7]인 모델을 파이프라인으로 변환- 추정 통합
2
3 from sklearn.linear_model import LinearRegression
4 from sklearn.preprocessing import PolynomialFeatures
5
6 degree = [1, 3, 7]
7 colors = ['magenta', 'green', 'red']
8
9 plt.scatter(X, y, color='b', alpha=0.2)
10 for i, d in enumerate(degree):
11     reg = make_pipeline(PolynomialFeatures(degree=d), \
12                        LinearRegression()).fit(X, y)
13     xx = np.arange(-3, 3, 0.01)[: , np.newaxis]
14     yy = reg.predict(xx)
15
16     plt.plot(xx, yy, color=colors[i], linestyle='--', \
17             label='degree={}'.format(d))
18 plt.legend()
19 plt.show()
```

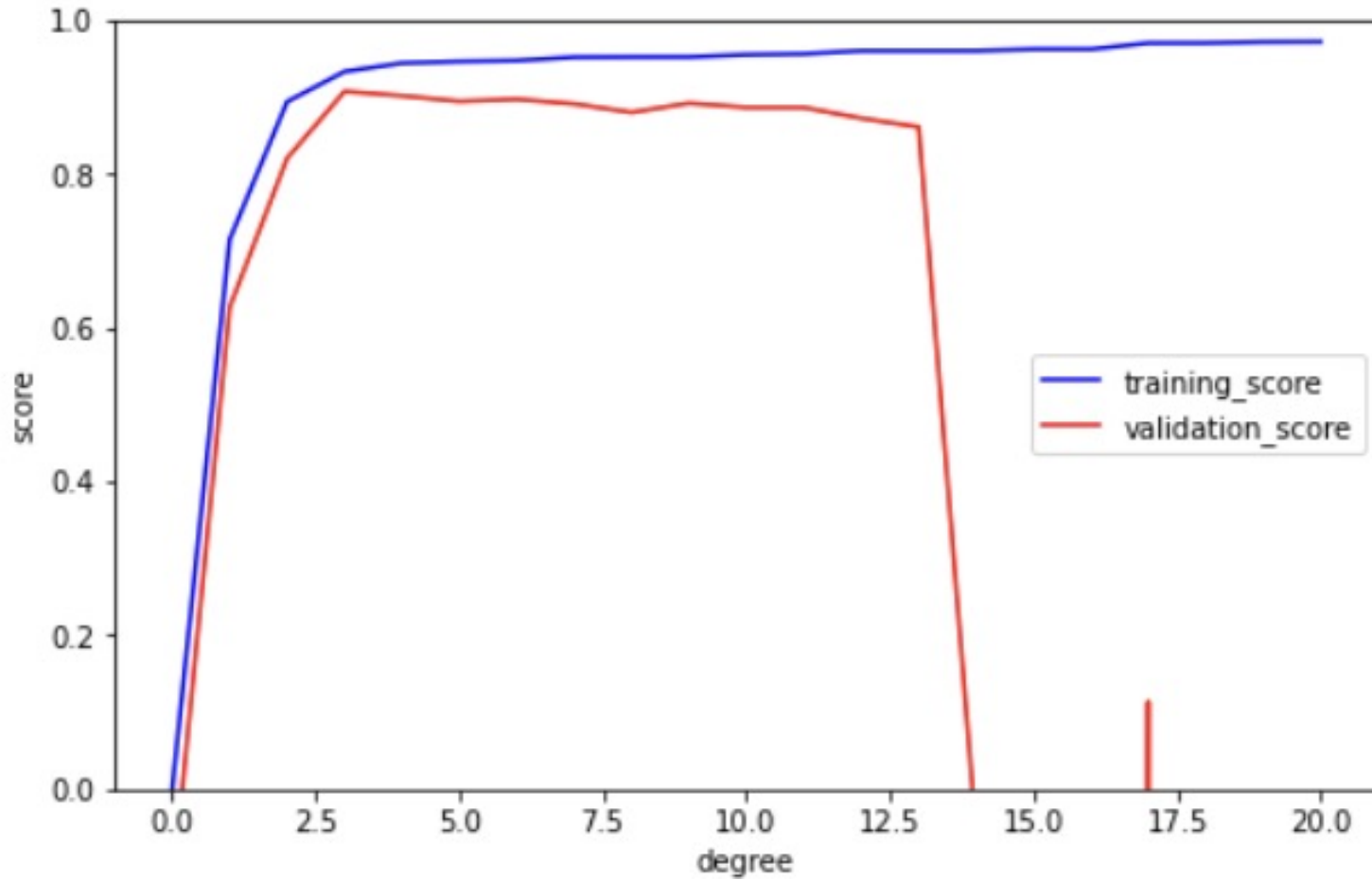


# 좋은 모델이란?



## 좋은 모델이란?

### ■ 차수별 학습 vs 검증 곡선 점수





## [응용1] 보스턴 집값 VS 방의 갯수

-보스턴 집값 데이터셋을 활용하여 방의 갯수 변수의 특성을 2차, 3차로 늘리고, 각각의 회귀 계수를 구하시오.

조건:

```
# 학습용, 검증용 데이터 분할  
  
X_train, X_test, y_train, y_test =\  
train_test_split(X, y, test_size=0.3, random_state=1)
```

결과:

	reg.coef_	reg.intercept_
원본 특성 데이터 (degree=1)	[8.46109164]	-30.571032410898336
degree=2	[ 0. -23.79086549  2.54284549]	70.45880734716539
degree=3	[ 0. -121.27686495  18.59069233 - 0.86283235]	263.66230448020355

## [응용2] 당뇨병 진행도 VS bmi

-당뇨병 데이터셋의 모든 특성을 조합해 2차, 3차 다항회귀식을 만들고 각 차수에 해당하는 특성의 수와 r2값을 비교해보시오(당뇨병 진행도 VS 모든 특성)

조건:

```
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=1)
```

결과:

	r2	특성 수
원본 특성 데이터 (degree=1)	0.438	10
degree=2	0.211	66
degree=3	0.167	286

### [응용3] 농어 무게 예측 - 다중다항회귀

농어의 길이, 두께, 폭을 특성으로 하고 무게를 예측하는 다중 다항 회귀분석을 하되, 1차, 2차, 3차 다항회귀를 각각 수행하라.

각 차수에 해당하는 특성의 수와  $r^2$ 값을 비교하라

`test_size=0.3`

`random_state=31`

## [응용4] 파이프라인

---

1. 보스턴 집값 예측 예제에서 각각의 차수와 추정기를 연결하는 파이프라인 모델을 구현하라
2. 농어 무게 예측 예제에서 스케일러, 각각의 차수, 추정기를 연결하는 파이프라인 모델을 구현하라

