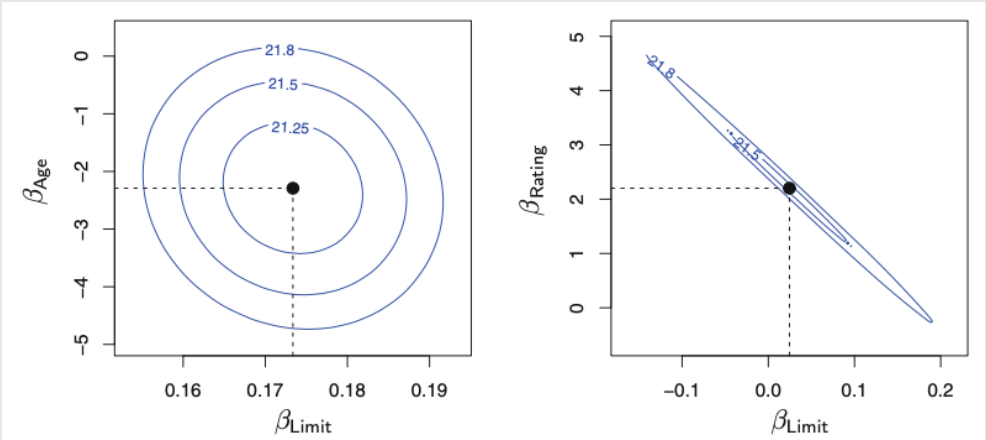
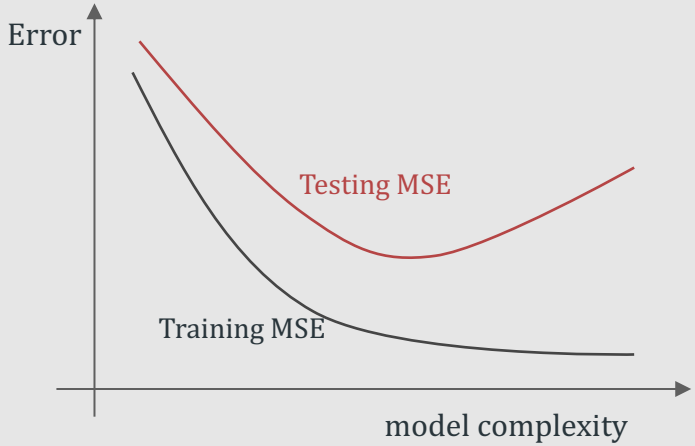
Linear Regression

Potential Problem – Collinearity



- When two predictors are highly correlated, the contours of the MSE run along a narrow valley, there is a broad range for the coefficient estimate.
- A small change in data could cause significant change in the parameter estimation.
- With $x_1$and x_2 linearly related (perfectly), X^TX has a 0 eigenvalue.
- So the level set $\{w|(w-\hat{w})^TX^TX(w-\hat{w})=C\}$ is no longer an ellipsoid. It's a degenerated ellipsoid – where contour lines will be pairs of lines in this case

Potential Problem – Overfitting



In linear regression, the more features X_j we include in the model, the lower training MSE will be. Adding too many features to the model may lead to overfitting.

Deciding on important variables

Subset selection:
We identify a subset of the p predictors that we believe to be related to the response.
We then fit a model using least squares on the reduced set of variables.

If there are p candidate predictors,

Forward Stepwise Selection

- Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model.
- In particular, at each step the variable that gives the greatest additional improvement to the fit is added to the model.

Forward Stepwise Feature Selection

- Let M_0 denote the null model, which contains no predictors.
- For $k = 0, p-1, p$:
  - Consider all $p-k$ models that augment the predictors in M_k with one additional predictor.
  - Choose the best among these $p-k$ models, and call it M_{k+1}. Here best is defined as having smallest SSE.
- Select a single best model from among $M_1, \ldots, M_p$ using cross-validated prediction error

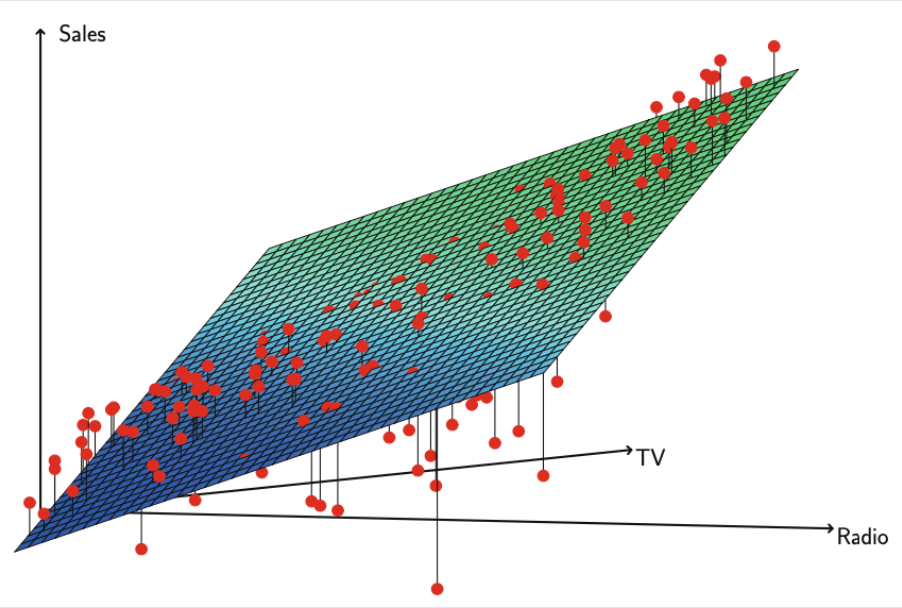
Backward Stepwise Selection

- Like forward stepwise selection, backward stepwise selection provides an efficient alternative to best subset selection.
- However, unlike forward stepwise selection, it begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time.

Backward Stepwise Selection

- Let M_p denote the full model, which contains all p predictors.
- For $k = p, p-1, \ldots 1$:
  - Consider all k models that contain all but one of the predictors in M_k, for a total of k − 1 predictors.
  - Choose the best among these k models, and call it M_{k-1}. Here best is defined as having smallest SSE.
- Select a single best model from among $M_1, \ldots, M_p$ using cross-validated prediction error

Potential problem: synergy between two predictors



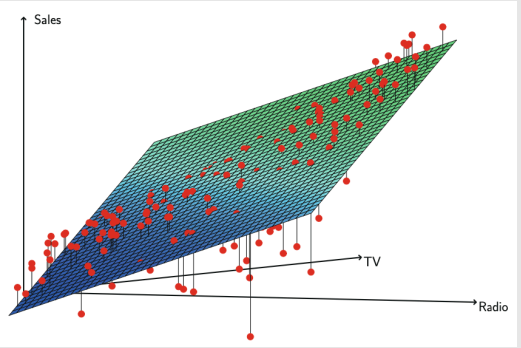
When there is a synergy or interaction effect between two predictors

Potential problem: synergy between two predictors

$$y = w_0 + w_1 x_1 + w_2 x_2$$

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2$$

Interaction term

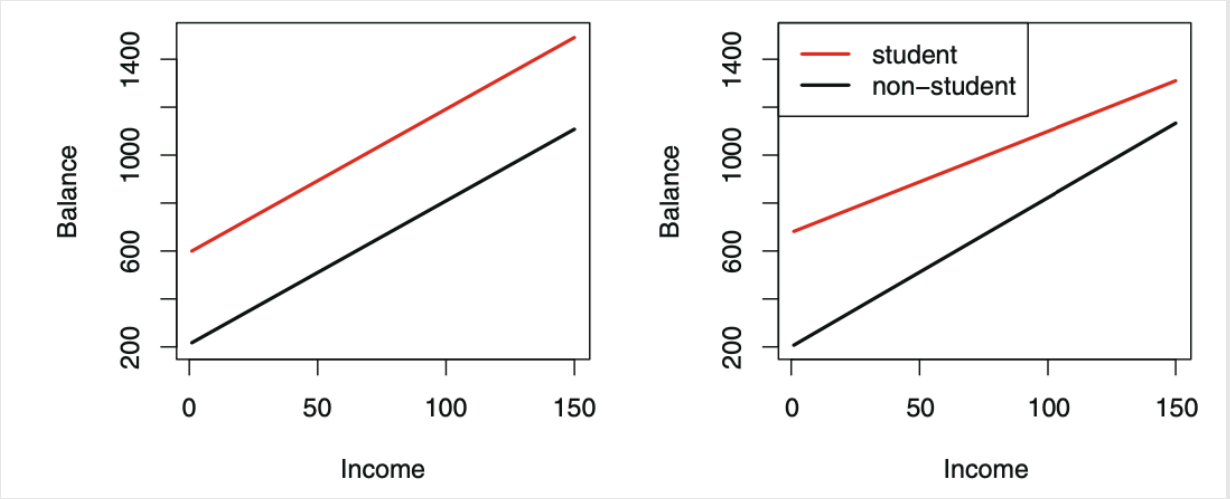


Potential problem: synergy between two predictors

Interaction between continuous and discrete variable

$$balance = w_0 + w_1 \times income + w_2 \times I_{s\ tudent}$$

$$balance = w_0 + w_1 \times income + w_2 \times I_{student} + w_3 \times income \times I_{student}$$



Non student

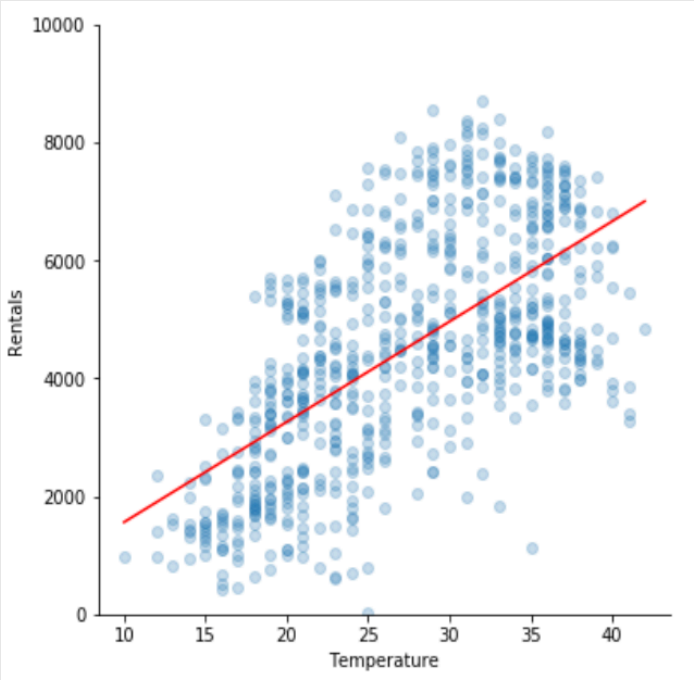
$w_0 + w_1 \times income$

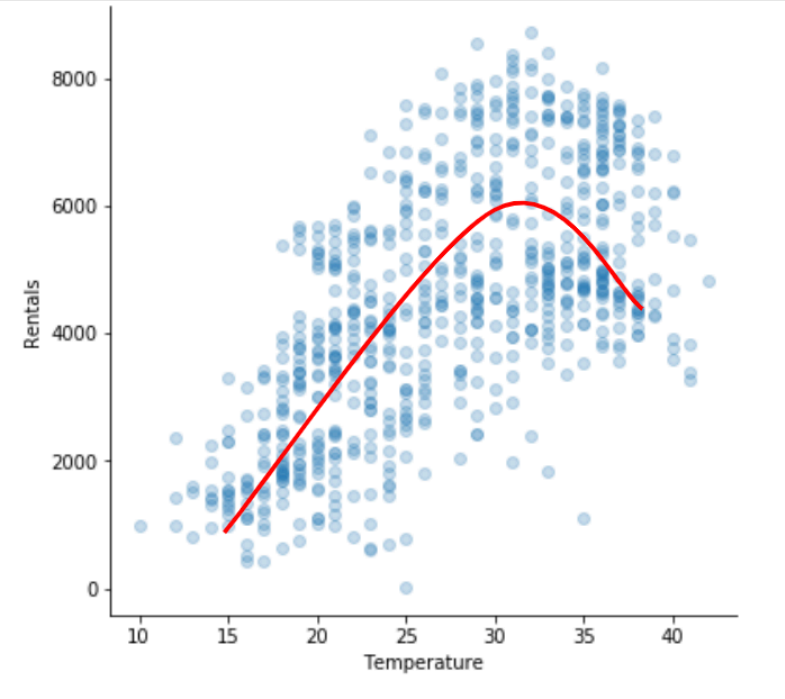
Student

$w_0 + (w_1 + w_3) \times income + w_2$

Moving Beyond Linearity

The world is not linear

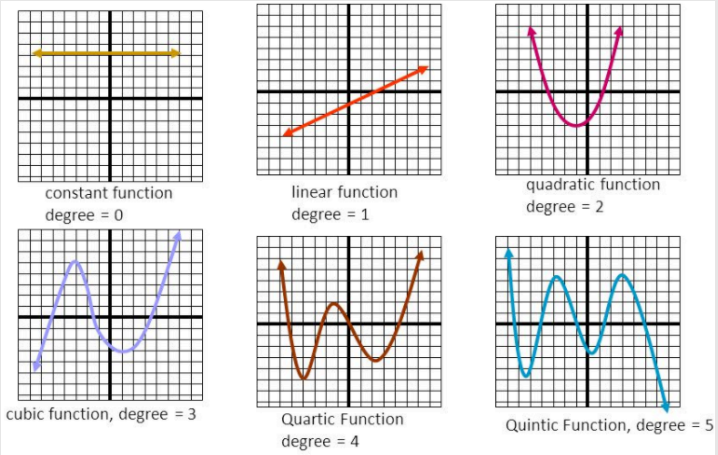




Polynomial Regression

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_D x^D$$
$$= w_0 + w_1 \phi_1(x) + w_2 \phi_2(x) + \cdots + w_D \phi_D(x)$$
$$= w^T \phi(x)$$

Polynomial function of different degrees



Polynomial Regression in sklearn

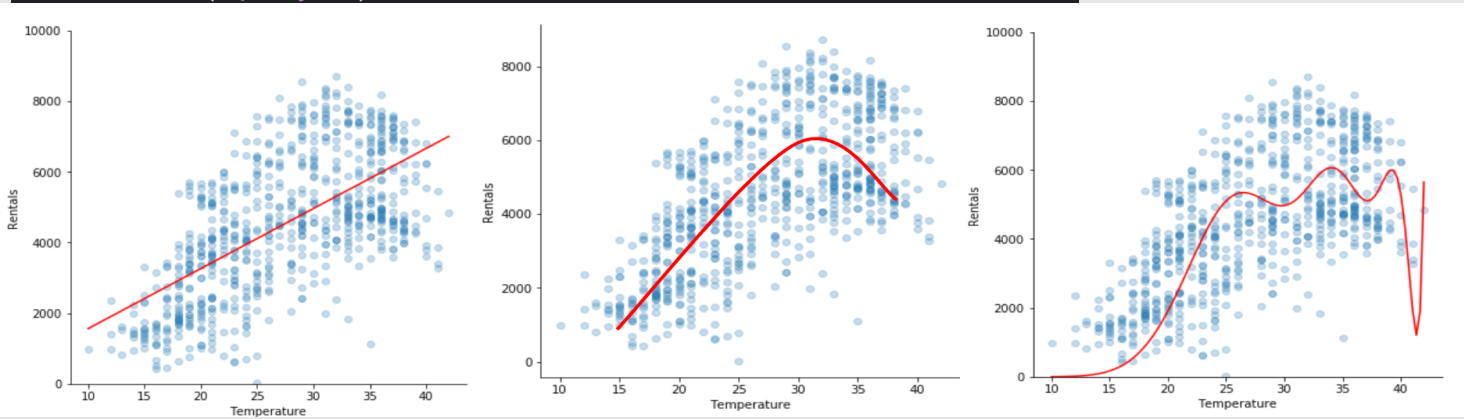
```
from sklearn.preprocessing import PolynomialFeatures

# degree of the polynomial
D       = ...

# adding polynomials of x is a preprocessing step!
poly    = PolynomialFeatures(degree=D)
Xpoly   = poly.fit_transform(X)

model = LinearRegression()
model.fit(Xpoly,Y)
```

Create a matrix containing powers of X

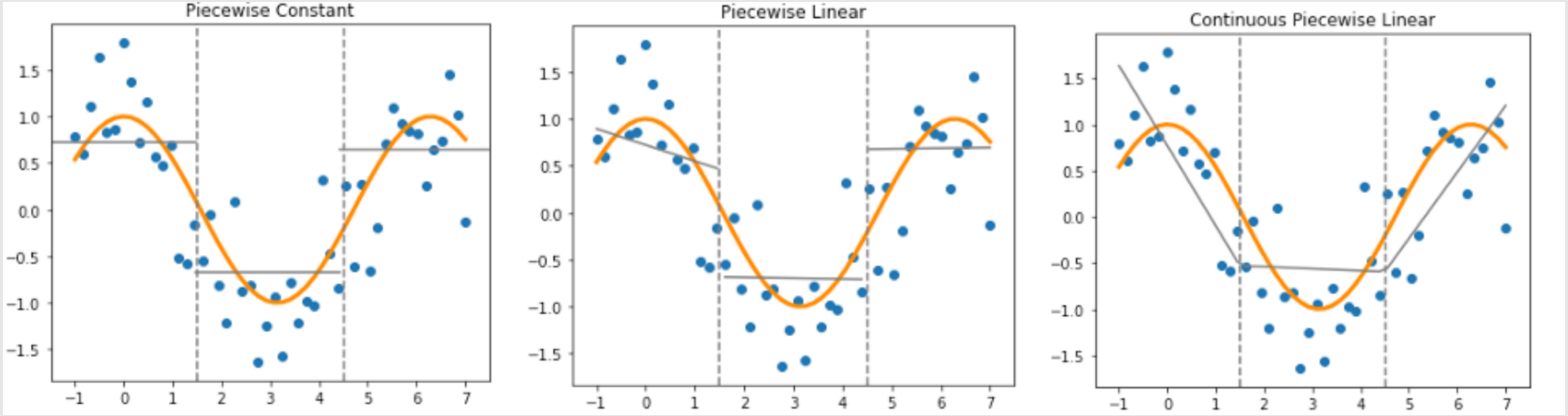


Piecewise Polynomials (Optional)

Divide the domain of X into contiguous intervals

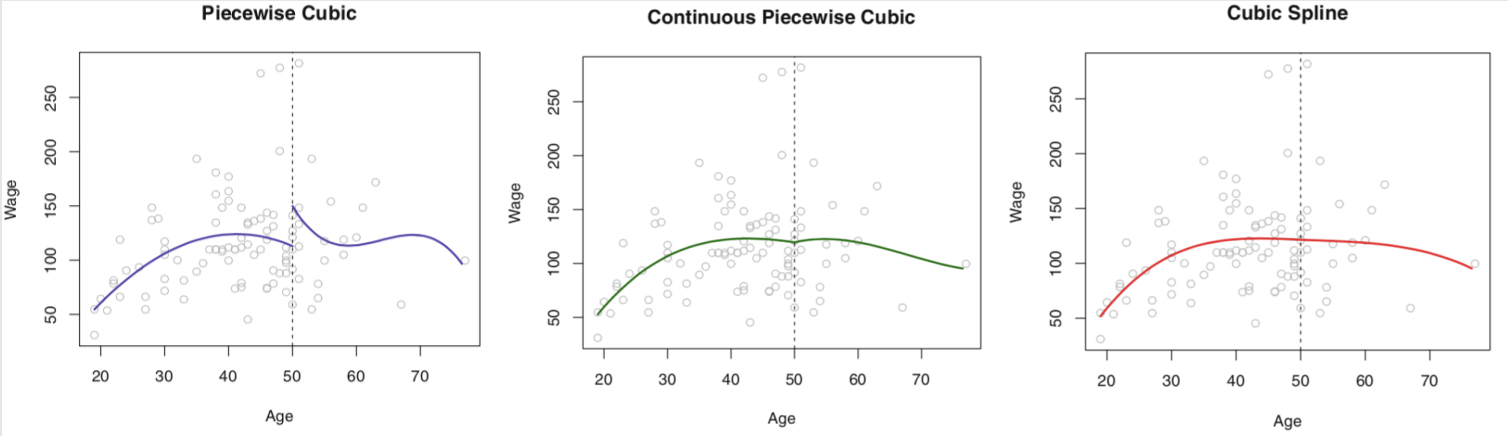
Represent the function by a separate polynomial in each interval.

In each interval, fit a separate polynomial regression model $y = w_0 + w_1 x + w_2 x^2 + \cdots + w_D x^D$

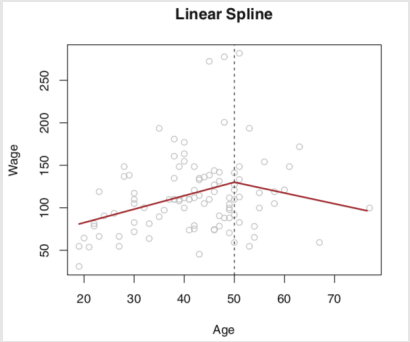


the function is discontinuous

Regression splines (Optional)



Degree-d spline is piecewise degree-d polynomial with continuity in derivatives up to degree d-1 at each knot



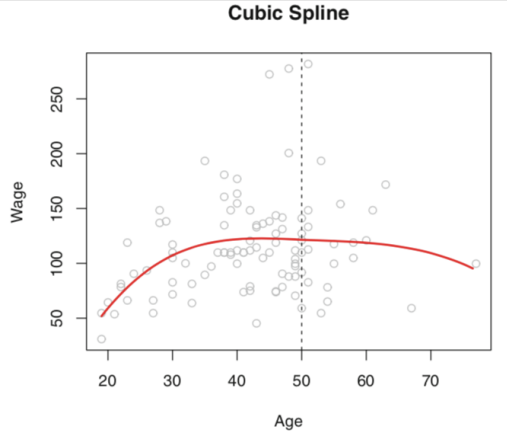
Cubic Spline (Optional)

Fit a regression model with the following predictors

$$X, X^2, X^3, h(X, \xi_1), h(X, \xi_2), \ldots, h(X, \xi_K)$$

$$h(x, \xi) = (x - \xi)_+^3 = \begin{cases} (x - \xi)^3 \text{ if } x > \xi \\ 0 \quad \text{otherwise} \end{cases}$$

where $\xi_1, \xi_2, \ldots \xi_K$ are the knots.



Linear Regression

Model:

$$y = w_0 + w_1 x_1 + \cdots + w_p x_p$$

Matrix representation:

$$y = Xw$$

The loss function:

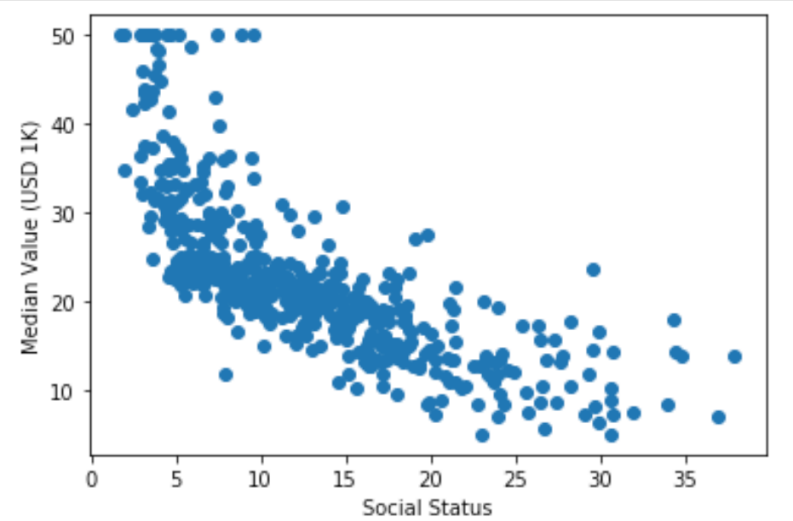
$$MSE = \frac{1}{n} (y - X^T w)^T (y - X^T w)$$

Least square estimation:

$$\widehat{w} = (X^T X)^{-1} X^T y$$

Exercise

Exercise: Boston housing dataset



Boston Housing

```
from sklearn.preprocessing import PolynomialFeatures

# create the input feature transformer, specify the degree
poly = PolynomialFeatures(degree = 3)
poly.fit(X)

# apply the tranformer to our X matrix
X_with_2nd_degree = poly.transform(X)

# now run the linear regression module
model       = LinearRegression()
model       = model.fit(X_with_2nd_degree, y)
y_predicted = model.predict(X_with_2nd_degree)

MSE_poly    = mean_squared_error(y, model.predict(X_with_2nd_degree))
print(MSE_poly)

sort_by_x = np.argsort(X.T)[0]
plt.scatter(X,y)
plt.plot(X[sort_by_x],y_predicted[sort_by_x],color="red");
```