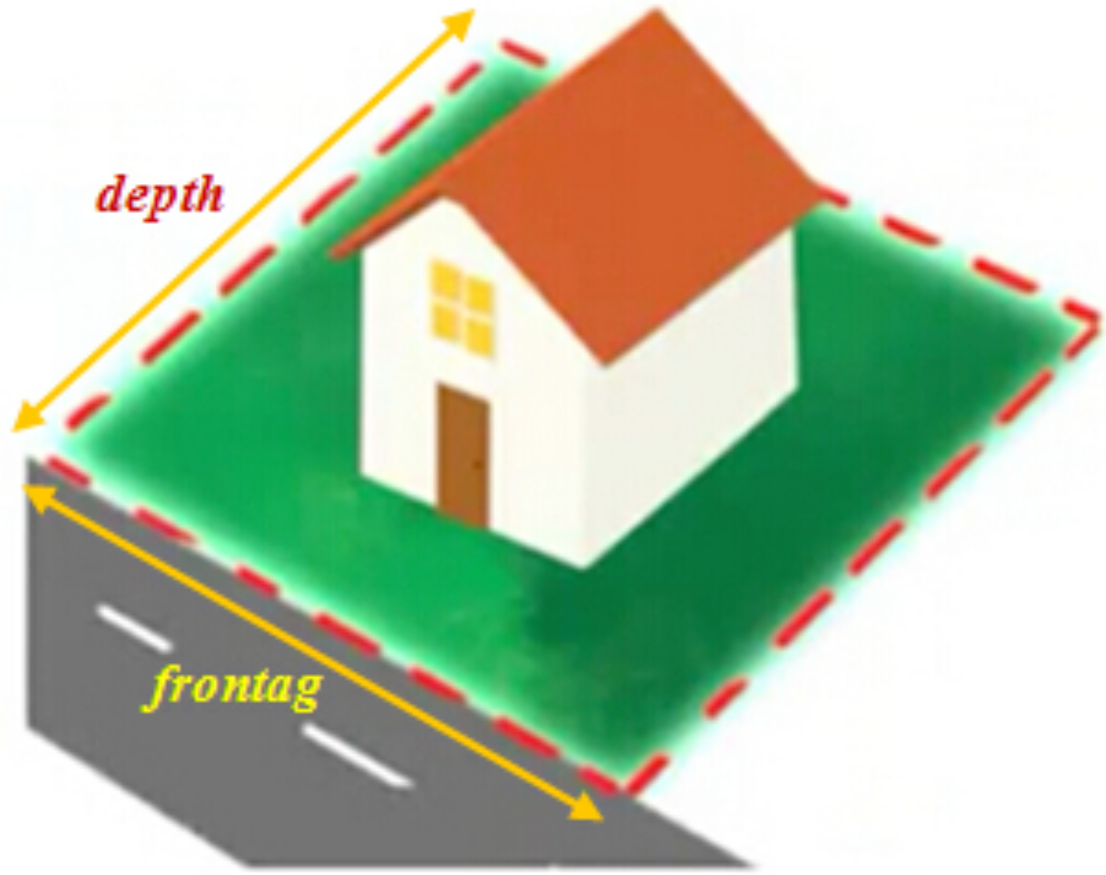
Polynomial Regression

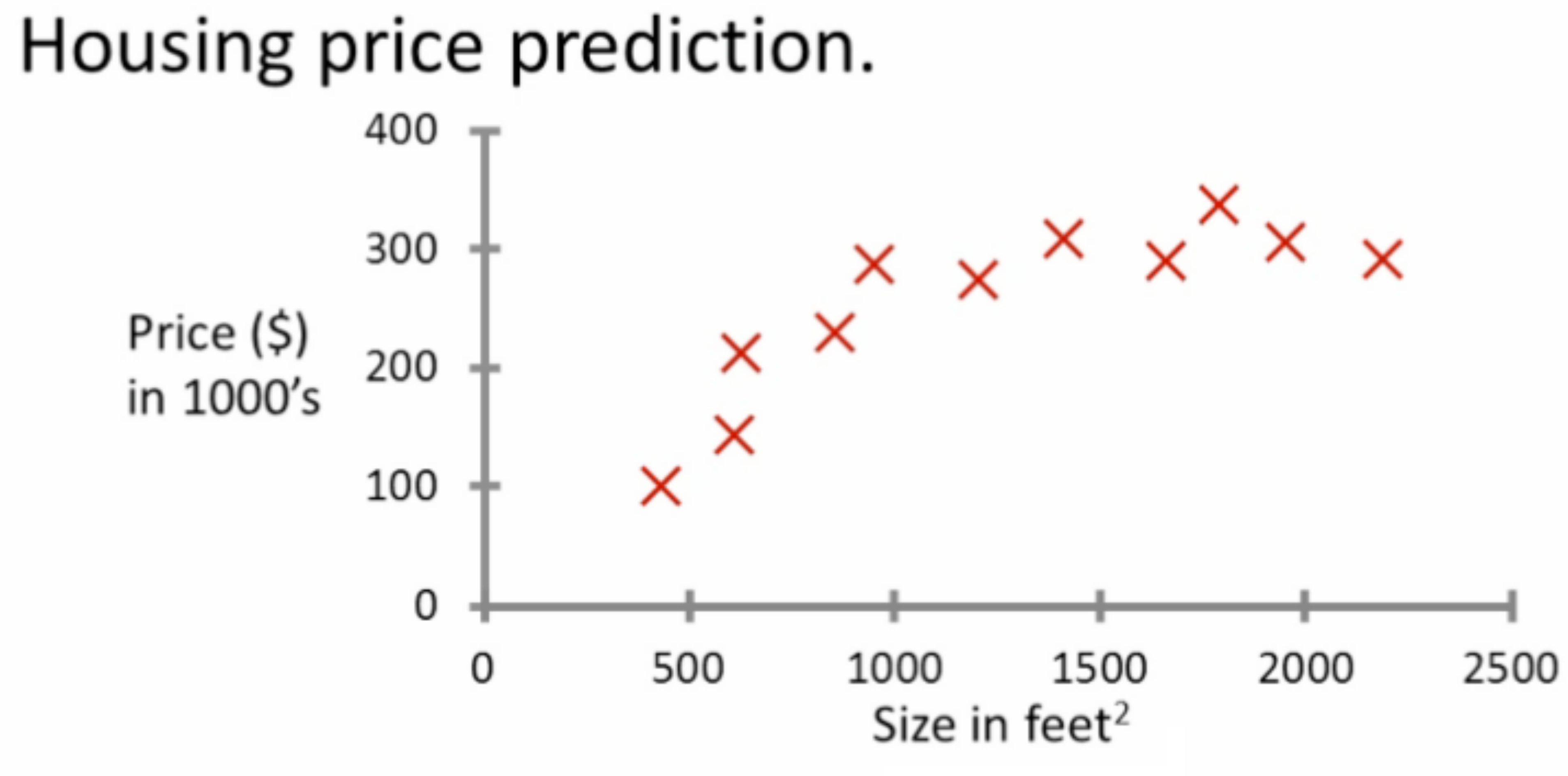
Linear Regression

Director of TEAMLAB
Sungchul Choi

집의 넓이와 집세의 상관관계



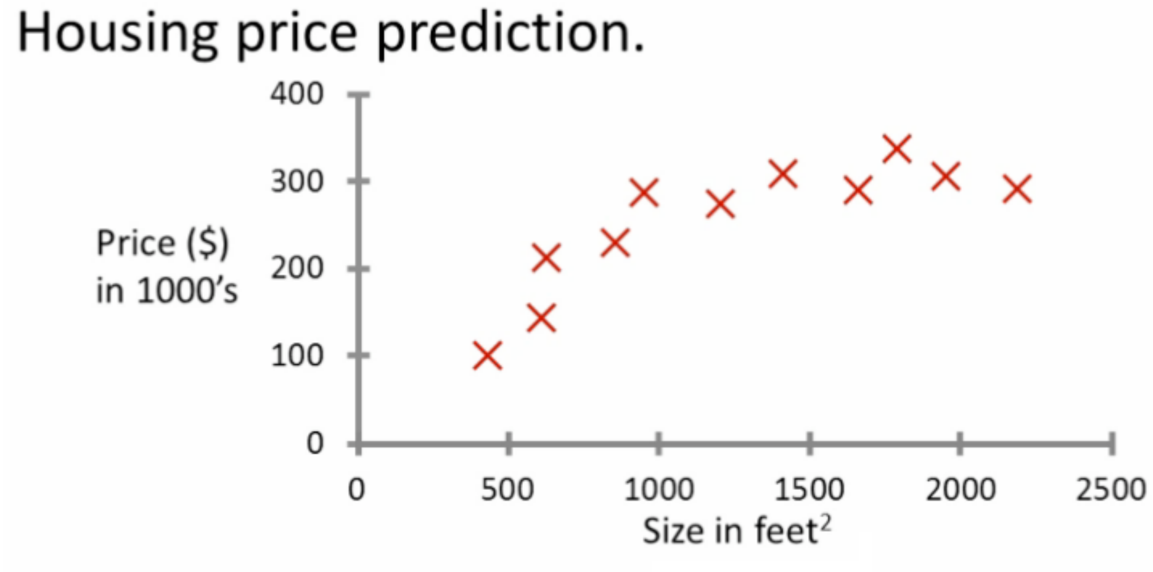
집의 넓이와 집세의 상관관계

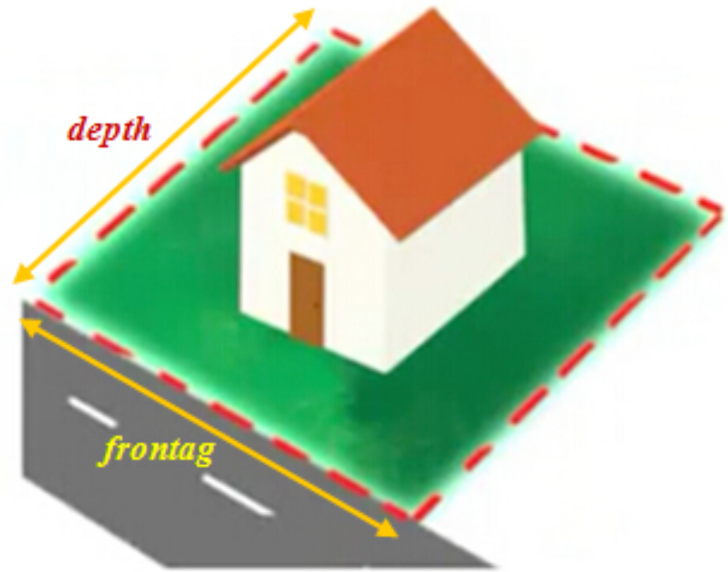


https://padamrajgurung.com.np/machine-learning-1/

Polynomial Regression
다항회귀

집의 넓이와 집세의 상관관계





$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

$$\theta_0 + \theta_1(depth) + \theta_2(frontag) + \theta_3(depth \times frontag)$$

https://padamrajgurung.com.np/machine-learning-1/

Polynomial Features

- **1차 방정식을 고차다항식으로 변경하는 기법**

$$x_1 + x_2 \rightarrow x_1 + x_2 + x_1x_2 + x_1^2 + x_2^2$$

- sklearn.preprocessing.PolynomialFeatures **사용**

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

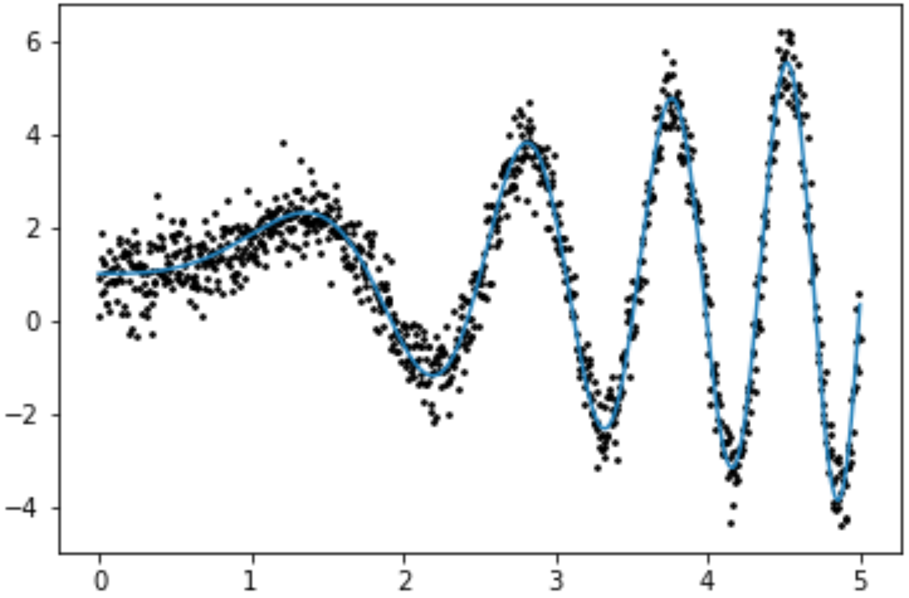
$$\theta_0 + \theta_1(depth) + \theta_2(frontag) + \theta_3(depth \times frontag)$$

```
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)
array([[  1.,   0.,   1.,   0.,   0.,   1.],
       [  1.,   2.,   3.,   4.,   6.,   9.],
       [  1.,   4.,   5.,  16.,  20.,  25.]])
>>> poly = PolynomialFeatures(interaction_only=True)
>>> poly.fit_transform(X)
array([[  1.,   0.,   1.,   0.],
       [  1.,   2.,   3.,   6.],
       [  1.,   4.,   5.,  20.]])
```

$$
\begin{aligned}
h_\theta(x) = &\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\
&\theta_0 + \theta_1(depth) + \theta_2(frontag) + \theta_3(depth \times frontag)
\end{aligned}
$$

Example

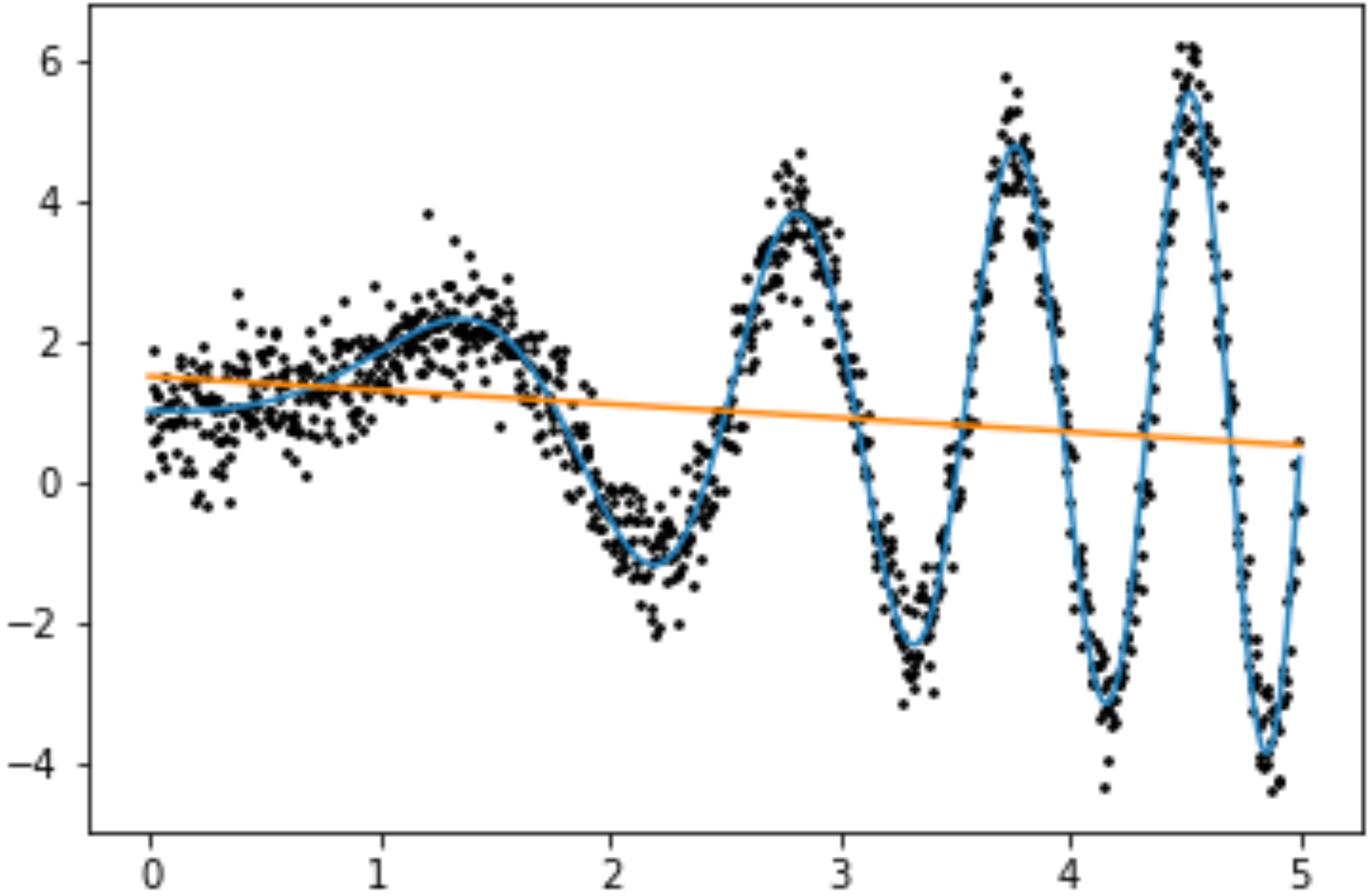
Dataset




```python
def f(size):
    x = np.linspace(0, 5, size)
    y = x * np.sin(x ** 2) + 1
    return (x,y)

def sample(size):
    x = np.linspace(0, 5, size)
    y = x * np.sin(x ** 2) + 1 + pl.randn(x.size)*0.5
    return (x,y)
```

Linear regression



Linear regression

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X,y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
f_x, f_y = f(1000)
plt.plot(f_x, f_y)
plt.scatter(X.flatten(), y.flatten(), s=3, c="black")
plt.plot(X.flatten(), lr.predict(X).flatten())
plt.show()
```

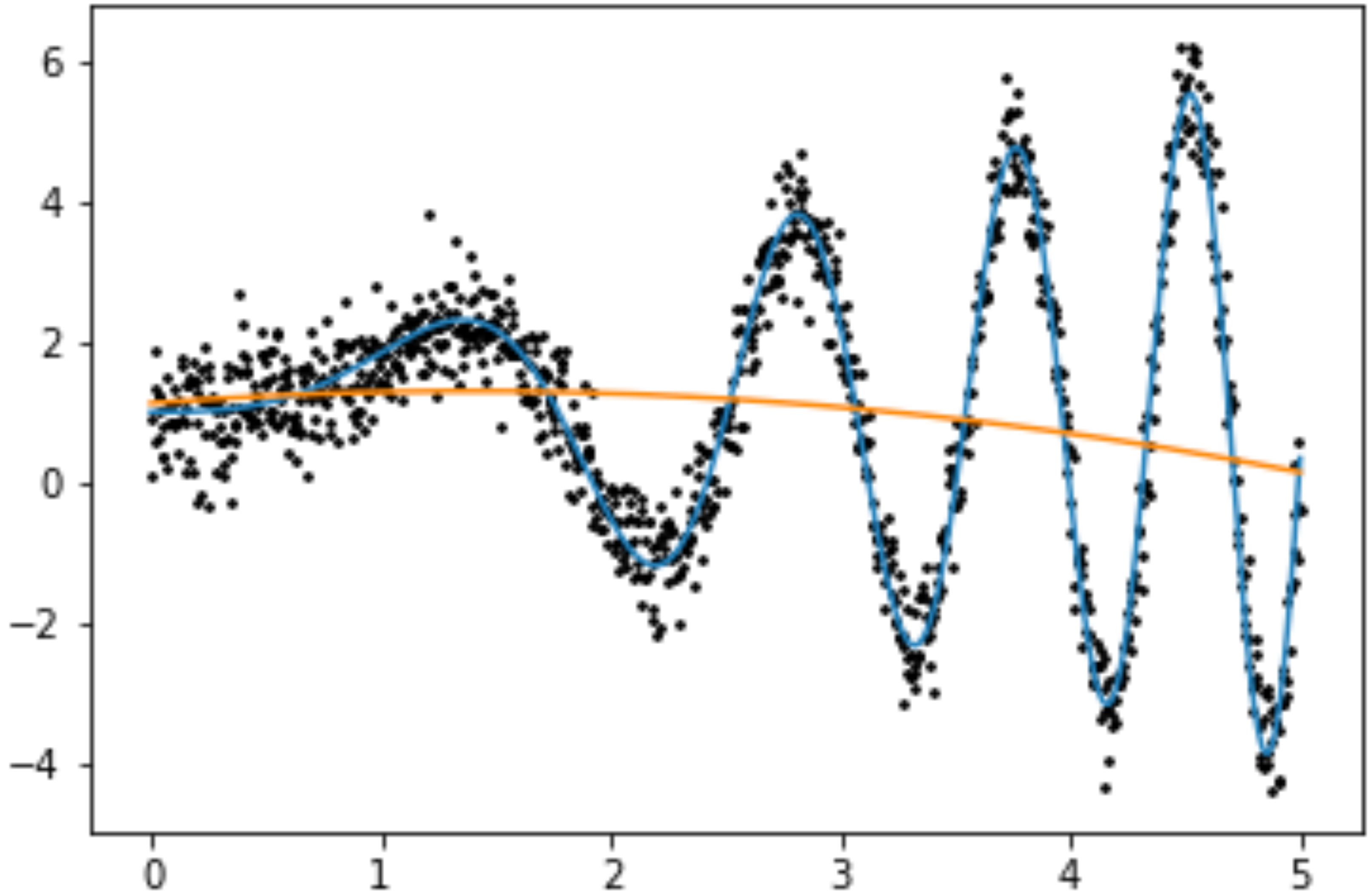
Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=2)
X_poly = poly_features.fit_transform(X)
X_poly[:10]
```

```
array([[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00],
       [  1.00000000e+00,   5.00500501e-03,   2.50500751e-05],
       [  1.00000000e+00,   1.00100100e-02,   1.00200300e-04],
       [  1.00000000e+00,   1.50150150e-02,   2.25450676e-04],
       [  1.00000000e+00,   2.00200200e-02,   4.00801202e-04],
       [  1.00000000e+00,   2.50250250e-02,   6.26251878e-04],
       [  1.00000000e+00,   3.00300300e-02,   9.01802704e-04],
       [  1.00000000e+00,   3.50350350e-02,   1.22745368e-03],
       [  1.00000000e+00,   4.00400400e-02,   1.60320481e-03],
       [  1.00000000e+00,   4.50450450e-02,   2.02905608e-03]])
```

Polynomial Regression

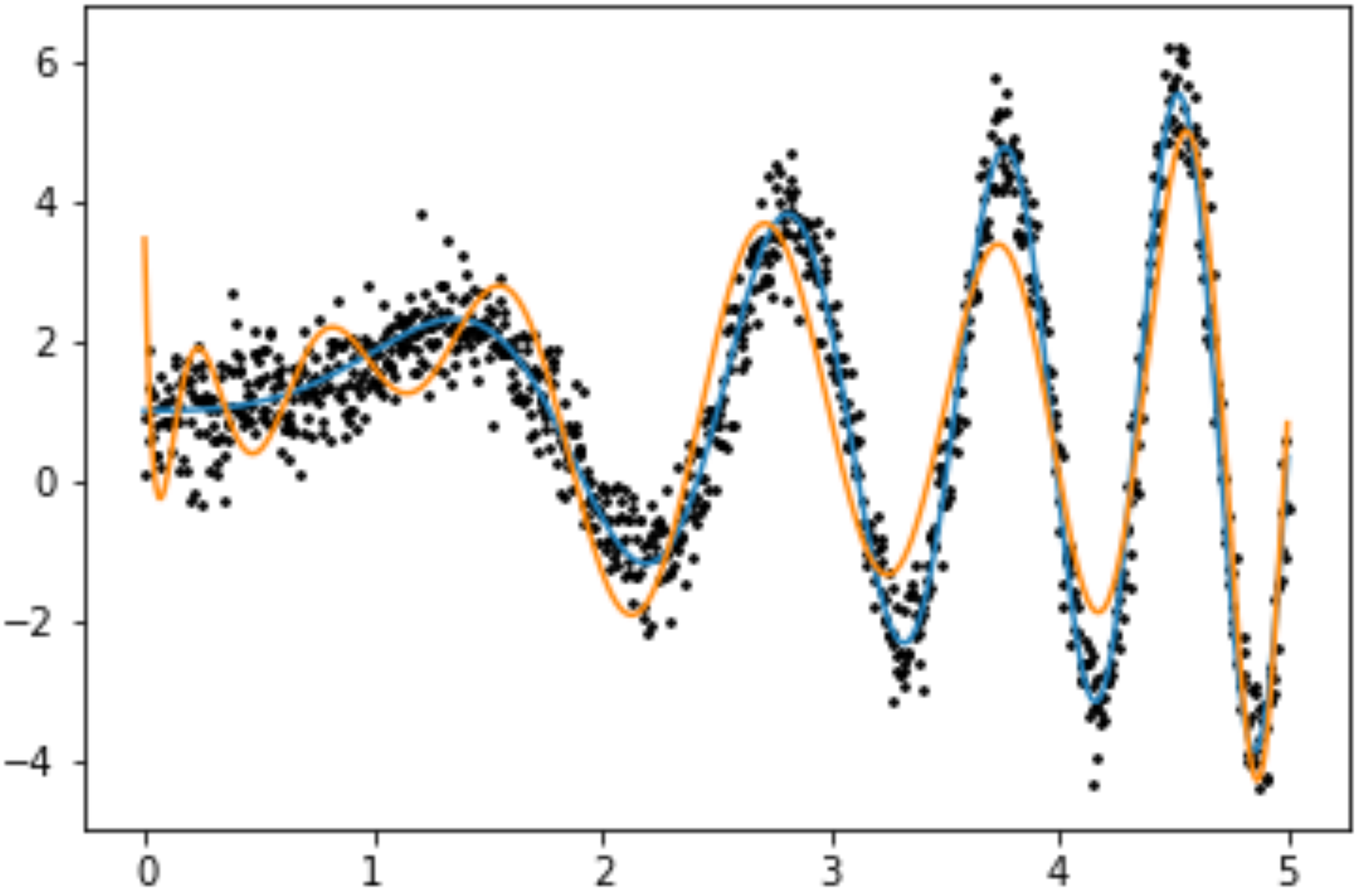


Polynomial Regression

```
poly_features = PolynomialFeatures(degree=15)
X_poly = poly_features.fit_transform(X)
X_poly[:3]
```

```
array([[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00],
       [  1.00000000e+00,   5.00500501e-03,   2.50500751e-05,
          1.25375751e-07,   6.27506263e-10,   3.14067198e-12,
          1.57190790e-14,   7.86740691e-17,   3.93764110e-19,
          1.97079134e-21,   9.86382051e-24,   4.93684710e-26,
          2.47089445e-28,   1.23668391e-30,   6.18960915e-33,
          3.09790248e-35],
       [  1.00000000e+00,   1.00100100e-02,   1.00200300e-04,
          1.00300601e-06,   1.00401002e-08,   1.00501504e-10,
          1.00602106e-12,   1.00702808e-14,   1.00803612e-16,
          1.00904517e-18,   1.01005522e-20,   1.01106629e-22,
          1.01207837e-24,   1.01309146e-26,   1.01410556e-28,
          1.01512068e-30]])
```

Polynomial Regression



How to optimize

- RMSE의 최소값을 찾자
- Ridge, Lasso, LR 모두다 써 보자
- Degree 를 10 ~ 50까지 써보기!
- 결과를 한눈에 정리해보기!!

```python
poly_range = list(range(10, 50))
rmse_lr_list = []
rmse_lasso_list = []
rmse_ridge_list = []

from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge

for poly_value in poly_range:
    poly_features = PolynomialFeatures(degree=poly_value)
    X_poly = poly_features.fit_transform(X)
    lr = LinearRegression()
    lr.fit(X_poly,y)

    rmse_lr_list.append(rmse(lr.predict(X_poly), y))

    lasso = Lasso()
    lasso.fit(X_poly,y)
    rmse_lasso_list.append(rmse(lasso.predict(X_poly), y))

    ridge = Ridge()
    ridge.fit(X_poly,y)
    rmse_ridge_list.append(rmse(ridge.predict(X_poly), y))
```

```python
import pandas as pd
from pandas import DataFrame
data = {"poly_range":poly_range, "lr_rmse":rmse_lr_list,
        "lasso_rmse":rmse_lasso_list,"ridge_rmse":rmse_ridge_list}
df = DataFrame(data).set_index("poly_range")
df
```

poly_range	lasso_rmse	lr_rmse	ridge_rmse
10	2.202944	1.881959	1.907225
11	2.212084	1.873851	1.887279
12	2.217680	1.610925	1.886012
13	2.220835	1.332842	1.858443
14	2.222724	1.315370	1.681269
15	2.223819	0.934578	1.465469
16	2.224356	0.731226	1.524860
17	2.224477	0.686098	1.280227
18	2.224292	0.707544	0.876097
19	2.223907	0.672758	0.753384
20	2.223435	0.658000	0.746286
21	2.222997	0.634819	0.623290
22	2.222719	0.553137	0.544725
23	2.222722	0.547207	0.583676
24	2.223110	0.630950	0.841203
30	2.235822	1.150520	1.044262
31	2.239474	1.110347	1.138093
32	2.243404	1.298196	2.567021
33	2.247543	1.329882	1.133565
34	2.252578	1.428437	1.195742
35	2.258469	1.323009	1.628810
36	2.264223	1.297421	2.116149
37	2.269635	1.270417	1.664779
38	2.274774	1.252174	1.461587
39	2.279632	1.248591	1.453577
40	2.284208	1.262911	1.603647
41	2.288505	1.285513	1.369352
42	2.292531	1.554041	1.731936
43	2.296293	1.491091	1.626011
44	2.299798	1.491933	1.389574
45	2.303052	1.594845	1.420752

```python
df.min()
```

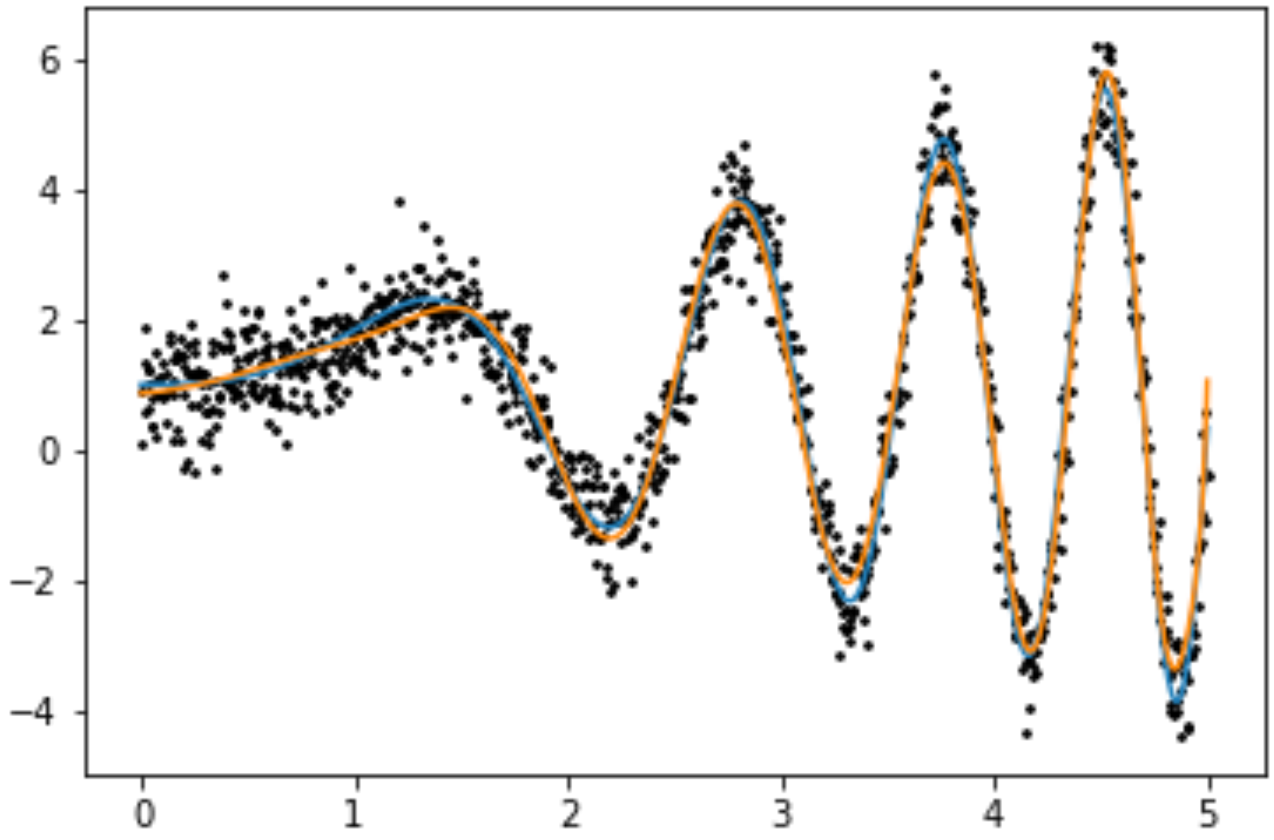
```
lasso_rmse    2.162528
lr_rmse       0.521856
ridge_rmse    0.520451
dtype: float64
```

```python
df["ridge_rmse"].sort_values().head()
```

```
poly_range
22    0.520451
23    0.524784
21    0.604382
24    0.619083
26    0.663233
Name: ridge_rmse, dtype: float64
```

```
poly_features = PolynomialFeatures(degree=22)
X_poly = poly_features.fit_transform(X)
ridge = Ridge()
ridge.fit(X_poly,y)
```

```
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```



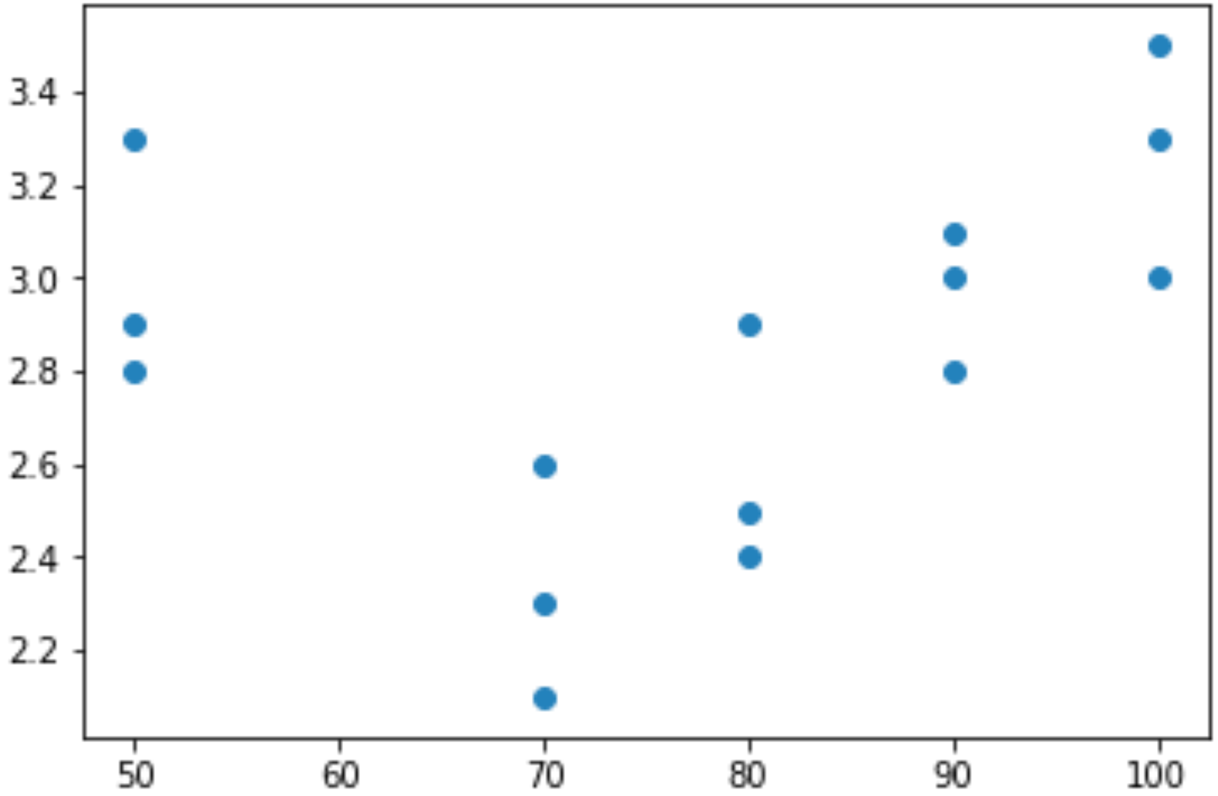
언제쓰나?

- 한개 변수가 Y값과 비선형적인 관계가 있다고 의심
- 주기적인 패턴을 보이는 Series 데이터
- 모델 자체가 복잡해지면 해결가능한 부분이 많음
  ➔ SVM, Tree-based models

Challenge

```
df = pd.read_csv("yield.csv",sep="\t")
df.head()
```

	i	Temp	Yield
0	1	50	3.3
1	2	50	2.8
2	3	50	2.9
3	4	70	2.3
4	5	70	2.6



TEAMLAB

Human knowledge belongs to the world.