Lab: Simple Linear Regression on the Boston Housing Data

In this lab, you will learn to:

- load data from a CSV file
- plot data
- perform simple mathematical manipulations,
- fit a simple linear regression model.

Before doing this lab, you can go through the demo (./auto_mpg.ipynb) to see an example of these operations on an automobile dataset. This lab use the Boston housing data set, a widely-used machine learning data set for illustrating basic concepts.

Loading the data

The Boston housing data set was collected in the 1970s to study the relationship between house price and various factors such as the house size, crime rate, socio-economic status, etc. Since the variables are easy to understand, the data set is ideal for learning basic concepts in machine learning. The raw data and a complete description of the dataset can be found on the UCI website:

https://archive.ics.uci.edu/ml/datasets/Housing (https://archive.ics.uci.edu/ml/datasets/Housing)

In the lab, you will complete all the code marked `TODO`.

First, complete the following code that uses the `pd.read_csv` command to read the data from the file located at

https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data (https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data)

I have supplied a list `names` of the column headers. You will have to set the options in the `read_csv` command to correctly delimit the data in the file and name the columns correctly.

In [1]:
```
import pandas as pd
import numpy as np
names =[
    'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
    'AGE',  'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'PRICE'
]

# TODO:  Complete the code
# df = pd.read_csv(...)
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/'+
                 'housing/housing.data',
                 header=None,delim_whitespace=True,names=names)
```

Display the first six rows of the data frame

In [2]:
```
# TODO
df.head(6)
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222.0	18.7	394.12

Basic Manipulations on the Data

What is the shape of the data? How many attributes are there? How many samples? Print a statement of the form:

```
num samples=xxx, num attributes=yy
```

In [3]:
```
# TODO
nsamp, natt = df.shape
print('num samples={0:d}, num attributes={1:d}'.format(nsamp,natt))
```

```
num samples=506, num attributes=14
```

Create a response vector y with the values in the column `PRICE`. The vector y should be a 1D `numpy.array` structure.

In [4]:
```
# TODO
# y = ...
y = np.array(df['PRICE'])
```

Use the response vector y to find the mean house price in thousands and the fraction of homes that are above $40k. (You may realize this is very cheap. Prices have gone up a lot since the 1970s!). Create print statements of the form:

```
The mean house price is xx.yy thousands of dollars.
Only x.y percent are above $40k.
```

In [5]:
```
# TODO
print('The mean house price is {0:5.2f} thousands of dollars'.format(np.mean(y
)))
print('Only {0:.1f} per cent are above $40k '.format(np.mean(y>40)*100))
```

```
The mean house price is 22.53 thousands of dollars
Only 6.1 per cent are above $40k
```

Visualizing the Data

Python's `matplotlib` has very good routines for plotting and visualizing data that closely follows the format of MATLAB programs. You can load the `matplotlib` package with the following commands.

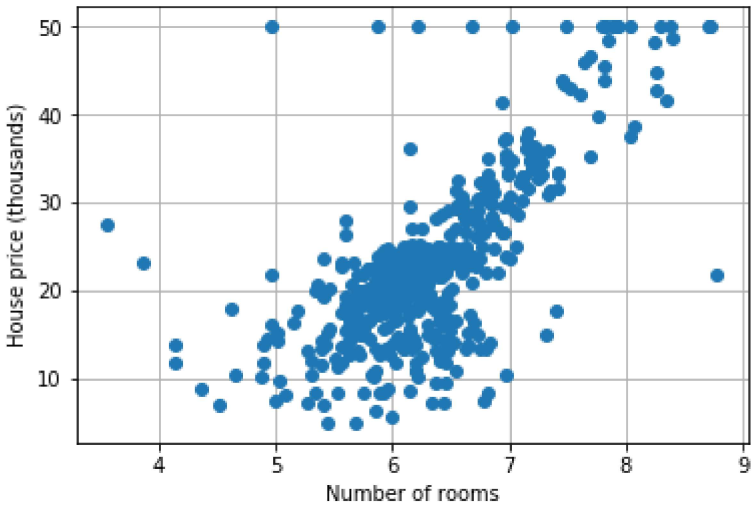
In [6]:
```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

Similar to the y vector, create a predictor vector x containing the values in the `RM` column, which represents the average number of rooms in each region.

In [7]:
```
# TODO
# x = ...
x = np.array(df['RM'])
```

Create a scatter plot of the price vs. the `RM` attribute. Make sure your plot has grid lines and label the axes with reasonable labels so that someone else can understand the plot.

In [8]:
```
# TODO
plt.plot(x,y,'o')
plt.xlabel('Number of rooms')
plt.ylabel('House price (thousands)')
plt.grid(True)
```



Fitting a Simple Linear Model

We will write a simple function to perform a linear fit. Use the formulae given in the class, to compute the parameters β_0, β_1 in the linear model

$$y = \beta_0 + \beta_1 x + \epsilon$$

as well as the coefficient of determination R^2.

In [9]:
```
def fit_linear(x,y):
    """
    Given vectors of data points (x,y), performs a fit for the linear model:
       yhat = beta0 + beta1*x,
    The function returns beta0, beta1 and rsq, where rsq is the coefficient of
 determination.
    """
    # TODO complete the following code
    xm = np.mean(x)
    ym = np.mean(y)
    sxx = np.mean((x-xm)**2)
    sxy = np.mean((x-xm)*(y-ym))
    syy = np.mean((y-ym)**2)
    beta1 = sxy/sxx
    beta0 = ym - beta1*xm
    rsq = sxy**2/sxx/syy
    return beta0, beta1, rsq
```

Using the function `fit_linear` above, print the values `beta0`, `beta1` and `rsq` for the linear model of price vs. number of rooms.

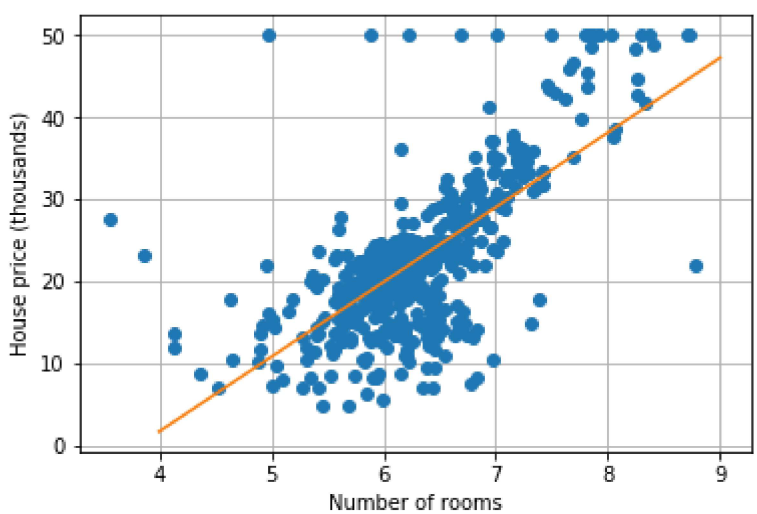
In [10]:
```
# TODO
beta0, beta1, rsq = fit_linear(x,y)
print("beta0={0:5.3f}, beta1={1:5.3f}, rsq={2:5.3f}".format(beta0,beta1,rsq))
```

```
beta0=-34.671, beta1=9.102, rsq=0.484
```

Replot the scatter plot above, but now with the regression line. You can create the regression line by creating points xp from say 4 to 9, computing the linear predicted values yp on those points and plotting yp vs. xp on top of the above plot.

In [11]:
```
# TODO
xp = np.linspace(4,9,100)
yp = beta0 + beta1*xp
```

In [12]:
```
# TODO
plt.plot(x,y,'o')
plt.plot(xp,yp,'-')
plt.xlabel('Number of rooms')
plt.ylabel('House price (thousands)')
plt.grid(True)
```



Compute coefficients of determination

We next compute the R^2 values for all the predictors and output the values in a table. Your table should look like the following, where each the first column is the attribute name and the second column is the R^2 value.

```
CRIM        0.151
ZN          0.130
INDUS       0.234
...         ...
```

To index over the set of colunms in the dataframe `df`, you can either loop over the items in the `names` lists (skipping over the final name `PRICE`) or loop over integer indices and use the method, `df.iloc`.

I will first show the method looping directly over the names in `names`. Note that the for loop in python can directly loop over any set of elements in a container.

In [13]:
```
# TODO
# Loop over names.
for name in names:

    # Skip over the case where the attribute is the target variable.
    if name != 'PRICE':
        # compute the r^2 value for the predictor
        x = np.array(df[name])
        beta0,beta1,rsq = fit_linear(x,y)

        # print the value.  note the syntax to format the string
        print('{0:10}  {1:.3f}'.format(name, rsq))
```

```
CRIM        0.151
ZN          0.130
INDUS       0.234
CHAS        0.031
NOX         0.183
RM          0.484
AGE         0.142
DIS         0.062
RAD         0.146
TAX         0.220
PTRATIO     0.258
B           0.111
LSTAT       0.544
```

Now, here is an implementation using a for-loop with an integer index. You can do either and get full credit.

In [14]:
```
p = len(names)-1       # number of predictors
for ind in range(p):

    # compute the r^2 value for the predictor
    x = np.array(df.iloc[:,ind])
    beta0,beta1,rsq = fit_linear(x,y)


    # print the value.  note the syntax to format the string
    print('{0:10}  {1:.3f}'.format(names[ind], rsq))
```

```
CRIM        0.151
ZN          0.130
INDUS       0.234
CHAS        0.031
NOX         0.183
RM          0.484
AGE         0.142
DIS         0.062
RAD         0.146
TAX         0.220
PTRATIO     0.258
B           0.111
LSTAT       0.544
```