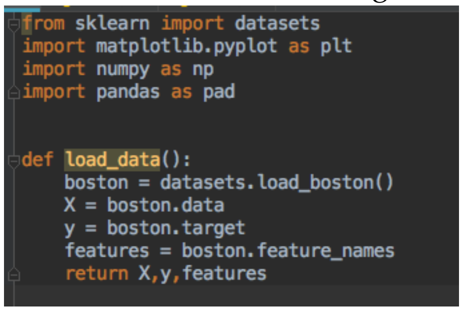
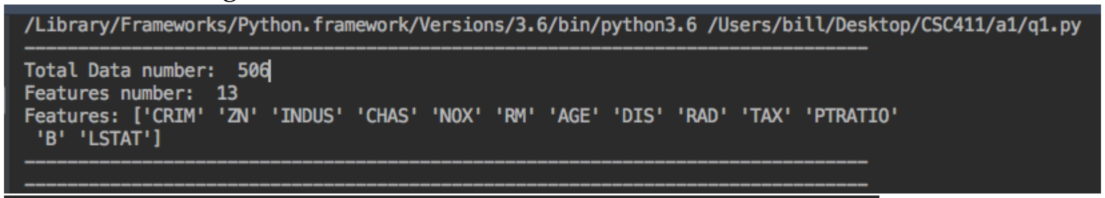
Q1:

1. Load the Boston housing data from the sklearn datasets module.

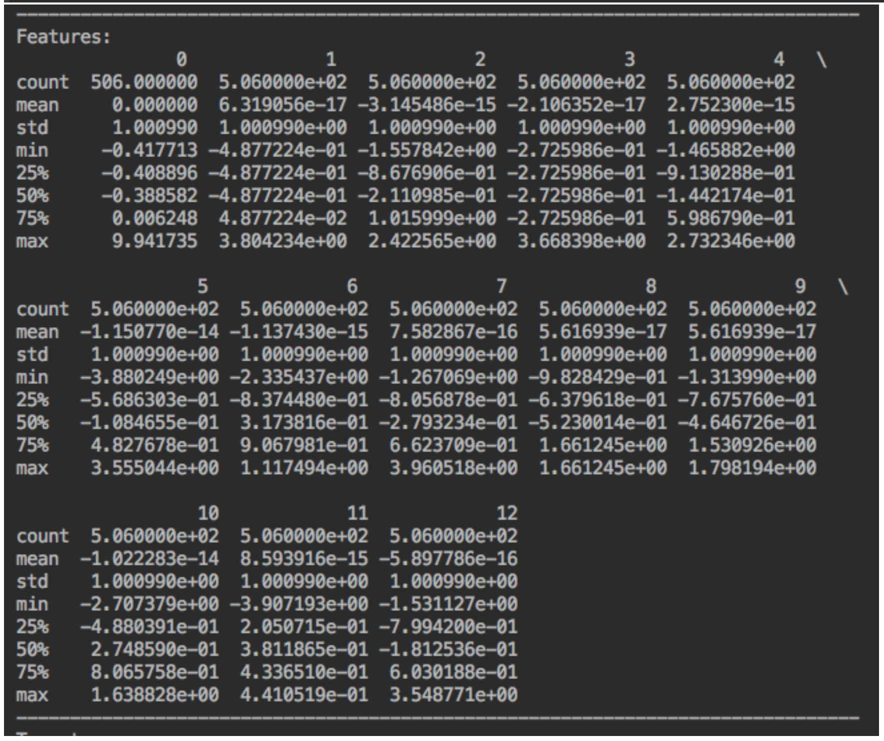


```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np
import pandas as pad


def load_data():
    boston = datasets.load_boston()
    X = boston.data
    y = boston.target
    features = boston.feature_names
    return X,y,features
```

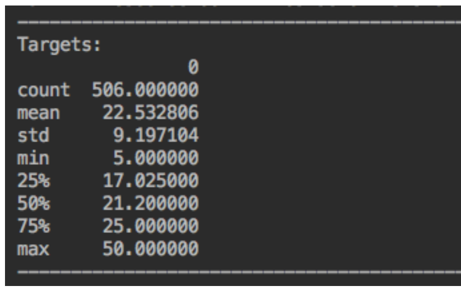
2. Describe and summarize the data in terms of number of data points, dimensions, target, etc.



```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/bill/Desktop/CSC411/a1/q1.py
--------------------------------------------------------------------------------
Total Data number:  506
Features number:  13
Features: ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```



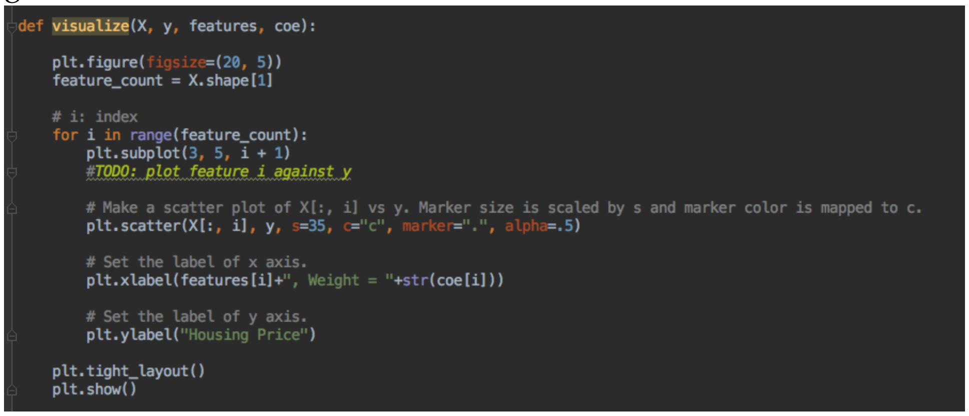
```
--------------------------------------------------------------------------------
Features:
                0             1             2             3             4  \
count  506.000000  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02
mean     0.000000  6.319056e-17 -3.145486e-15 -2.106352e-17  2.752300e-15
std      1.000990  1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00
min     -0.417713 -4.877224e-01 -1.557842e+00 -2.725986e-01 -1.465882e+00
25%     -0.408896 -4.877224e-01 -8.676906e-01 -2.725986e-01 -9.130288e-01
50%     -0.388582 -4.877224e-01 -2.110985e-01 -2.725986e-01 -1.442174e-01
75%      0.006248  4.877224e-02  1.015999e+00 -2.725986e-01  5.986790e-01
max      9.941735  3.804234e+00  2.422565e+00  3.668398e+00  2.732346e+00

                  5             6             7             8             9  \
count  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02
mean  -1.150770e-14 -1.137430e-15  7.582867e-16  5.616939e-17  5.616939e-17
std    1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00
min   -3.880249e+00 -2.335437e+00 -1.267069e+00 -9.828429e-01 -1.313990e+00
25%   -5.686303e-01 -8.374480e-01 -8.056878e-01 -6.379618e-01 -7.675760e-01
50%   -1.084655e-01  3.173816e-01 -2.793234e-01 -5.230014e-01 -4.646726e-01
75%    4.827678e-01  9.067981e-01  6.623709e-01  1.661245e+00  1.530926e+00
max    3.555044e+00  1.117494e+00  3.960518e+00  1.661245e+00  1.798194e+00

                 10            11            12
count  5.060000e+02  5.060000e+02  5.060000e+02
mean  -1.022283e-14  8.593916e-15 -5.897786e-16
std    1.000990e+00  1.000990e+00  1.000990e+00
min   -2.707379e+00 -3.907193e+00 -1.531127e+00
25%   -4.880391e-01  2.050715e-01 -7.994200e-01
50%    2.748590e-01  3.811865e-01 -1.812536e-01
75%    8.065758e-01  4.336510e-01  6.030188e-01
max    1.638828e+00  4.410519e-01  3.548771e+00
--------------------------------------------------------------------------------
```



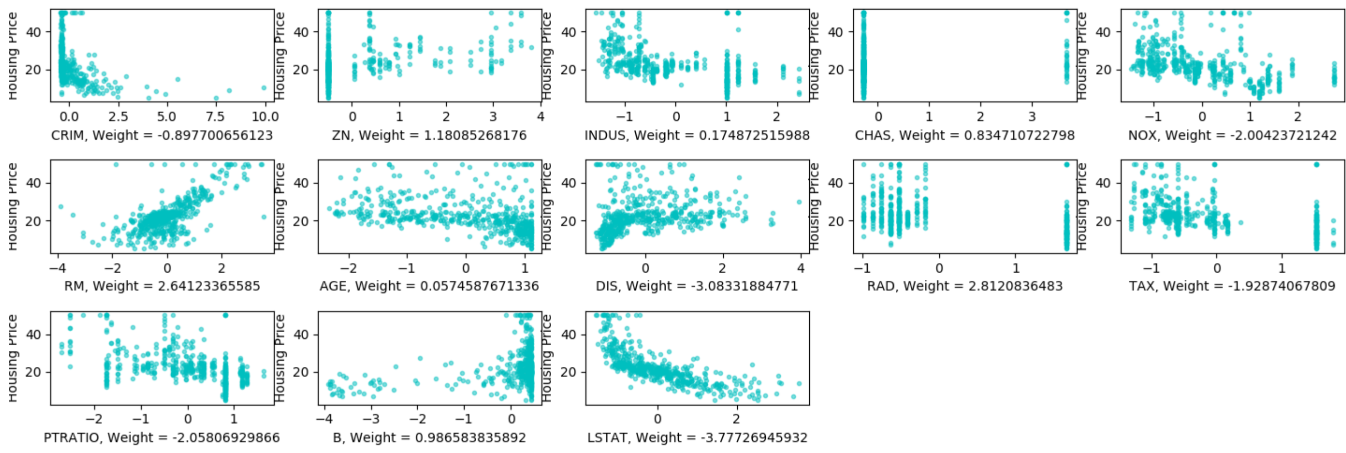
```
--------------------------------------------------------------------------------
Targets:
                0
count  506.000000
mean    22.532806
std      9.197104
min      5.000000
25%     17.025000
50%     21.200000
75%     25.000000
max     50.000000
--------------------------------------------------------------------------------
```

The ”number of data points” is 506, which shows as ”Total Data number” in image above. This number tell us the total data number we need to use as training or testing.
The ”dimensions” is 13, which shows as ”Features number” in image above. This number tell us the how many features are in a single data. The details of features show in image above.
The ”target” also shows in image above.
I keep these printing information in my code in case someone need to check these again.

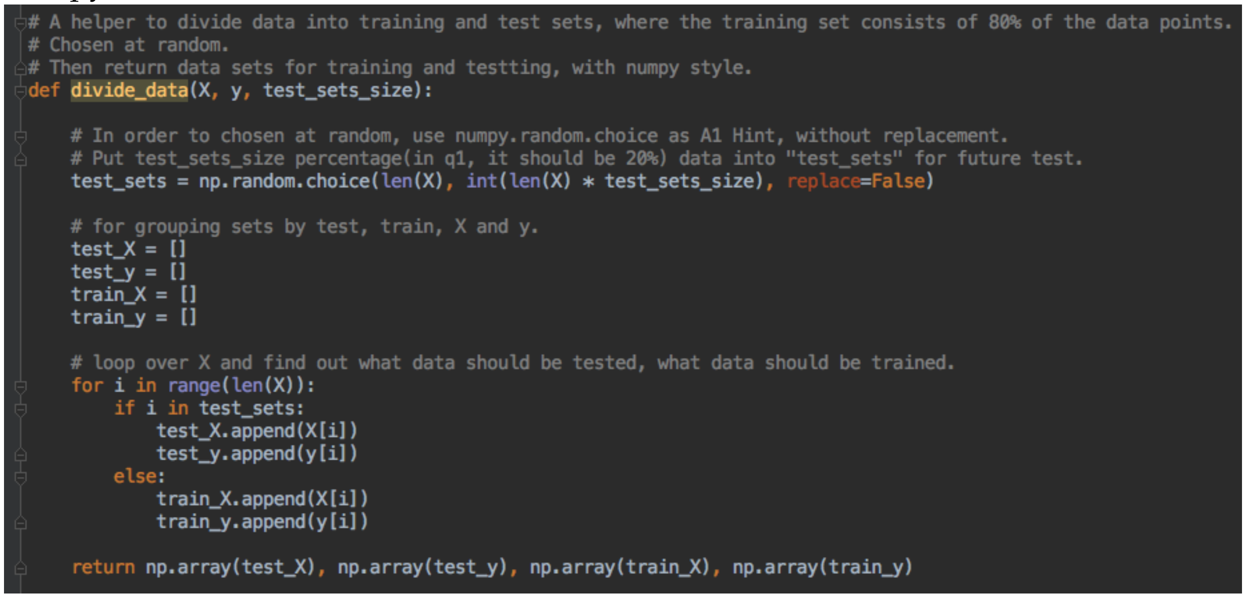
3. Visualization: present a single grid containing plots for each feature against the target. Choose the appropriate axis for dependent vs. independent variables. Hint: use pyplot.tight layout function to make your grid readable.



```
def visualize(X, y, features, coe):

    plt.figure(figsize=(20, 5))
    feature_count = X.shape[1]

    # i: index
    for i in range(feature_count):
        plt.subplot(3, 5, i + 1)
        #TODO: plot feature i against y

        # Make a scatter plot of X[:, i] vs y. Marker size is scaled by s and marker color is mapped to c.
        plt.scatter(X[:, i], y, s=35, c="c", marker=".", alpha=.5)

        # Set the label of x axis.
        plt.xlabel(features[i]+", Weight = "+str(coe[i]))

        # Set the label of y axis.
        plt.ylabel("Housing Price")

    plt.tight_layout()
    plt.show()
```

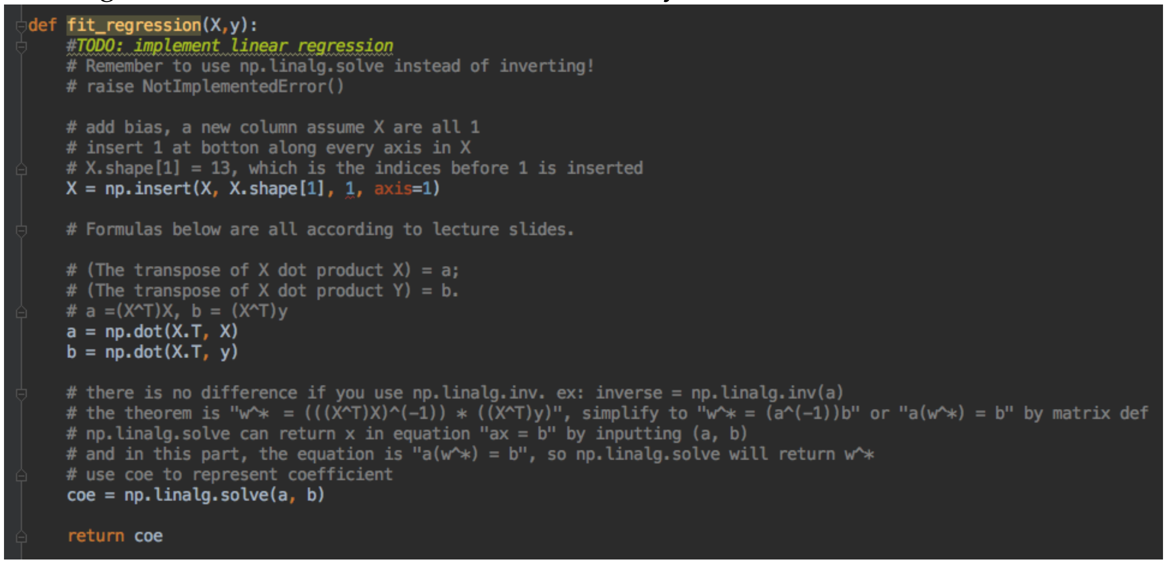


4. Divide your data into training and test sets, where the training set consists of 80% of the data points (chosen at random). Hint: You may find numpy.random.choice useful.



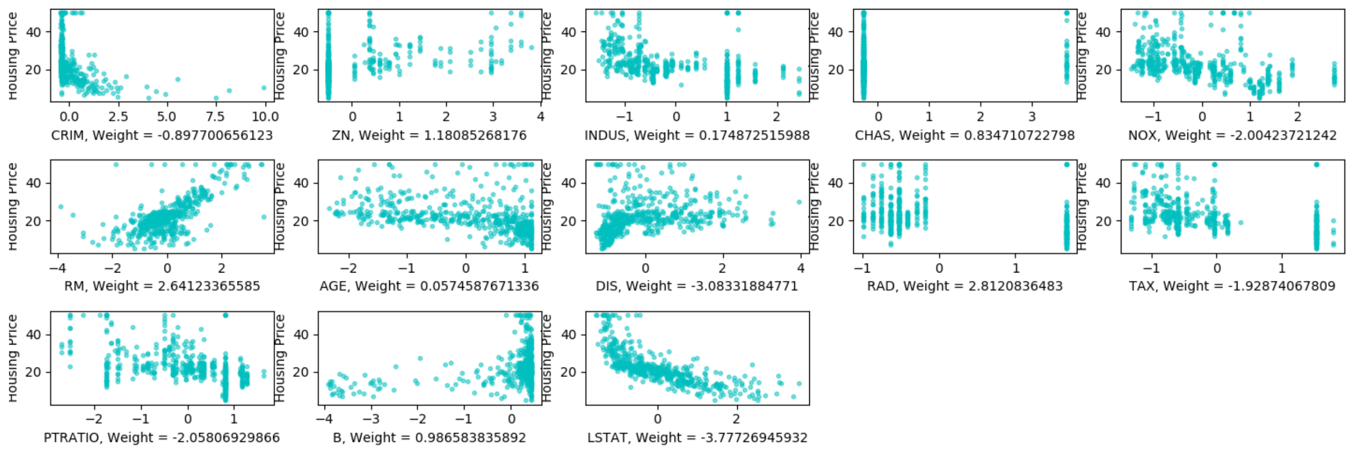
```python
# A helper to divide data into training and test sets, where the training set consists of 80% of the data points.
# Chosen at random.
# Then return data sets for training and testting, with numpy style.
def divide_data(X, y, test_sets_size):

    # In order to chosen at random, use numpy.random.choice as A1 Hint, without replacement.
    # Put test_sets_size percentage(in q1, it should be 20%) data into "test_sets" for future test.
    test_sets = np.random.choice(len(X), int(len(X) * test_sets_size), replace=False)

    # for grouping sets by test, train, X and y.
    test_X = []
    test_y = []
    train_X = []
    train_y = []

    # loop over X and find out what data should be tested, what data should be trained.
    for i in range(len(X)):
        if i in test_sets:
            test_X.append(X[i])
            test_y.append(y[i])
        else:
            train_X.append(X[i])
            train_y.append(y[i])

    return np.array(test_X), np.array(test_y), np.array(train_X), np.array(train_y)
```

5. Write code to perform linear regression to predict the targets using the training data. Remember to add a bias term to your model.



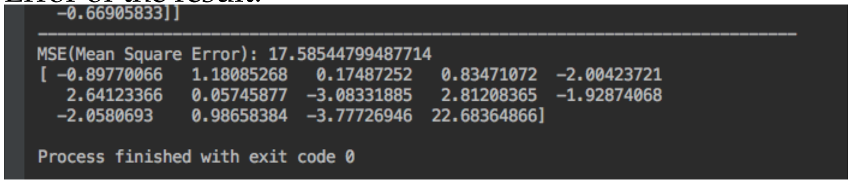
```python
def fit_regression(X,y):
    #TODO: implement linear regression
    # Remember to use np.linalg.solve instead of inverting!
    # raise NotImplementedError()

    # add bias, a new column assume X are all 1
    # insert 1 at botton along every axis in X
    # X.shape[1] = 13, which is the indices before 1 is inserted
    X = np.insert(X, X.shape[1], 1, axis=1)

    # Formulas below are all according to lecture slides.

    # (The transpose of X dot product X) = a;
    # (The transpose of X dot product Y) = b.
    # a =(X^T)X, b = (X^T)y
    a = np.dot(X.T, X)
    b = np.dot(X.T, y)

    # there is no difference if you use np.linalg.inv. ex: inverse = np.linalg.inv(a)
    # the theorem is "w^* = (((X^T)X)^(-1)) * ((X^T)y)", simplify to "w^* = (a^(-1))b" or "a(w^*) = b" by matrix def
    # np.linalg.solve can return x in equation "ax = b" by inputting (a, b)
    # and in this part, the equation is "a(w^*) = b", so np.linalg.solve will return w^*
    # use coe to represent coefficient
    coe = np.linalg.solve(a, b)

    return coe
```

6. Tabulate each feature along with its associated weight and present them in a table. Explain what the sign of the weight means in the third column (INDUS) of this table. Does the sign match what you expected? Why?



The weight of INDUS is not completely stable (actually there are no completely stable features), but it is positive in the most of runs on q1.py. INDUS's weight is also small, compare to others. So, these information tell us that the relation between INDUS and housing price may not strong.

7. Test the fitted model on your test set and calculate the Mean Square Error of the result.



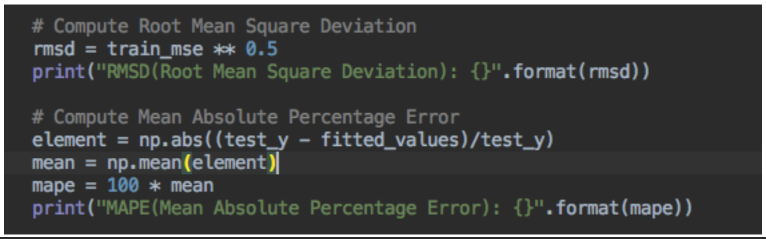
```
  -0.66905833]]
--------------------------------------------------------------------------------
MSE(Mean Square Error): 17.58544799487714
[ -0.89770066   1.18085268   0.17487252   0.83471072  -2.00423721
   2.64123366   0.05745877  -3.08331885   2.81208365  -1.92874068
  -2.0580693    0.98658384  -3.77726946  22.68364866]

Process finished with exit code 0
```

8. Suggest and calculate two more error measurement metrics; justify your choice.
There are 3 common error measurement metrics in wikipedia except MSE: Root Mean Square Deviation (RMSD), Mean Absolute Scaled Error (MASE) and Mean Absolute Percentage Error (MAPE). Let's choose Mean Absolute Percentage Error (MAPE) and Root Mean Square Deviation (RMSD) as examples.

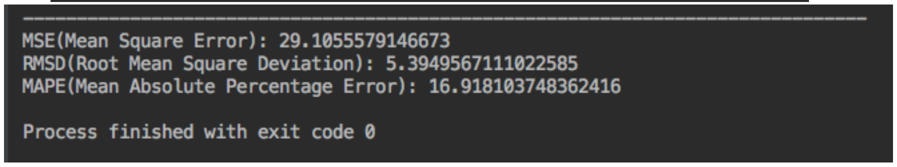
(1). Mean Absolute Percentage Error (MAPE):

Formula: $$\mathrm{M} = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

(2). Root Mean Square Deviation (RMSD):
Formula: RMSD = square root of MSE



```
# Compute Root Mean Square Deviation
rmsd = train_mse ** 0.5
print("RMSD(Root Mean Square Deviation): {}".format(rmsd))

# Compute Mean Absolute Percentage Error
element = np.abs((test_y - fitted_values)/test_y)
mean = np.mean(element)
mape = 100 * mean
print("MAPE(Mean Absolute Percentage Error): {}".format(mape))
```



```
------------------------------------------------------------------------------
MSE(Mean Square Error): 29.1055579146673
RMSD(Root Mean Square Deviation): 5.3949567111022585
MAPE(Mean Absolute Percentage Error): 16.918103748362416

Process finished with exit code 0
```

9. Feature Selection: Based on your results, what are the most significant features that best predict the price? Justify your answer.
Most significant features, which best predict the price, are depended by weights. If this feature weights more, then this feature is more significant. According to images above, RM, PTRATIO, DIS, LSTAT, RAD, NOX are important, especially RM, DIS and LSTAT.