Supervised Learning-Linear Regression

October 22, 2019

```
In [1]: from sklearn import datasets
        iris = datasets.load_iris()
```

```
In [4]: print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
In [7]: print(iris.target)
        print(iris.target_names)
```

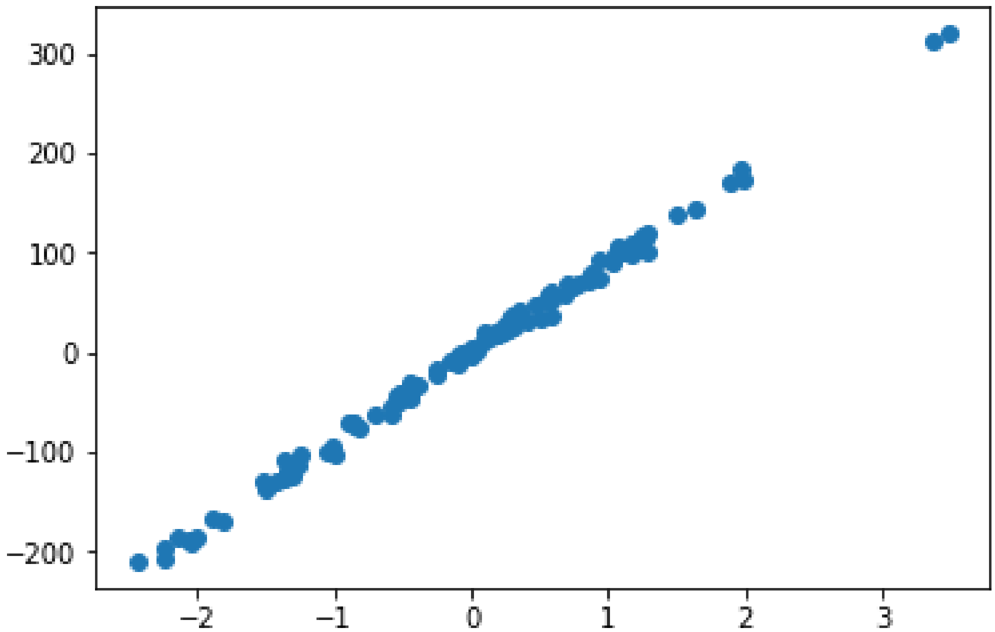
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
```

```
In [8]: import pandas as pd
        df = pd.DataFrame(iris.data)

        print(df.head())
```

```
     0    1    2    3
0  5.1  3.5  1.4  0.2
1  4.9  3.0  1.4  0.2
2  4.7  3.2  1.3  0.2
3  4.6  3.1  1.5  0.2
4  5.0  3.6  1.4  0.2
```

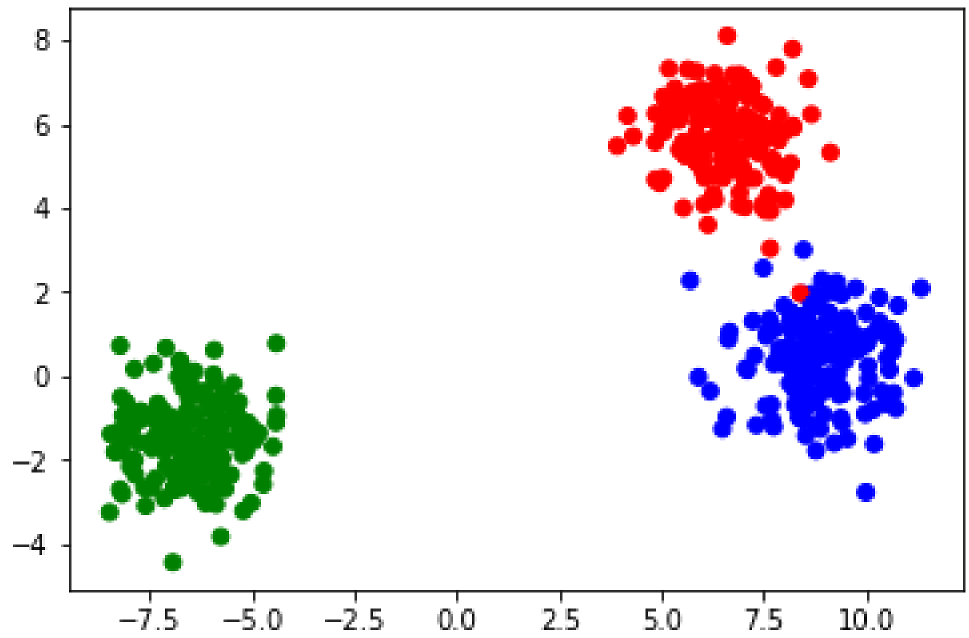
```
In [9]: %matplotlib inline
        from matplotlib import pyplot as plt
        from sklearn.datasets.samples_generator import make_regression

        X, y = make_regression(n_samples=100, n_features=1, noise=5.4)
        plt.scatter(X,y)
```

Out[9]: <matplotlib.collections.PathCollection at 0x21831584208>

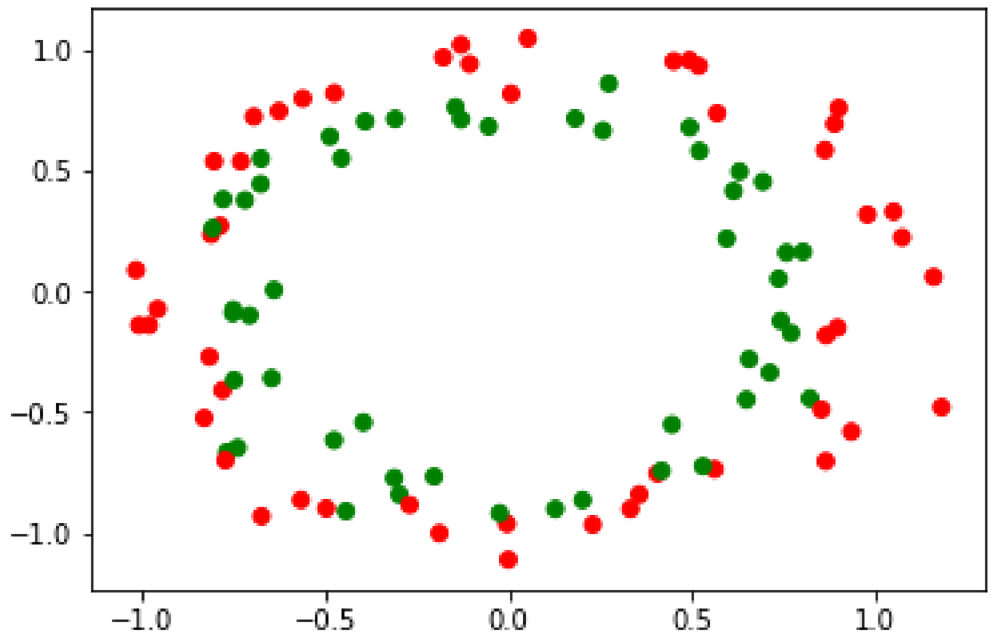


In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_blobs

X, y = make_blobs(500, centers=3)
rgb = np.array(['r', 'g', 'b'])
plt.scatter(X[:, 0], X[:, 1], color=rgb[y])
```

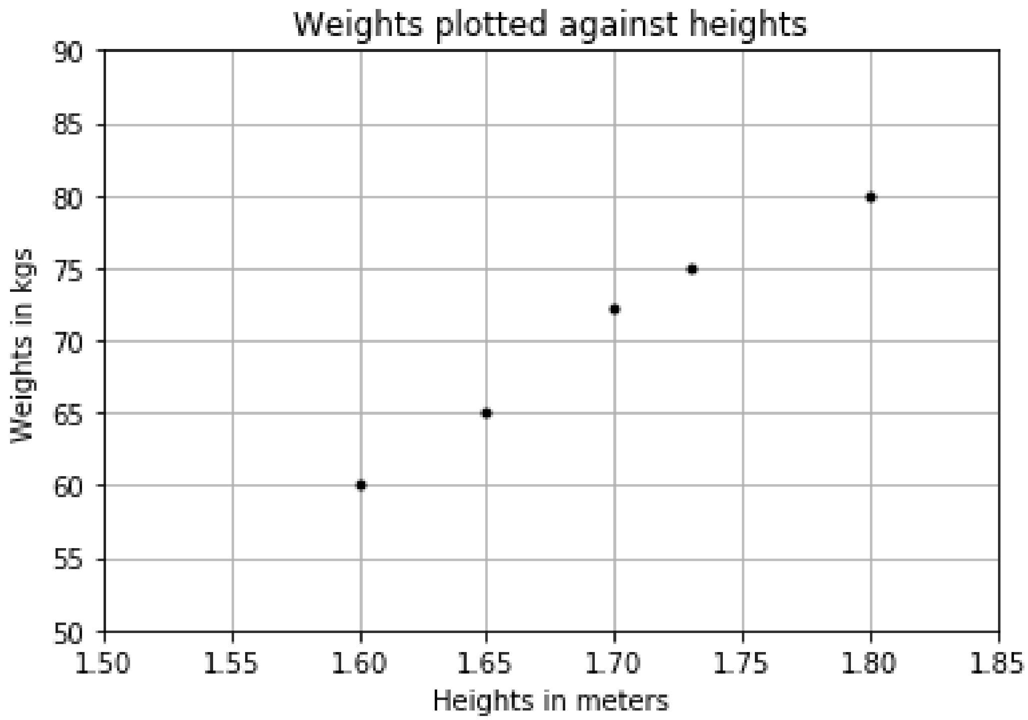
Out[10]: <matplotlib.collections.PathCollection at 0x218325eb908>




```
In [13]: %matplotlib inline
         import matplotlib.pyplot as plt
         import numpy as np
         from sklearn.datasets import make_circles

         X, y = make_circles(n_samples=100, noise=0.09)

         rgb=np.array(['r', 'g', 'b'])
         plt.scatter(X[:, 0], X[:, 1], color=rgb[y])

Out[13]: <matplotlib.collections.PathCollection at 0x2183268f9b0>
```



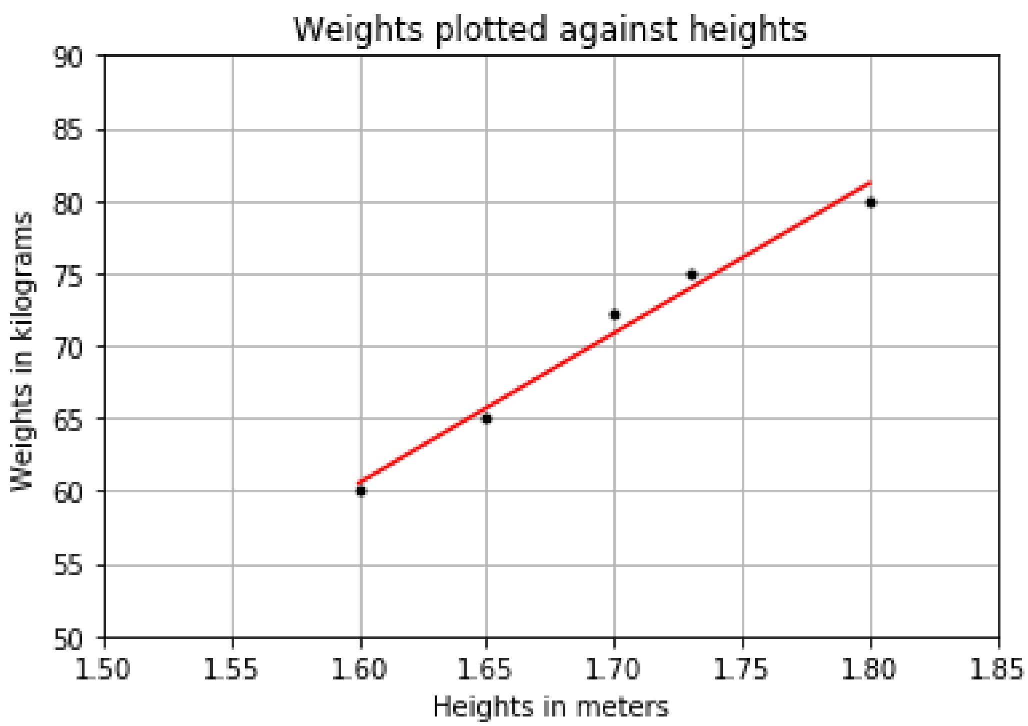
In [15]:
```
%matplotlib inline
import matplotlib.pyplot as plt
# represents the heights of a group of people in meters
heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]

# represents the weights of a group of people in kgs
weights = [[60], [65], [72.3], [75], [80]]

plt.title('Weights plotted against heights')
plt.xlabel('Heights in meters')
plt.ylabel('Weights in kgs')

plt.plot(heights, weights, 'k.')

# axis range for x and y
plt.axis([1.5, 1.85, 50, 90])
plt.grid(True)
```




```
In [16]: from sklearn.linear_model import LinearRegression
         # Create and fit the model
         model = LinearRegression()
         model.fit(X=heights, y=weights)

Out[16]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)

In [17]: # make prediction
         weight = model.predict([[1.75]])[0][0]
         print(round(weight,2))

76.04

In [18]: import matplotlib.pyplot as plt

         heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]
         weights = [[60], [65], [72.3], [75], [80]]

         plt.title('Weights plotted against heights')
         plt.xlabel('Heights in meters')
         plt.ylabel('Weights in kilograms')
```

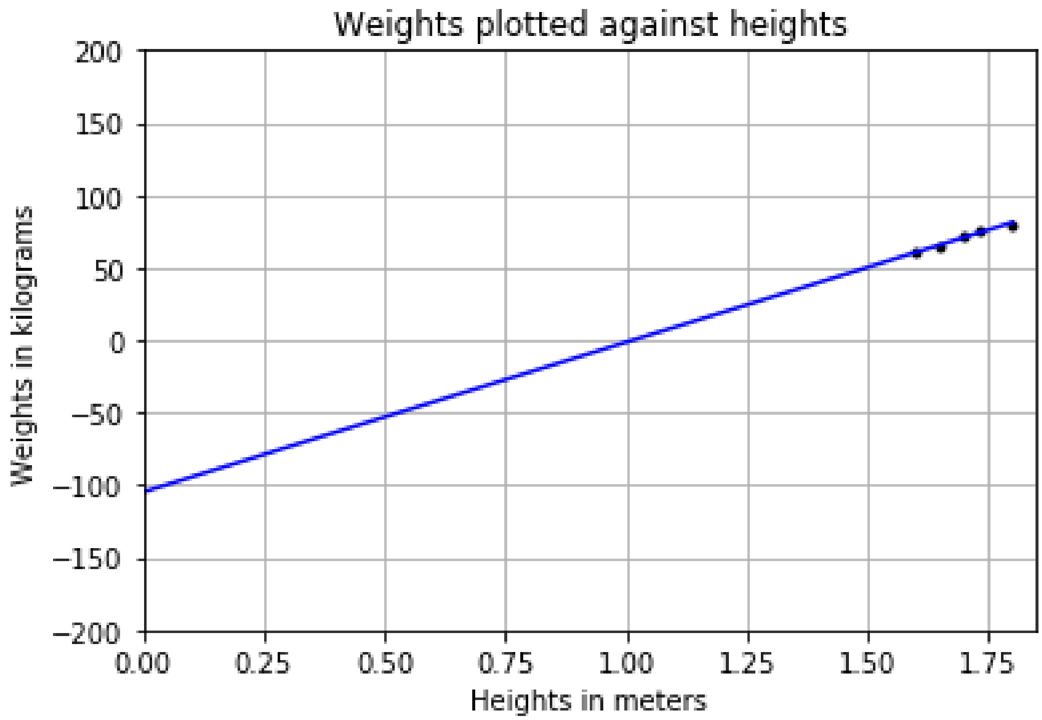
```
plt.plot(heights, weights, 'k.')

plt.axis([1.5, 1.85, 50, 90] )
plt.grid(True)

#plt the regression line
plt.plot(heights, model.predict(heights), color='r')
```

Out[18]: [<matplotlib.lines.Line2D at 0x2183273c080>]



In [22]:
```
plt.title('Weights plotted against heights')
plt.xlabel('Heights in meters')
plt.ylabel('Weights in kilograms')

plt.plot(heights, weights, 'k.')

plt.axis([0, 1.85, -200, 200])
plt.grid(True)
#plot the regression line
extreme_heights = [[0], [1.8]]
plt.plot(extreme_heights, model.predict(extreme_heights), color='b')
```

Out[22]: [<matplotlib.lines.Line2D at 0x21832b232b0>]




```
In [24]: round(model.predict([[0]])[0][0],2)
```

```
Out[24]: -104.75
```

```
In [25]: print(round(model.intercept_[0],2))
```

```
-104.75
```

```
In [26]: print(round(model.coef_[0][0],2))
```

```
103.31
```

```
In [27]: import numpy as np

         print('Residual sum of squares: %.2f' % np.sum((weights-model.predict(heights))**2))
```

```
Residual sum of squares: 5.34
```

```
In [29]: heights_test = [[1.58], [1.62], [1.69], [1.76], [1.82]]
         weights_test = [[58], [63], [72], [73], [85]]

         #Total Sum of squares (TSS)
```

```
weights_test_mean = np.mean(np.ravel(weights_test))
TSS = np.sum((np.ravel(weights_test)-weights_test_mean)**2)
print("TSS: %.2f" % TSS)
#Residual Sum of Squares (RSS)
RSS = np.sum((np.ravel(weights_test)-np.ravel(model.predict(heights_test)))**2)
print("RSS: %.2f" % RSS)

# R_squared
R_squared = 1-(RSS/TSS)
print("R-squared: %.2f" % R_squared)
```

```
TSS: 430.80
RSS: 24.62
R-squared: 0.94
```

In [30]:
```
# using scikit-learn to calculates r-squared
print('R-squared: %.4f' % model.score(heights_test, weights_test))
```

```
R-squared: 0.9429
```

In [31]:
```
import pickle
```

In [32]:
```
# save the model to disk
filename = 'HeightsAndWeights_model.sav'
# write to the file using write and binary mode
pickle.dump(model, open(filename, 'wb'))
```

In [33]:
```
# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(heights_test, weights_test)
```

In [34]:
```
print(result)
```

```
0.9428592885995253
```

In [35]:
```
from sklearn.externals import joblib
```

In [36]:
```
# save the model to disk
filename = 'HeightsAndWeights_model2.sav'
joblib.dump(model, filename)
# load the model from disk
loaded_model = joblib.load(filename)
result = loaded_model.score(heights_test, weights_test)
print(result)
```

```
0.9428592885995253
```

```
In [37]: # Clearing Rows with NaNs
         import pandas as pd
         df = pd.read_csv('NaNDataset.csv')
         df.isnull().sum()
```

```
Out[37]: A    0
         B    2
         C    0
         dtype: int64
```

```
In [38]: # replaces all the NaNs in column B with the average of column B
         df.B = df.B.fillna(df.B.mean())
         print(df)
```

```
    A     B   C
0   1   2.0   3
1   4  11.0   6
2   7  11.0   9
3  10  11.0  12
4  13  14.0  15
5  16  17.0  18
```

```
In [39]: df = pd.read_csv('NaNDataset.csv')
         df = df.dropna()
         print(df)
```

```
    A     B   C
0   1   2.0   3
3  10  11.0  12
4  13  14.0  15
5  16  17.0  18
```

```
In [41]: df = df.reset_index(drop=True)
         print(df)
```

```
    A     B   C
0   1   2.0   3
1  10  11.0  12
2  13  14.0  15
3  16  17.0  18
```

```
In [45]: import pandas as pd
         df = pd.read_csv('DuplicateRows.csv')
         print(df.duplicated(keep=False))
```

```
0     False
1      True
```

```
2     True
3    False
4    False
5     True
6     True
7    False
8    False
dtype: bool
```

```
In [47]: print(df[df.duplicated(keep=False)])
```

```
    A   B   C
1   4   5   6
2   4   5   6
5  10  11  12
6  10  11  12
```

```
In [51]: df.drop_duplicates(keep='first', inplace = True)
         print(df)
```

```
    A   B   C
0   1   2   3
1   4   5   6
3   7   8   9
4   7  18   9
5  10  11  12
7  13  14  15
8  16  17  18
```

```
In [53]: df.drop_duplicates(subset=['A', 'C'], keep='last', inplace = True)
         print(df)
```

```
    A   B   C
0   1   2   3
1   4   5   6
4   7  18   9
5  10  11  12
7  13  14  15
8  16  17  18
```

```
In [55]: import pandas as pd
         from sklearn import preprocessing

In [56]: df = pd.read_csv('NormalizeColumns.csv')
```

In [57]:
```
x=df.values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df=pd.DataFrame(x_scaled, columns=df.columns)
print(df)
```

```
     A         B    C
0  0.6  0.000000  0.0
1  0.2  0.200000  0.2
2  0.4  0.266667  0.4
3  0.0  0.600000  0.6
4  0.8  0.800000  0.8
5  1.0  1.000000  1.0
```

In [63]:
```
import pandas as pd
df = pd.read_csv("http://www.mosaic-web.org/go/datasets/galton.csv")
print(df.head(20))
```

```
   family  father  mother sex  height  nkids
0       1    78.5    67.0   M    73.2      4
1       1    78.5    67.0   F    69.2      4
2       1    78.5    67.0   F    69.0      4
3       1    78.5    67.0   F    69.0      4
4       2    75.5    66.5   M    73.5      4
5       2    75.5    66.5   M    72.5      4
6       2    75.5    66.5   F    65.5      4
7       2    75.5    66.5   F    65.5      4
8       3    75.0    64.0   M    71.0      2
9       3    75.0    64.0   F    68.0      2
10      4    75.0    64.0   M    70.5      5
11      4    75.0    64.0   M    68.5      5
12      4    75.0    64.0   F    67.0      5
13      4    75.0    64.0   F    64.5      5
14      4    75.0    64.0   F    63.0      5
15      5    75.0    58.5   M    72.0      6
16      5    75.0    58.5   M    69.0      6
17      5    75.0    58.5   M    68.0      6
18      5    75.0    58.5   F    66.5      6
19      5    75.0    58.5   F    62.5      6
```

In [60]:
```
import numpy as np
def outliers_iqr(data):
    q1, q3 = np.percentile(data, [25, 75])
    iqr = q3-q1
    lower_bound = q1-(iqr*1.5)
    upper_bound = q3+(iqr*1.5)
    return np.where((data>upper_bound)|(data<lower_bound))
```

```
In [61]: for i in outliers_iqr(df.height)[0]:
             print(df[i:i+1])
```

```
    family  father  mother sex  height  nkids
288     72    70.0    65.0   M    79.0      7
```

```
In [64]: def outliers_z_score(data):
             threshold =3
             mean = np.mean(data)
             std = np.std(data)
             z_scores = [(y-mean)/std for y in data]
             return np.where(np.abs(z_scores)>threshold)

         for i in outliers_z_score(df.height)[0]:
             print(df[i:i+1])
         print()
```

```
    family  father  mother sex  height  nkids
125     35    71.0    69.0   M    78.0      5
    family  father  mother sex  height  nkids
288     72    70.0    65.0   M    79.0      7
    family  father  mother sex  height  nkids
672    155    68.0    60.0   F    56.0      7
```

```
In [65]: # Supervised learning - Linear Regression
         import matplotlib.pyplot as plt
         import pandas as pd
         import numpy as np

         from sklearn.datasets import load_boston
         dataset = load_boston()

In [66]: print(dataset)
```

```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
```

```
 7.8800e+00]]), 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 1
18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
 7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
 8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
 9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]), 'feature_names': ar
'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'), 'DESCR': ".. _boston_dataset:\n\nBoston I
```

```
In [68]: print(dataset.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

```
In [69]: print(dataset.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is us

    :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 river; 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 1940
        - DIS      weighted distances to five Boston employment centres
        - 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 blacks 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.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


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

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
```

```
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regressi
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources o
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on th
```

```
In [70]: print(dataset.target) # The prices of houses is the information we are seeking
                               # and it can be accessed via the target property
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
```

```
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
```

In [72]:
```
# load the dat into a Pandas DataFrame
df = pd.DataFrame(dataset.data, columns=dataset.feature_names)
df.head()
```

Out[72]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

In [73]:
```
#add the prices of the houses to the DataFrame, add a new column to the DataFrame and
df['MEDV'] = dataset.target
df.head()
```

Out[73]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

In [74]:
```
#Data cleansing
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
```

```
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
MEDV       506 non-null float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [75]:
```
print(df.isnull().sum())
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [76]:
```
# Feature selection
# choose those features that directly influence the result(that is prices of houses)
# corr() function computes the pairwise correlation of columns
corr = df.corr()
print(corr)
```

```
            CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
CRIM    1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247  0.352734
ZN     -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.569537
INDUS   0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.644779
CHAS   -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251  0.086518
NOX     0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.731470
```

```
RM      -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.240265
AGE      0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.000000
DIS     -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.747881
RAD      0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.456022
TAX      0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.506456
PTRATIO  0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.261515
B       -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.273534
LSTAT    0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.602339
MEDV    -0.388305  0.360445 -0.483725  0.175260 -0.427321  0.695360 -0.376955

              DIS       RAD       TAX   PTRATIO         B     LSTAT      MEDV
CRIM    -0.379670  0.625505  0.582764  0.289946 -0.385064  0.455621 -0.388305
ZN       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
B        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
MEDV     0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

In [77]:
```
print(df.corr().abs().nlargest(3, 'MEDV').index)#top 3 correlation index
```

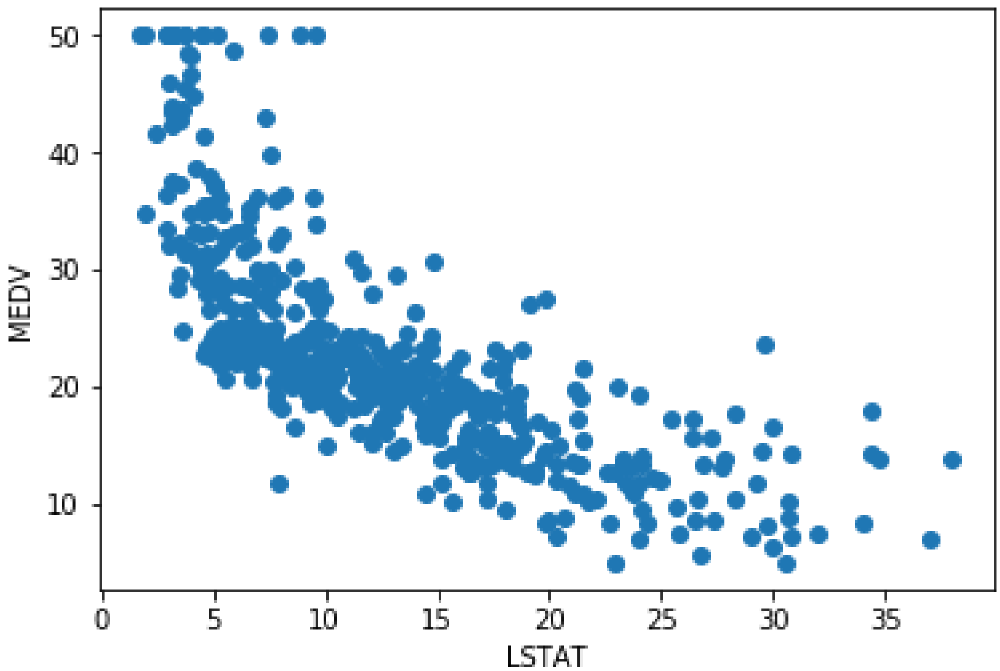
```
Index(['MEDV', 'LSTAT', 'RM'], dtype='object')
```

In [79]:
```
print(df.corr().abs().nlargest(3, 'MEDV').values[:,13])#top3 correlation values
```

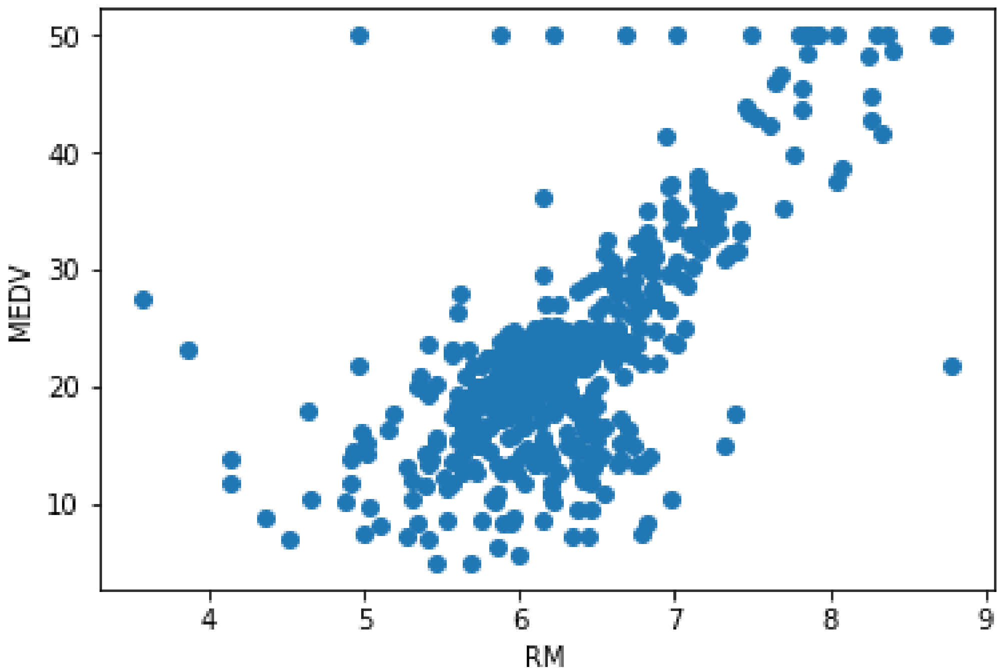
```
[1.         0.73766273 0.69535995]
```

In [80]:
```
#Since RM and LSTAT have high correlation values, we will use these two features to t
#Multiple Regression
# 2 or more independent variables are used to predict the value of a dependent variab
# Plot a scatter plot showing the relationship between the LSTAT feature and the MEDV
plt.scatter(df['LSTAT'], df['MEDV'], marker='o')
plt.xlabel('LSTAT')
plt.ylabel('MEDV')
```

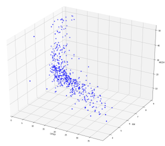
Out[80]:
```
Text(0, 0.5, 'MEDV')
```




```
In [81]: # RM and MEDV label
         plt.scatter(df['RM'], df['MEDV'], marker='o')
         plt.xlabel('RM')
         plt.ylabel('MEDV')

Out[81]: Text(0, 0.5, 'MEDV')
```




```
In [82]: from mpl_toolkits.mplot3d import Axes3D

In [83]: fig = plt.figure(figsize=(18,15))
         ax=fig.add_subplot(111, projection='3d')
         ax.scatter(df['LSTAT'], df['RM'], df['MEDV'], c='b')
         ax.set_xlabel("LSTAT")
         ax.set_ylabel("RM")
         ax.set_zlabel("MEDV")
         plt.show()
```



```
In [88]: # Training the model
         x = pd.DataFrame(np.c_[df['LSTAT'], df['RM']], columns=['LSTAT', 'RM'])
         Y = df['MEDV']

In [89]: from sklearn.model_selection import train_test_split
         x_train, x_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.3, random_state=
```

In [90]:
```
print(x_train.shape)
print(Y_train.shape)
```

```
(354, 2)
(354,)
```

In [91]:
```
# x training set 354rows, 2col   Y 354rows 1col
print(x_test.shape)
print(Y_test.shape)
```

```
(152, 2)
(152,)
```

In [93]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, Y_train)
```

Out[93]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [94]:
```
price_pred = model.predict(x_test)
```

In [95]:
```
print('R-Squared: %.4f' % model.score(x_test, Y_test))
```

```
R-Squared: 0.6162
```

In [96]:
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(Y_test, price_pred)
print(mse)
```

```
36.49422110915324
```

In [97]:
```
plt.scatter(Y_test, price_pred)
plt.xlabel("Actual Prices")
plt.ylabel('Predicted prices')
plt.title("Actual prices vs Predicted prices")
```

Out[97]:
```
Text(0.5, 1.0, 'Actual prices vs Predicted prices')
```


```
In [98]: print(model.intercept_)
         print(model.coef_)
```

```
0.38437936780346504
[-0.65957972  4.83197581]
```

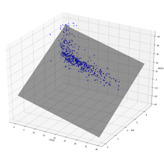
```
In [100]: print(model.predict([[30, 5]]))
```

```
[4.75686695]
```

```
In [103]: import matplotlib.pyplot as plt
          import pandas as pd
          import numpy as np
          from mpl_toolkits.mplot3d import Axes3D
          from sklearn.datasets import load_boston
          dataset = load_boston()
          df = pd.DataFrame(dataset.data, columns=dataset.feature_names)
          df['MEDV'] = dataset.target
```

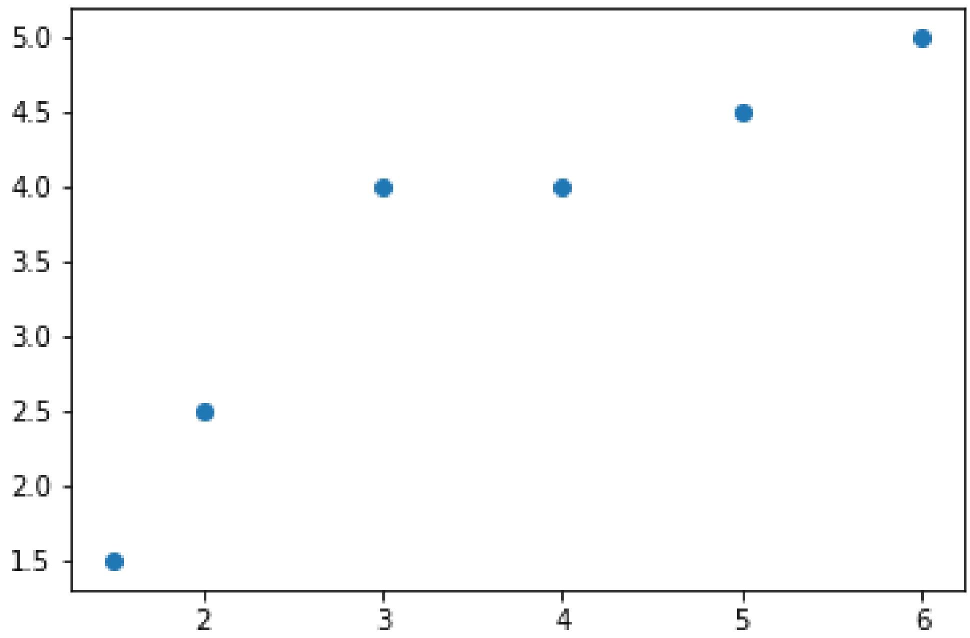
```
In [108]: x = pd.DataFrame(np.c_[df['LSTAT'], df['RM']], columns=['LSTAT', 'RM'])
          Y = df['MEDV']
          fig = plt.figure(figsize=(18,15))
```

```
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x['LSTAT'], x['RM'], Y, c='b')
ax.set_xlabel("LSTAT")
ax.set_ylabel("RM")
ax.set_zlabel("MEDV")

x_surf = np.arange(0,40,1)
y_surf = np.arange(0,10,1)
x_surf, y_surf = np.meshgrid(x_surf, y_surf)

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x, Y)

z = lambda x,y: (model.intercept_+model.coef_[0] * x + model.coef_[1]*y)

ax.plot_surface(x_surf, y_surf, z(x_surf,y_surf),
                rstride=1,
                cstride=1,
                color='None',
                alpha=0.4)
plt.show()
```

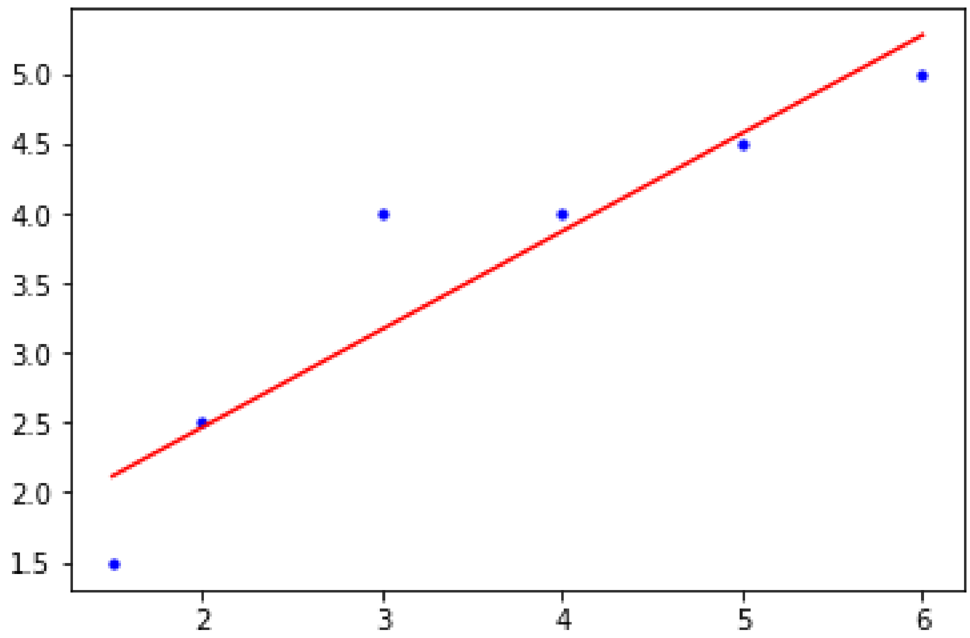


In [109]:
```
# Polynomial Regression
df = pd.read_csv('polynomial.csv')
plt.scatter(df.x,df.y)
```

Out[109]: <matplotlib.collections.PathCollection at 0x2183323ee80>



In [111]:
```
x=df.x[0:6, np.newaxis]# convert to 2D array
y=df.y[0:6, np.newaxis]
model.fit(x,y)
# perform prediction
y_pred = model.predict(x)
# plot the training point
plt.scatter(x, y, s=10, color='b')
# plot the straight line
plt.plot(x, y_pred, color='r')
plt.show()
# calculate R-Squared
print('R-Squared for training set: %.4f' % model.score(x,y))
```




```
R-Squared for training set: 0.8658
```

```
In [129]: from sklearn.preprocessing import PolynomialFeatures
          degree=5
          polynomial_features = PolynomialFeatures(degree=degree)
          x_poly=polynomial_features.fit_transform(x)
          print(x_poly)
```

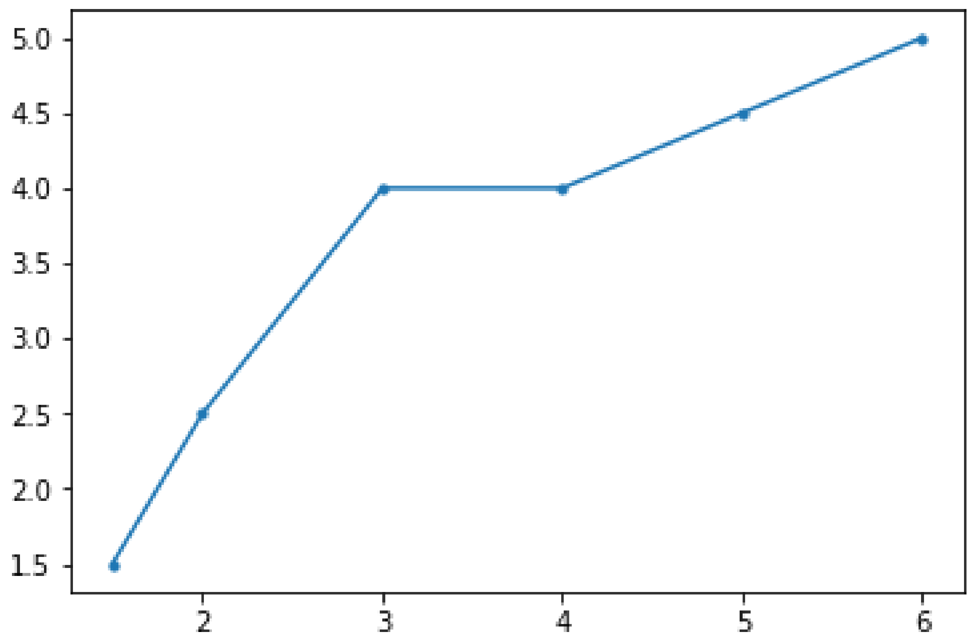
```
[[1.00000e+00 1.50000e+00 2.25000e+00 3.37500e+00 5.06250e+00 7.59375e+00]
 [1.00000e+00 2.00000e+00 4.00000e+00 8.00000e+00 1.60000e+01 3.20000e+01]
 [1.00000e+00 3.00000e+00 9.00000e+00 2.70000e+01 8.10000e+01 2.43000e+02]
 [1.00000e+00 4.00000e+00 1.60000e+01 6.40000e+01 2.56000e+02 1.02400e+03]
 [1.00000e+00 5.00000e+00 2.50000e+01 1.25000e+02 6.25000e+02 3.12500e+03]
 [1.00000e+00 6.00000e+00 3.60000e+01 2.16000e+02 1.29600e+03 7.77600e+03]]
```

```
In [130]: print(polynomial_features.get_feature_names('x'))
```

```
['1', 'x', 'x^2', 'x^3', 'x^4', 'x^5']
```

```
In [131]: model = LinearRegression()
          model.fit(x_poly, y)
          y_poly_pred = model.predict(x_poly)
          # plot the points
```

```
plt.scatter(x,y, s=10)
# plot the regression line
plt.plot(x, y_poly_pred)
plt.show()
print(model.intercept_)
print(model.coef_)
```



```
[14.02380952]
[[  0.         -27.35119048  20.692791    -6.71693122   0.99371693
   -0.05489418]]
```

In [132]:
```
print('R-squared for training set: % .4f' % model.score(x_poly, y))
```

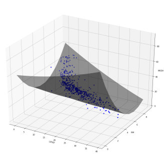
```
R-squared for training set:  1.0000
```

In [133]:
```
# Using Polynominal Multiple Regression on the Boston Dataset
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
```

```
In [135]: dataset = load_boston()
          df = pd.DataFrame(dataset.data, columns = dataset.feature_names)
          df['MEDV'] = dataset.target
          x=pd.DataFrame(np.c_[df['LSTAT'], df['RM']], columns=['LSTAT', 'RM'])
          Y=df['MEDV']
          from sklearn.model_selection import train_test_split
          x_train, x_test, Y_train, Y_test = train_test_split(x, Y, test_size=0.3, random_state

In [136]: degree = 2
          polynomial_features = PolynomialFeatures(degree = degree)
          x_train_poly = polynomial_features.fit_transform(x_train)

In [138]: print(polynomial_features.get_feature_names(['x', 'y']))

['1', 'x', 'y', 'x^2', 'x y', 'y^2']


In [140]: model = LinearRegression()
          model.fit(x_train_poly, Y_train)

Out[140]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                   normalize=False)

In [141]: x_test_poly = polynomial_features.fit_transform(x_test)
          print('R-squared: %.4f' % model.score(x_test_poly, Y_test))

R-squared: 0.7340


In [142]: print(model.intercept_)
          print(model.coef_)

26.93343052383913
[ 0.00000000e+00  1.47424550e+00 -6.70204730e+00  7.93570743e-04
 -3.66578385e-01  1.17188007e+00]


In [143]: import matplotlib.pyplot as plt
          import pandas as pd
          import numpy as np
          from mpl_toolkits.mplot3d import Axes3D
          from sklearn.preprocessing import PolynomialFeatures
          from sklearn.linear_model import LinearRegression
          from sklearn.datasets import load_boston

In [145]: dataset = load_boston()
          df['MEDV']=dataset.target
          x=pd.DataFrame(np.c_[df['LSTAT'], df['RM']], columns = ['LSTAT', 'RM'])
          Y=df['MEDV']
```

In [150]:
```
fig=plt.figure(figsize=(18,15))
ax=fig.add_subplot(111, projection='3d')
ax.scatter(x['LSTAT'],x['RM'],Y,c='b')
ax.set_xlabel("LSTAT")
ax.set_ylabel("RM")
ax.set_zlabel("MEDV")

x_surf = np.arange(0,40,1)
y_surf = np.arange(0,10,1)
x_surf, y_surf = np.meshgrid(x_surf, y_surf)

degree=2
polynomial_features = PolynomialFeatures(degree=degree)
x_poly = polynomial_features.fit_transform(x)
print(polynomial_features.get_feature_names(['x','y']))
model = LinearRegression()
model.fit(x_poly, Y)
z=lambda x, y: (model.intercept_+
                (model.coef_[1]* x)+
                (model.coef_[2]* y)+
                (model.coef_[3]* x**2)+
                (model.coef_[4]* x*y)+
                (model.coef_[5]* y**2)
               )
ax.plot_surface(x_surf, y_surf, z(x_surf, y_surf),
               rstride=1,
               cstride=1,
               color='None',
               alpha=0.4)
plt.show()
```

```
['1', 'x', 'y', 'x^2', 'x y', 'y^2']
```



In [ ]: