



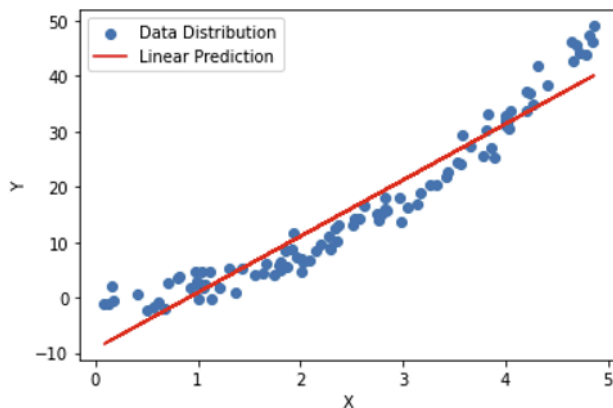
✓ 3. Polynomial Regression(다항 회귀)

독립변수들과 종속변수가 2차 함수 이상의 관계를 가지는 것으로 따라서 곡선이나 좀 더 복잡한 모양을 표현하는 것이 가능하다. 다만 다항회귀 역시 선형회귀로 간주되는 데 선형회귀의 정의가 단순히 직선이나 곡선과 같은 모양에 있는 것이 아니라 가중치와 독립변수의 선형결합 여부에 있기 때문이다. 아래 다항 회귀식에서 거듭제곱인 독립변수들을 $X_1, X_2, X_3..$ 과 같은 변수로 대체하면 다중 선형회귀식과 같아지는 것을 알 수 있다.

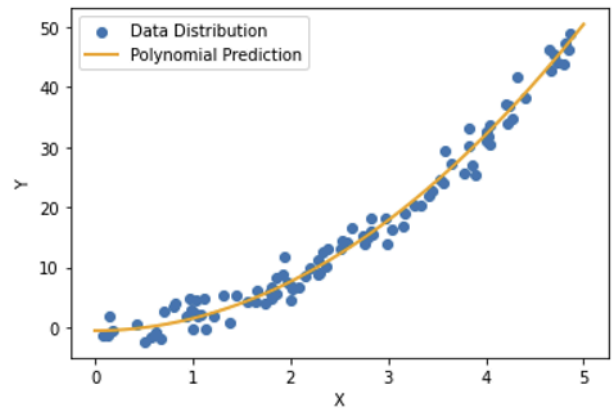
$$y = w_0 + w_1x + w_2x^2 + \dots + w_dx^d$$

다항회귀는 별도의 알고리즘이 있는 것이 아니라 독립변수를 제공, 서로 곱한 값 등 좀 더 복잡한 값으로 만들어 선형회귀에 넣어 학습시키는 것을 말한다. 한 개의 독립변수와 종속변수가 아래와 같은 비선형 모양을 띤다면 단순히 독립변수를 선형회귀에 넣어 학습시킨 것보다 독립변수를 제공해서 선형회귀에 입력하면 성능이 향상되는 것은 볼 수 있다.

Train R2 0.9245336879294013
Test R2 0.9131813763544481



Train R2 0.9846395067796223
Test R2 0.9769579169852202



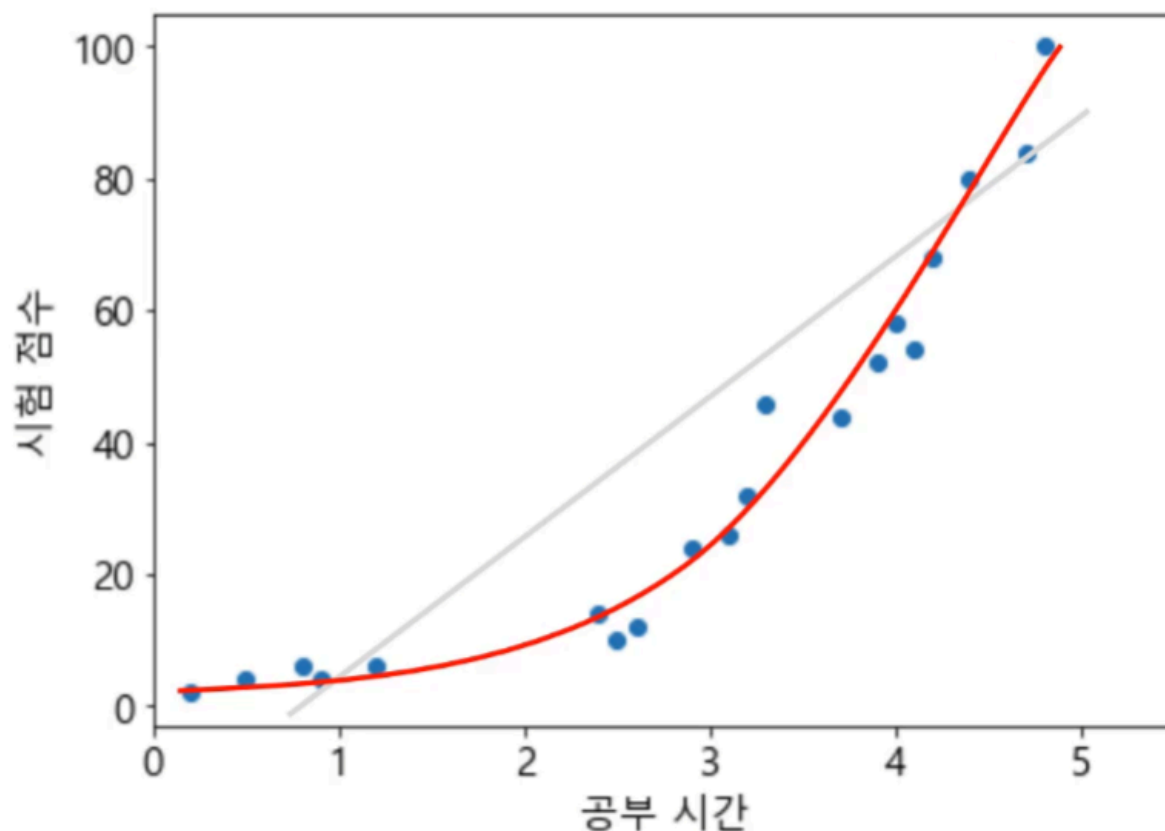
X 를 넣은 경우 vs. X^2 로 해서 넣은 경우

다항회귀 적용하기

독립변수와 종속변수가 곡선 또는 비선형 관계를 보인다면 변수들을 2차항 이상의 값으로 변환해 선형회귀의 성능 향상을 시도해 볼 수 있을 것이다.

앞에서 사용한 공부시간 예측(Housing Prices) 데이터셋을 가지고 다항회귀를 어떻게 사용할 수 있는지 살펴보도록 하자. 이번에는 우등생 데이터셋으로 가정해 보면

데이터를 가장 잘 표현하는 선?



[다항회귀를 경험할수 있는 사이트](#)

✓ 공부 시간에 따른 시험 점수 (우등생)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# 머신러닝에서 많이 사용
```

```
from google.colab import drive # 구글드라이버에 있는 파일 사용하기
drive.mount('/content/drive')
```

↔ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c

```
dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/활용편7-머신러닝/ScikitLearn/Polynom
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

✓ 3-1. 단순 선형 회귀 (Simple Linear Regression)

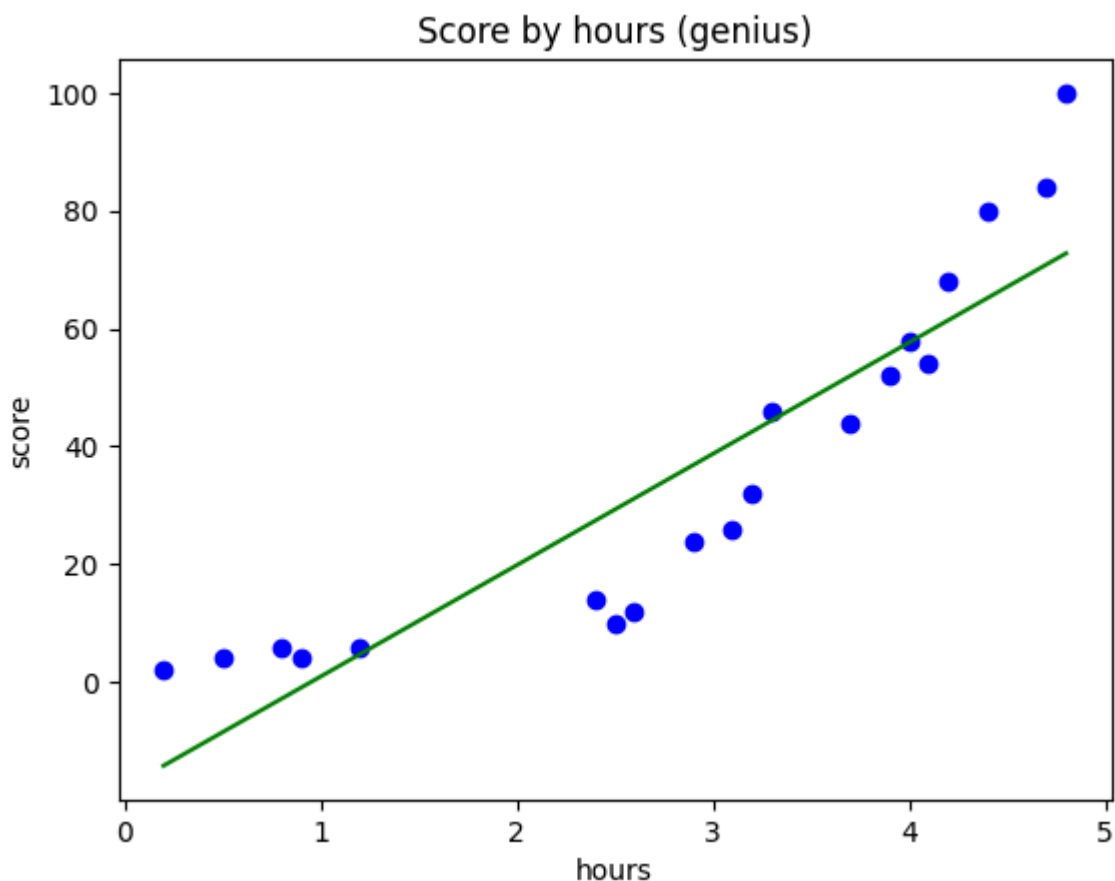
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X, y) # 전체 데이터로 학습
```



▼ LinearRegression
LinearRegression()

✓ 데이터 시각화 (전체)

```
plt.scatter(X, y, color='blue') # 산점도
plt.plot(X, reg.predict(X), color='green') # 선 그래프
plt.title('Score by hours (genius)') # 제목
plt.xlabel('hours') # X 축 이름
plt.ylabel('score') # Y 축 이름
plt.show()
```



```
reg.score(X, y) # 전체 데이터를 통한 모델 평가 81.7점
```



0.8169296513411765

✓ 3-2. 다항 회귀 (Polynomial Regression)

```
from sklearn.preprocessing import PolynomialFeatures # 현재 데이터를 다항식 형태로 변경하는 함수
poly_reg = PolynomialFeatures(degree=2) # 2차 다항식으로(다항회귀)
X_poly = poly_reg.fit_transform(X) # poly_reg.fit(), poly_reg.transform() 이렇게 따로 호출할수 도
X_poly[:5] # [x] -> [x^0, x^1, x^2] -> x 가 30이라면 [1, 3, 9] 으로 변환
```

```
⇒ array([[1. , 0.2 , 0.04],
        [1. , 0.5 , 0.25],
        [1. , 0.8 , 0.64],
        [1. , 0.9 , 0.81],
        [1. , 1.2 , 1.44]])
```

X[:5] # 원본 데이터

```
⇒ array([[0.2],
        [0.5],
        [0.8],
        [0.9],
        [1.2]])
```

poly_reg.get_feature_names_out() # [x^0, x^1, x^2]

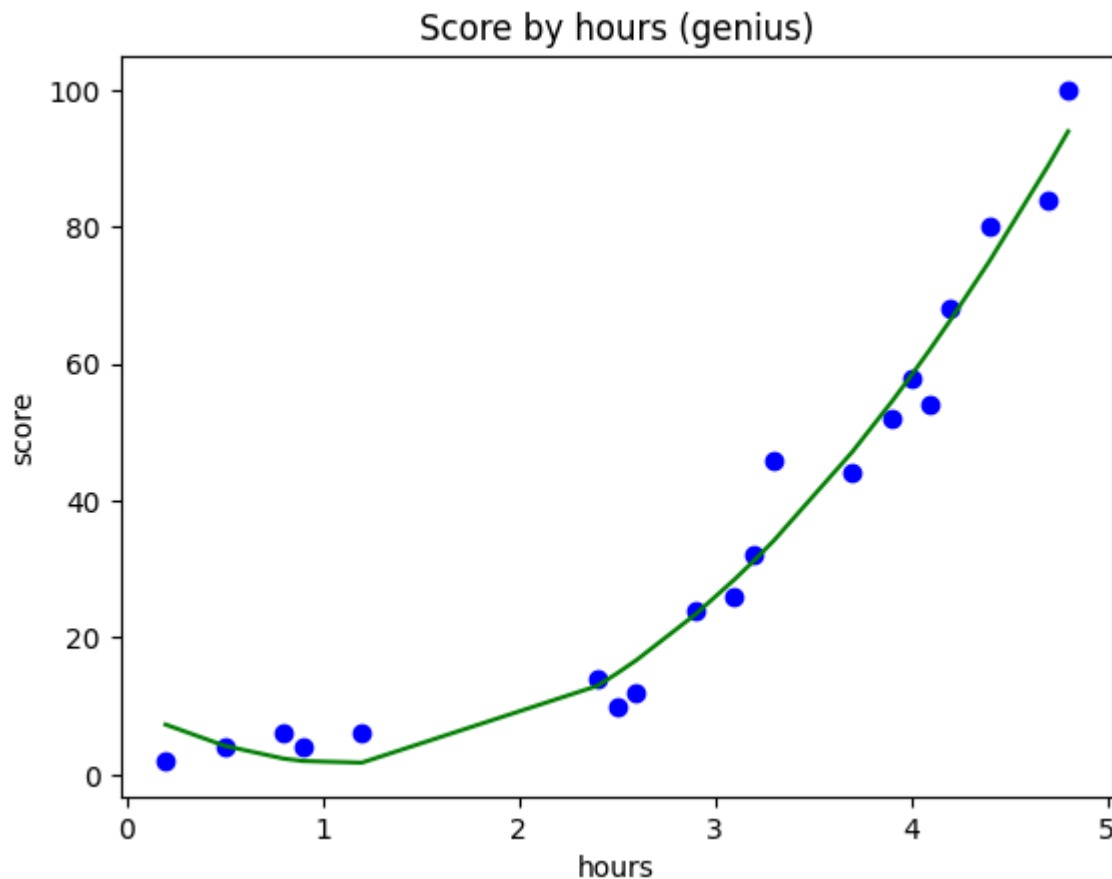
```
⇒ array(['1', 'x0', 'x0^2'], dtype=object)
```

```
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y) # 변환된 X 와 y 를 가지고 모델 생성 (학습)
```

```
⇒ ▾ LinearRegression
   LinearRegression()
```

✓ 데이터 시각화 (변환된 X 와 y)

```
plt.scatter(X, y, color='blue')
plt.plot(X, lin_reg.predict(poly_reg.fit_transform(X)), color='green') # predict : 예측하는 함수,
plt.title('Score by hours (genius)') # 제목
plt.xlabel('hours') # X 축 이름
plt.ylabel('score') # Y 축 이름
plt.show()
```



위에 그래프를 좀더 부드럽게 하기위해 넘파이를 이용해 데이터 생성

```
X_range = np.arange(np.min(X), np.max(X), 0.1) # X 의 최소값에서 최대값까지의 범위를 0.1 단위로 잘
X_range
```

```
array([0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2, 1.3, 1.4,
       1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7,
       2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4. ,
       4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7])
```

X_range.shape # X_range 컬럼 개수 46 1차원 배열

```
(46,)
```

X[:5] # 원래 데이터 5개 의 2차원 배열임.

```
array([[0.2],
       [0.5],
       [0.8],
       [0.9],
       [1.2]])
```

X.shape # 원래 데이터 컬럼 개수 20이고 2차원 배열

```
(20, 1)
```

```
X_range = X_range.reshape(-1, 1) # -1은 row 개수는 자동으로 계산, column 개수는 1개인 2차원 배열로
X_range.shape
```

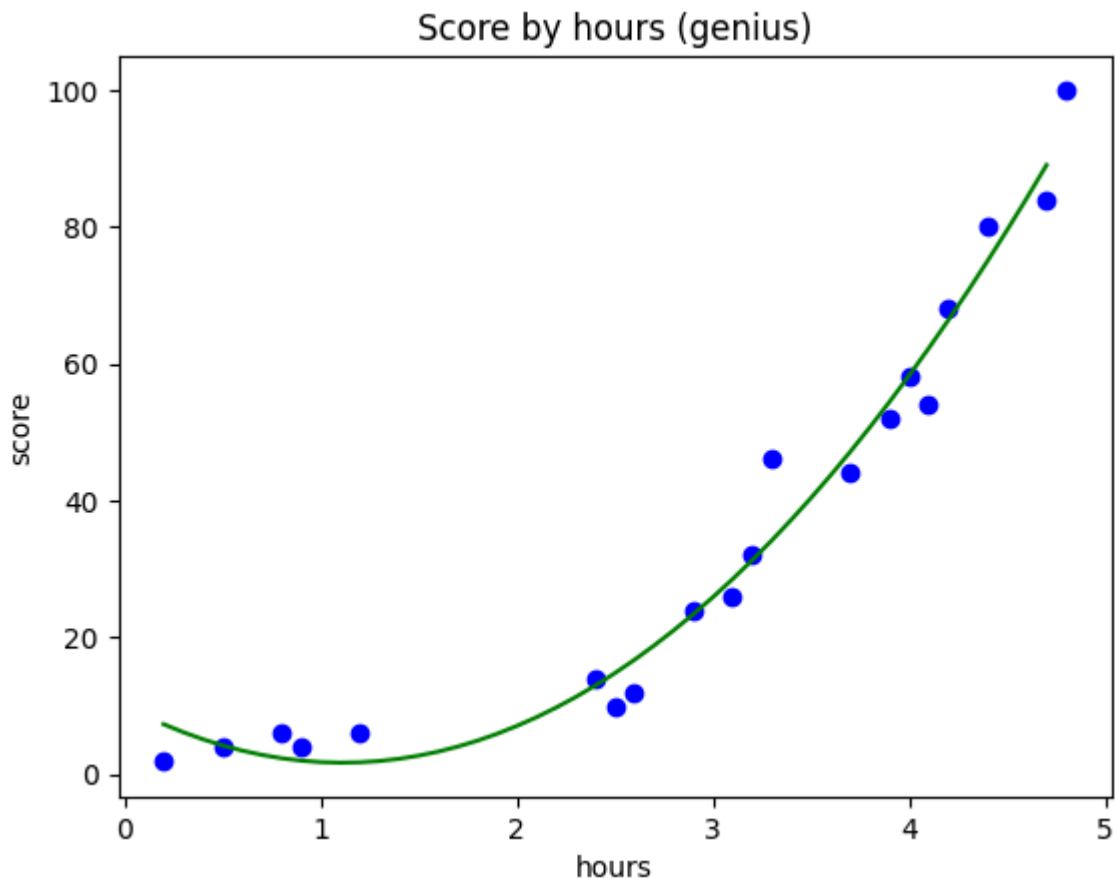
↗ (46, 1)

`X_range[:5]` # 2차원 배열로 변환된 것을 알수 있음.

↗ `array([[0.2],
[0.3],
[0.4],
[0.5],
[0.6]])`

```
plt.scatter(X, y, color='blue')
plt.plot(X_range, lin_reg.predict(poly_reg.fit_transform(X_range)), color='green')
plt.title('Score by hours (genius)') # 제목
plt.xlabel('hours') # X 축 이름
plt.ylabel('score') # Y 축 이름
plt.show() # 선이 더 부드러운 그래프가 생성됨.
# 4차로 바꿔서 해보면 좀더 정확한 그래프가 됨.
```

↗




✓ 공부 시간에 따른 시험 성적 예측


`reg.predict([[2]])` # 2시간을 공부했을 때 선형 회귀 모델의 예측

↗ `array([19.85348988])`

`lin_reg.predict(poly_reg.fit_transform([[2]]))` # 2시간을 공부했을 때 다항 회귀 모델의 예측. `poly_r`

 `array([7.05092142])`

```
lin_reg.score(X_poly, y) # 다항 회귀에서의 점수
```

 `0.9755457185555199`

차원을 너무 높이면 과대 적합이 오므로 훈련데이터에만 너무 치중해서 훈련데이터로 확인했을 때는 점수가 굉장히 잘 나오지만 일반적인 데이터를 넣었을 때는 점수가 형편없이 나오는 과대 적합이 오거나 학습이 너무 덜 되어서 훈련데이터조차 점수가 낮게 나오는 과소 접합을 주의해야 한다.