(교차 검증)

In [7]:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5)

from sklearn.model_selection import cross_val_score
cross_val_score(model, X, y, cv=10)
```

Out[7]:

```
array([1.        , 0.93333333, 1.        , 1.        , 0.86666667,
       0.93333333, 0.93333333, 1.        , 1.        , 1.        ])
```

In [ ]:

```
array([1.        , 0.93333333, 1.        , 0.93333333, 0.86666667,
       1.        , 0.86666667, 1.        , 1.        , 1.        ])
```

In [ ]:

```
X1, y1 = load_iris(return_X_y=True)
print(X1.shape)
print(y1.shape)
```

In [5]:

```
from sklearn import datasets
iris = datasets.load_iris(return_X_y=True)
```

In [ ]:

```

```

In [ ]:

```

```

In [8]:

```
from sklearn.model_selection import LeaveOneOut
scores = cross_val_score(model, X, y, cv=LeaveOneOut())
scores
```

Out[8]:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [9]:

```
scores.mean()
```

Out[9]:

```
0.9666666666666667
```

In [ ]:

```

```

(편향 - 분산 트레이드 오프)

In [57]:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

cnt = 0

def PolynomialRegression(degree=2, **kwargs):
    print(" degree = ", degree )

    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
```

In [50]:

```
import numpy as np

def make_data(N, err=1.0, seed=1):
    np.random.seed(seed)

    X = np.random.rand(N, 1) **2
    y = 10 -1. / (X.ravel() + 0.1)
    if err > 0:
        y += err * np.random.randn(N)
    return X, y

X, y = make_data(40)
```

In [51]:

```
X.shape
```

Out[51]:

```
(40, 1)
```

In [52]:

```
X[:5]
```

Out[52]:

```
array([[1.73907352e-01],
       [5.18867376e-01],
       [1.30815988e-08],
       [9.14049845e-02],
       [2.15372915e-02]])
```

In [53]:

```
X.ravel()[:5]
```

Out[53]:

```
array([1.73907352e-01, 5.18867376e-01, 1.30815988e-08, 9.14049845e-02,
       2.15372915e-02])
```

In [ ]:

```

```

In [36]:

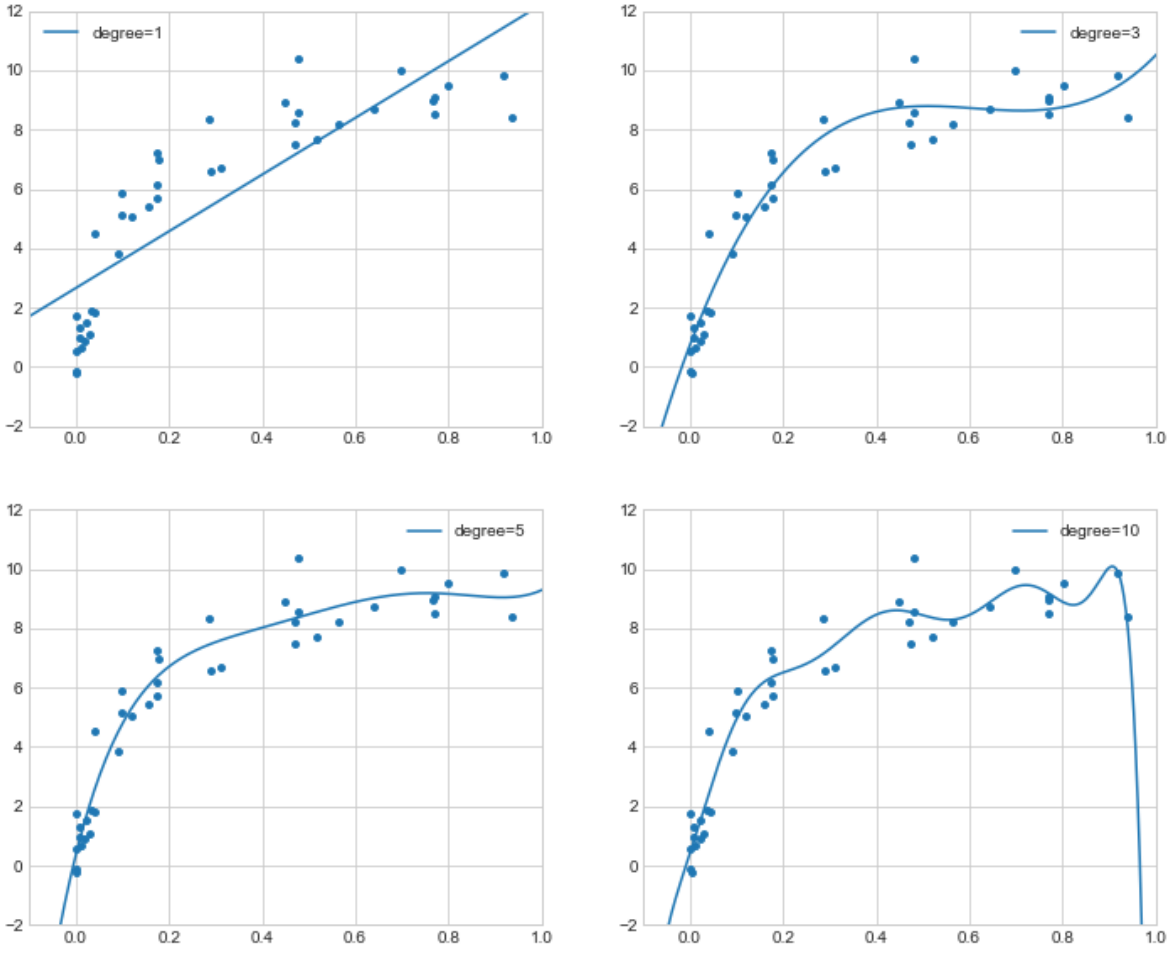
```
# ravel() vs reshape() vs flatten()
# https://m.blog.naver.com/wideeyed/221533365486
```

In [ ]:

```

```

In [54]:

```
#%matplotlib inline

import matplotlib.pyplot as plt
plt.style.use("seaborn-whitegrid")
X_test = np.linspace(-01., 1.1, 500).reshape(-1, 1)

fig = plt.figure(figsize=(12,10))

for i, degree in enumerate([1,3,5,10], start=1):
    ax = fig.add_subplot(2,2,i)
    ax.scatter(X.ravel(), y, s=15)

    y_test = make_pipeline(PolynomialFeatures(degree), LinearRegression()).fit(X,y).predict(X_t

    ax.plot(X_test.ravel(), y_test, label='degree={0}'.format(degree))
    ax.set_xlim(-0.1, 1.0)
    ax.set_ylim(-2, 12)
    ax.legend(loc='best')
```

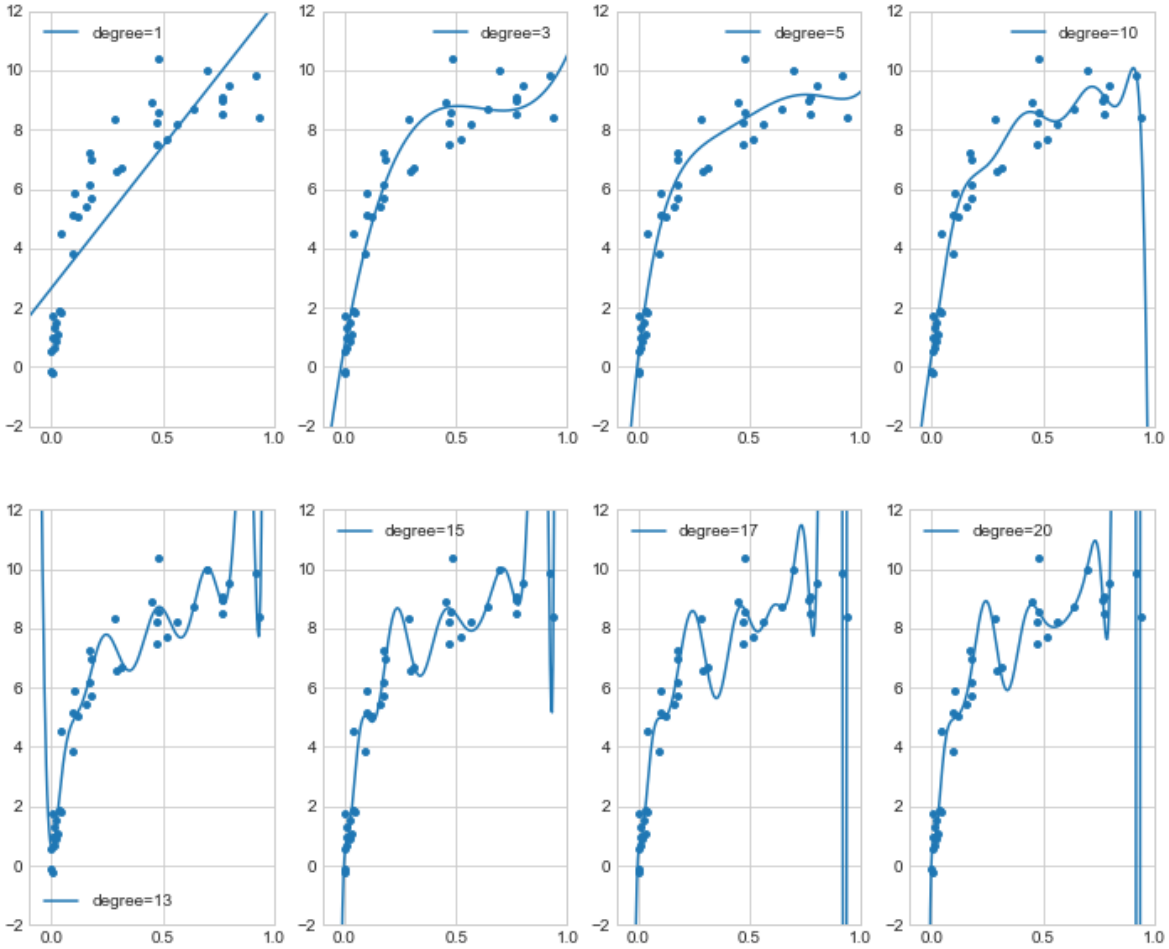


In [ ]:

```

```

In [55]:

```
#%matplotlib inline

import matplotlib.pyplot as plt
plt.style.use("seaborn-whitegrid")
X_test = np.linspace(-01., 1.1, 500).reshape(-1, 1)

fig = plt.figure(figsize=(12,10))

for i, degree in enumerate([1,3,5,10,13,15,17,20], start=1):
    ax = fig.add_subplot(2,4,i)
    ax.scatter(X.ravel(), y, s=15)

    y_test = make_pipeline(PolynomialFeatures(degree), LinearRegression()).fit(X,y).predict(X_t

    ax.plot(X_test.ravel(), y_test, label='degree={0}'.format(degree))
    ax.set_xlim(-0.1, 1.0)
    ax.set_ylim(-2, 12)
    ax.legend(loc='best')
```



In [ ]:

```
1
```

In [44]:

```
for j, degree in enumerate([1,3,5,10], start=3):
    print("i = ", j, " degree = ", degree)
```

```
i =  3  degree =  1
i =  4  degree =  3
i =  5  degree =  5
i =  6  degree =  10
```

In [26]:

```
i  = 1
for degree in ([1,3,5,10]):
    print("i = ", i,  " degree = ", degree)
    i = i + 1
```

```
i =  1  degree =  1
i =  2  degree =  3
i =  3  degree =  5
i =  4  degree =  10
```

In [ ]:

```

```

(최적 모델)

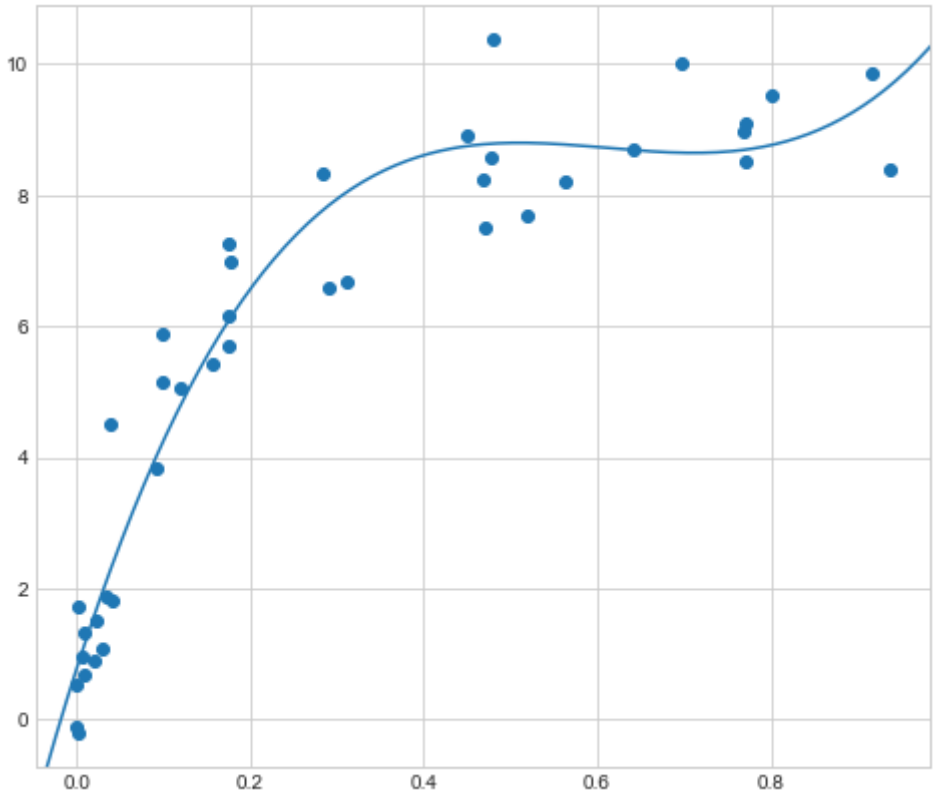
In [58]:

```
plt.figure(figsize=(8,7))
plt.scatter(X.ravel(), y)
lim = plt.axis()
y_test = PolynomialRegression(3).fit(X,y).predict(X_test)
plt.plot(X_test.ravel(), y_test)
plt.axis(lim)
```

```
degree =  3
```

Out[58]:

```
(-0.04687651021505175,
 0.9844070023112612,
 -0.7308177116555796,
 10.902869392322714)
```



In [ ]:

```

```

(그리드 서치)

In [ ]:

```
def PolynomialRegression(degree=2, **kwargs):
    print(" degree = ", degree )

    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
```

In [64]:

```
from sklearn.model_selection import GridSearchCV

param_grid = {'polynomialfeatures__degree' : np.arange(50),
              'linearregression__fit_intercept':[True, False],
              'linearregression__normalize':[True, False]}
grid = GridSearchCV(PolynomialRegression(), param_grid, cv=11)
```

```
 degree =  2
```

In [65]:

```
grid.fit(X,y)
```

Out[65]:

```
GridSearchCV(cv=11,
             estimator=Pipeline(steps=[('polynomialfeatures',
                                        PolynomialFeatures()),
                                       ('linearregression',
                                        LinearRegression())]),
             param_grid={'linearregression__fit_intercept': [True, False],
                         'linearregression__normalize': [True, False],
                         'polynomialfeatures__degree': array([ 0,  1,  2,  3,  4,
5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

In [66]:

```
grid.best_params_
```

Out[66]:

```
{'linearregression__fit_intercept': False,
 'linearregression__normalize': True,
 'polynomialfeatures__degree': 4}
```

In [16]:

```
#dir(grid)
```

In [67]:

```
1 dir(grid)
```

Out[67]:

```
['__abstractmethods__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_abc_impl',
 '_check_is_fitted',
 '_check_n_features',
 '_estimator_type',
 '_format_results',
 '_get_param_names',
 '_get_tags',
 '_more_tags',
 '_pairwise',
 '_repr_html_',
 '_repr_html_inner',
 '_repr_mimebundle_',
 '_required_parameters',
 '_run_search',
 '_validate_data',
 'best_estimator_',
 'best_index_',
 'best_params_',
 'best_score_',
 'classes_',
 'cv',
 'cv_results_',
 'decision_function',
 'error_score',
 'estimator',
 'fit',
```

```
 'get_params',
 'iid',
 'inverse_transform',
 'multimetric_',
 'n_features_in_',
 'n_jobs',
 'n_splits_',
 'param_grid',
 'pre_dispatch',
 'predict',
 'predict_log_proba',
 'predict_proba',
 'refit',
 'refit_time_',
 'return_train_score',
 'score',
 'scorer_',
 'scoring',
 'set_params',
 'transform',
 'verbose']
```

In [68]:

```
1 grid.best_estimator_
```

Out[68]:

```
Pipeline(steps=[('polynomialfeatures', PolynomialFeatures(degree=4)),
                ('linearregression',
                 LinearRegression(fit_intercept=False, normalize=True))])
```

In [71]:

```
1 model = LinearRegression(fit_intercept=False, normalize=True)
```

In [70]:

```
1 model.
```

Out[70]:

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

In [ ]:

```
1
```

In [73]:

```
model.get_params()
```

Out[73]:

```
{'memory': None,
 'steps': [('polynomialfeatures', PolynomialFeatures(degree=4)),
  ('linearregression', LinearRegression(fit_intercept=False, normalize=True))],
 'verbose': False,
 'polynomialfeatures': PolynomialFeatures(degree=4),
 'linearregression': LinearRegression(fit_intercept=False, normalize=True),
 'polynomialfeatures__degree': 4,
 'polynomialfeatures__include_bias': True,
 'polynomialfeatures__interaction_only': False,
 'polynomialfeatures__order': 'C',
 'linearregression__copy_X': True,
 'linearregression__fit_intercept': False,
 'linearregression__n_jobs': None,
 'linearregression__normalize': True}
```

In [ ]:

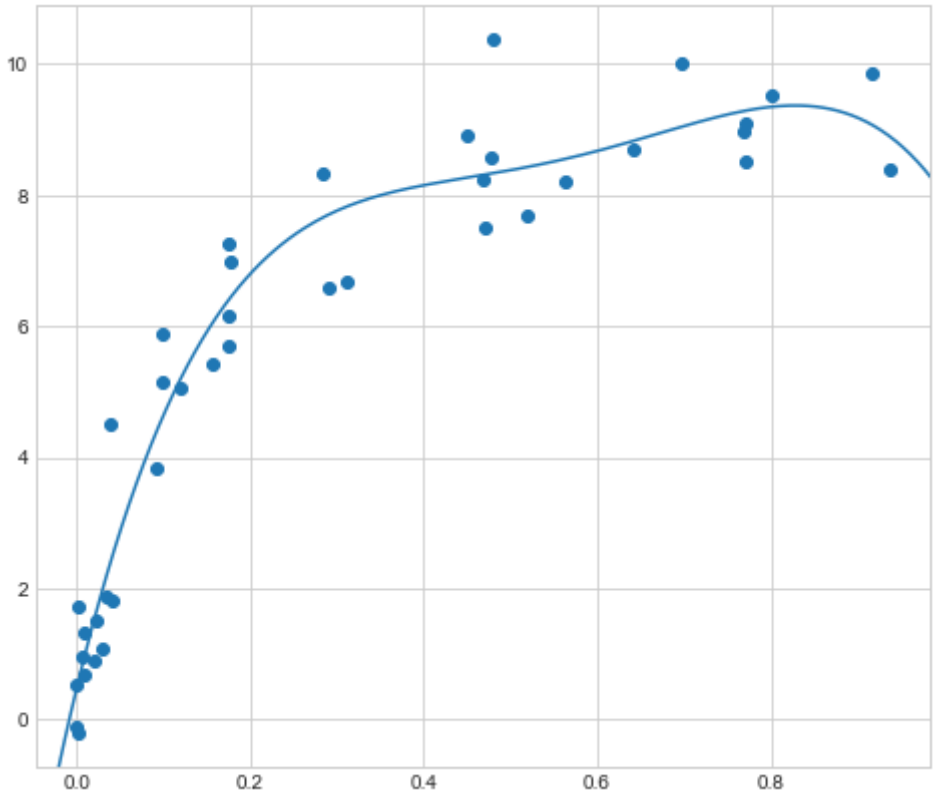
```

```

In [84]:

```
model = grid.best_estimator_

plt.figure(figsize=(8,7))
plt.scatter(X.ravel(), y)
lim = plt.axis()
y_test = model.fit(X,y).predict(X_test)
plt.plot(X_test.ravel(), y_test)
#plt.axis()
plt.axis(lim)
```

Out[84]:

```
(-0.04687651021505175,
 0.9844070023112612,
 -0.7308177116555796,
 10.902869392322714)
```



In [80]:

```
lim
```

Out[80]:

```
(-0.04687651021505175,
 0.9844070023112612,
 -0.7308177116555796,
 10.902869392322714)
```

In [ ]:

```

```

In [ ]:

```
1
```

(실습 코드)

6-01.py

6-02.py

In [ ]:

```
1
```

In [76]:

```
1 # 모델 저장 & 복원
2 import pickle
3 s = pickle.dumps(model)
```

In [88]:

```
1 # 모델 복원
2 model2 = pickle.loads(s)
3 model2
```

Out[88]:

```
Pipeline(steps=[('polynomialfeatures', PolynomialFeatures(degree=4)),
                ('linearregression',
                 LinearRegression(fit_intercept=False, normalize=True))])
```

In [95]:

```
1 import time
2 time.time()
```

Out[95]:

```
1610440749.7487457
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [100]:

```
1 from joblib import dump, load
2 import time
3 # 훈련된 모델을 외부 파일로 저장
4 dump_name = "model_" + str(time.time()) + ".joblib"
5
6 dump(model, dump_name)
```

Out[100]:

```
['model_1610440832.5420282.joblib']
```

In [ ]:

```
1 # 모델 복원
2 model2 = load('filename.joblib')
```

In [ ]:

```
1
```

(특징 공학)

범주 특징의 변환

In [47]:

```
1 data = [
2     {"price":1200000000, "rooms":3, "location":"잠실동"},
3     {"price":700000000, "rooms":4, "location":"천호동"},
4     {"price":1200000000, "rooms":4, "location":"신천동"},
5     {"price":700000000, "rooms":2, "location":"천호동"},
6 ]
```

In [48]:

```
import numpy as np
from sklearn.feature_extraction import DictVectorizer

dv = DictVectorizer(sparse=False, dtype=np.int)
dv.fit_transform(data)
```

Out[48]:

```
array([[         0,          1,          0, 1200000000,          3],
       [         0,          0,          1,  700000000,          4],
       [         1,          0,          0, 1200000000,          4],
       [         0,          0,          1,  700000000,          2]])
```

In [ ]:

```

```

In [49]:

```
dv.get_feature_names()
```

Out[49]:

```
['location=신천동', 'location=잠실동', 'location=천호동', 'price', 'rooms']
```

In [50]:

```
dv = DictVectorizer(sparse=True, dtype=np.int)
dv.fit_transform(data)
```

Out[50]:

```
<4x5 sparse matrix of type '<class 'numpy.int32'>'
        with 12 stored elements in Compressed Sparse Row format>
```

In [ ]:

```

```

(특징 공학)

텍스트 특징의 변환

In [51]:

```
text = [
    "mobile phone",
    "battlegrounds mobile game",
    "phone game"
]
```

In [53]:

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()
X = cv.fit_transform(text)
X
```

Out[53]:

```
<3x4 sparse matrix of type '<class 'numpy.int64'>'
        with 7 stored elements in Compressed Sparse Row format>
```

In [54]:

```
import pandas as pd
pd.DataFrame(X.toarray(), columns = cv.get_feature_names())
```

Out[54]:

	battlegrounds	game	mobile	phone
0	0	0	1	1
1	1	1	1	0
2	0	1	0	1

In [ ]:

```

```

In [55]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer()
X = tv.fit_transform(text)
X
```

Out[55]:

```
<3x4 sparse matrix of type '<class 'numpy.float64'>'
        with 7 stored elements in Compressed Sparse Row format>
```

In [56]:

```
pd.DataFrame(X.toarray(), columns=tv.get_feature_names())
```

Out[56]:

	battlegrounds	game	mobile	phone
0	0.000000	0.000000	0.707107	0.707107
1	0.680919	0.517856	0.517856	0.000000
2	0.000000	0.707107	0.000000	0.707107

In [ ]:

```
1
```

(특징 공학)

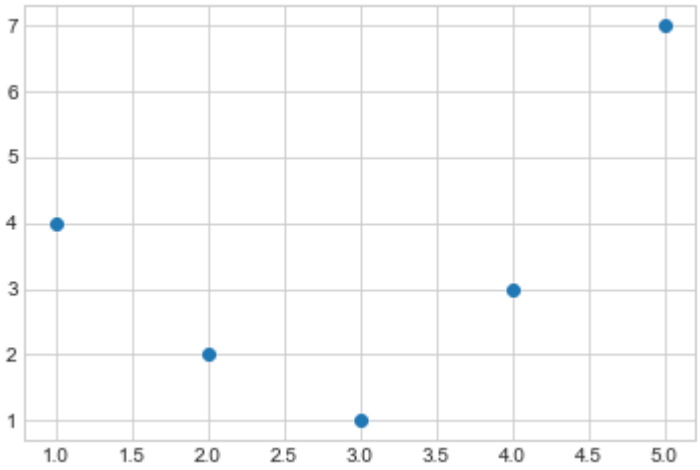
유도 특징의 추가

In [57]:

```
1 #%matplotlib inline
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 x = np.arange(1, 6)
6 y = np.array([4,2,1,3,7])
7 plt.scatter(x,y)
```

Out[57]:

```
<matplotlib.collections.PathCollection at 0x17baeb395c8>
```



In [ ]:

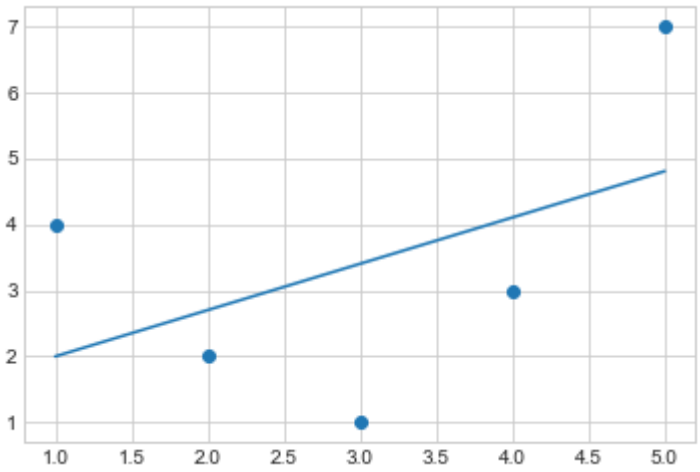
```
1
```

In [58]:

```
from sklearn.linear_model import LinearRegression
X = x.reshape(-1, 1)
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

plt.scatter(x, y)
plt.plot(x, y_pred)
```

Out[58]:

```
[<matplotlib.lines.Line2D at 0x17bac5364c8>]
```



In [ ]:

```

```

In [61]:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=3, include_bias=False)
X2 = poly.fit_transform(X)
print(X2)
```

```
[[  1.   1.   1.]
 [  2.   4.   8.]
 [  3.   9.  27.]
 [  4.  16.  64.]
 [  5.  25. 125.]]
```

In [ ]:

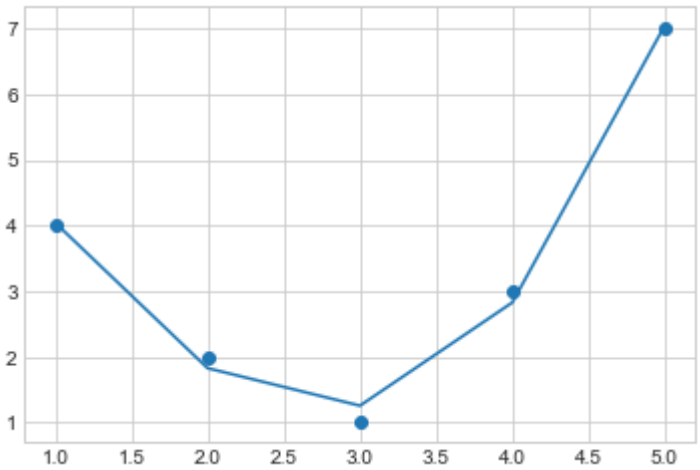
```

```

In [62]:

```
model = LinearRegression()
model.fit(X2, y)
y_pred = model.predict(X2)
plt.scatter(x,y)
plt.plot(x, y_pred)
```

Out[62]:

```
[<matplotlib.lines.Line2D at 0x17baeb770c8>]
```



In [ ]:

```

```

(특징공학)

누락 데이터 대체

In [63]:

```
X = np.array([[np.nan, 0, 3],
              [3, 7, 9],
              [3, 5, 2],
              [4, np.nan, 6],
              [8, 8, 1]])
y = np.array([14, 16, -1, 8, -5])
```

In [68]:

```
from sklearn.preprocessing import Imputer
imp = Imputer(strategy='mean')
X2 = imp.fit_transform(X)
X2
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-68-cb131507e8c7> in <module>
----> 1 from sklearn.preprocessing import Imputer
      2 imp = Imputer(strategy='mean')
      3 X2 = imp.fit_transform(X)
      4 X2

ImportError: cannot import name 'Imputer' from 'sklearn.preprocessing' (C:₩Anaconda
3_64₩lib₩site-packages₩sklearn₩preprocessing₩__init__.py)
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [69]:

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='mean')
X2 = imp.fit_transform(X)
X2
```

Out[69]:

```
array([[4.5, 0. , 3. ],
       [3. , 7. , 9. ],
       [3. , 5. , 2. ],
       [4. , 5. , 6. ],
       [8. , 8. , 1. ]])
```

In [71]:

```
model = LinearRegression()
model.fit(X2, y)
y_pred = model.predict(X2)
y_pred
```

Out[71]:

```
array([13.14869292, 14.3784627 , -1.15539732, 10.96606197, -5.33782027])
```

In [ ]:

```

```

(특징 공학)

특징 파이프 라인

In [72]:

```
from sklearn.pipeline import make_pipeline

model = make_pipeline(Imputer(strategy='mean'),
                      PolynomialFeatures(degree=2),
                      Linearregression())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-72-ed435a1766f2> in <module>
      1 from sklearn.pipeline import make_pipeline
      2
----> 3 model = make_pipeline(Imputer(strategy='mean'),
      4                       PolynomialFeatures(degree=2),
      5                       Linearregression())

NameError: name 'Imputer' is not defined
```

In [ ]:

```

```

In [ ]:

```

```

In [74]:

```
from sklearn.pipeline import make_pipeline

model = make_pipeline(SimpleImputer(strategy='mean'),
                      PolynomialFeatures(degree=2),
                      LinearRegression())
```

In [75]:

```
model.fit(X, y)
```

Out[75]:

```
Pipeline(steps=[('simpleimputer', SimpleImputer()),
                ('polynomialfeatures', PolynomialFeatures()),
                ('linearregression', LinearRegression())])
```

In [76]:

```
print(y)
```

```
[14 16 -1  8 -5]
```

In [77]:

```
print(model.predict(X))
```

```
[14. 16. -1.  8. -5.]
```

In [ ]:

```

```