# Boston Housing Datasetを統計分析する

## データの前処理、発見的な探索

# 回帰分析、ランダムフォレストなどを行う

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import sklearn


import seaborn as sns
from matplotlib import rcParams
sns.set_style('whitegrid')
sns.set_context('poster')
```

sklearnデータセットからBoston Housing Datasetをインポートし、bostonという変数に格納する

In [3]:
```
# Importing Boston Housing Dataset from sklearn datasets and storing inside a variab
le called boston

from sklearn.datasets import load_boston

boston = load_boston()
print(boston.keys())
```

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

データセットに506行13列があると判断します。

In [4]:
```
# determines that the dataset has 506 rows and 13 coloumns
print(boston.data.shape,boston.target.shape)
```

```
(506, 13) (506,)
```

In [5]:
```
boston.target[:10]
```

Out[5]:
```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9])
```

属性情報（順）：

- 
  - CRIM: 町ごとの一人当たりの犯罪率
  - ZN: 住宅地のZN比率が25,000平方フィートを超える敷地に区画されている。
  - INDUS: 町あたりの非小売業エーカーのINDUS比率
  - CHAS: Charles Riverダミー変数（トラクトが川の境界にある場合は1、それ以外の場合は0）
  - NOX: 一酸化窒素濃度（1000万分の1）
  - RM: 住居ごとの平均部屋数
  - AGE: 1940年以前に建設された所有者居住ユニットのAGE比率
  - DIS: 5つのボストンの雇用センターまでのDIS加重距離
  - RAD: ラジアルハイウェイへのアクセス可能性のRAD指数
  - TAX：10,000ドルあたりの全額固定資産税率
  - PTRATIO： 町による生徒教師比率
  - B： 1000（Bk - 0.63）^ 2 Bkは町による黒人の割合である
  - LSTAT：%人口の地位が低い
  - MEDV： 1000ドル単位での所有者居住住宅の中央値

In [6]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attrib
ute 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 othe
rwise)
        - 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 Me
llon 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 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 add
ress 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. In Proc
eedings on the Tenth International Conference of Machine Learning, 236-243, Univer
sity of Massachusetts, Amherst. Morgan Kaufmann.
```

In [7]:
```
# Determines the column names
print(boston.feature_names)
```

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

bostonデータをパンダデータフレームに変換

In [8]:
```
#converting boston data into pandas dataframe using pd.Dataframe()

boston_df = pd.DataFrame(boston.data)
boston_df.head()
```

Out[8]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 4.98 |
| **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 | 9.14 |
| **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 | 4.03 |
| **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 | 2.94 |
| **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 | 5.33 |

In [9]:
```
# Replacing integers with feature names as columns.
boston_df.columns = boston.feature_names
boston_df.head()
```

Out[9]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 4.98 |
| **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 | 9.14 |
| **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 | 4.03 |
| **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 | 2.94 |
| **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 | 5.33 |

In [10]:
```
# Adding "Price" as another feature in the current dataset which is a part of anothe
r attribute called "target"
boston_df['PRICE'] = boston.target
boston_df.head()
```

Out[10]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | PRICE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 4.98 | 24.0 |
| **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 | 9.14 | 21.6 |
| **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 | 4.03 | 34.7 |
| **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 | 2.94 | 33.4 |
| **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 | 5.33 | 36.2 |

In [11]:
```
boston_df.shape
```

Out[11]: (506, 14)

各列の統計的な概要を表示する

In [12]:
```
# Showing summary of each columns using describe()
boston_df.describe()
```

Out[12]:

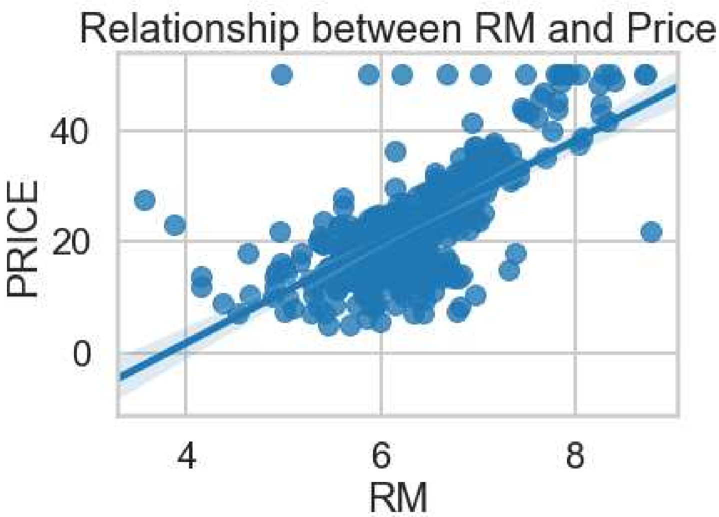
| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | R. |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.0000 |
| **mean** | 3.613524 | 11.363636 | 11.136779 | 0.069170 | 0.554695 | 6.284634 | 68.574901 | 3.795043 | 9.5494 |
| **std** | 8.601545 | 23.322453 | 6.860353 | 0.253994 | 0.115878 | 0.702617 | 28.148861 | 2.105710 | 8.7072 |
| **min** | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 | 2.900000 | 1.129600 | 1.0000 |
| **25%** | 0.082045 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.885500 | 45.025000 | 2.100175 | 4.0000 |
| **50%** | 0.256510 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.208500 | 77.500000 | 3.207450 | 5.0000 |
| **75%** | 3.677083 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 | 94.075000 | 5.188425 | 24.0000 |
| **max** | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 12.126500 | 24.0000 |

相関性

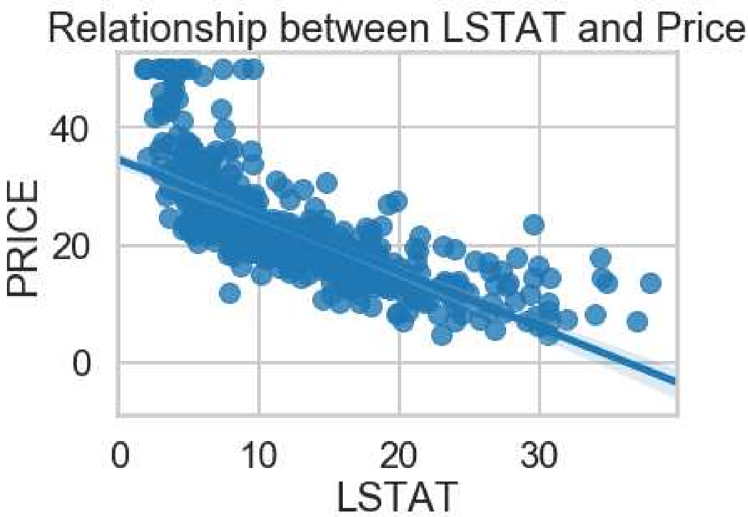
部屋数と価格の関係

In [13]:
```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
```

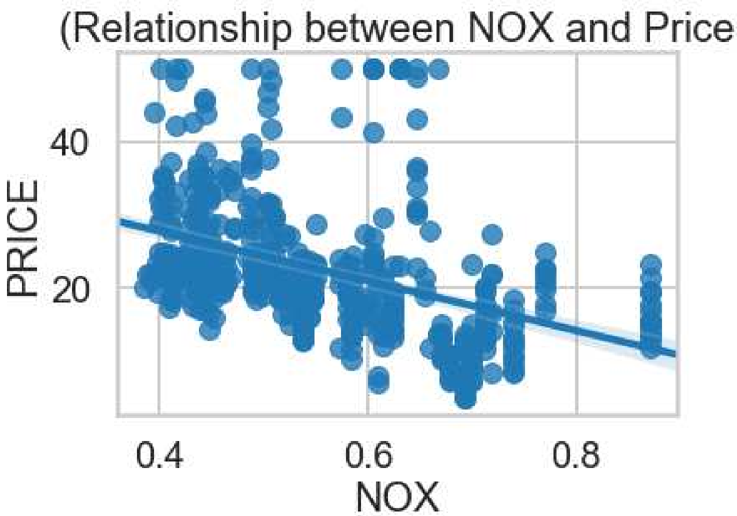
In [14]:
```
# Between prices and No. of rooms
#部屋数と価格の関係
sns.regplot(x="RM",y="PRICE", data=boston_df, fit_reg=True)
plt.title("Relationship between RM and Price")
plt.show()
```



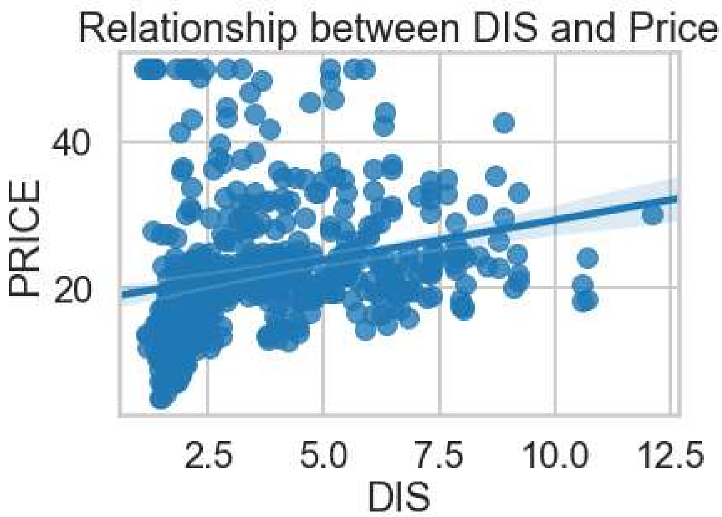
In [15]:
```
# Between prices and Lower Status Population
#低地位人口と価格の関係
sns.regplot(y="PRICE",x="LSTAT", data=boston_df, fit_reg= True)
plt.title("Relationship between LSTAT and Price")
plt.show()
```



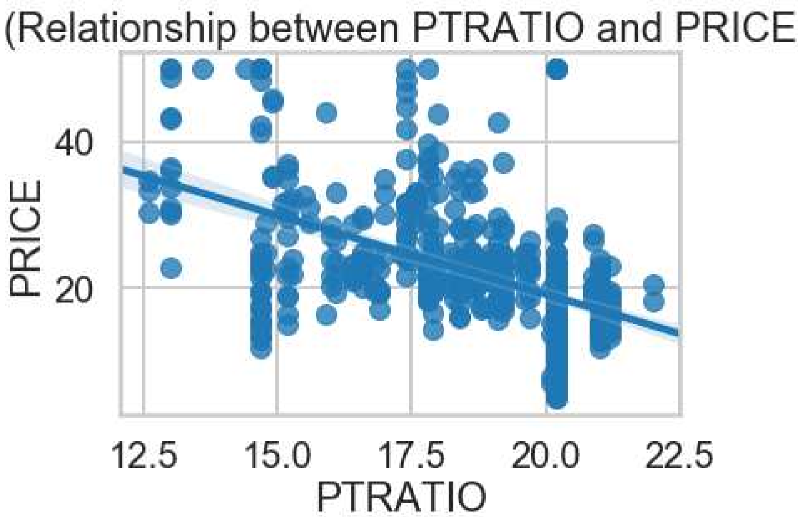

In [16]:
```
# Between prices and Nitrous Oxide Concentration
#亜酸化窒素濃度と価格の関係
sns.regplot(y="PRICE",x="NOX", data=boston_df, fit_reg= True)
plt.title("(Relationship between NOX and Price")
plt.show()
```



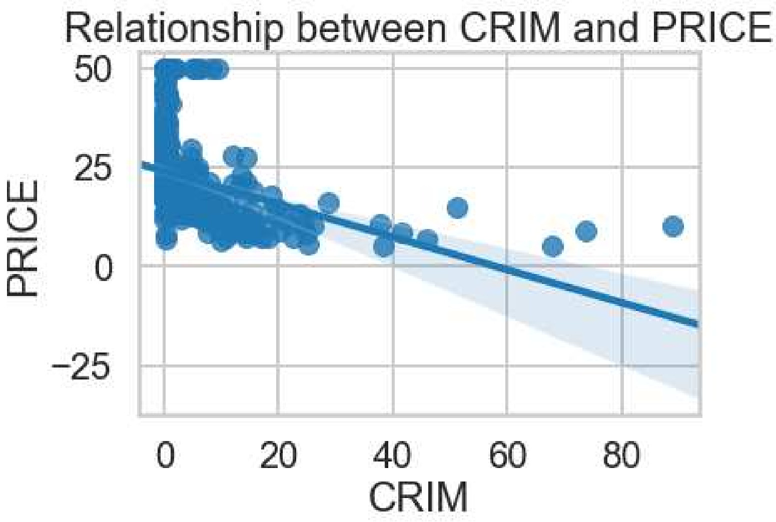
In [17]:
```
# Between prices and Weighted distance between 5 Boston Employment Center
#5つボストン雇用センター間の加重距離と価格の関係
sns.regplot(y="PRICE",x="DIS", data=boston_df, fit_reg= True)
plt.title("Relationship between DIS and Price")
plt.show()
```



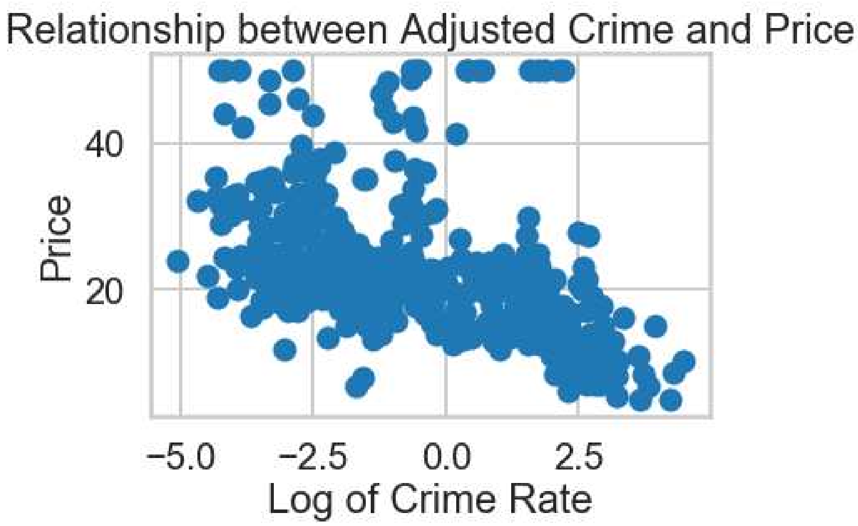

In [18]:
```
# Between prices and Pupil-Teacher ratio by town
#町による生徒教師比率と価格の関係
sns.regplot(y="PRICE",x="PTRATIO", data=boston_df, fit_reg= True)
plt.title("(Relationship between PTRATIO and PRICE")
plt.show()
```



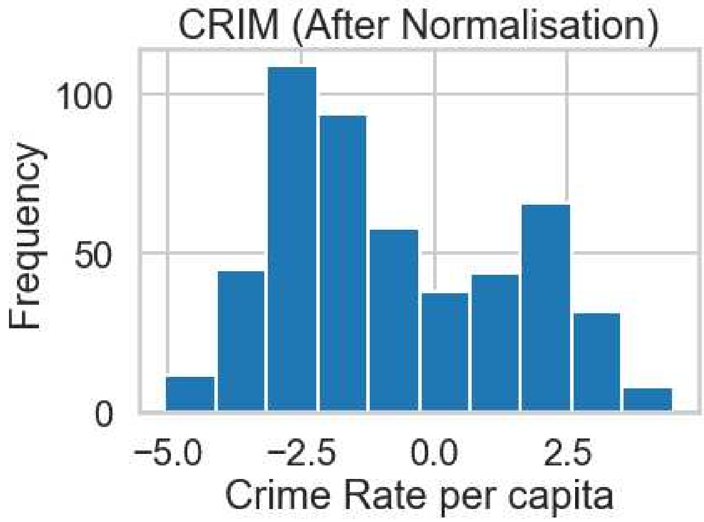
In [19]:
```
# Between prices and per capita crime rate by town
#町による一人当たりの犯罪率と価格の関係
sns.regplot(y="PRICE",x="CRIM", data=boston_df, fit_reg= True)
plt.title("Relationship between CRIM and PRICE")
plt.show()
```



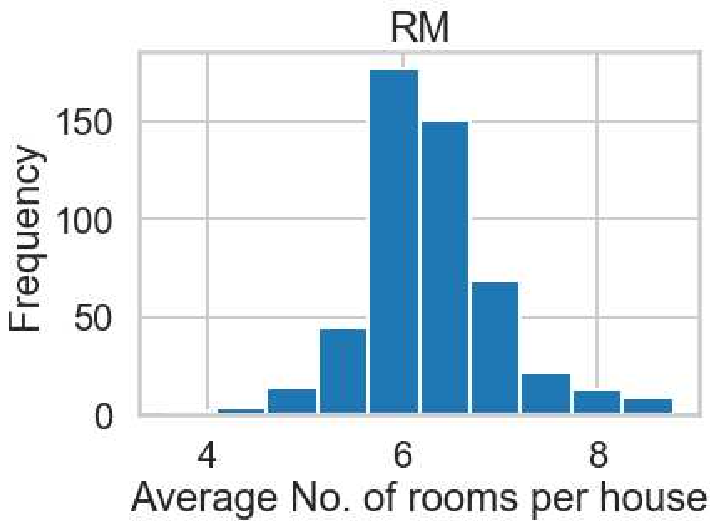

In [20]:
```
#We saw that the scatter plot between Price and Crime Rate an observed an exponentia
l decay of crimes. This can be overcome by normalising.
#町による一人当たりの犯罪率(log)と価格の関係
adj_CRIM = np.log(boston_df.CRIM)
plt.scatter(adj_CRIM , boston_df.PRICE)
plt.xlabel("Log of Crime Rate")
plt.ylabel("Price")
plt.title("Relationship between Adjusted Crime and Price")
plt.show()
```



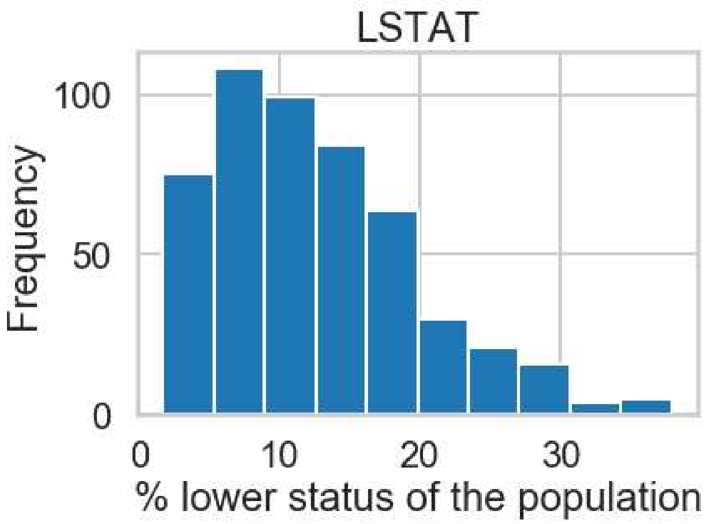
In [21]:
```
plt.hist(adj_CRIM)
plt.xlabel("Crime Rate per capita")
plt.ylabel("Frequency")
plt.title("CRIM (After Normalisation)")
plt.show()
```



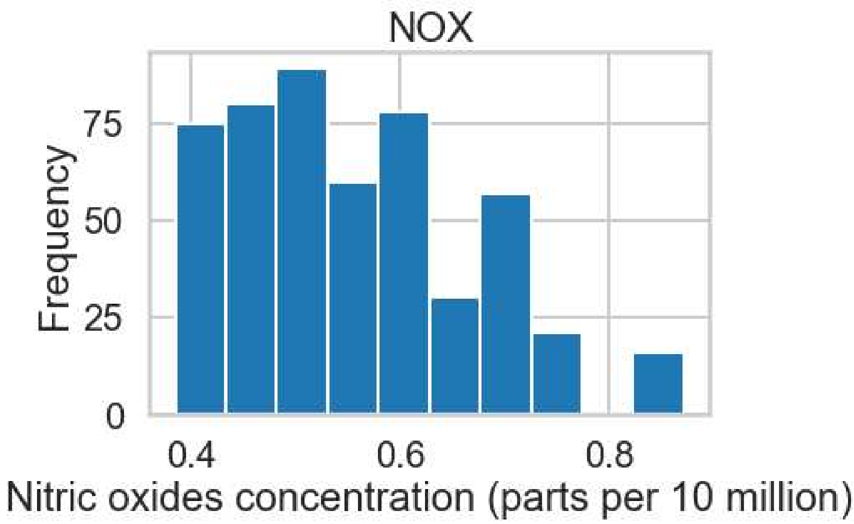

In [22]:
```
plt.hist(boston_df.RM)
plt.xlabel("Average No. of rooms per house")
plt.ylabel("Frequency")
plt.title("RM")
plt.show()
```



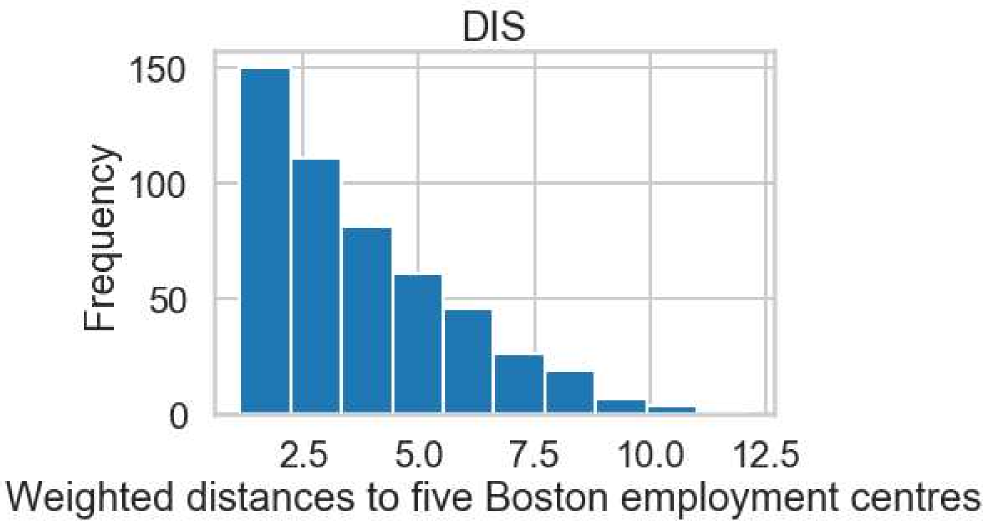
In [23]:
```
plt.hist(boston_df.LSTAT)
plt.xlabel("% lower status of the population")
plt.ylabel("Frequency")
plt.title("LSTAT")
plt.show()
```




In [24]:
```
plt.hist(boston_df.NOX)
plt.xlabel("Nitric oxides concentration (parts per 10 million)")
plt.ylabel("Frequency")
plt.title("NOX")
plt.show()
```



In [25]:
```
plt.hist(boston_df.DIS)
#plt.hist(np.log(boston_df.DIS))
plt.xlabel("Weighted distances to five Boston employment centres")
plt.ylabel("Frequency")
plt.title("DIS")
plt.show()
```



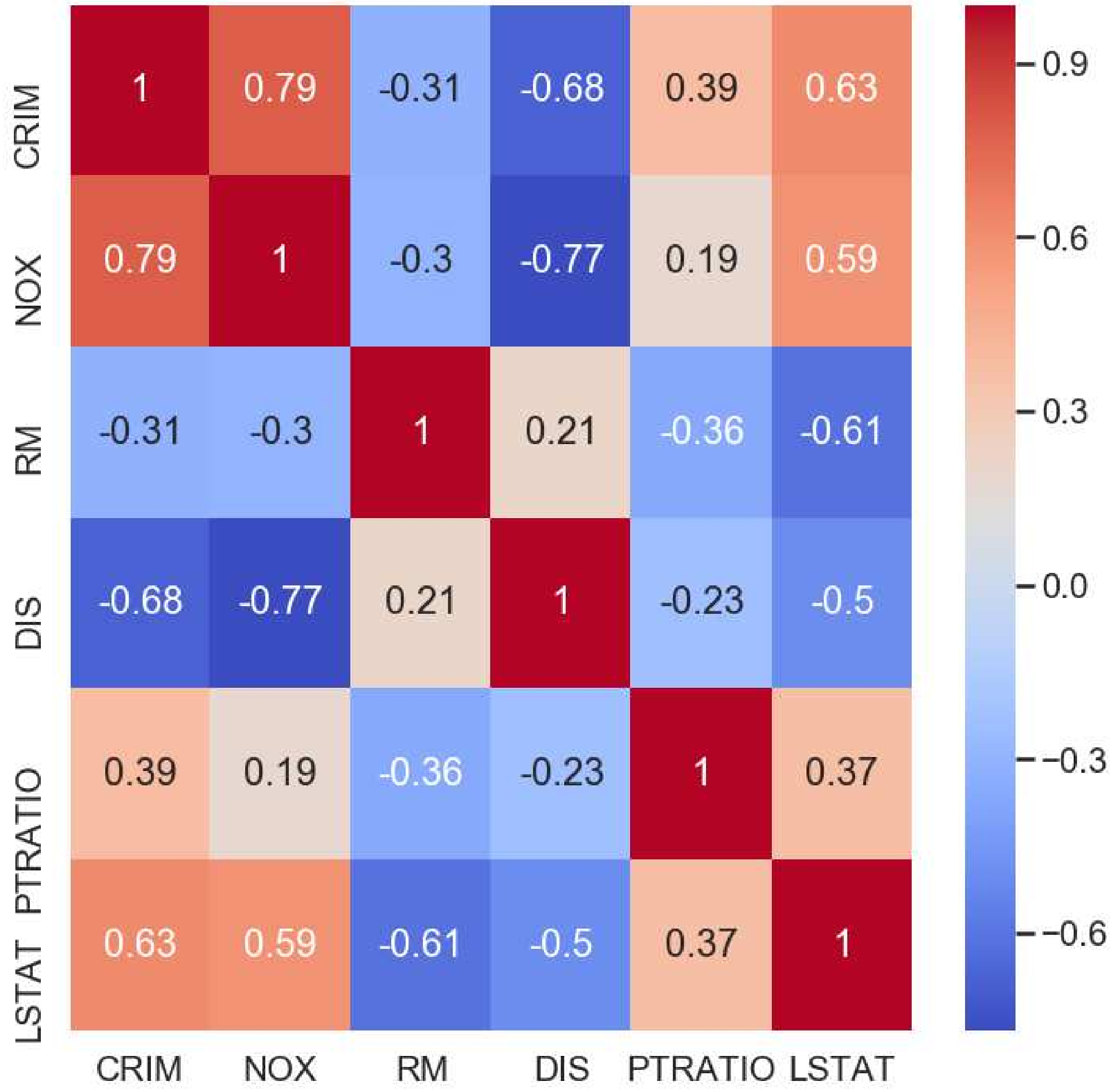

In [26]:
```
bos_df = boston_df
bos_df['CRIM'] = np.log(bos_df['CRIM'])
df = bos_df.iloc[:,[0,4,5,7,10,12]]
df.corr()
```

Out[26]:

| | CRIM | NOX | RM | DIS | PTRATIO | LSTAT |
|---|---|---|---|---|---|---|
| **CRIM** | 1.000000 | 0.788616 | -0.306943 | -0.681903 | 0.389554 | 0.626615 |
| **NOX** | 0.788616 | 1.000000 | -0.302188 | -0.769230 | 0.188933 | 0.590879 |
| **RM** | -0.306943 | -0.302188 | 1.000000 | 0.205246 | -0.355501 | -0.613808 |
| **DIS** | -0.681903 | -0.769230 | 0.205246 | 1.000000 | -0.232471 | -0.496996 |
| **PTRATIO** | 0.389554 | 0.188933 | -0.355501 | -0.232471 | 1.000000 | 0.374044 |
| **LSTAT** | 0.626615 | 0.590879 | -0.613808 | -0.496996 | 0.374044 | 1.000000 |

In [27]:
```
plt.figure(figsize=(12, 12))
sns.heatmap(df.corr(), annot = True, cmap = 'coolwarm')
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x2945802e940>




## ボストンの住宅データを使用した線形回帰の例

In [28]:
```
#Linear regression with Boston housing data example
#importing regression modules
#ols- stands for Ordinary least squares: a method for estimating unknown parameters
in a linear regression model
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

In [29]:
```
m = ols('PRICE ~ PTRATIO + NOX + RM + LSTAT + DIS ', bos_df).fit()
print(m.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  PRICE   R-squared:                       0.708
Model:                            OLS   Adj. R-squared:                  0.705
Method:                 Least Squares   F-statistic:                     242.6
Date:                Fri, 24 May 2019   Prob (F-statistic):          3.67e-131
Time:                        11:42:01   Log-Likelihood:                -1528.7
No. Observations:                 506   AIC:                             3069.
Df Residuals:                     500   BIC:                             3095.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     37.4992      4.613      8.129      0.000      28.436      46.562
PTRATIO       -1.0458      0.114     -9.212      0.000      -1.269      -0.823
NOX          -17.9966      3.261     -5.519      0.000     -24.403     -11.590
RM             4.1633      0.412     10.104      0.000       3.354       4.973
LSTAT         -0.5811      0.048    -12.122      0.000      -0.675      -0.487
DIS           -1.1847      0.168     -7.034      0.000      -1.516      -0.854
==============================================================================
Omnibus:                      187.456   Durbin-Watson:                   0.971
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              885.498
Skew:                           1.584   Prob(JB):                    5.21e-193
Kurtosis:                       8.654   Cond. No.                         545.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
```

係数の解釈

上記の結果にはたくさんの情報があります。 係数表にのみ集中しています（中央）

非常に小さい（基本的にゼロ）p（p> | t |）値の解釈を始めます。 それは私たちの機能が私たちのPRICEの統計的に有意な予測因子であることを意味します。

一般的に、各係数は対応するフィーチャの単位増加による増減として解釈できます。たとえば、2つの町のグループを比較すると、平均部屋数は5、他のグループは同じです。 全室6部屋あります。 これら2つのグループの住宅価格の平均差は約9.1（千単位）なので約\ 9,100差はRMのcoefにすぎません。 信頼区間により、この差について（¥8,279、¥9,925）というもっともらしい値の範囲がわかります。

平均して単位面積あたりのNOX濃度の増加を言うNOXによって示されたもう一つの重要な特徴は、他の変数を除いて最終的に住宅価格を18,000ドル減少させるでしょう。 信頼区間により、この差について（\ 11,000、\ 24,000）について妥当な値の範囲がわかります。

DISで示されるもう1つの重要な特徴は、平均してDISの1単位の増加（5つのボストンの雇用センターまでの加重距離）は、住宅価格を最終的に他の変数の正味1,000ドル減少させることです。 信頼区間から、この差についてもっともらしい値の範囲（約¥854、¥1,500）がわかります。

モデル係数の信頼区間の解釈

統計モデルは私たちのモデル係数の95%信頼区間を計算します。これは次のように解釈することができます - このサンプルが採取された母集団が100回サンプリングされると、それらの信頼区間の約95%に真の係数が含まれます。

> 95%は単なる慣例ですが
>
> > 90%信頼区間を作成できます（これはより狭くなります）。
> > 99%の信頼区間を作成できます（これはもっと広くなります）。
>
> 上記の範囲は、係数が含まれる可能性のある範囲です。

R2乗値の解釈

決定係数または単にR 2乗値とも呼ばれます。 回帰直線が実際のデータポイントにどの程度近似しているかを示す統計的尺度。線形モデルの全体的な近似を評価する方法。

与えられたモデルでは、R2乗値は0.708です。これは基本的に、価格の総分散の約70%が現在の回帰モデルによって決定できることを意味します。

In [30]:
```
predicted_prices = m.fittedvalues
plt.scatter(  bos_df.PRICE , predicted_prices)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Vs Predicted Price")
plt.show()
```

In [31]:
```
#Spliting test and train set

from sklearn.model_selection import train_test_split
X = bos_df.drop('PRICE', axis = 1)
Y = bos_df['PRICE']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =0.33,random_sta
te = 5 )
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

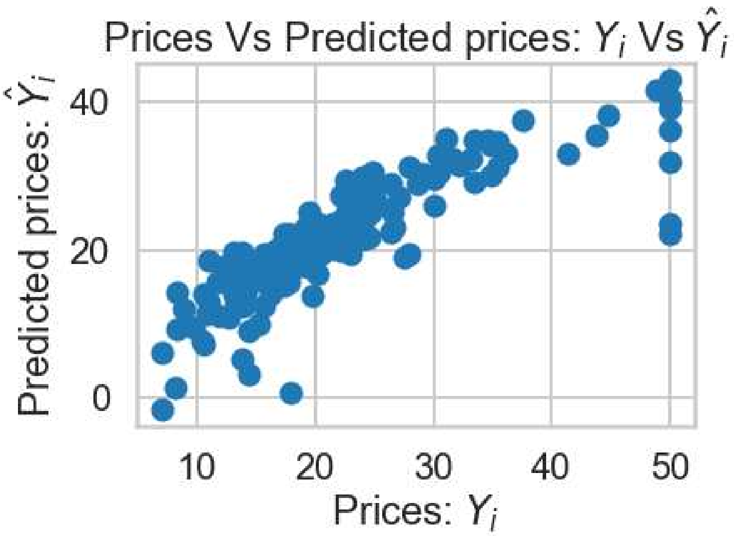
```
(339, 13)
(167, 13)
(339,)
(167,)
```

In [32]:
```
from sklearn.linear_model import LinearRegression

LinReg = LinearRegression()
LinReg.fit(X_train,Y_train)

Y_pred = LinReg.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices Vs Predicted prices: $Y_i$ Vs $\hat{Y}_i$")
plt.show()
```




In [33]:
```
# displaying the coefficients of parameters
print("Coefficients: \n", LinReg.coef_)

# displaying the R-squared score
print(LinReg.score(X_test , Y_test))
```

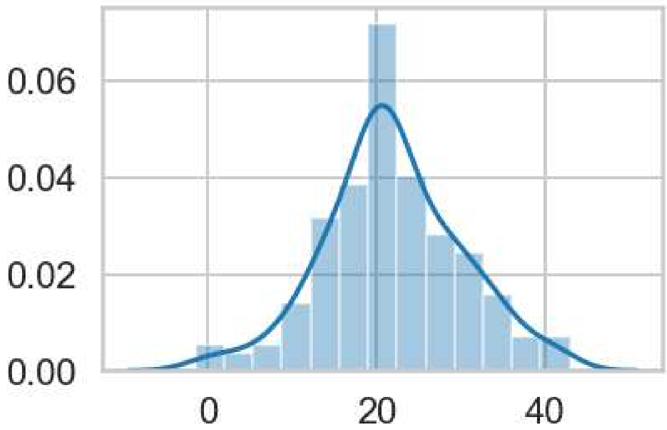
```
Coefficients:
 [ 2.91429472e-01  3.63624438e-02 -2.47555447e-02  1.10925522e+00
 -1.29834557e+01  3.89110219e+00 -1.18923107e-02 -1.24562550e+00
  2.23495573e-01 -1.36600521e-02 -9.70026335e-01  1.17800958e-02
 -5.34376398e-01]
0.7086789571201157
```

In [34]:
```
# generating the mean squared error
mse = sklearn.metrics.mean_squared_error(Y_test, Y_pred)
mse
```

Out[34]:
```
27.309558311498133
```

In [35]:
```
sns.distplot(Y_pred)
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x294580213c8>




In [36]:
```
# mean absolute error which is the average of all predicted error values ,where all
predicted error values are forced to be positive
print(sklearn.metrics.mean_absolute_error(Y_test, Y_pred))

#root mean squared error is the root of the average of the squared predicted error v
alues.
print(np.sqrt(mse))
```

```
3.3781785953800116
5.225854792423736
```

In [37]:
```
#Linear Regression (5 features only)
from sklearn.model_selection import train_test_split
X1 = bos_df[['NOX','RM','DIS','PTRATIO','LSTAT' ]]
Y1 = bos_df['PRICE']

X_train, X_test, Y_train, Y_test = train_test_split(X1, Y1, test_size =0.33,random_s
tate = 5 )
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

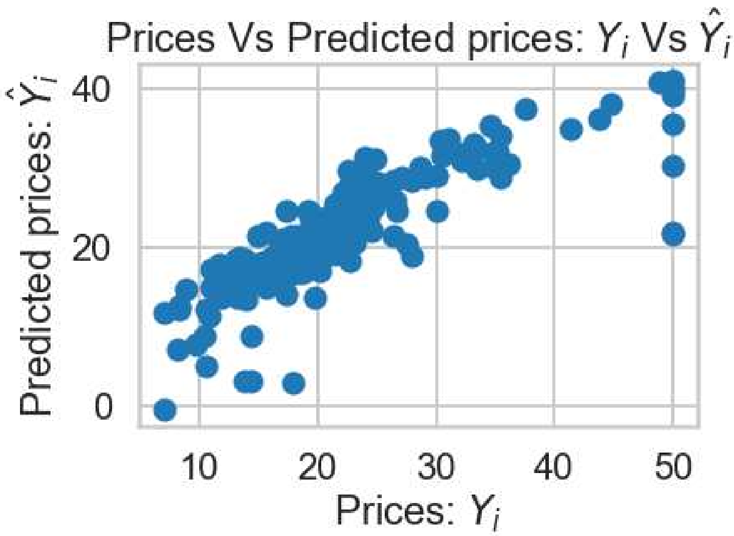
```
(339, 5)
(167, 5)
(339,)
(167,)
```

In [38]:
```
from sklearn.linear_model import LinearRegression

LinReg = LinearRegression()
LinReg.fit(X_train,Y_train)

Y_pred = LinReg.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices Vs Predicted prices: $Y_i$ Vs $\hat{Y}_i$")
plt.show()
```




In [39]:
```
print(LinReg.score(X_test , Y_test))
cv_results = sklearn.model_selection.cross_val_score(LinReg, X_train, Y_train, cv = 
5, scoring = 'r2')
msg = "%s: %f (%f)" % ('r2 score', cv_results.mean(),cv_results.std())
print(msg)
```

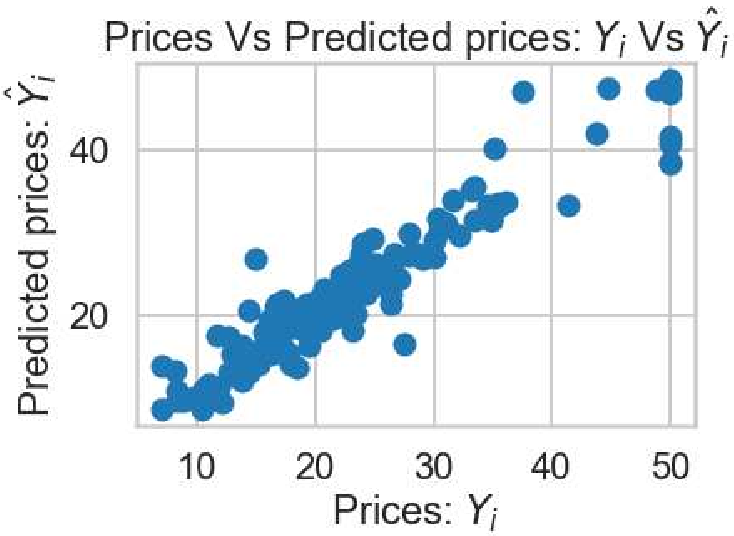
```
0.6840953220739522
r2 score: 0.694977 (0.029360)
```

## ランダムフォレスト回帰

In [40]:
```
#RandomForrest Regression
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=500, oob_score=True, random_state=0)
rf.fit(X_train , Y_train)
Y_pred = rf.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices Vs Predicted prices: $Y_i$ Vs $\hat{Y}_i$")
plt.show()
#print(rf.score(X_test , Y_test))
```




In [41]:
```
print(sklearn.metrics.mean_absolute_error(Y_test, Y_pred))
print(np.sqrt(mse))
```

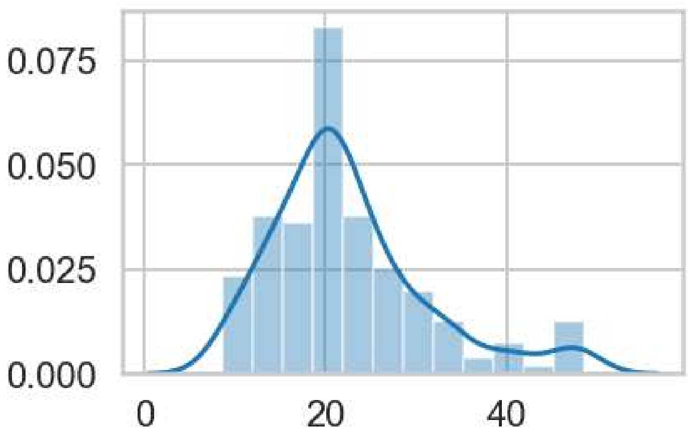
```
2.2367425149700546
5.225854792423736
```

In [42]:
```
print(rf.score(X_test , Y_test))
```

```
0.8935026873334214
```

In [43]:
```
sns.distplot(Y_pred)
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x29457eb8dd8>



In [44]:
```
from sklearn.metrics import r2_score
from scipy.stats import spearmanr, pearsonr

test_score = r2_score(Y_test , Y_pred)
spearman = spearmanr(Y_test, Y_pred)
pearson = pearsonr(Y_test, Y_pred)
print("Out of Bag R2 Score" , rf.oob_score_)
print("Test Data R2 Score:" , test_score)
print("Test Data Spearman Correlation:" ,  round(spearman[0], 3))
print("Test Data Pearson Correlation" , round(pearson[0],3))
```

```
Out of Bag R2 Score 0.8211453241420169
Test Data R2 Score: 0.8935026873334214
Test Data Spearman Correlation: 0.919
Test Data Pearson Correlation 0.946
```

In [45]:
```
type(boston)
```

Out[45]:
```
sklearn.utils.Bunch
```

In [46]:
```
type(boston.data)
```

Out[46]:
```
numpy.ndarray
```

In [ ]:

In [ ]: