1) Identify the data scale for all the attributes in the dataset

In [1]:

```
import pandas as  pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:

```
from sklearn.datasets import load_boston
```

In [3]:

```
bt = load_boston()
bt.keys()
```

Out[3]:

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [4]:

```
print(bt['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 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 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/ (https://
archive.ics.uci.edu/ml/machine-learning-databases/housing/)


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

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 diagnostic
s
...', 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 th
at address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
```

```
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. I
n Proceedings on the Tenth International Conference of Machine Learning, 236
-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [5]:

```
print(bt['feature_names'])
```

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

In [6]:

```
boston_df = bt.data
```

In [7]:

```
boston_df
```

Out[7]:

```
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]])
```

In [8]:

```
print(bt['filename'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\datasets\data\boston_hous
e_prices.csv
```

In [9]:

```
bost = pd.DataFrame(data = bt.data, columns = bt.feature_names)
bost.head()
```

Out[9]:

	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	[illegible]
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	[illegible]
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	[illegible]
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	[illegible]
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	[illegible]

In [10]:

```
bost['Price'] = bt.target
bost.head()
```

Out[10]:

	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	[illegible]
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	[illegible]
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	[illegible]
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	[illegible]
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	[illegible]

2) Compute the central tendency measure for any attribute in the dataset

In [18]:

```
print("Mean values of the deviation-")
bost[['CRIM','AGE','TAX']].mean()
```

```
Mean values of the deviation-
```

Out[18]:

```
CRIM      3.613524
AGE      68.574901
TAX     408.237154
dtype: float64
```

In [20]:

```
print ("Median Values in the Distribution")
bost[['CRIM','AGE','TAX']].median()
```

```
Median Values in the Distribution
```

Out[20]:

```
CRIM      0.25651
AGE      77.50000
TAX     330.00000
dtype: float64
```

In [21]:

```
print ("Mode Values in the Distribution")
bost[['CRIM','AGE','TAX']].mode()
```

```
Mode Values in the Distribution
```

Out[21]:

	CRIM	AGE	TAX
0	0.01501	100.0	666.0
1	14.33370	NaN	NaN

```
or
```

In [11]:

```
bost.isnull().sum()
```

Out[11]:

```
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
Price      0
dtype: int64
```

In [12]:

```
print(bost.describe())
```

```
             CRIM          ZN       INDUS        CHAS         NOX           R
M  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.00000
0
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.28463
4
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.70261
7
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.56100
0
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.88550
0
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.20850
0
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.62350
0
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.78000
0

              AGE         DIS         RAD         TAX     PTRATIO
B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.00000
0
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.67403
2
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.29486
4
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.32000
0
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.37750
0
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.44000
0
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.22500
0
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.90000
0

            LSTAT       Price
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

3)Compute the quartile measures for any attribute in the dataset

In [13]:

```python
bost.describe()
```

Out[13]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901
std	8.601545	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%	3.677083	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

In [14]:

```python
bost.count()
```

Out[14]:

```
CRIM       506
ZN         506
INDUS      506
CHAS       506
NOX        506
RM         506
AGE        506
DIS        506
RAD        506
TAX        506
PTRATIO    506
B          506
LSTAT      506
Price      506
dtype: int64
```

In [17]:

```python
print("Quartile Range of Population Column of a Data set:")
bost ['CRIM'].quantile([0.25,0.5,0.75])
```

```
Quartile Range of Population Column of a Data set:
```

Out[17]:

```
0.25    0.082045
0.50    0.256510
0.75    3.677083
Name: CRIM, dtype: float64
```

4)Compute variance and standard deviation for any attribute in the dataset

In [23]:

```
import statistics
p=bost['CRIM']
r=bost['TAX']
u=bost['AGE']
print("Variance of Crime rate  is % s" %(statistics.variance(p)))
print("Variance of Tax   is % s" %(statistics.variance(r)))
print("Variance of Age  is % s" %(statistics.variance(u)))
```

```
Variance of Crime rate  is 73.9865781990693
Variance of Tax   is 28404.759488122727
Variance of Age  is 792.358398505068
```

In [24]:

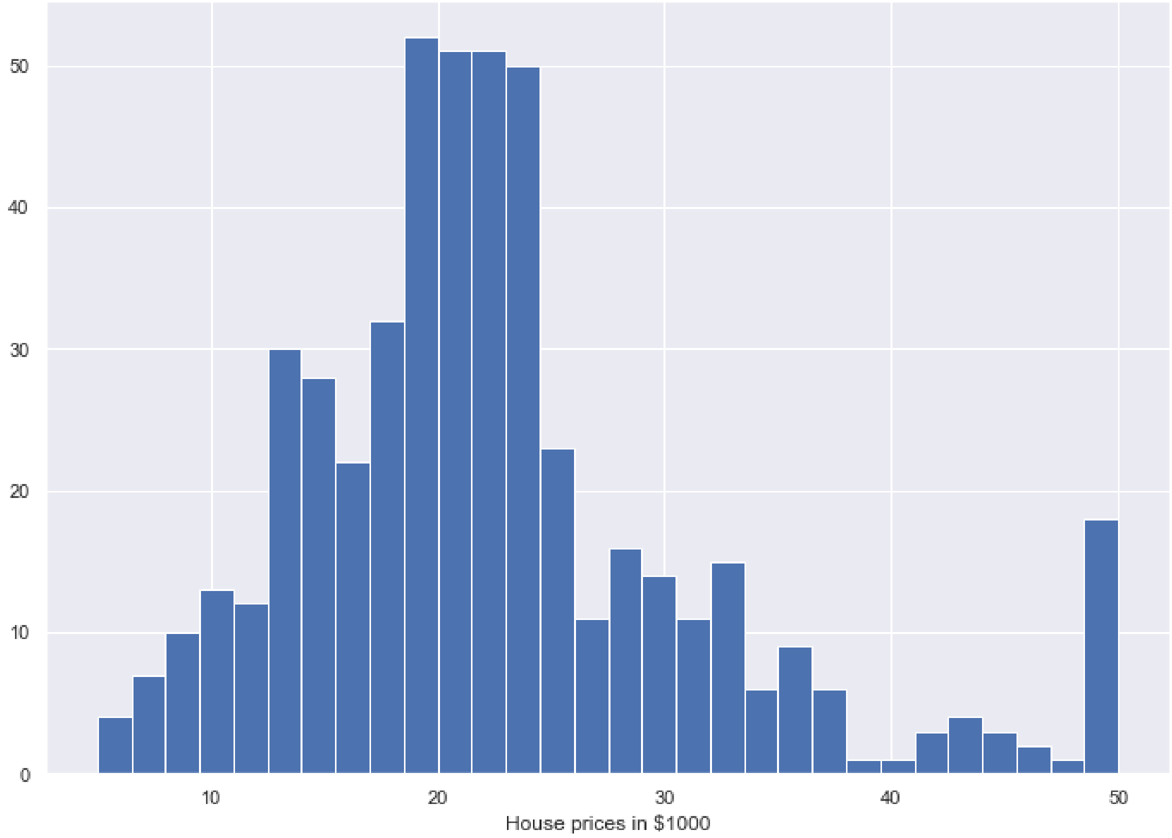
```
import statistics
p=bost['CRIM']
r=bost['TAX']
u=bost['AGE']
print("Variance of Crime rate  is % s" %(statistics.stdev(p)))
print("Variance of Tax   is % s" %(statistics.stdev(r)))
print("Variance of Age  is % s" %(statistics.stdev(u)))
```

```
Variance of Crime rate  is 8.60154510533249
Variance of Tax   is 168.53711605495903
Variance of Age  is 28.148861406903617
```

5)Compute dissimilarity measure for a binary / discreate attribute in the dataset

In [30]:

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
plt.hist(bost['Price'], bins=30)
plt.xlabel("House prices in $1000")
plt.show()
```



6)Compute Correlation coeffience for any two attribute

In [41]:

```
from numpy.random import randn
from numpy.random import seed
from numpy import cov

r=bost['Price']
u=bost['RM']
# seed random number generator
seed(1)
# prepare data
r = 20 * randn(1000) + 100
u = r + (10 * randn(1000) + 50)
# calculate covariance matrix
covariance = cov(r, u)
print(covariance)
```

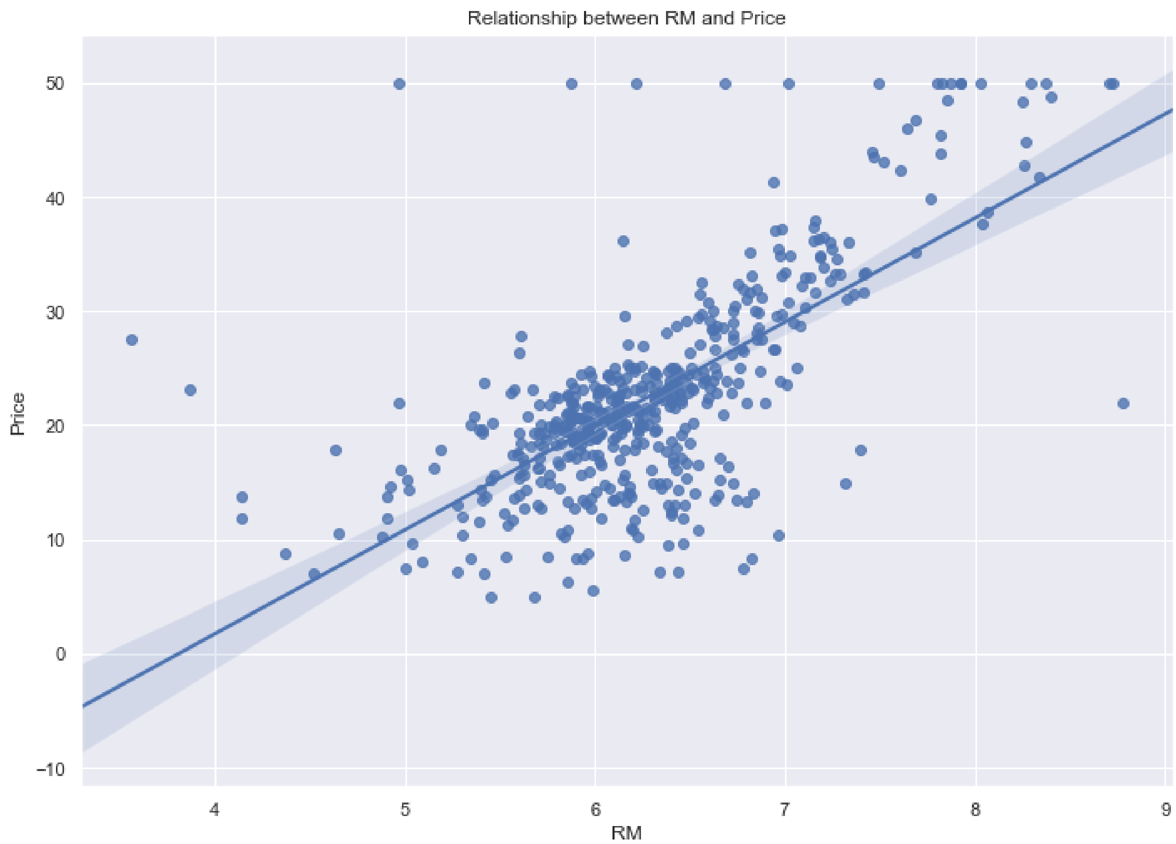
```
[[385.33297729 389.7545618 ]
 [389.7545618  500.38006058]]
```

In [42]:

```
sns.regplot(y="Price", x="RM", data=bost, fit_reg = True)
plt.title("Relationship between RM and Price")
```

Out[42]:

```
Text(0.5, 1.0, 'Relationship between RM and Price')
```



In [52]:

```
#negative corelation
sns.regplot(y="Price", x="LSTAT", data=bost, fit_reg = True)
plt.title("Relationship between LSTAT and Price")
```

Out[52]:

```
Text(0.5, 1.0, 'Relationship between LSTAT and Price')
```

7)Apply Z-score Normalization

In [53]:

```
import numpy as np
from scipy import stats
r=bost['Price']
u=bost['RM']

print ("\nZ-score for PRICE : \n", stats.zscore(r))
print ("\nZ-score for RM : \n", stats.zscore(u))
```

```
Z-score for PRICE :
 [ 0.15968566 -0.10152429  1.32424667  1.18275795  1.48750288  0.6712218
  0.03996443  0.49708184 -0.65659542 -0.39538548 -0.81985164 -0.39538548
 -0.09064054 -0.23212926 -0.47157171 -0.286548    0.06173193 -0.54775795
 -0.25389676 -0.47157171 -0.97222411 -0.31919924 -0.79808414 -0.87427038
 -0.75454915 -0.93957286 -0.64571167 -0.84161913 -0.44980422 -0.16682677
 -1.07017784 -0.87427038 -1.0157591  -1.02664285 -0.98310786 -0.39538548
 -0.27566425 -0.16682677  0.23587189  0.89978051  1.34601416  0.4426631
  0.30117438  0.23587189 -0.14505928 -0.35185049 -0.27566425 -0.64571167
 -0.88515413 -0.34096674 -0.30831549 -0.22124551  0.26852314  0.09438317
 -0.39538548  1.4004329   0.23587189  0.98685049  0.08349942 -0.31919924
 -0.41715297 -0.71101416 -0.0362218   0.26852314  1.13922296  0.10526692
 -0.34096674 -0.0579893  -0.55864169 -0.17771052  0.18145315 -0.09064054
  0.02908069  0.09438317  0.17056941 -0.12329178 -0.27566425 -0.18859427
 -0.14505928 -0.24301301  0.59503557  0.14880191  0.24675564  0.03996443
  0.14880191  0.4426631  -0.00357056 -0.0362218   0.11615067  0.6712218
  0.00731319 -0.0579893   0.03996443  0.26852314 -0.21036176  0.63857056
 -0.12329178  1.75959658  2.31466771  1.16099045  0.54061683  0.43177935
```

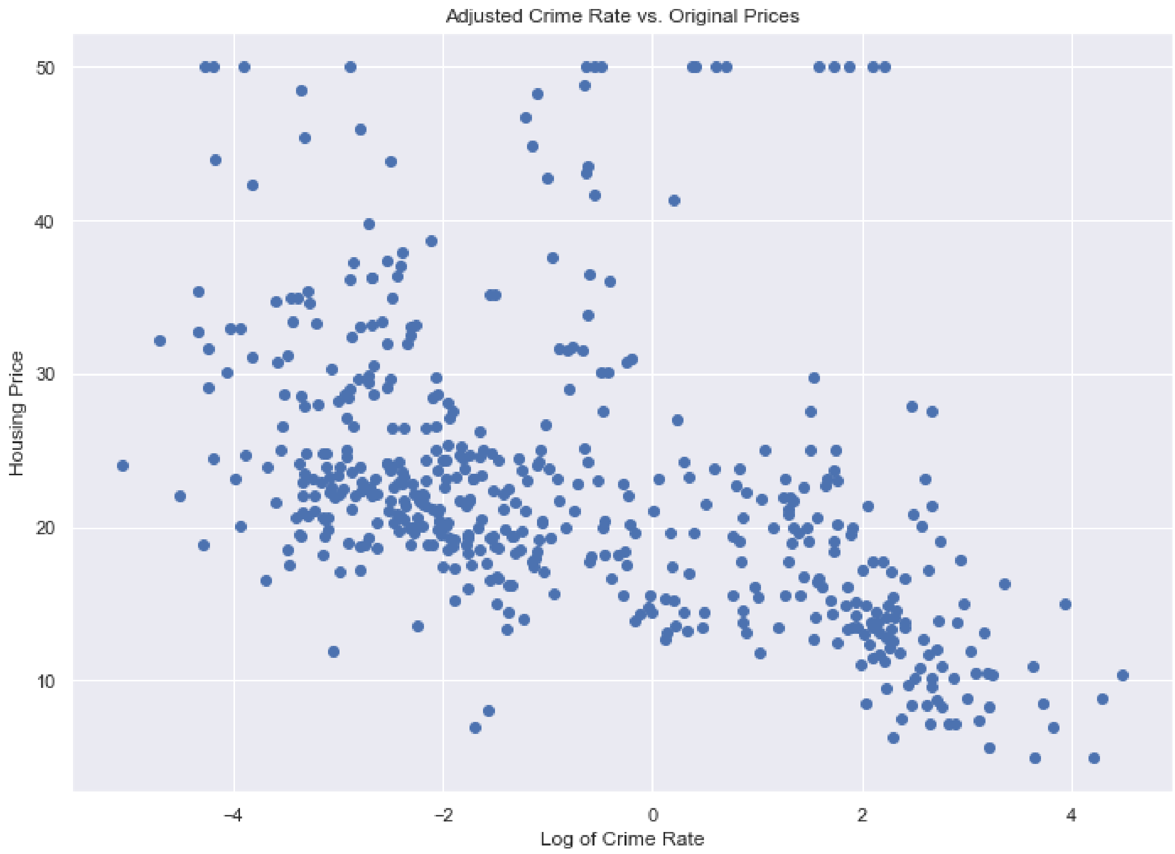
In [56]:

```
x = np.log(bost.CRIM)
plt.scatter(x, bost.Price)

plt.xlabel("Log of Crime Rate")
plt.ylabel("Housing Price")
plt.title("Adjusted Crime Rate vs. Original Prices")
```

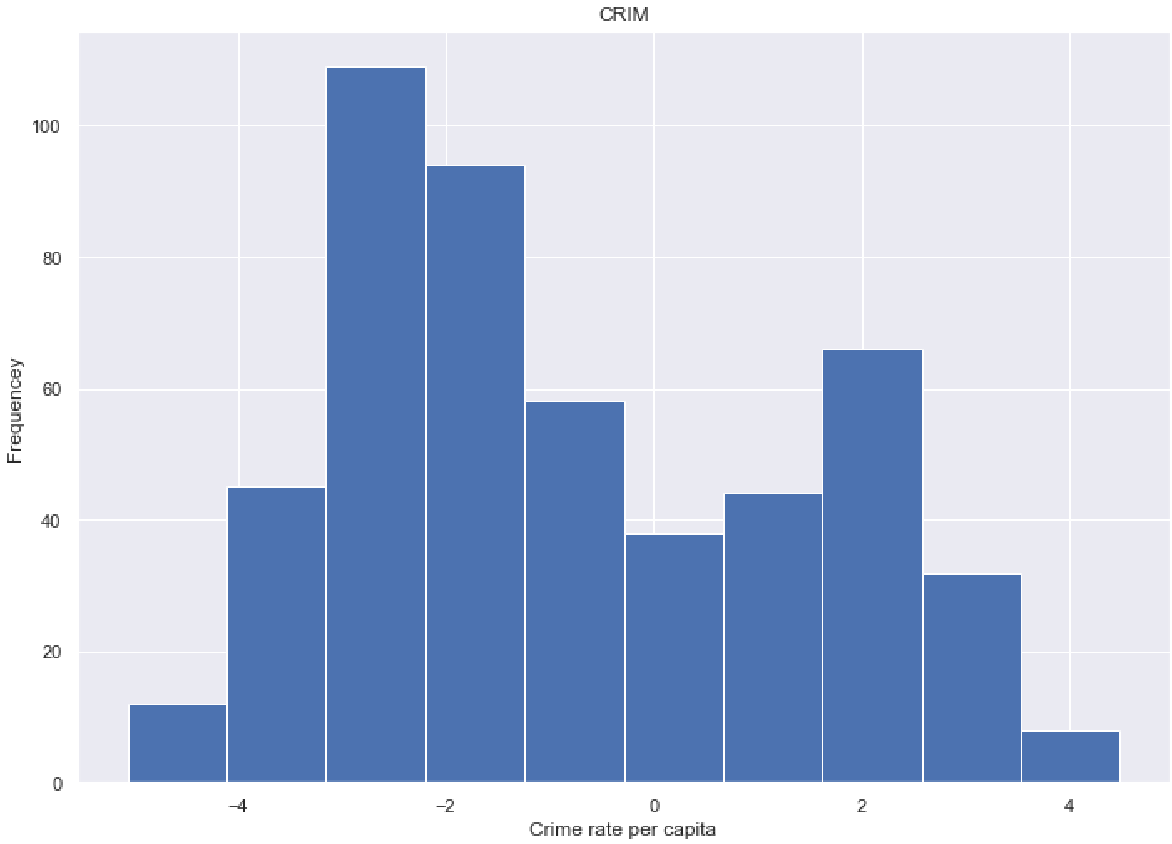
Out[56]:

```
Text(0.5, 1.0, 'Adjusted Crime Rate vs. Original Prices')
```



In [57]:

```
#Crime Rate now has a normal distribution:
plt.hist(np.log(bost.CRIM))
plt.title("CRIM")
plt.xlabel("Crime rate per capita")
plt.ylabel("Frequencey")
plt.show()
```



8)Min-Max normalization

In [58]:

```
from sklearn.preprocessing import MinMaxScaler
stats.zscore(u)
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler()
print(scaler.fit(data))


print(scaler.data_max_)

print(scaler.transform(data))




print(scaler.transform([[0, 1]]))
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
[ 1. 18.]
[[0.   0.  ]
 [0.25 0.25]
 [0.5  0.5 ]
 [1.   1.  ]]
[[ 0.5    -0.0625]]
```

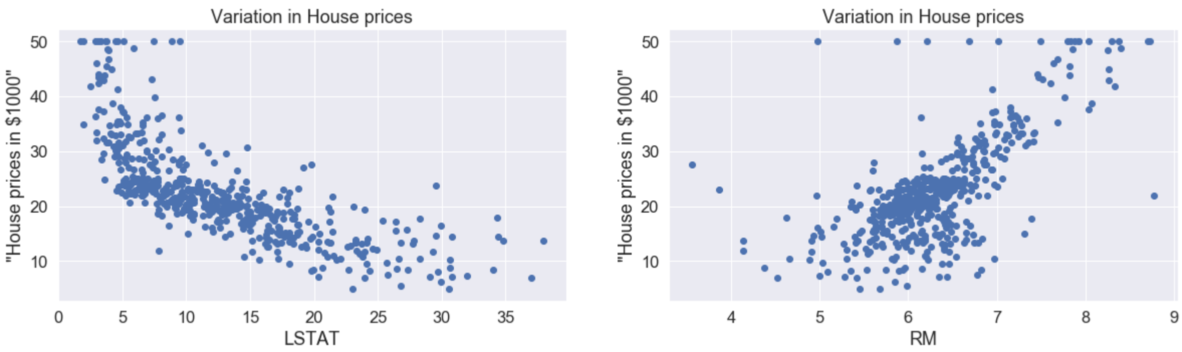
9)Perform data discretization for any discrete or continuous valued attribute. Apply Bin mean,Bin median, Bin average methods.

In [62]:

```
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = bost['Price']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = bost[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('"House prices in $1000"')
```



In [73]:

```
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error
```

In [83]:

```
X_rooms = bost.RM
y_price = bost.Price


X_rooms = np.array(X_rooms).reshape(-1,1)
y_price = np.array(y_price).reshape(-1,1)

print(X_rooms.shape)
print(y_price.shape)
```

```
(506, 1)
(506, 1)
```

In [84]:

```
X_train_1, X_test_1, Y_train_1, Y_test_1 = train_test_split(X_rooms, y_price, test_size = 0

print(X_train_1.shape)
print(X_test_1.shape)
print(Y_train_1.shape)
print(Y_test_1.shape)
```

```
(404, 1)
(102, 1)
(404, 1)
(102, 1)
```

In [85]:

```
reg_1 = LinearRegression()
reg_1.fit(X_train_1, Y_train_1)

y_train_predict_1 = reg_1.predict(X_train_1)
rmse = (np.sqrt(mean_squared_error(Y_train_1, y_train_predict_1)))
r2 = round(reg_1.score(X_train_1, Y_train_1),2)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 6.972277149440585
R2 score is 0.43
```

In [86]:

```


y_pred_1 = reg_1.predict(X_test_1)
rmse = (np.sqrt(mean_squared_error(Y_test_1, y_pred_1)))
r2 = round(reg_1.score(X_test_1, Y_test_1),2)

print("The model performance for training set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
Root Mean Squared Error: 4.895963186952216
R^2: 0.69
```

In [77]:

```
rediction_space = np.linspace(min(X_rooms), max(X_rooms)).reshape(-1,1)
plt.scatter(X_rooms,y_price)
plt.plot(prediction_space, reg_1.predict(prediction_space), color = 'black', linewidth = 3)
plt.ylabel('value of house/1000($)')
plt.xlabel('number of rooms')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-77-c51911137cb3> in <module>
      1 rediction_space = np.linspace(min(X_rooms), max(X_rooms)).reshape(-1
,1)
      2 plt.scatter(X_rooms,y_price)
----> 3 plt.plot(prediction_space, reg_1.predict(prediction_space), color =
'black', linewidth = 3)
      4 plt.ylabel('value of house/1000($)')
      5 plt.xlabel('number of rooms')

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

In [ ]:

In [89]:

```
X = bost.drop('Price', axis = 1)
y = bost['Price']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=42)

reg_all = LinearRegression()
reg_all.fit(X_train, y_train)

# model evaluation for training set

y_train_predict = reg_all.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = round(reg_all.score(X_train, y_train),2)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 4.6520331848801675
R2 score is 0.75
```

In [90]:

```
# model evaluation for test set

y_pred = reg_all.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
r2 = round(reg_all.score(X_test, y_test),2)

print("The model performance for training set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
```

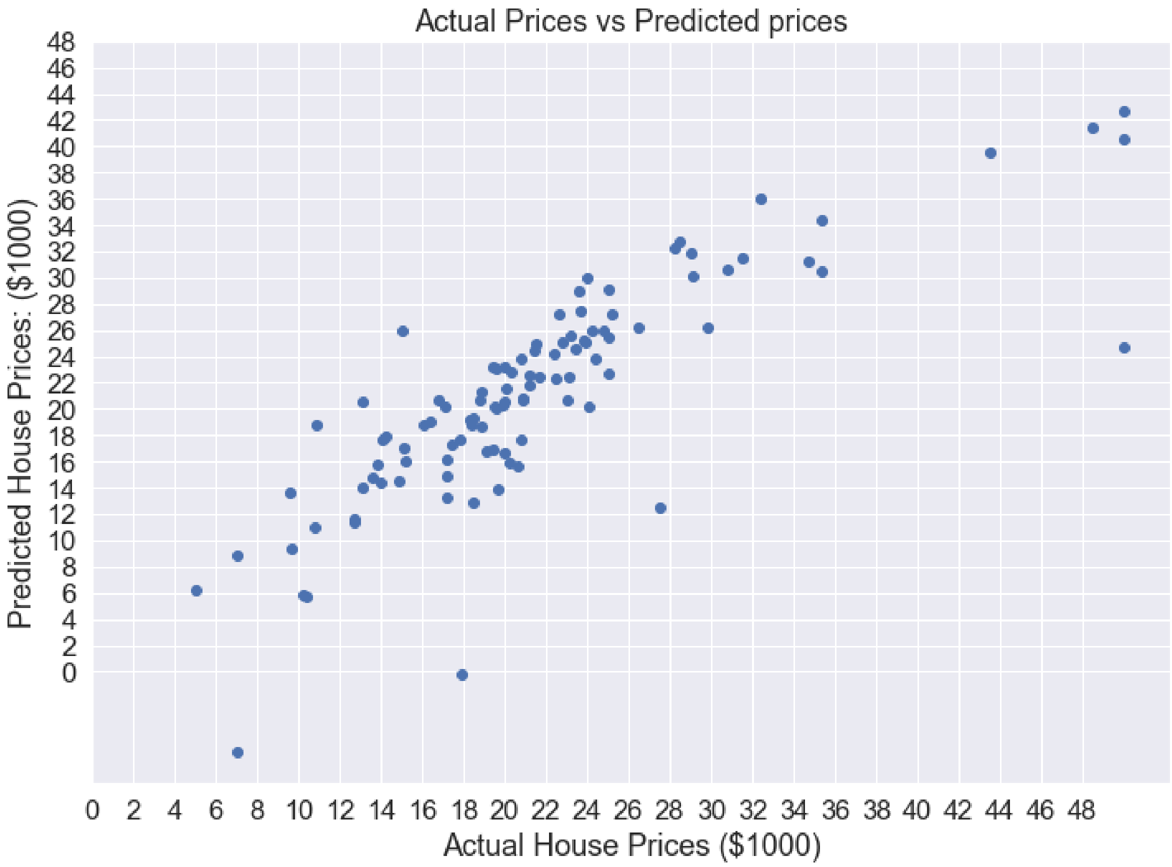
```
The model performance for training set
--------------------------------------
Root Mean Squared Error: 4.928602182665355
R^2: 0.67
```

In [92]:

```
plt.scatter(y_test, y_pred)
plt.xlabel("Actual House Prices ($1000)")
plt.ylabel("Predicted House Prices: ($1000)")
plt.xticks(range(0, int(max(y_test)),2))
plt.yticks(range(0, int(max(y_test)),2))
plt.title("Actual Prices vs Predicted prices")
```

Out[92]:

```
Text(0.5, 1.0, 'Actual Prices vs Predicted prices')
```



State your interpretations as a data analyst.

```
The boston dataset aims to find the factors affecting the domestic property value in the
city of Boston. Factors like per capita income, environmental factors, educational
facilities, property size, etc were taken into consideration to determine the most
significant parameters. We create multiple linear regression model using forward stepwise
selection and compare its performance with the linear regression model containing all the
variables.
```

In [ ]:

```
```