DUT STID, Université de la Côte d'Azur

Data mining

Régression linéaire

Prof. Dario Malchiodi

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

UNIVERSITÉ CÔTE D'AZUR

In [1]:

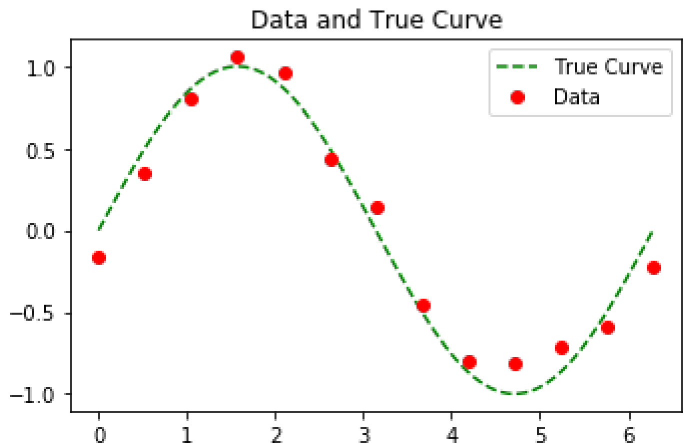
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
def regression_example_points():
    x = np.linspace(0, 2*np.pi, 13);
    # np.random.randn generates gaussian samples
    y = np.sin(x) + np.random.randn(x.shape[0]) * 0.2;
    xx = np.linspace(0, 2*np.pi, 100);
    plt.figure(figsize=(12,7.5))
    plt.subplot(221)
    plt.plot(xx, np.sin(xx), "g", linestyle='--')
    plt.plot(x, y, "or")
    plt.legend(['True Curve','Data'])
    plt.title('Data and True Curve')
    plt.show()
    return x, xx, y
```

In [3]:

```
x, xx, y = regression_example_points()
```



In [4]:

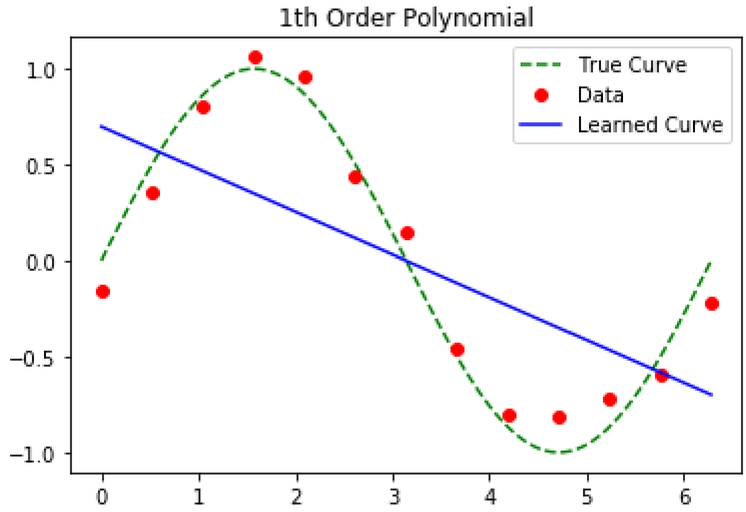
```
from IPython.display import display, Math

def regression_example_draw(x, xx, y, degree, verbose=False):
    coeffs = np.polyfit(x, y, degree)

    poly = np.poly1d(coeffs)
    plt.plot(xx, np.sin(xx), "g", linestyle='--')
    plt.plot(x, y, "or")
    plt.plot(xx, poly(xx), color='b', linestyle='-')
    plt.legend(['True Curve','Data','Learned Curve'])
    plt.title(str(degree)+'th Order Polynomial')

    exprsn=''
    for i in range(degree+1):
        if i==0:
            exprsn += '{:.3f}'.format(coeffs[i])
        if i==1:
            exprsn += '{}{:.3f}x'.format('+' if coeffs[i]>0 else '', coeffs[i])
        elif i>0 and coeffs[i]>0:
            exprsn += '%c%.3fx^{%d}' %('+' if coeffs[i]>0 else '', coeffs[i], i)

    if verbose:
        display(Math(r'\text{The expression for the polynomial is}'))
        display(Math(r'{}'.format(exprsn)))
```

In [5]:

```
regression_example_draw(x, xx, y, 1, True)
```

The expression for the polynomial is

$-0.222 + 0.698x$

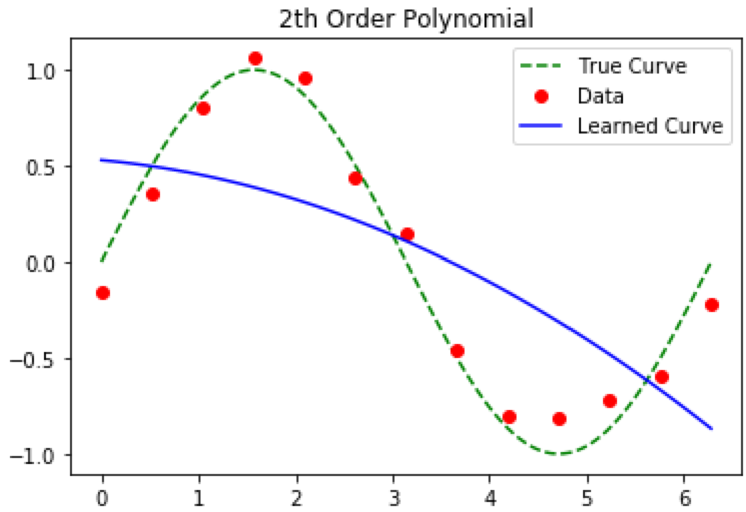


In [6]:

```
regression_example_draw(x, xx, y, 2, True)
```

The expression for the polynomial is

$-0.028 - 0.046x + 0.529x^2$

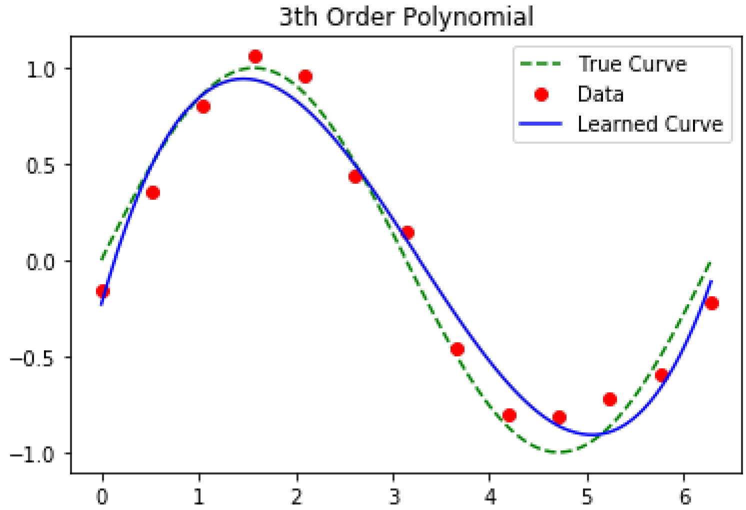


In [7]:

```
regression_example_draw(x, xx, y, 3, True)
```

The expression for the polynomial is

$$0.080 - 0.782x + 1.773x^2$$

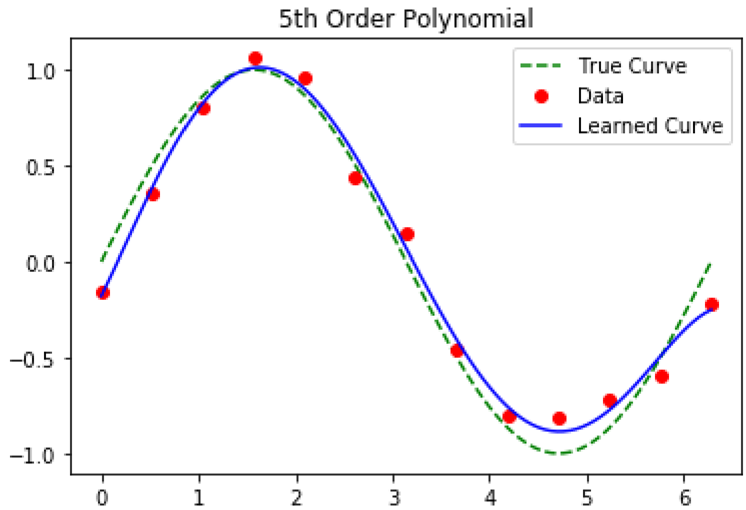


In [8]:

```
regression_example_draw(x, xx, y, 5, True)
```

The expression for the polynomial is

$$-0.006 + 0.098x + 0.297x^3 + 1.021x^4$$

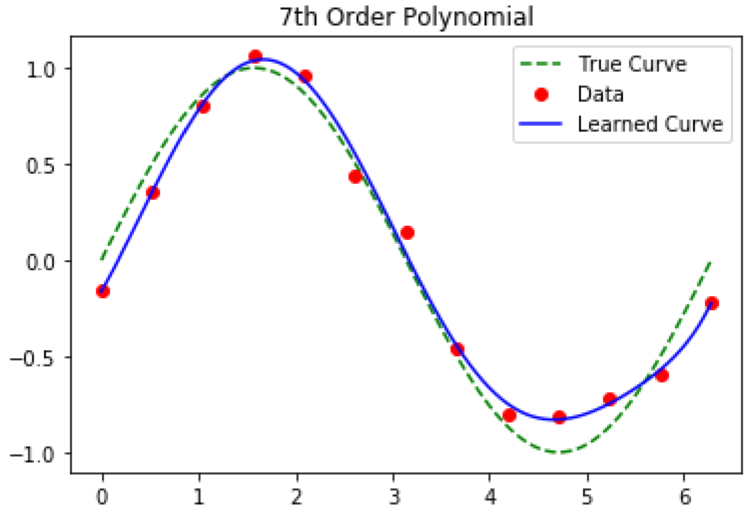


In [9]:

```
regression_example_draw(x, xx, y, 7, True)
```

The expression for the polynomial is

$$0.000 - 0.009x + 0.059x^2 + 0.301x^5 + 0.886x^6$$

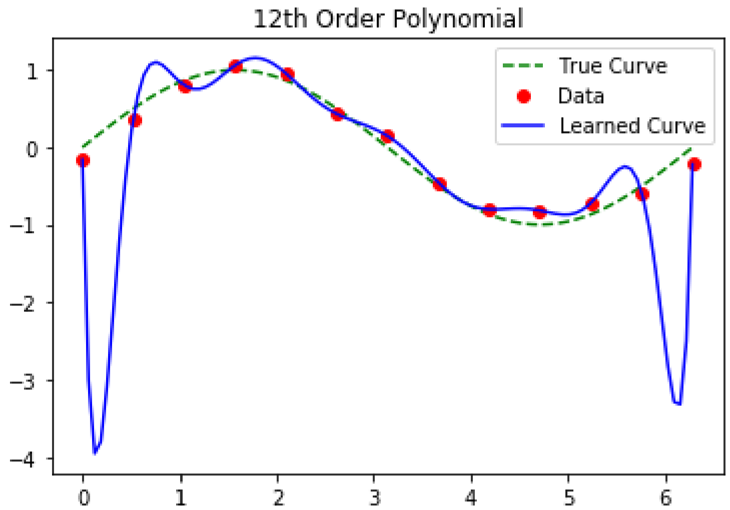


In [10]:

```
regression_example_draw(x, xx, y, 12, True)
```

The expression for the polynomial is

$$0.001 - 0.034x + 0.577x^2 + 35.290x^4 + 426.049x^6 + 1057.751x^8 + 367.456x^{10}$$



In [11]:

```
import sklearn as sk
from sklearn import datasets as ds
boston = ds.load_boston()
```

In [12]:

```
print(boston['DESCR'])
```

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over
25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds r
iver; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1
940
        - DIS      weighted distances to five Boston employment centre
s
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of bla
cks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing (http://archive.ics.uc
i.edu/ml/datasets/Housing)


This dataset was taken from the StatLib library which is maintained at
Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedon
ic
prices and the demand for clean air', J. Environ. Economics & Manageme
nt,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diag
nostics
...', Wiley, 1980.   N.B. Various transformations are used in the tabl
e on
pages 244-261 of the latter.
```

```
The Boston house-price data has been used in many machine learning pap
ers that address regression
problems.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influ
ential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learn
ing. In Proceedings on the Tenth International Conference of Machine L
earning, 236-243, University of Massachusetts, Amherst. Morgan Kaufman
n.
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)
 (http://archive.ics.uci.edu/ml/datasets/Housing))
```

In [13]:

```
x_boston = boston['data']
y_boston = boston['target']
```

In [14]:

```
import seaborn as sns
import pandas as pd

boston_df = pd.DataFrame(x_boston, columns=boston['feature_names'])
boston_df['Price'] = y_boston
boston_df.head()
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

In [15]:

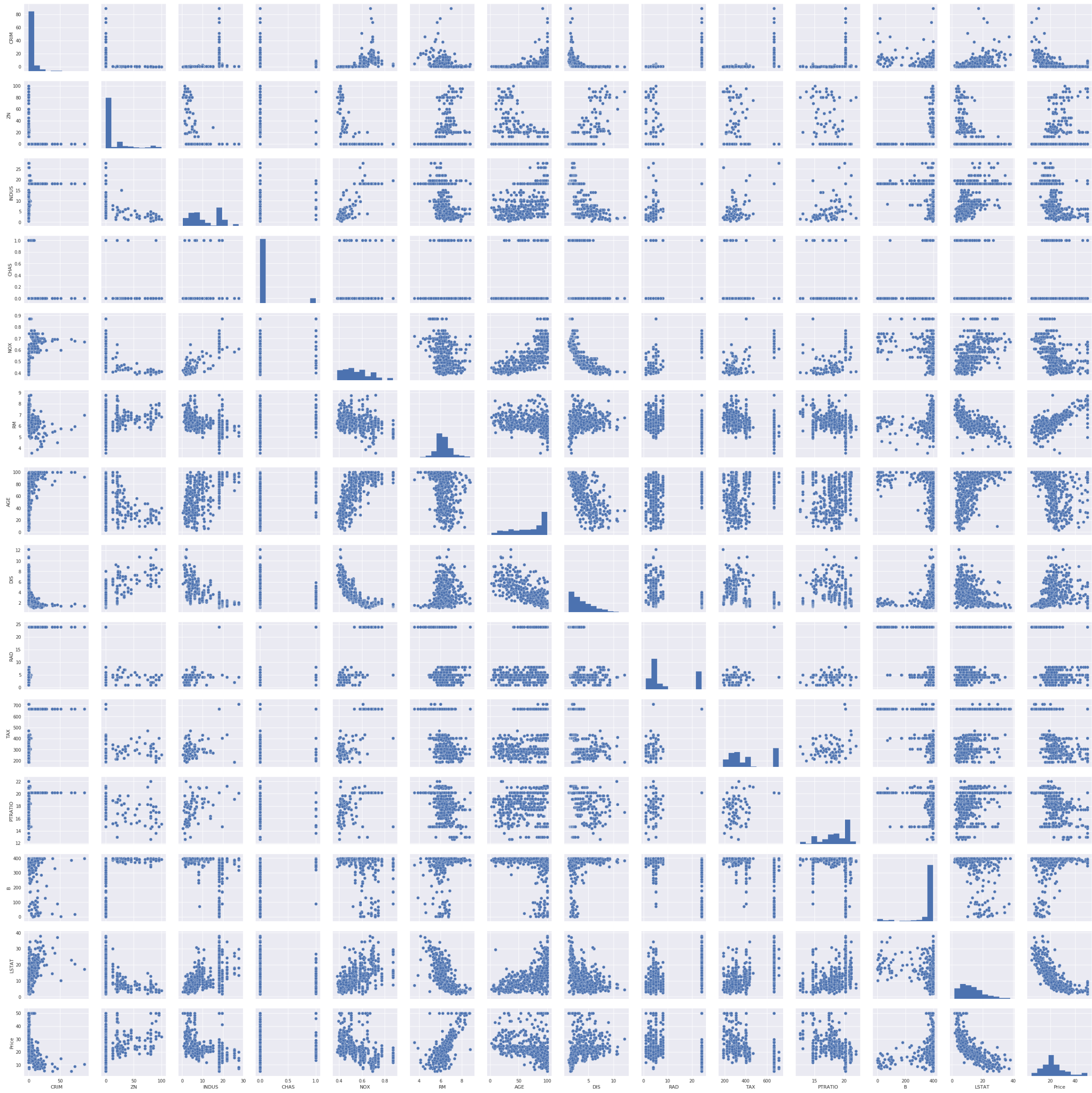
```
boston_df.describe()
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	50
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	1

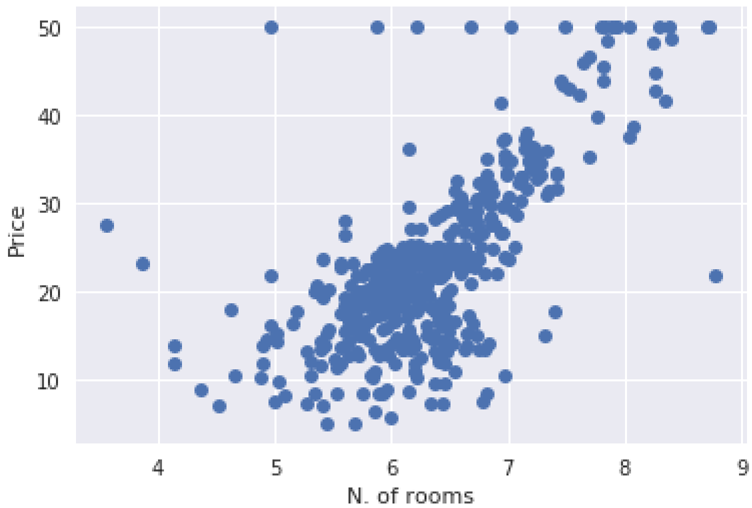
In [16]:

```
sns.set()
sns.pairplot(boston_df)
plt.show()
```



In [17]:

```
rooms = boston_df[['RM']]
prices = boston_df['Price']
plt.plot(rooms, prices, "o")
plt.xlabel('N. of rooms')
plt.ylabel('Price')
plt.show()
```



In [18]:

```
X = np.insert(rooms.values, 1, 1, axis=1)
y = prices

np.linalg.inv((X.T).dot(X)).dot(X.T).dot(y)
```

Out[18]:

```
array([  9.10210898, -34.67062078])
```

In [19]:

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(rooms, prices)
(lr.coef_, lr.intercept_)
```

Out[19]:

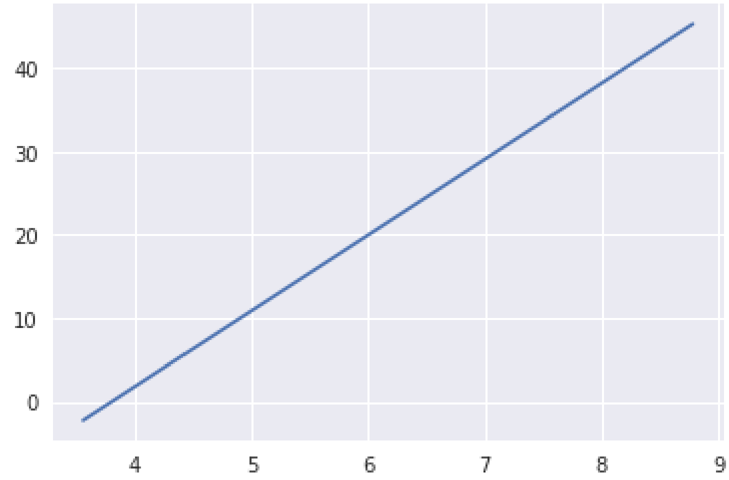
```
(array([9.10210898]), -34.67062077643857)
```

In [20]:

```
room_min = rooms.min()
room_max = rooms.max()
def predict_price(rooms):
    return rooms * lr.coef_[0] + lr.intercept_
```

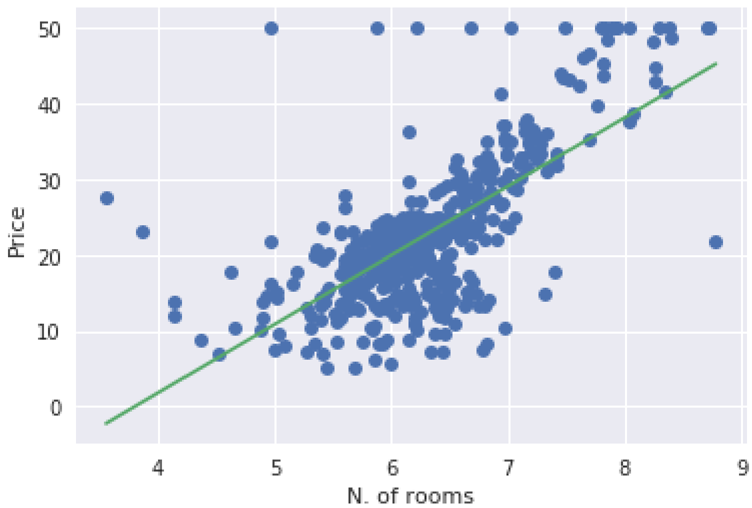
In [21]:

```
plt.plot((room_min, room_max),
         (predict_price(room_min), predict_price(room_max)))
plt.show()
```



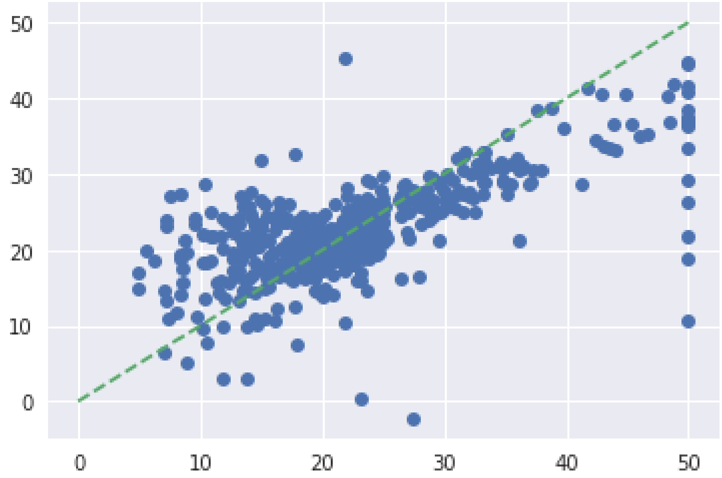
In [22]:

```
plt.plot(rooms, prices, "o")
plt.plot((room_min, room_max),
         (predict_price(room_min), predict_price(room_max)))
plt.xlabel('N. of rooms')
plt.ylabel('Price')
plt.show()
```



In [23]:

```
predicted_prices = lr.predict(rooms)
plt.plot(prices, predicted_prices, "o")
plt.plot([0, 50], [0, 50], "--")
plt.show()
```



In [24]:

```
mse = sk.metrics.mean_squared_error
mse(prices, predicted_prices)
```

Out[24]:

43.60055177116956

In [25]:

```
sum([(y - yhat)**2 for y, yhat in zip(prices, predicted_prices)])/len(prices)
```

Out[25]:

43.60055177116958

In [26]:

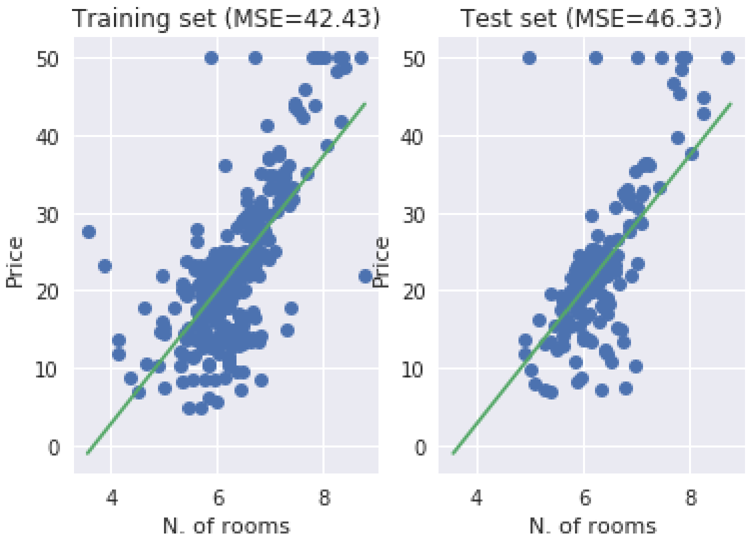
```
from sklearn import model_selection
rooms_train, rooms_test, \
prices_train, prices_test = model_selection.train_test_split(rooms, prices, test_si
                                                              #random_state=42)
```

In [27]:

```
def boston_regression(rooms_train, rooms_test, prices_train, prices_test, regressor
    regressor.fit(rooms_train, prices_train)

    def predict_price(rooms):
        return rooms * regressor.coef_[0] + regressor.intercept_

    train_mse = mse(regressor.predict(rooms_train), prices_train)
    test_mse = mse(regressor.predict(rooms_test), prices_test)

    plt.subplot(121)
    plt.plot(rooms_train, prices_train, "o")
    plt.plot((room_min, room_max),
             (predict_price(room_min), predict_price(room_max)))
    plt.xlabel('N. of rooms')
    plt.ylabel('Price')
    plt.title('Training set (MSE={:.2f})'.format(train_mse))

    plt.subplot(122)
    plt.plot(rooms_test, prices_test, "o")
    plt.plot((room_min, room_max),
             (predict_price(room_min), predict_price(room_max)))
    plt.xlabel('N. of rooms')
    plt.ylabel('Price')
    plt.title('Test set (MSE={:.2f})'.format(test_mse))
    plt.show()
```

In [28]:

```
boston_regression(rooms_train, rooms_test, prices_train, prices_test, lr)
```



In [29]:

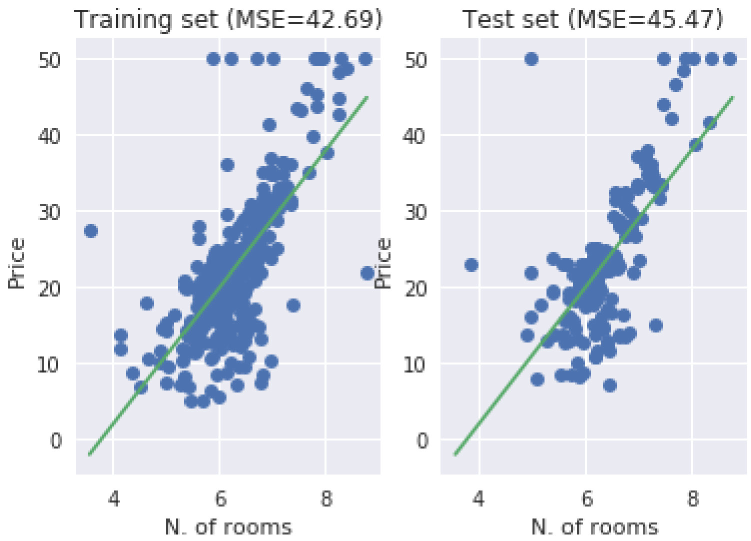
```
num_holdout = 100
mean_train_mse = 0
mean_test_mse = 0

for _ in range(num_holdout):
    rooms_train, rooms_test, \
    prices_train, prices_test = model_selection.train_test_split(rooms, prices, tes
    lr.fit(rooms_train, prices_train)
    mean_train_mse += mse(lr.predict(rooms_train), prices_train)
    mean_test_mse += mse(lr.predict(rooms_test), prices_test)

results = pd.DataFrame([[mean_train_mse/num_holdout, mean_test_mse/num_holdout]],
                       index=['Linear regression (RM)'],
                       columns=['Train MSE', 'Test MSE'])
results
```

Out[29]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524

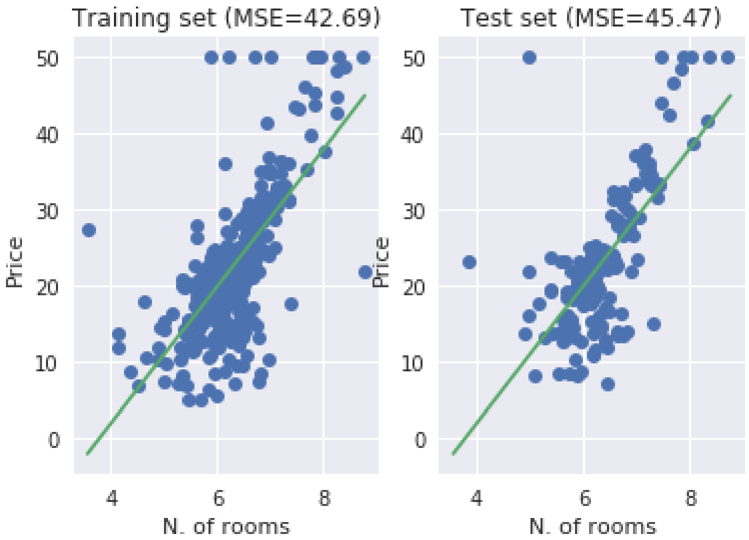
In [30]:

```
from sklearn import linear_model
rlr = linear_model.Ridge(alpha=0.5)
boston_regression(rooms_train, rooms_test, prices_train, prices_test, rlr)
```



In [31]:

```
rlr = linear_model.Ridge(alpha=.1)
boston_regression(rooms_train, rooms_test, prices_train, prices_test, rlr)
```

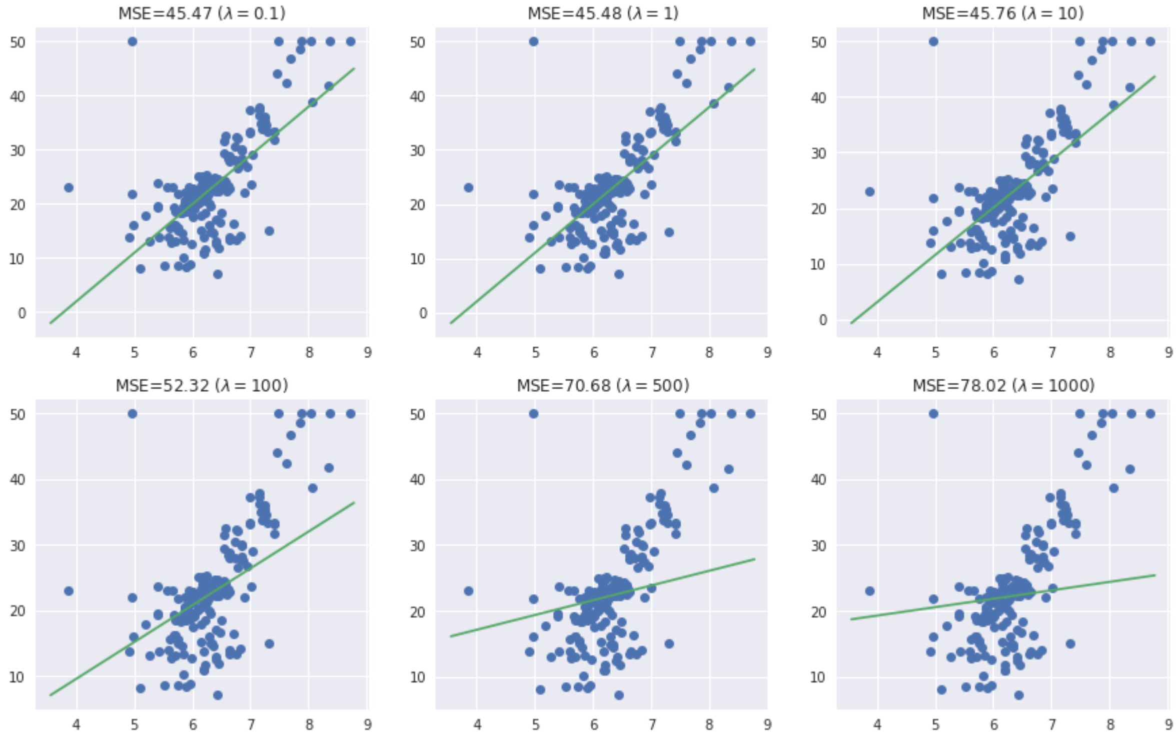


In [32]:

```
def ridge_experiment(alpha):
    regressor = linear_model.Ridge(alpha=alpha)
    regressor.fit(rooms_train, prices_train)
    test_mse = mse(regressor.predict(rooms_test), prices_test)

    def predict_price(rooms):
        return rooms * regressor.coef_[0] + regressor.intercept_

    plt.plot(rooms_test, prices_test, "o")
    plt.plot((room_min, room_max),
             (predict_price(room_min), predict_price(room_max)))

    plt.title('MSE={:.2f} ($\\lambda={}$)'.format(test_mse, alpha))
```

In [33]:

```
plt.figure(figsize=(15, 5))
plt.subplots_adjust(top = 1.5, bottom=0.1, hspace=0.2, wspace=0.2)
plt.subplot(231)
ridge_experiment(.1)
plt.subplot(232)
ridge_experiment(1)
plt.subplot(233)
ridge_experiment(10)
plt.subplot(234)
ridge_experiment(100)
plt.subplot(235)
ridge_experiment(500)
plt.subplot(236)
ridge_experiment(1000)
```



In [34]:

```
def cv_ridge_experiment(rooms_train, rooms_test, prices_train, prices_test):
    reg = linear_model.RidgeCV(alphas=[0.1, 1, 10, 50], cv=3)
    reg.fit(rooms_train, prices_train)
    return (reg.alpha_, mse(prices_test, reg.predict(rooms_test)))
```

In [35]:

```
cv_ridge_experiment(rooms_train, rooms_test, prices_train, prices_test)
```

Out[35]:

```
(1, 45.48184965121321)
```

In [36]:

```
def regression_experiment(label, regressor, xs, ys, num_holdout=100):
    mean_train_mse = 0
    mean_test_mse = 0

    for _ in range(num_holdout):
        x_train, x_test, \
        y_train, y_test = model_selection.train_test_split(xs, ys, test_size = 0.33
        regressor.fit(x_train, y_train)
        mean_train_mse += mse(regressor.predict(x_train), y_train)
        mean_test_mse += mse(regressor.predict(x_test), y_test)


    new_row = pd.DataFrame([[mean_train_mse/num_holdout, mean_test_mse/num_holdout]
                           index=[label],
                           columns=['Train MSE', 'Test MSE'])
    return new_row
```

In [37]:

```
ridge = linear_model.RidgeCV(alphas=[0.1, 1, 10, 50], cv=3)
new_row = regression_experiment('Ridge regression (RM)', ridge, rooms, prices)

results = pd.concat((results, new_row))
results
```

Out[37]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524
Ridge regression (RM)	43.648515	44.003620

In [38]:

```
lasso = linear_model.LassoCV(alphas=[0.1, 1, 10, 50], cv=3)
new_row = regression_experiment('LASSO regression (RM)', lasso, rooms, prices)

results = pd.concat((results, new_row))
results
```

Out[38]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524
Ridge regression (RM)	43.648515	44.003620
LASSO regression (RM)	43.773153	43.640314

In [39]:

```
gd = linear_model.SGDRegressor(max_iter=1000, tol=1e-3, loss='squared_loss')
new_row = regression_experiment('Gradient descent (RM)', gd, rooms, prices)

results = pd.concat((results, new_row))
results
```

Out[39]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524
Ridge regression (RM)	43.648515	44.003620
LASSO regression (RM)	43.773153	43.640314
Gradient descent (RM)	59.127410	58.948480

In [40]:

```
x_boston = boston_df.iloc[:,:-1]
y_boston = boston_df.iloc[:,-1]
```

In [41]:

```
lr_all = LinearRegression()
new_row = regression_experiment('Linear regression (all features)', lr_all, x_bosto

results = pd.concat((results, new_row))
results
```

Out[41]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524
Ridge regression (RM)	43.648515	44.003620
LASSO regression (RM)	43.773153	43.640314
Gradient descent (RM)	59.127410	58.948480
Linear regression (all features)	21.394194	24.302297

In [53]:

```
ridge = linear_model.RidgeCV(alphas=[0.1, 1, 10, 50], cv=3)
new_row = regression_experiment('Ridge regression (all features)', ridge, x_boston,

results = pd.concat((results, new_row))
results
```

Out[53]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524
Ridge regression (RM)	43.648515	44.003620
LASSO regression (RM)	43.773153	43.640314
Gradient descent (RM)	59.127410	58.948480
Linear regression (all features)	21.394194	24.302297
Ridge regression (all features)	21.407517	25.214320
LASSO regression (all features)	22.682098	25.218771
Linear regression (extended features)	5.938976	17.696634
Ridge regression (extended features)	6.443780	16.857679
Ridge regression (all features)	21.353100	25.155228

In [54]:

```
lasso = linear_model.LassoCV(alphas=[0.1, 1, 10, 50], cv=3)
new_row = regression_experiment('LASSO regression (all features)', lasso, x_boston,

results = pd.concat((results, new_row))
results
```

Out[54]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524
Ridge regression (RM)	43.648515	44.003620
LASSO regression (RM)	43.773153	43.640314
Gradient descent (RM)	59.127410	58.948480
Linear regression (all features)	21.394194	24.302297
Ridge regression (all features)	21.407517	25.214320
LASSO regression (all features)	22.682098	25.218771
Linear regression (extended features)	5.938976	17.696634
Ridge regression (extended features)	6.443780	16.857679
Ridge regression (all features)	21.353100	25.155228
LASSO regression (all features)	22.522220	25.452278

In [44]:

```
gd = linear_model.SGDRegressor(max_iter=1000, tol=1e-3, loss='squared_loss')
new_row = regression_experiment('Gradient descent (all features)', gd, x_boston, y_

new_row
```

Out[44]:

	Train MSE	Test MSE
Gradient descent (all features)	4.955369e+28	4.981200e+28

In [45]:

```
gd = linear_model.SGDRegressor(max_iter=1000, tol=1e-3, loss='squared_loss', alpha=
new_row = regression_experiment('Gradient descent (extended features)', gd, x_bosto

new_row
```

Out[45]:

	Train MSE	Test MSE
Gradient descent (extended features)	80.758405	80.689981

In [46]:

```
gd = linear_model.SGDRegressor(max_iter=100000, tol=1e-7, loss='epsilon_insensitive
new_row = regression_experiment('Gradient descent (extended features)', gd, x_bosto

new_row
```

Out[46]:

	Train MSE	Test MSE
Gradient descent (extended features)	84.090035	84.616345

In [57]:

```
v = [1, 2, 3]
[e+f for e in v for f in v]
```

Out[57]:

```
[2, 3, 4, 3, 4, 5, 4, 5, 6]
```

In [58]:

```
def extract_features(x):
    return [e_1*e_2 for e_1 in x for e_2 in x]
```

In [59]:

```
extract_features([1, 2, 3])
```

Out[59]:

```
[1, 2, 3, 2, 4, 6, 3, 6, 9]
```

In [60]:

```
new_x_boston = [extract_features(x) for x in x_boston.values]
```

In [61]:

```
new_x_boston[0]
```

Out[61]:

```
[3.99424e-05,
 0.11376,
 0.014599200000000001,
 0.0,
 0.00340016,
 0.041554,
 0.41206400000000004,
 0.025848799999999998,
 0.00632,
 1.87072,
 0.096696,
 2.5084079999999997,
 0.031473600000000004,
 0.11376,
 324.0,
 41.58,
 0.0,
 9.684000000000001,
```

In [50]:

```
lr_all = LinearRegression()
new_row = regression_experiment('Linear regression (extended features)',
                                lr_all, new_x_boston, y_boston)

results = pd.concat((results, new_row))
results
```

Out[50]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524
Ridge regression (RM)	43.648515	44.003620
LASSO regression (RM)	43.773153	43.640314
Gradient descent (RM)	59.127410	58.948480
Linear regression (all features)	21.394194	24.302297
Ridge regression (all features)	21.407517	25.214320
LASSO regression (all features)	22.682098	25.218771
Linear regression (extended features)	5.938976	17.696634

In [51]:

```
ridge = linear_model.RidgeCV(alphas=[0.1, 1, 10, 50], cv=3)
new_row = regression_experiment('Ridge regression (extended features)', ridge, new_

results = pd.concat((results, new_row))
results
```

Out[51]:

	Train MSE	Test MSE
Linear regression (RM)	42.973596	45.315524
Ridge regression (RM)	43.648515	44.003620
LASSO regression (RM)	43.773153	43.640314
Gradient descent (RM)	59.127410	58.948480
Linear regression (all features)	21.394194	24.302297
Ridge regression (all features)	21.407517	25.214320
LASSO regression (all features)	22.682098	25.218771
Linear regression (extended features)	5.938976	17.696634
Ridge regression (extended features)	6.443780	16.857679

In [52]:

```
gd = linear_model.SGDRegressor(max_iter=1000, tol=1e-7, learning_rate='constant', e
new_row = regression_experiment('Gradient descent (extended features)', gd, new_x_b

new_row
```

Out[52]:

	Train MSE	Test MSE
Gradient descent (extended features)	8.151851e+29	8.123493e+29

In [ ]:

```

```