Boston Housing with Linear Regression

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [3]:
```
# Importing DataSet and take a look at Data
BostonTrain = pd.read_csv("boston_train.csv")
```

** Here we can look at the BostonTrain data **

In [4]:
```
BostonTrain.head()
```

Out[4]:

	ID	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lsta
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
3	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
4	7	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43

In [5]:
```
BostonTrain.info()
BostonTrain.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 333 entries, 0 to 332
Data columns (total 15 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   ID       333 non-null    int64
 1   crim     333 non-null    float64
 2   zn       333 non-null    float64
 3   indus    333 non-null    float64
 4   chas     333 non-null    int64
 5   nox      333 non-null    float64
 6   rm       333 non-null    float64
 7   age      333 non-null    float64
 8   dis      333 non-null    float64
 9   rad      333 non-null    int64
 10  tax      333 non-null    int64
 11  ptratio  333 non-null    float64
 12  black    333 non-null    float64
 13  lstat    333 non-null    float64
 14  medv     333 non-null    float64
dtypes: float64(11), int64(4)
memory usage: 39.1 KB
```

Out[5]:

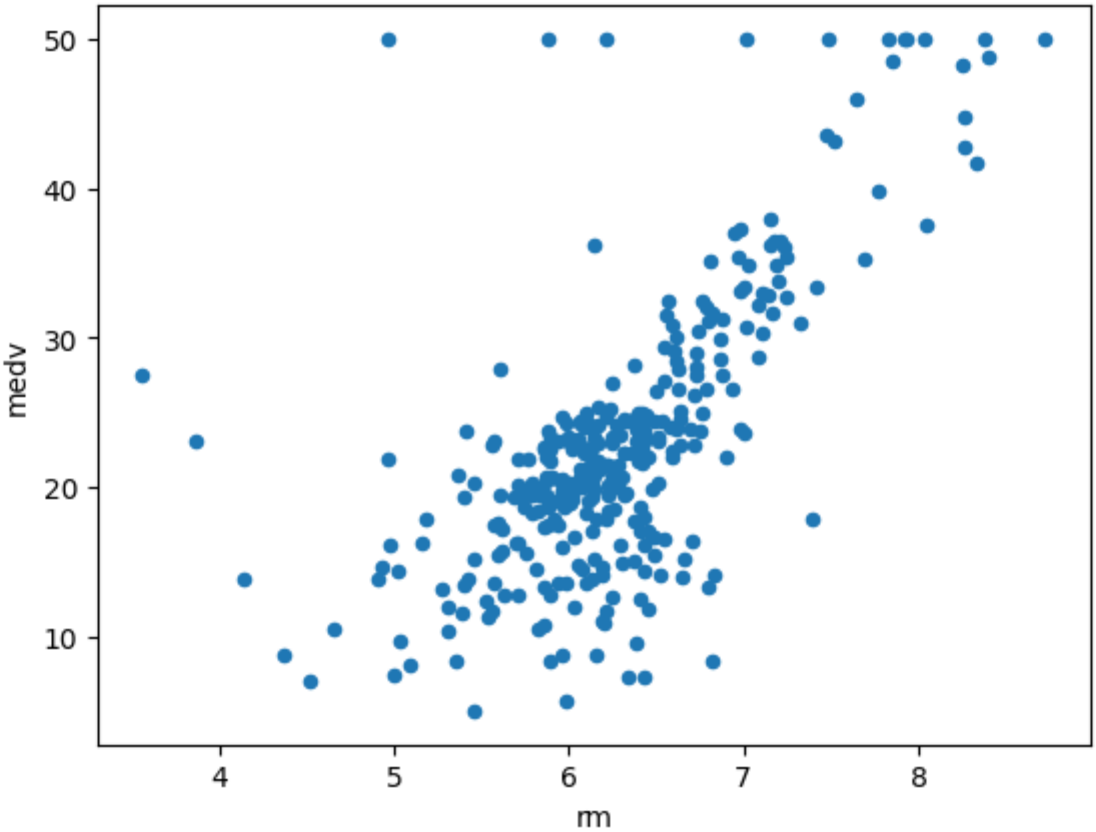
	ID	crim	zn	indus	chas	nox	rm
count	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000
mean	250.951952	3.360341	10.689189	11.293483	0.060060	0.557144	6.265619
std	147.859438	7.352272	22.674762	6.998123	0.237956	0.114955	0.703952
min	1.000000	0.006320	0.000000	0.740000	0.000000	0.385000	3.561000
25%	123.000000	0.078960	0.000000	5.130000	0.000000	0.453000	5.884000
50%	244.000000	0.261690	0.000000	9.900000	0.000000	0.538000	6.202000
75%	377.000000	3.678220	12.500000	18.100000	0.000000	0.631000	6.595000
max	506.000000	73.534100	100.000000	27.740000	1.000000	0.871000	8.725000

** Now, or goal is think about the columns, and discovery which columns is relevant to build our model, because if we consider to put columns with not relevant with our objective "medv" the model may be not efficient **

In [6]:
```
#ID columns does not relevant for our analysis.
BostonTrain.drop('ID', axis = 1, inplace=True)
```

In [7]:
```
BostonTrain.plot.scatter('rm', 'medv')
```

Out[7]: <Axes: xlabel='rm', ylabel='medv'>

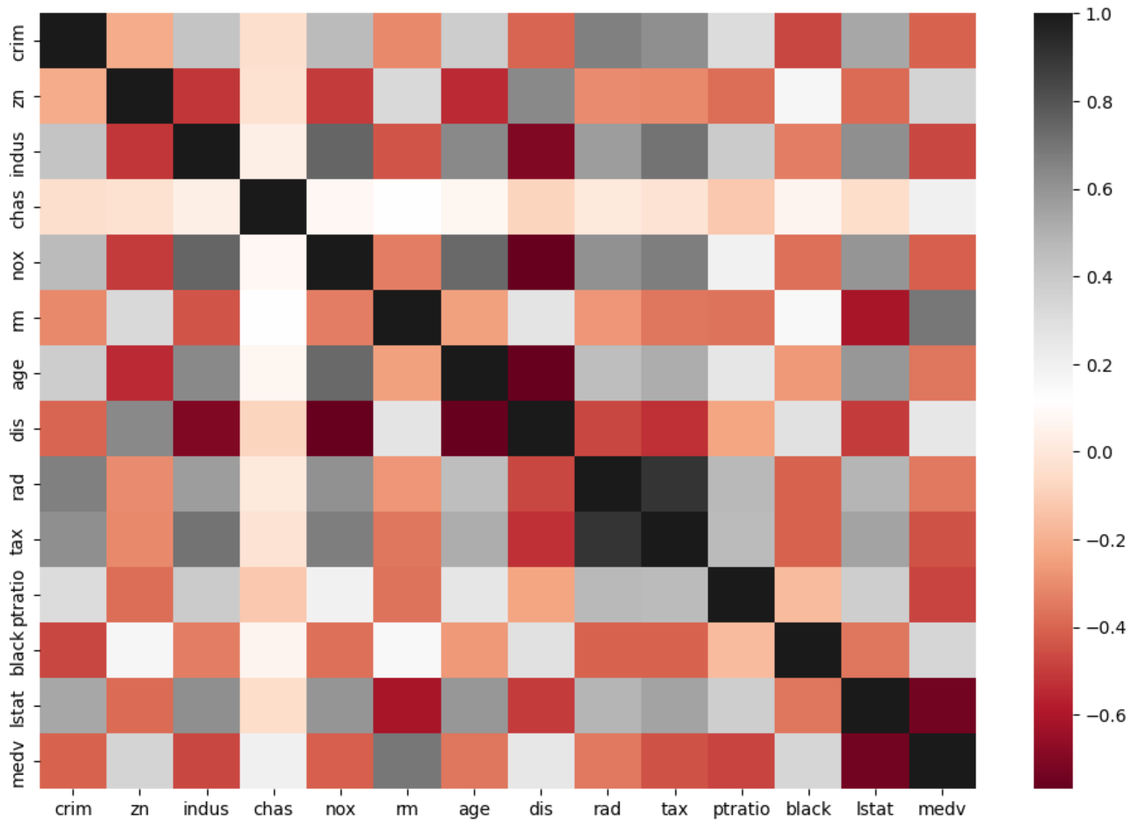


In this plot its clearly to see a linear pattern. Wheter more average number of rooms per dwelling, more expensive the median value is.

** Now lets take a loot how the all variables relate to each other. **

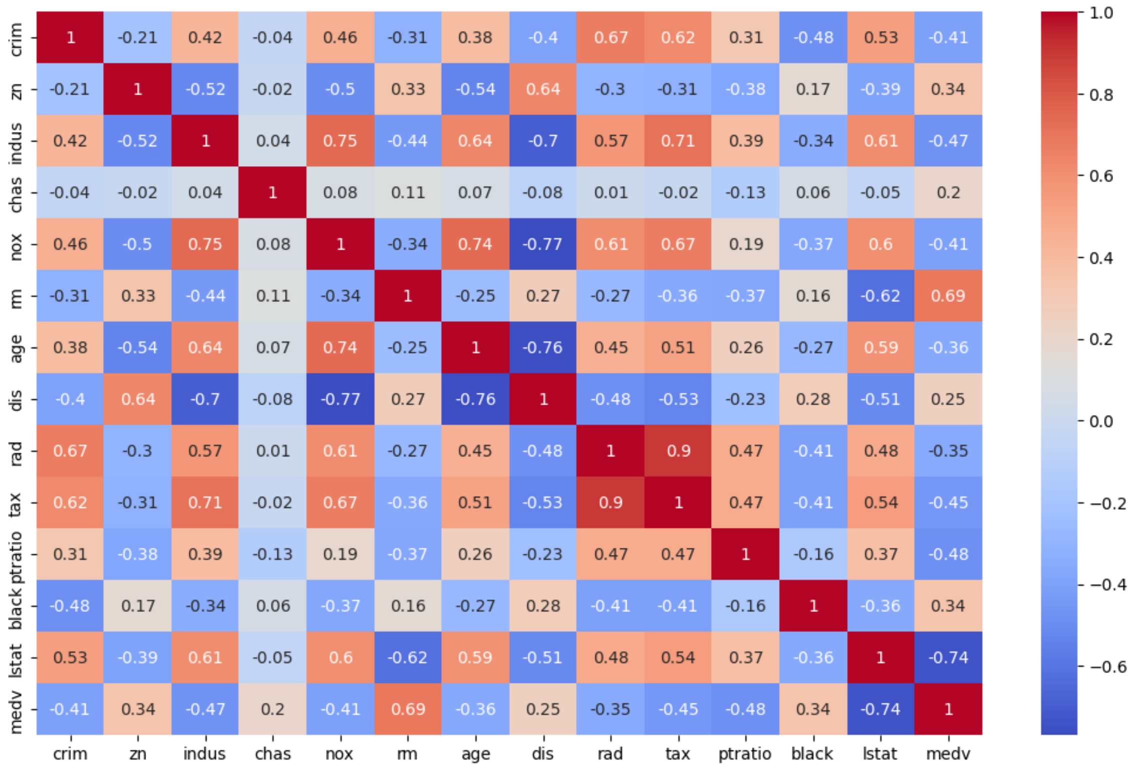
In [8]:
```
plt.subplots(figsize=(12,8))
sns.heatmap(BostonTrain.corr(), cmap = 'RdGy')
```

Out[8]: <Axes: >



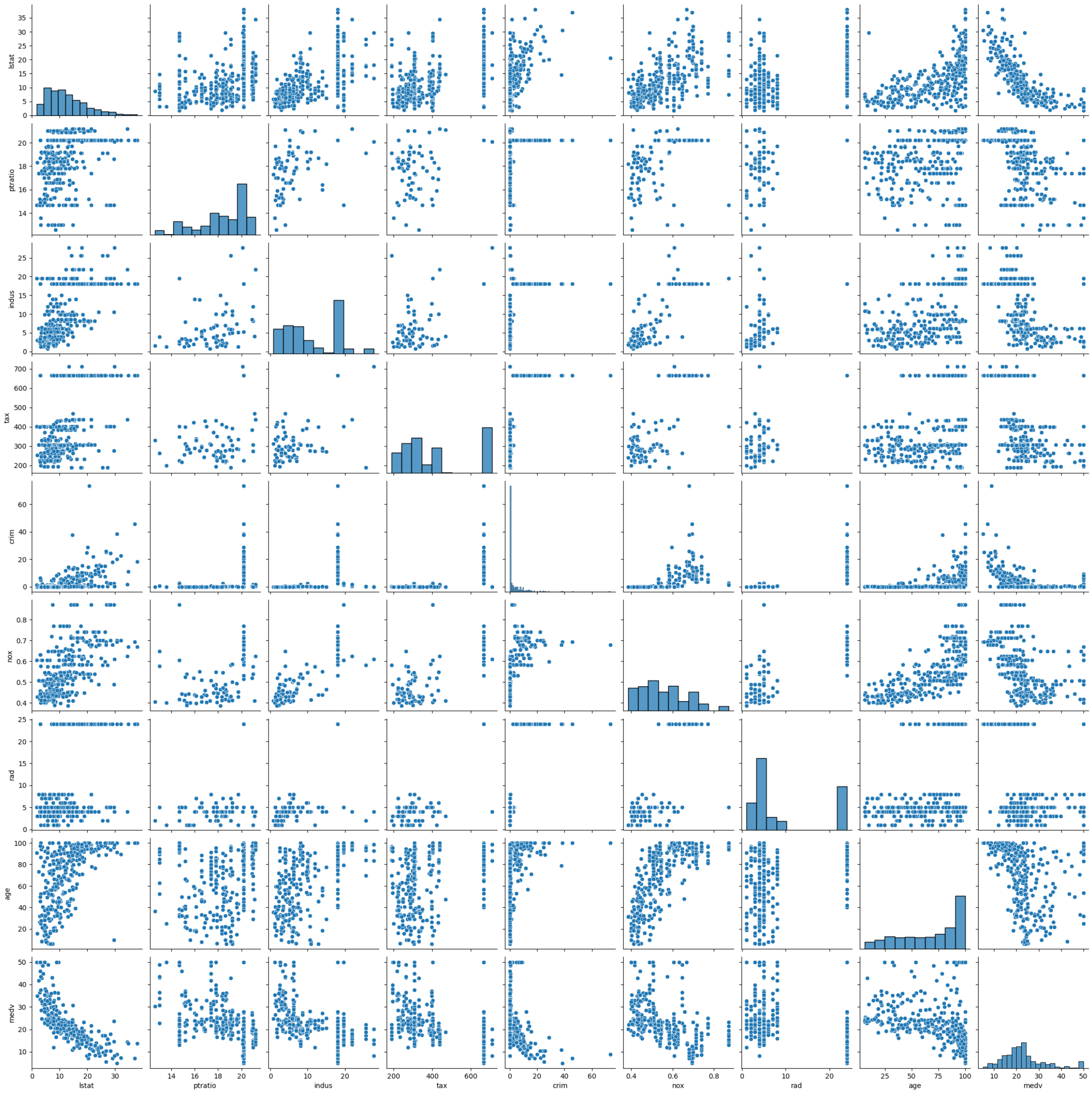
In [9]:
```python
# Plot the correlation heatmap
plt.figure(figsize=(13, 8))
corr_matrix = BostonTrain.corr().round(2)
sns.heatmap(data=corr_matrix,cmap='coolwarm',annot=True)
```

Out[9]:
```
<Axes: >
```



In [10]:
```
sns.pairplot(BostonTrain, vars = ['lstat', 'ptratio', 'indus', 'tax',
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x7f7eb4298f70>
```



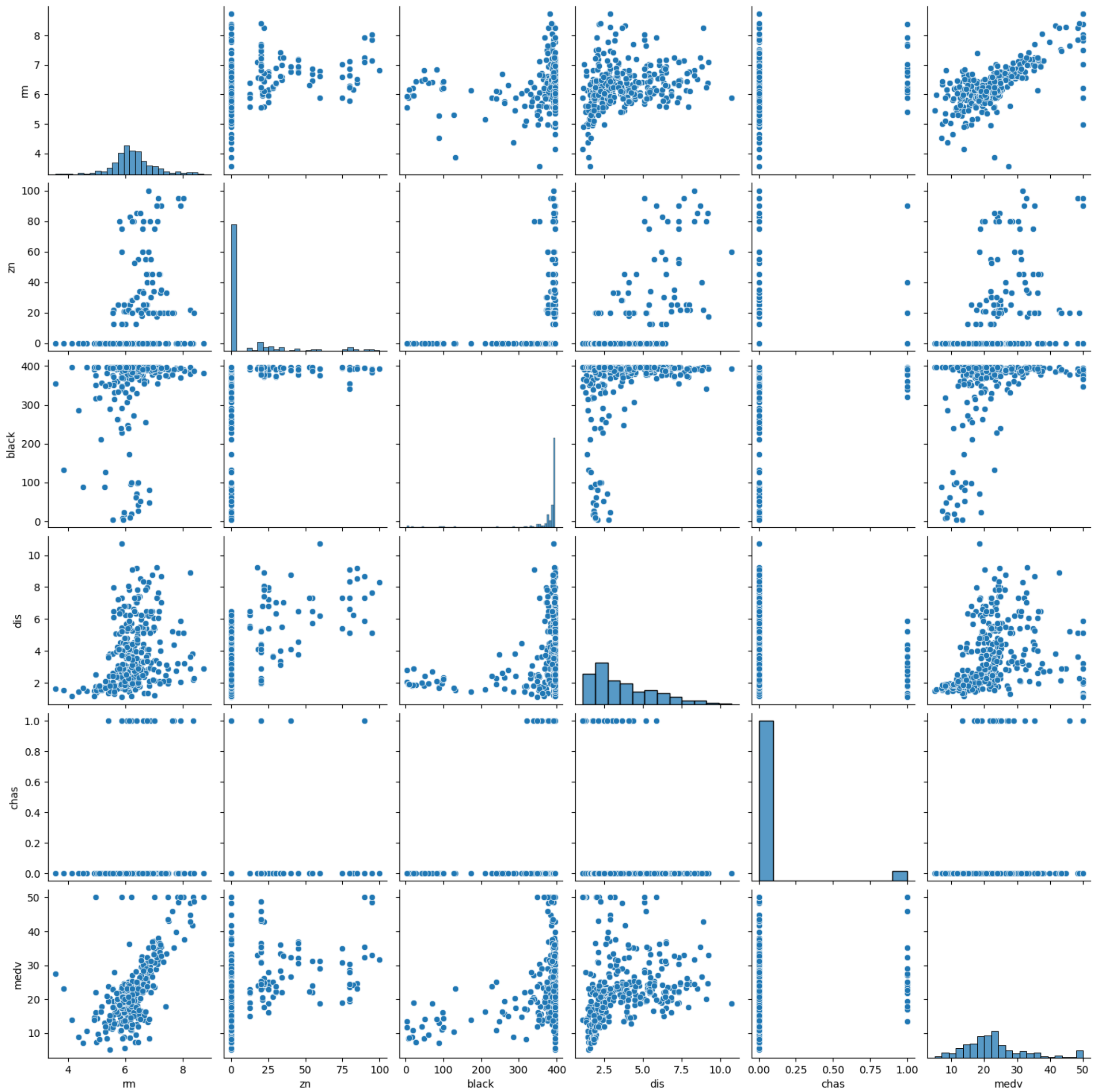
Zero Correlation. When x and y are completely independent

Positive Correlation. When x and y go together

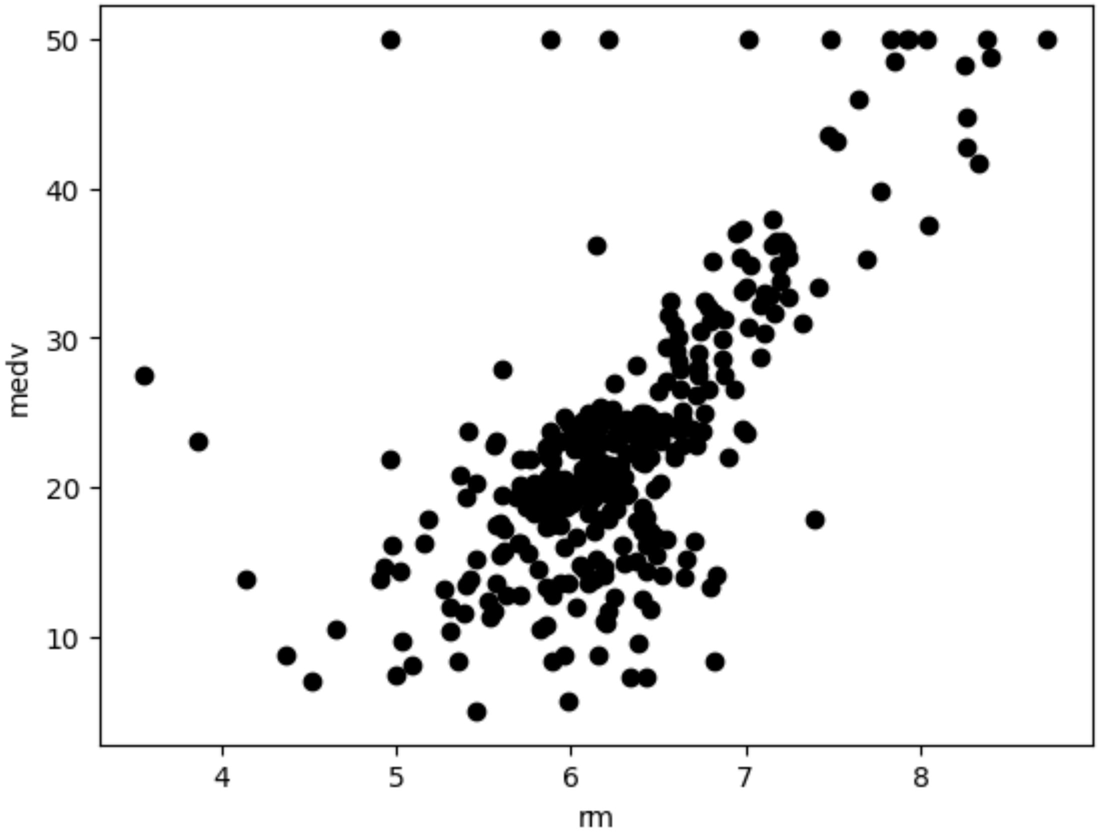
to the right more independent.

In [11]:
```
sns.pairplot(BostonTrain, vars = ['rm', 'zn', 'black', 'dis', 'chas',
```

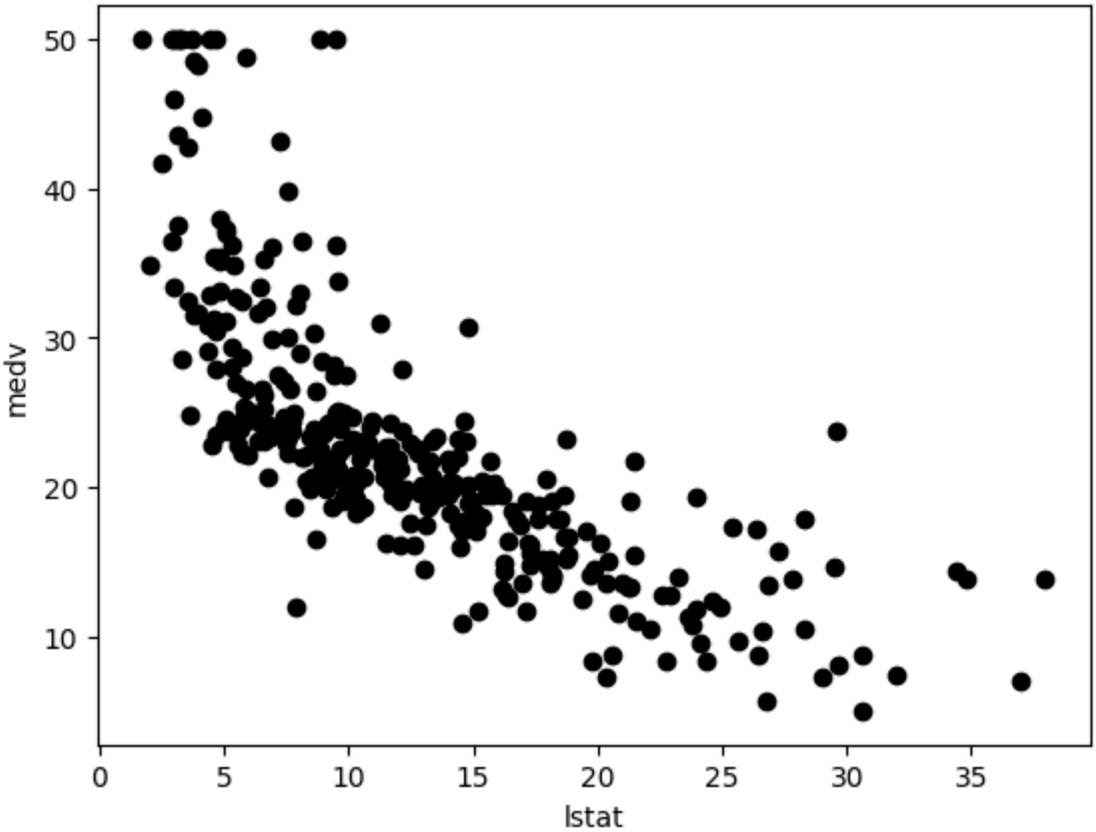
Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x7f7eb0ee4a00>
```



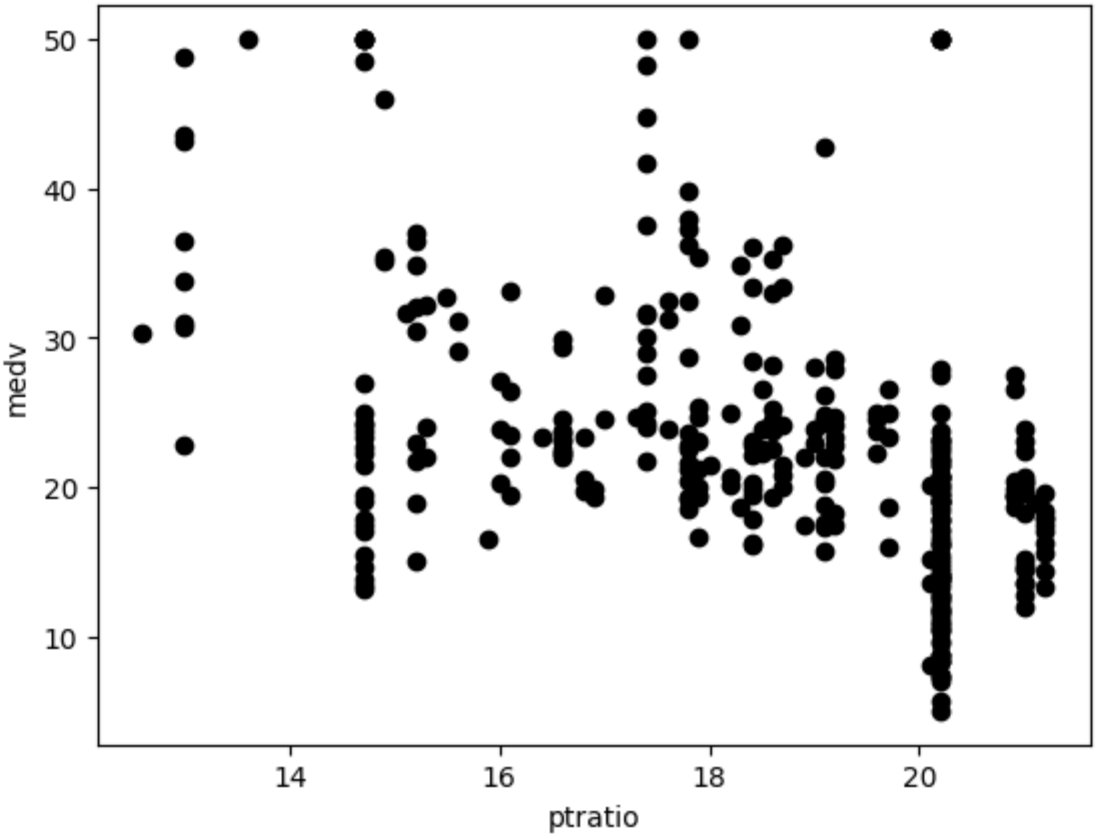
In [12]:
```
plt.scatter(BostonTrain.rm, BostonTrain.medv, color="black")
plt.xlabel("rm")
plt.ylabel("medv")
plt.show()
```



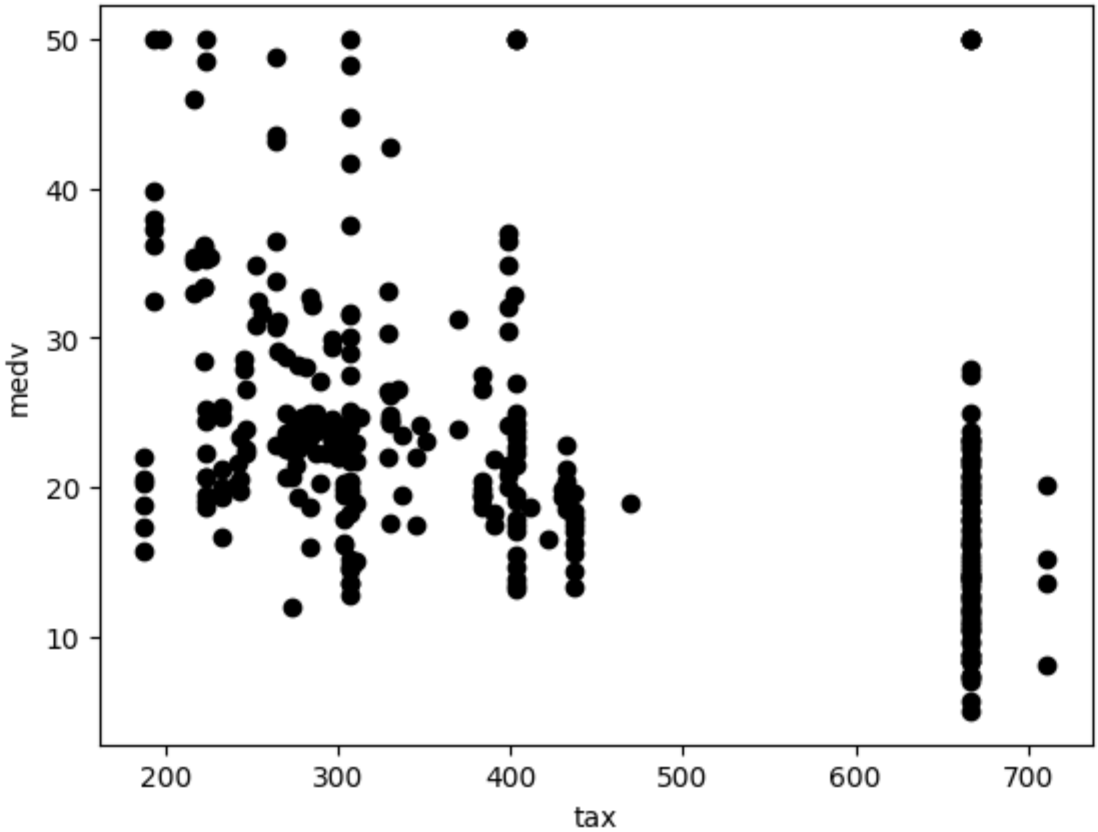
In [13]:
```
plt.scatter(BostonTrain.lstat, BostonTrain.medv, color="black")
plt.xlabel("lstat")
plt.ylabel("medv")
plt.show()
```



In [14]:
```
plt.scatter(BostonTrain.ptratio, BostonTrain.medv, color="black")
plt.xlabel("ptratio")
plt.ylabel("medv")
plt.show()
```



In [15]:
```
plt.scatter(BostonTrain.tax, BostonTrain.medv, color="black")
plt.xlabel("tax")
plt.ylabel("medv")
plt.show()
```



Trainning Linear Regression Model

Define X and Y

X: Varibles named as predictors, independent variables, features.
Y: Variable named as response or dependent variable

In [16]:
```
X = BostonTrain[['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age',
       'ptratio', 'black', 'lstat']]
y = BostonTrain['medv']
```

Import sklearn librarys:
train_test_split, to split our data in two DF, one for build a model and other to validate.
LinearRegression, to apply the linear regression.

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=(
```

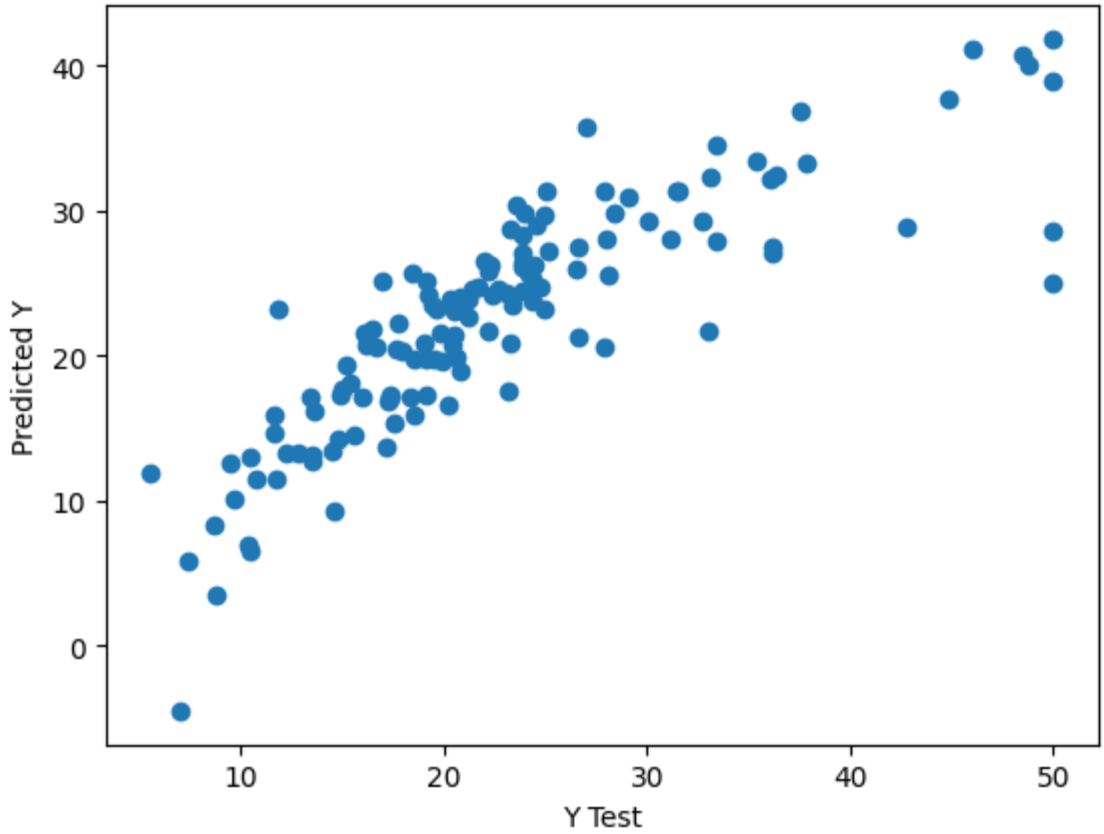
In [19]:
```
lm = LinearRegression()
lm.fit(X_train,y_train)
```

Out[19]:
```
▾ LinearRegression
LinearRegression()
```

In [20]:
```
predictions = lm.predict(X_test)
```

In [21]:
```
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[21]: `Text(0, 0.5, 'Predicted Y')`



In [22]:
```
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions
```

```
MAE: 3.598745438922565
MSE: 26.543390153530236
RMSE: 5.152027771036394
```

Considering the RMSE: we can conclude that this model average error is RMSE at medv, which means RMSE *1000 in money

In [23]:
```
sns.distplot((y_test-predictions),bins=50);
```

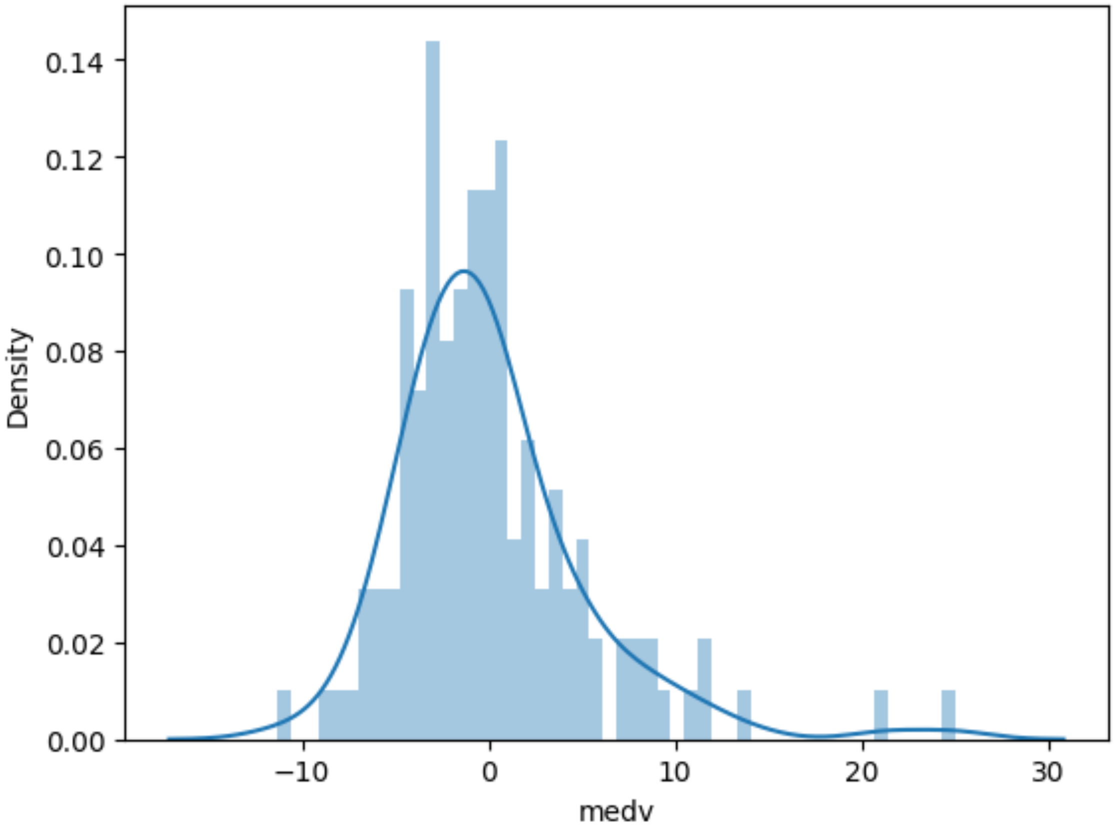
```
/tmp/ipykernel_18633/1326397652.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level func
tion with
similar flexibility) or `histplot` (an axes-level function for hist
ograms).

For a guide to updating your code to use the new functions, please
see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (h
ttps://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot((y_test-predictions),bins=50);
```



As more normal distribution, better it is.

In [24]:
```
coefficients = pd.DataFrame(lm.coef_,X.columns)
coefficients.columns = ['coefficients']
coefficients
```

Out[24]:

	coefficients
crim	-0.101348
zn	0.041950
indus	0.082293
chas	4.528386
nox	-17.282491
rm	3.527580
age	-0.005519
dis	-1.638703
rad	0.227202
tax	-0.009251
ptratio	-0.912317
black	0.010462
lstat	-0.568786