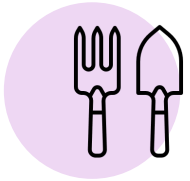
波士頓房價預測

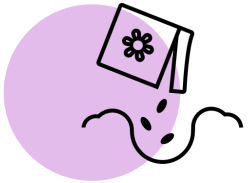
金塊隊
406170267 資數四 胡嘉芮
406170279 資數四 王芷涵

流程表



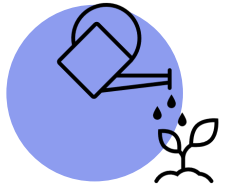
探索數據

1. 安裝套件，抓取資料，了解資料統計量有哪些?
2. 整理dataframe數據集，透過相關係數探討房價與變數的相關性(r>0.5)
3. 分出訓練與驗證數據，檢查遺失值



可視化

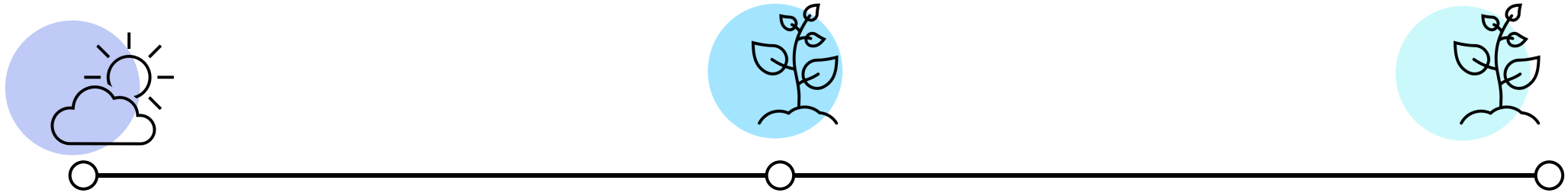
1. 列出房價統計量，分類連續跟非連續
2. 畫出散佈圖或盒鬚圖
3. 透過圖了解房價與各房子特徵的關係



特徵工程

1. 建立回歸模型，找出閾值
2. 將特徵進行標準化，使執行結果更準確
3. 作特徵篩選，將係數與閾值(True:大於閾值影響大)
4. 用圖表呈現出特徵重要性的排序

流程表



拆分數據

1. 前幾項重要性高的特徵，建立新的分析數據集(7項)
2. 使用決策樹進行拆分工程

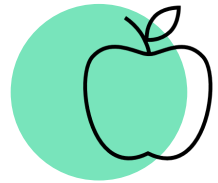
建立模型

1. 利用新數據計算 r2_train,r2_test 及 r2_adj
2. 建立七項變數的迴歸模型
3. 建立 XGB 模型

交互驗證

1. 用 k-fold 和 XGboost 的方式，將 train 數據拆分進行驗證

流程表



評估修正模型

1. 透過新數據計算判定係數 R square、R square adj、MSE 來評估修正模型
2. 最後得出用來預測的最佳化回歸模型

殘差圖表

顯示觀測的預測值與觀測的殘差之間的關係。(觀測的殘差為預測回應值與實際回應值之間的差異)

探索數據

```
#load data from sklearn datasets
import pandas as pd
from sklearn import datasets
boston = datasets.load_boston()
X, y = boston.data, boston.target
print(boston.keys())
X, boston.feature_names
```

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

(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]]),
 array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'))
```

抓取波士頓房價的樣本資料，得到影響房價的參考特徵，並整理dataframe

探索數據

```
# 分出訓練 跟 驗證
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.197, random_state=1)

print("number of test samples :", X_test.shape[0])
print("number of training samples:",X_train.shape[0])
```

```
number of test samples : 100
number of training samples: 406
```

```
traindata.shape
```

```
(406, 14)
```

用506個樣本去解釋13個房子特徵與房價之間的關係。
其中406是訓練樣本, 剩下的100組數據是驗證樣本。

探索數據

```
X_train.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
dtype: int64
```

檢查遺失值，觀察是否有問題資料

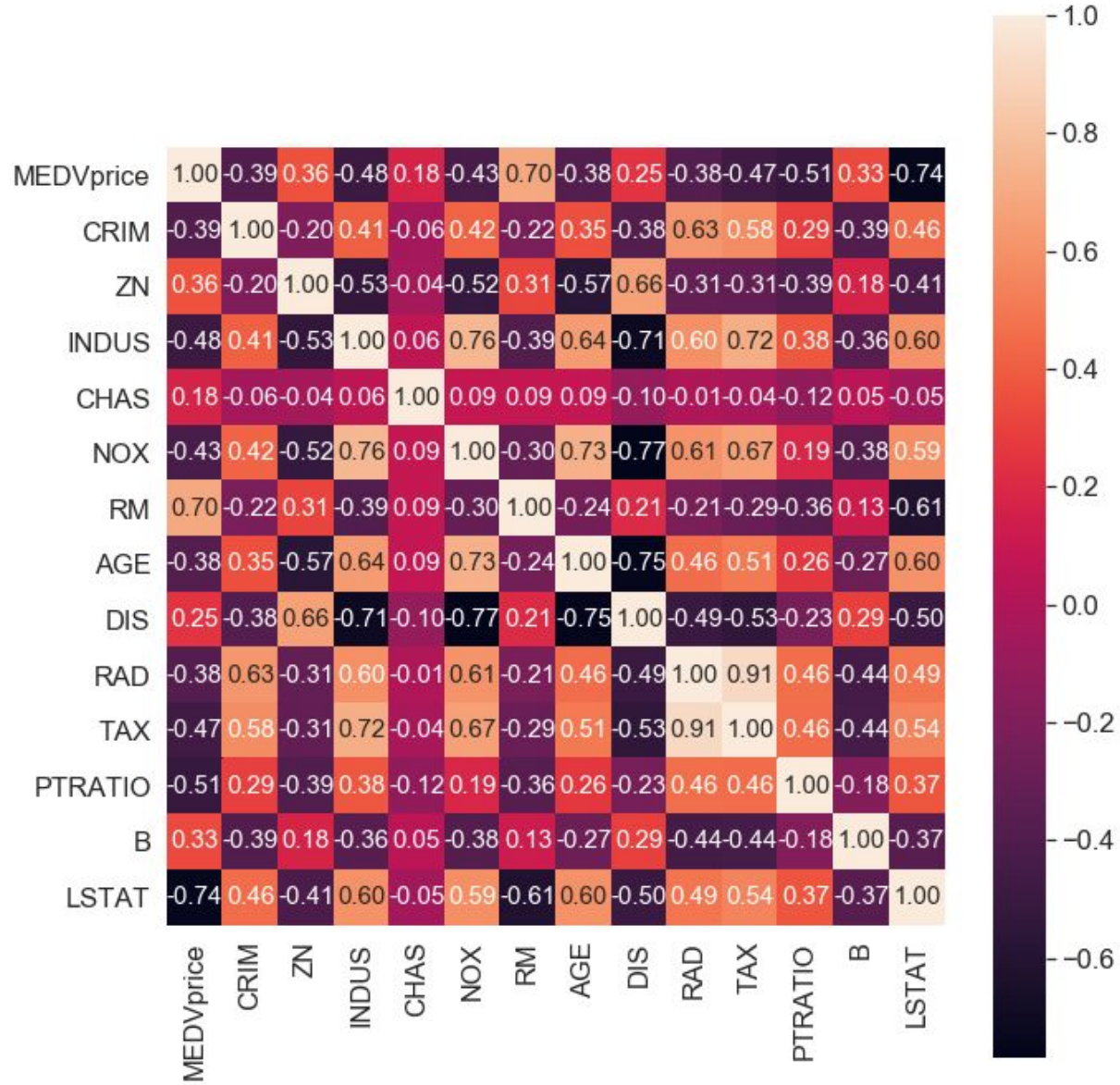
房子特徵與每月房價相關性

	CRIM	ZN	INDUS	CHAS	NOX	RM
MEDVprice	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360

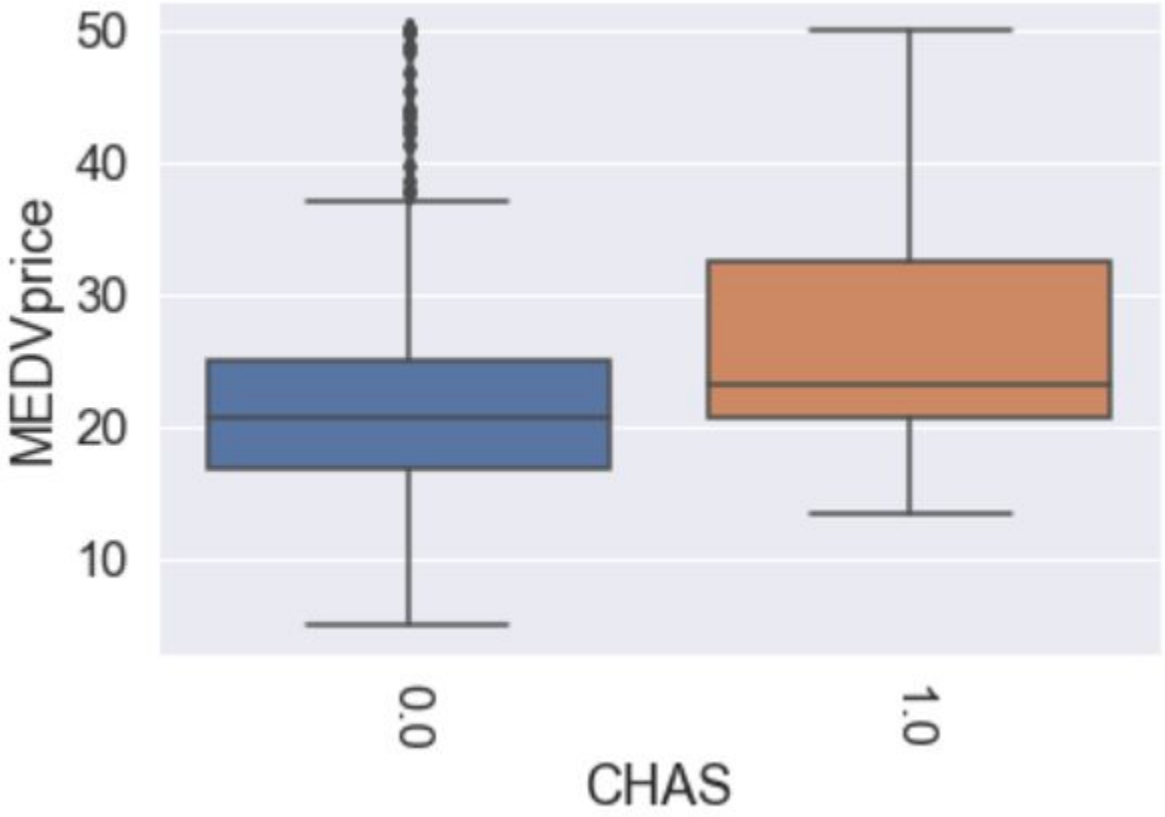
	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
MEDVprice	-0.376955	0.249929	-0.381626	-0.468536	-0.507787	0.333461	-0.737663

相關係數|r|>0.5表示該特徵與房價的相關程度越大

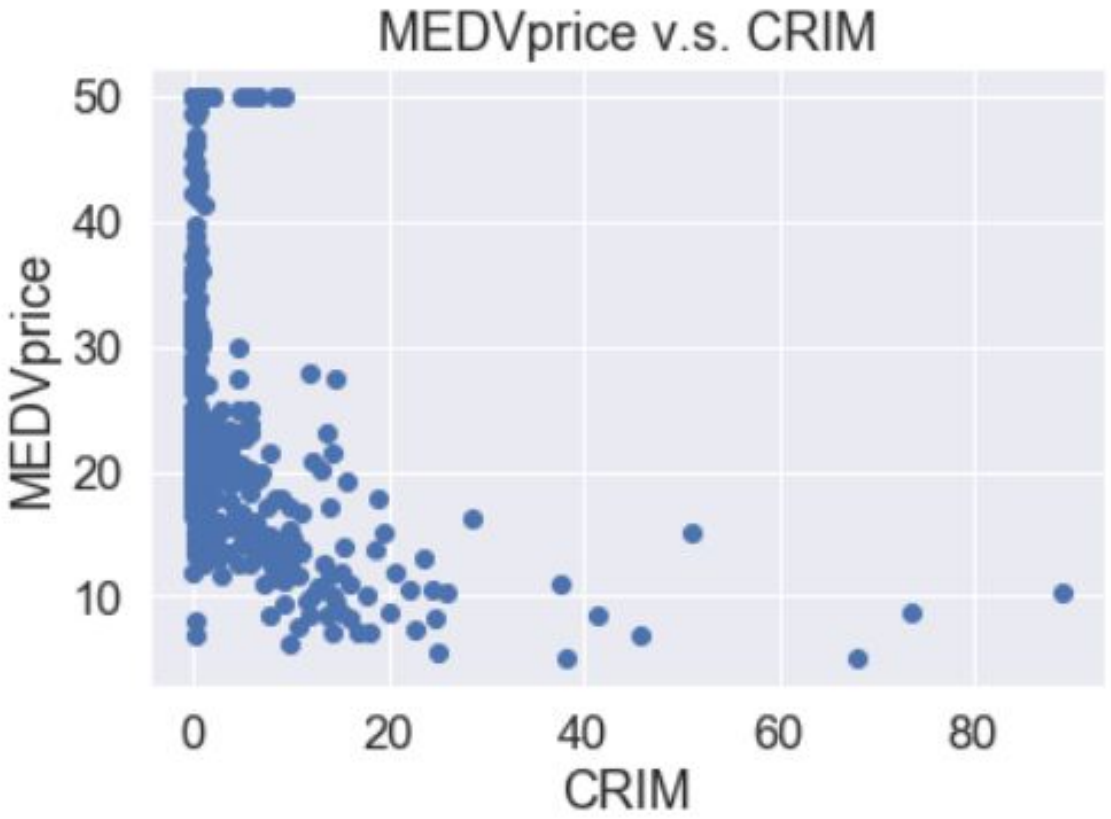
房子特徵與每月房價相關性



可視化

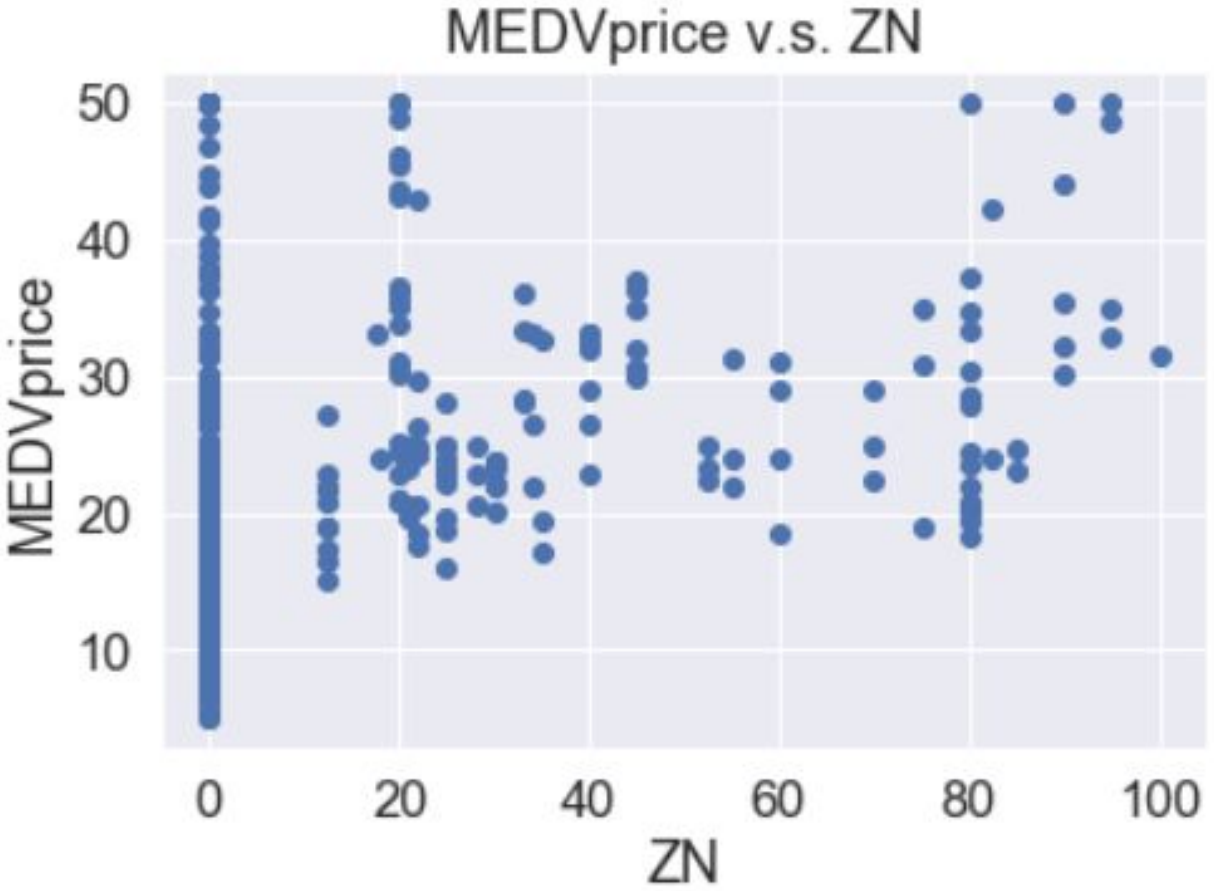


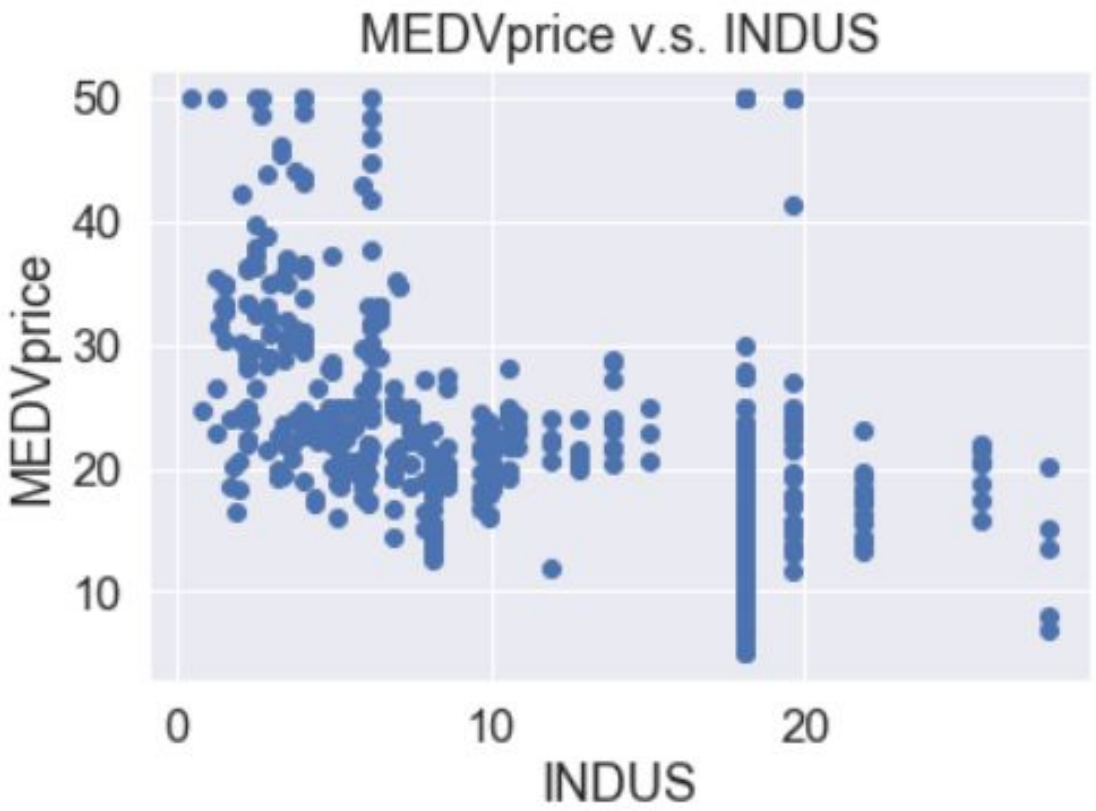
自用住宅的房價中位數(單位:1,000 美金)與查理斯河(如果房屋靠近河邊, 則為1;否則為0)的關係圖。



自用住宅的房價中位數與城鎮的人均犯罪率關係圖。

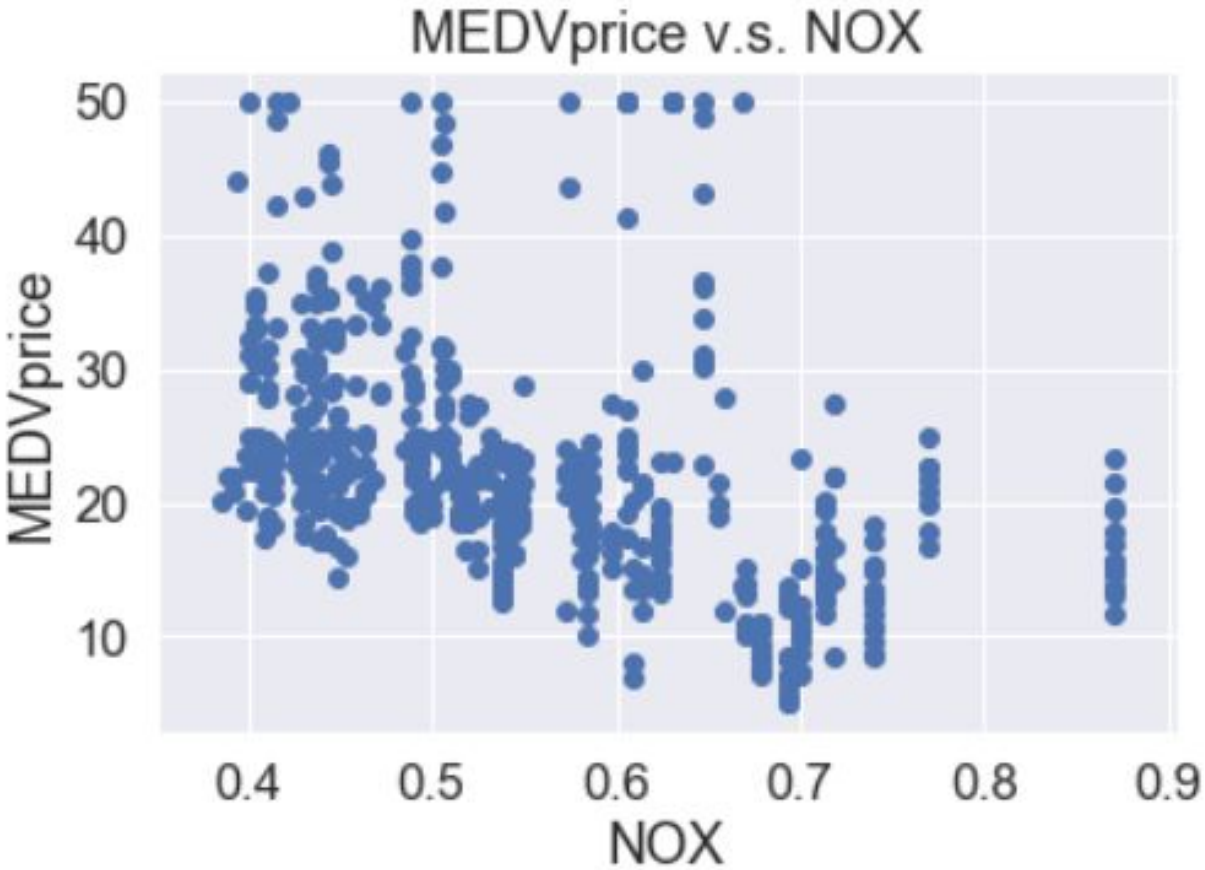
自用住宅的房價中位數與超過25,000平方呎的住宅用地所占的比例關係圖。

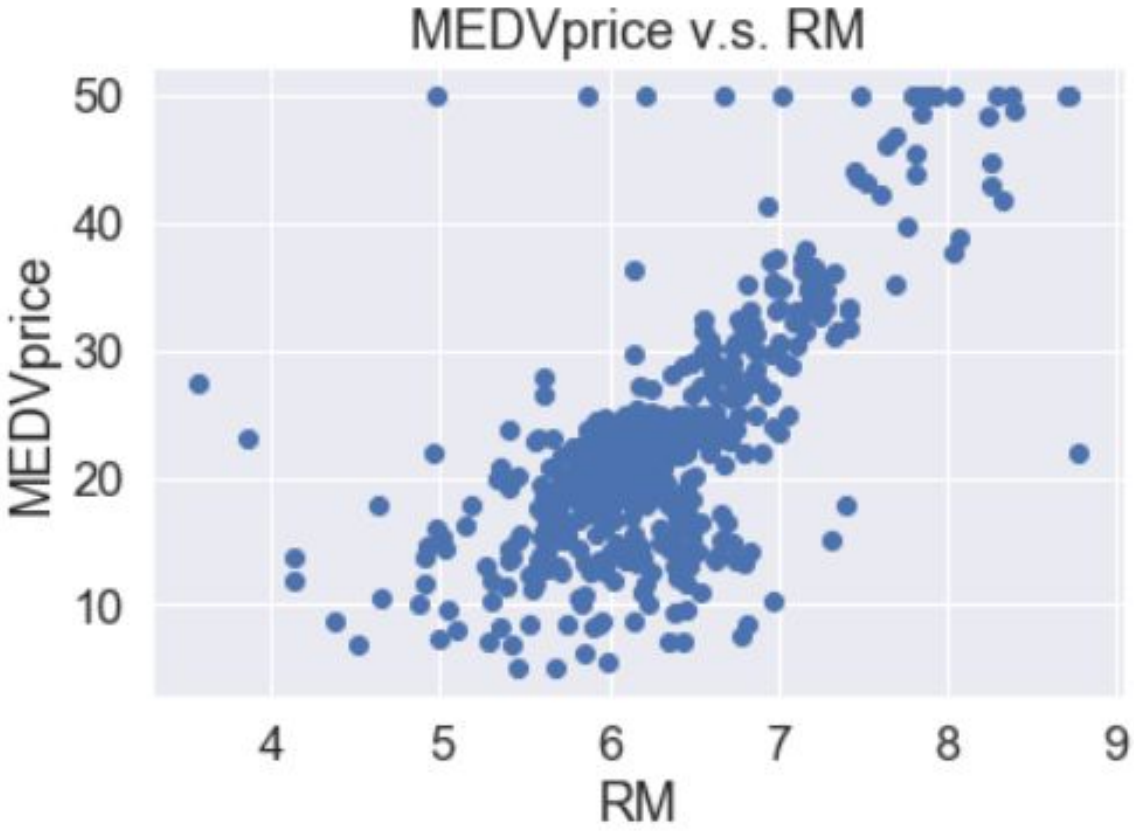




自用住宅的房價中位數與某城鎮非零售商業用地的比例(英畝)關係圖。

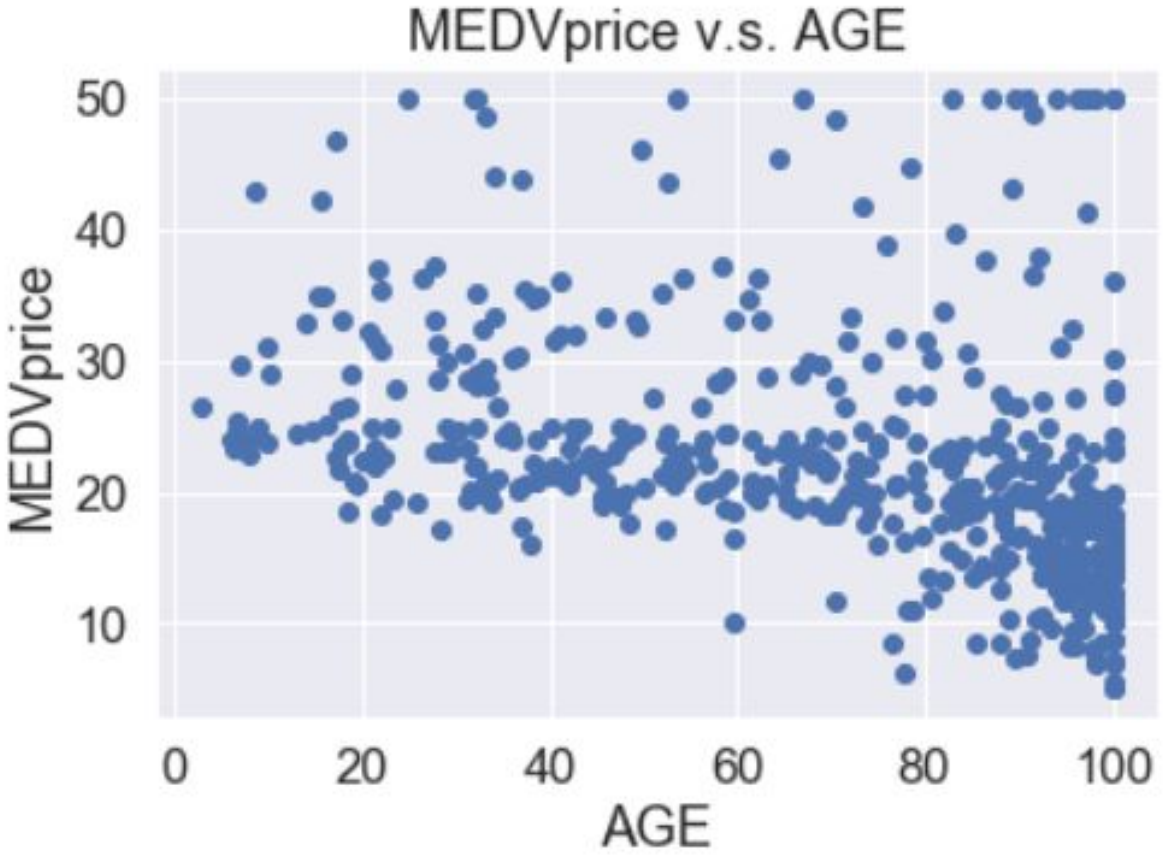
自用住宅的房價中位數與一氧化氮濃度(以10 ppm 為單位)的關係圖。

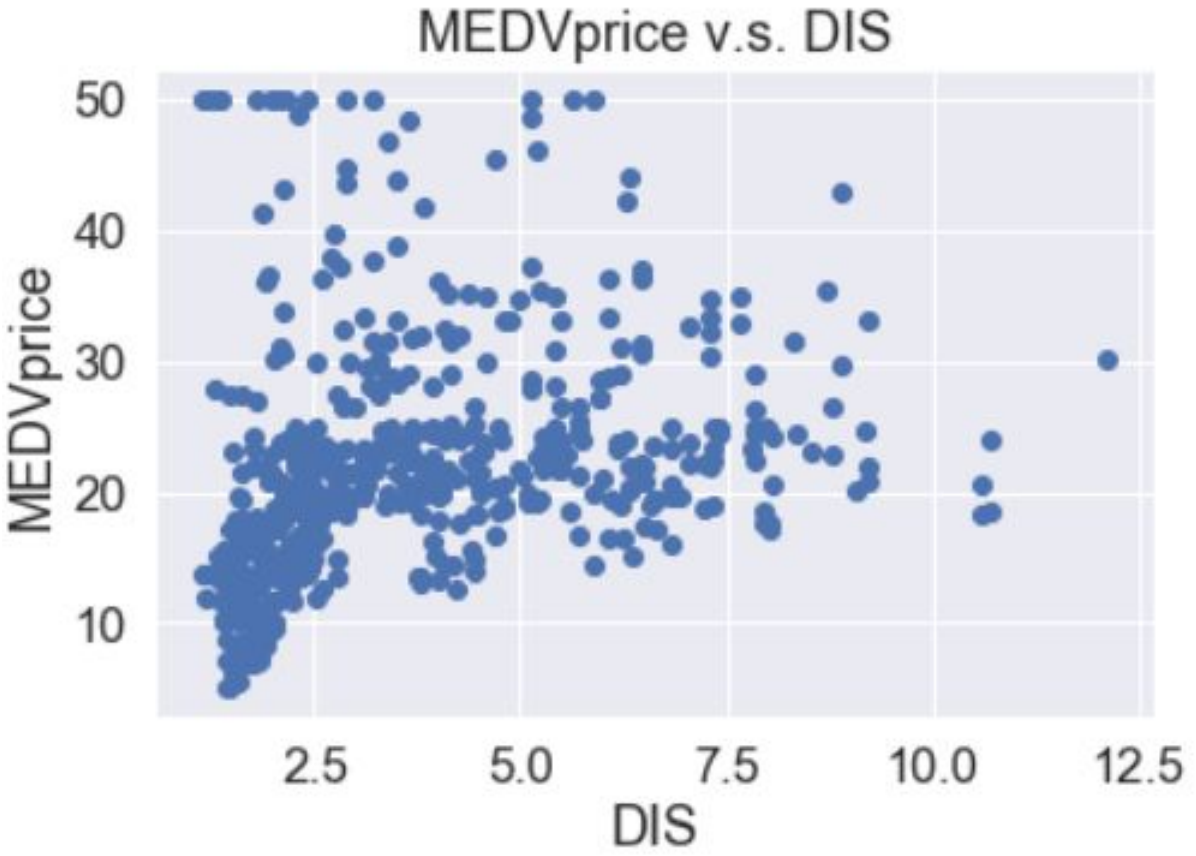




自用住宅的房價中位數與每戶住宅平均房間數關係圖。

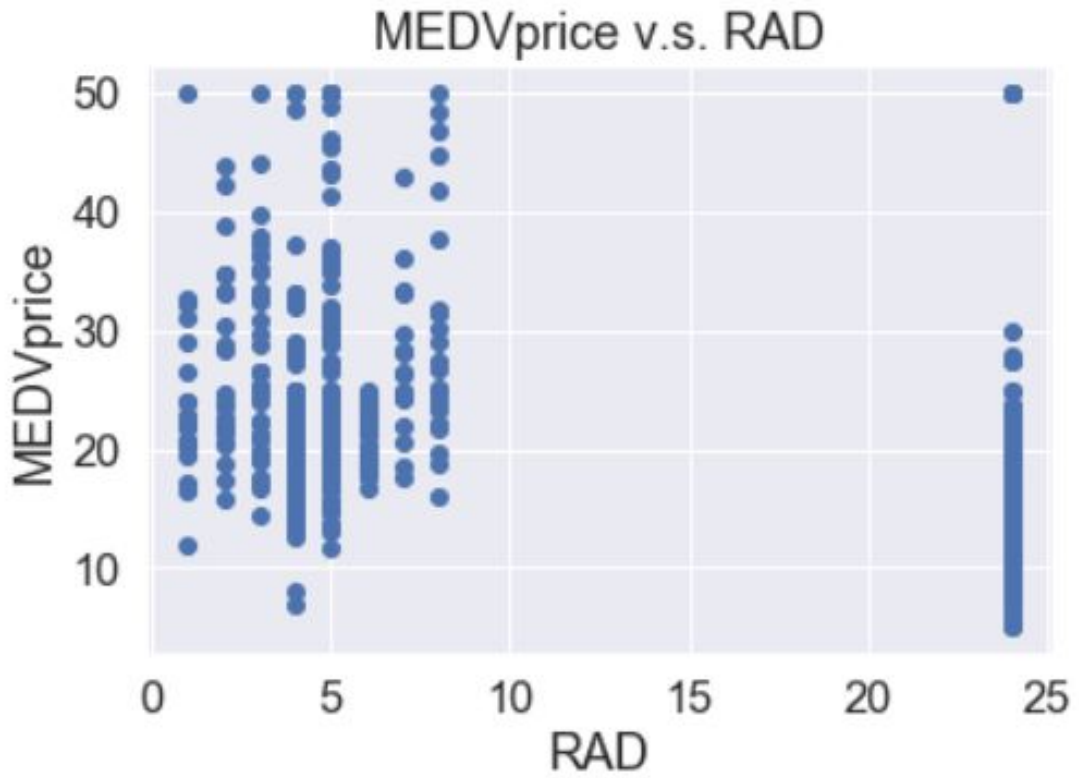
自用住宅的房價中位數與在1940年之前所建的自用房屋比例關係圖。

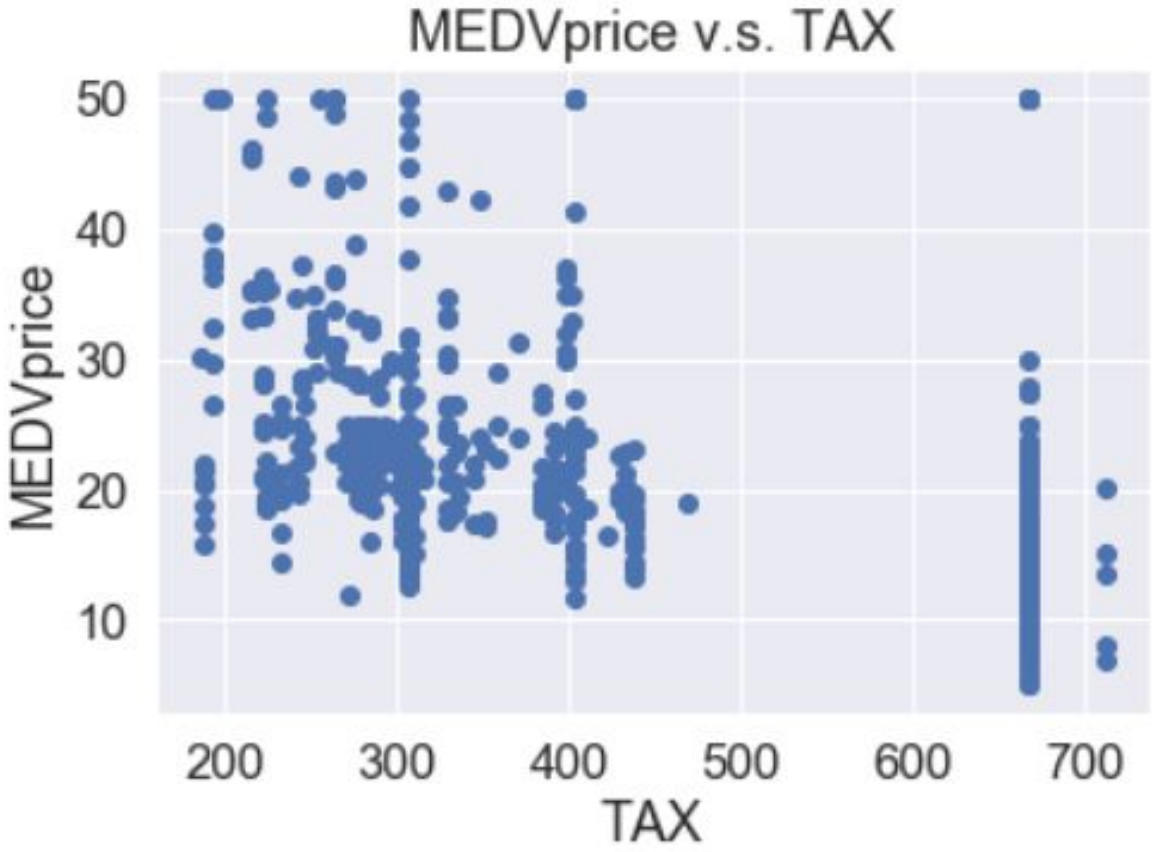




自用住宅的房價中位數與到波士頓五個中心區域的加權距離關係圖。

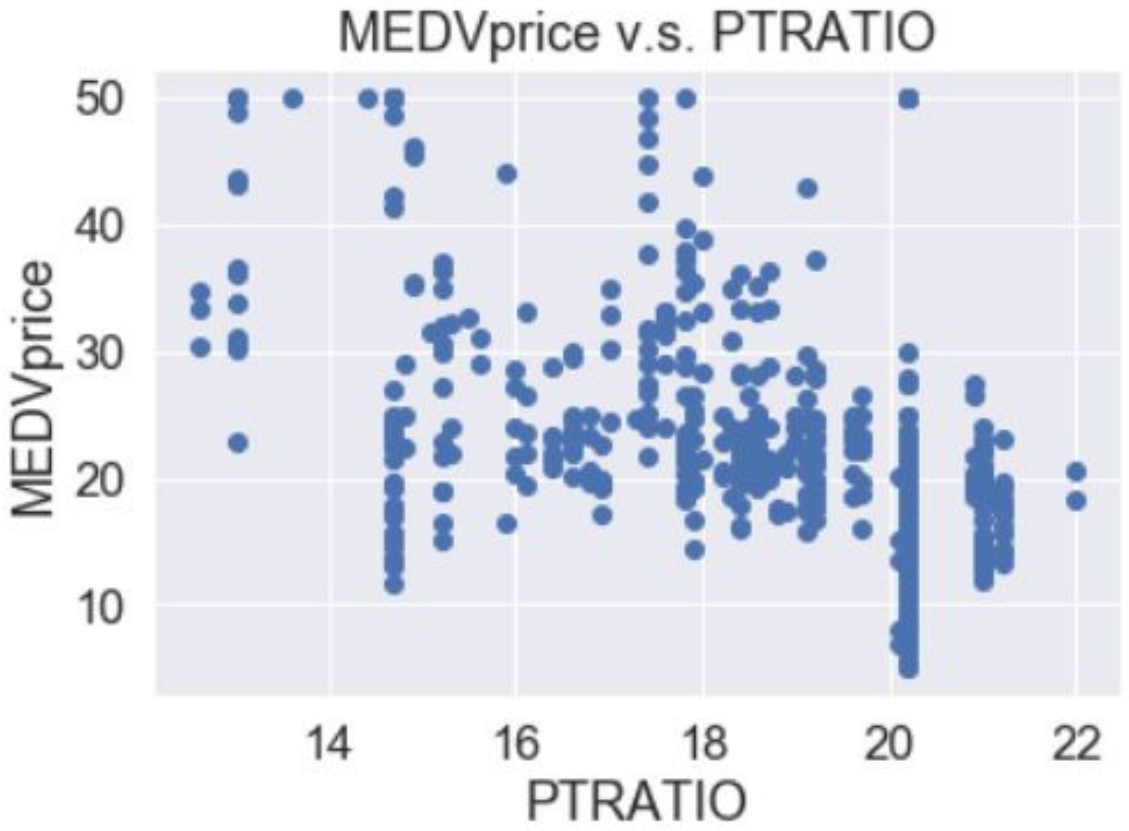
自用住宅的房價中位數與使用高速公路方便性的指數關係圖。





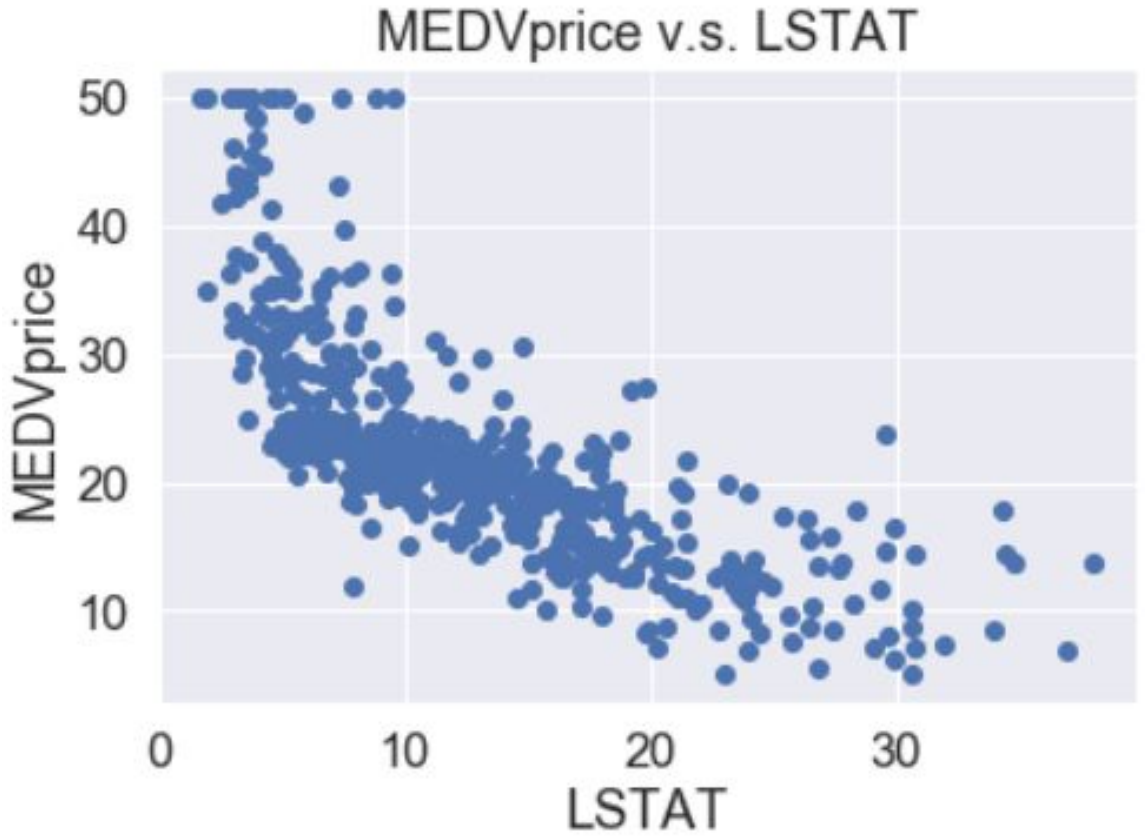
自用住宅的房價中位數與財產稅率(單位:10,000 美金)關係圖。

自用住宅的房價中位數與城鎮師生比例關係圖。



自用住宅的房價中位數與低社經地位的人口比例 (低收人口比例)關係圖。

自用住宅的房價中位數與1000(Bk-0.63)^2, 其中Bk指城鎮中黑人的比例關係圖。



特徵工程

```
from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()
X_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.fit_transform(X_test)
```

```
from sklearn.linear_model import LinearRegression
selector = SelectFromModel(estimator=LinearRegression()).fit(X_std, y_train)
```

```
selector.estimator_.coef_
```

```
array([-1.01350474,  1.34600685,  0.11557489,  0.57619956, -2.25207327,
        2.13772192,  0.15680574, -3.13809674,  2.62646459, -1.88649713,
       -2.13976217,  0.73920792, -3.90753383])
```

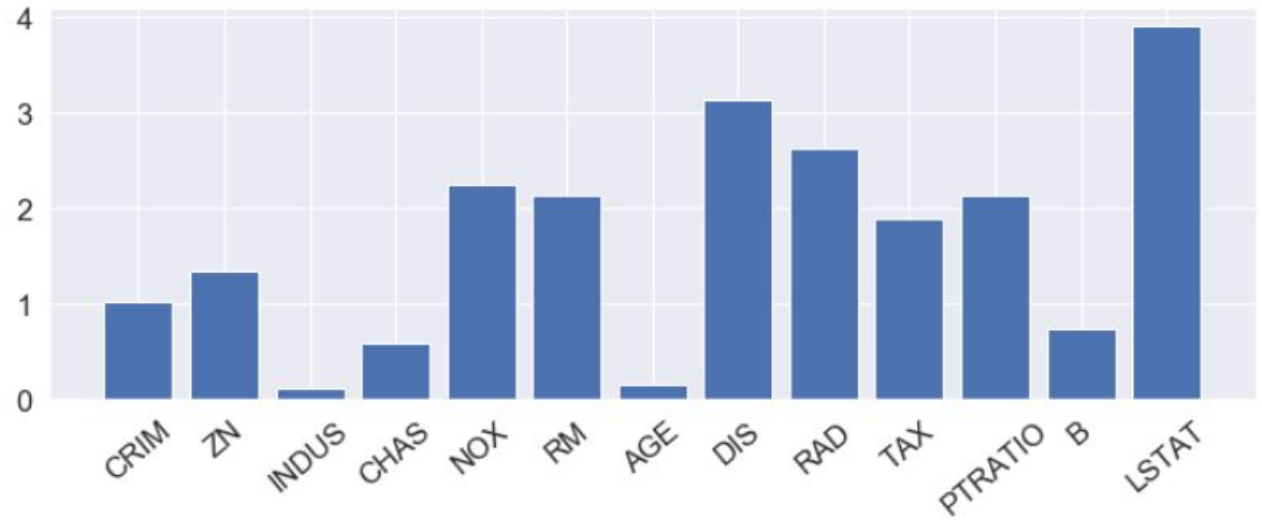
將特徵標準化, 避免範圍較大的特徵計算誤差平方距離時受到原本特徵尺度的影響, 以提高資料的精準度

特徵工程

```
#創建線性回歸
lm = LinearRegression()
lm.fit(X_std, y_train)
print(lm.intercept_ , lm.coef_)
# 畫圖 特徵重要性

feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT']
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.bar(feature_names , abs(lm.coef_))
plt.xticks(rotation=40)
plt.show()
```

```
22.47438423645322 [-1.01350474  1.34600685  0.11557489  0.57619956 -2.25207327  2.13772192
  0.15680574 -3.13809674  2.62646459 -1.88649713 -2.13976217  0.73920792
 -3.90753383]
```

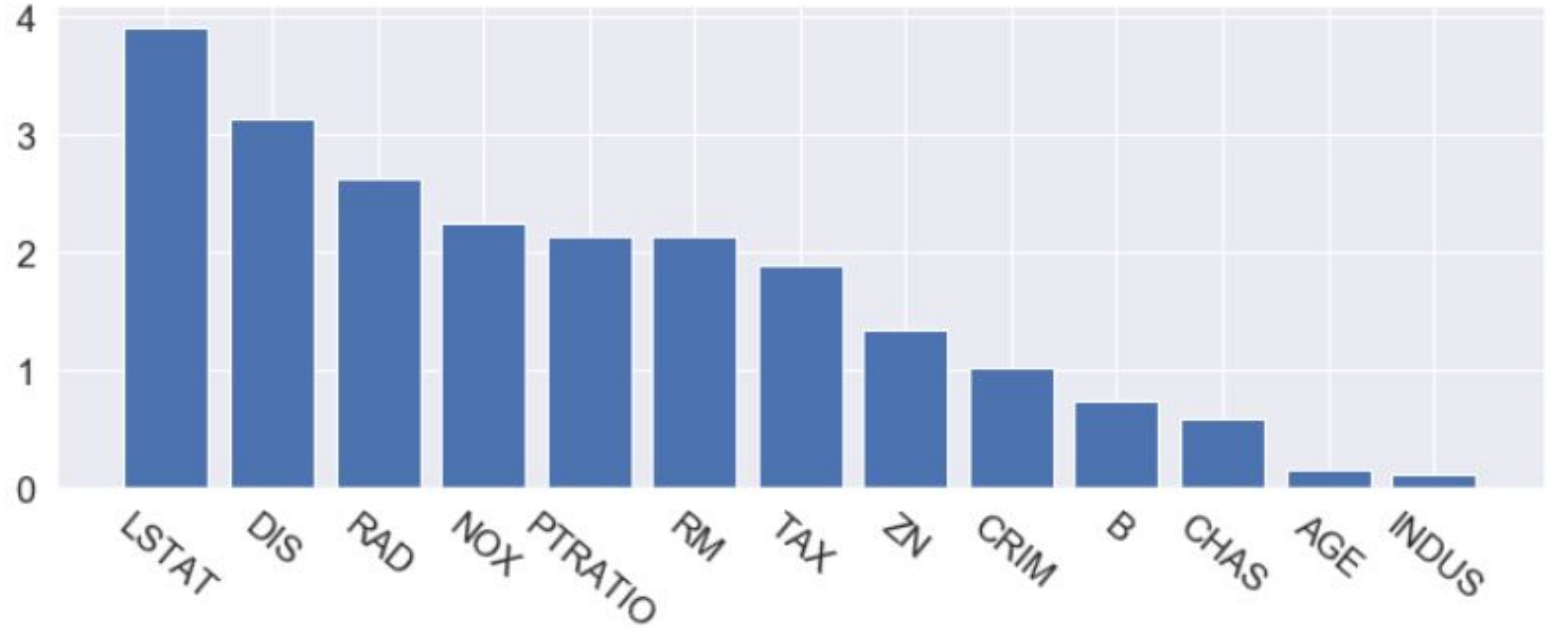


依照線性迴歸係數，判斷特徵重要性

將絕對值後的數值用長方圖表示，並由高到低排列

特徵工程

```
# 畫圖 從高到低 去畫出 特徵重要性
# 使用table 方式進行排序
plt.figure(figsize=(12, 4))
feature_sort = pd.DataFrame( abs(lm.coef_), columns=['score'])
feature_sort['feature'] = feature_names
feature_sort = feature_sort.sort_values(by='score', ascending=False)
# 使用table 方式進行排序
plt.bar(feature_sort['feature'] , feature_sort['score'])
plt.xticks(rotation=-40)
plt.show()
feature_sort
```



	score	feature
12	3.907534	LSTAT
7	3.138097	DIS
8	2.626465	RAD
4	2.252073	NOX
10	2.139762	PTRATIO
5	2.137722	RM
9	1.886497	TAX
1	1.346007	ZN
0	1.013505	CRIM
11	0.739208	B
3	0.576200	CHAS
6	0.156806	AGE
2	0.115575	INDUS

拆分數據

```
from sklearn.linear_model import LinearRegression
selector = SelectFromModel(estimator=LinearRegression()).fit(X_std, y_train)
```

```
selector.estimator_.coef_
```

```
array([-1.01350474,  1.34600685,  0.11557489,  0.57619956, -2.25207327,
        2.13772192,  0.15680574, -3.13809674,  2.62646459, -1.88649713,
       -2.13976217,  0.73920792, -3.90753383])
```

```
selector.threshold_
```

```
1.6950345655365462
```

用SelectFromModel選擇特徵
計算每個特徵的coef, 將平均訂為閾值進行篩選
將較不重要的特徵去除(小於閾值), 能提高模型的解釋力

拆分數據

```
selector.get_support(), boston.feature_names
```

```
(array([False, False, False, False,  True,  True, False,  True,  True,
         True,  True, False,  True]),
 array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'))
```

```
boston.feature_names[selector.get_support()]
```

```
array(['NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT'], dtype='<U7')
```

留下7個篩選結果為True的特徵

相較於RFE用疊代的方式進行篩選
，SelectFromModel更簡單直觀，但也較不可靠

拆分數據

```
selector.transform(X_train).shape
```

```
(406, 7)
```

```
selector.transform(X_train)
```

```
array([[  0.584 ,   5.837 ,   1.9976, ..., 666.    ,  20.2   ,  15.69  ],
       [  0.74  ,   6.485 ,   1.9784, ..., 666.    ,  20.2   ,  18.85  ],
       [  0.448 ,   6.169 ,   5.7209, ..., 233.    ,  17.9   ,   5.81  ],
       ...,
       [  0.693 ,   6.405 ,   1.6768, ..., 666.    ,  20.2   ,  19.37  ],
       [  0.507 ,   6.086 ,   3.6519, ..., 307.    ,  17.4   ,  10.88  ],
       [  0.499 ,   5.85  ,   3.9342, ..., 279.    ,  19.2   ,   8.77  ]])
```

重新定義資料(剩餘7個特徵)

建立模型

```
from sklearn.linear_model import LinearRegression
#創建線性回歸
lm = LinearRegression()
lm.fit(X_train, y_train) # use all features and train data to create model
print(lm.intercept_,lm.coef_)
r2_train= lm.score(X_train, y_train)
r2_test= lm.score(X_test, y_test)
print(r2_train, r2_test)
```

```
42.28626298344847 [ -0.11103262   0.05797726   0.01689505   2.13839616 -19.27485205
   3.11124463   0.00550103  -1.48228753   0.30162441  -0.01113592
  -0.98836392   0.00797486  -0.5429076 ]
0.7293635681762962 0.7652426102750567
```

對資料所有特徵做回歸模型

建立模型

```
p=7
lm.fit(selector.transform(X_train), y_train) # use train data to create model
r2_train= lm.score(selector.transform(X_train), y_train)
r2_test= lm.score(selector.transform(X_test), y_test)
r2_adj = 1 - ((1 - r2_test) * ((y_test.shape[0] - 1)/(y_test.shape[0] - p-1)) )
print(r2_train, r2_test, r2_adj)
```

```
0.703109162977364 0.7464748757092425 0.7271849206001633
```

```
lm.intercept_ , lm.coef_
```

```
(45.6425765715869,
 array([-18.0086473 ,   3.38104635,  -1.08639755,   0.19455093,
         -0.00887728,  -1.13665962,  -0.58922586]))
```

剩餘七個特徵的回歸模型

交互驗證

K-Fold

在K-fold CV中，我們在每次疊代後對模型進行評分，並計算所有評分的平均值。這樣就可以更好地表示該方法與只使用一個訓練和驗證集相比，模型的表現是怎樣的。

```
# k-fold CV (using selected 7 variables)
from sklearn.model_selection import cross_val_score

lm = LinearRegression()

scores = cross_val_score(lm, selector.transform(X_train), y_train, scoring='r2', cv=5)
scores
```

```
array([0.74774285, 0.66866317, 0.66579656, 0.63196233, 0.71394953])
```

```
# the other way of doing the same thing (more explicit)
from sklearn.model_selection import KFold
# create a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(lm, X_train, y_train, scoring='r2', cv=folds)
scores, scores.mean()
```

```
(array([0.685874  , 0.73705663, 0.52507792, 0.80110411, 0.71132109]),
 0.69208675147360 34)
```

K-Fold

得到最後平均分為 0.69, 以及它的95%信賴區間

```
# k-fold CV (using selected 7 variables)
from sklearn.model_selection import cross_val_score

lm = LinearRegression()

scores = cross_val_score(lm, selector.transform(X_train), y_train, scoring='r2', cv=5)
scores
```

```
array([0.74774285, 0.66866317, 0.66579656, 0.63196233, 0.71394953])
```

```
# the other way of doing the same thing (more explicit)
from sklearn.model_selection import KFold
# create a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(lm, X_train, y_train, scoring='r2', cv=folds)
scores, scores.mean()
```

```
(array([0.685874  , 0.73705663, 0.52507792, 0.80110411, 0.71132109]),
 0.6920867514736038)
```

```
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.69 (+/- 0.18)
```

XGBoost

比線性模型複雜, 但效能更高

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	17.046801	0.096223	17.159788	0.473955
1	12.331832	0.068324	12.558894	0.443266
2	9.004968	0.050496	9.387201	0.542747
3	6.677148	0.054352	7.334745	0.633012
4	5.051480	0.038922	5.992695	0.735889
5	3.925740	0.045741	5.185162	0.819386
6	3.166202	0.041303	4.688294	0.906837
7	2.638451	0.049779	4.405766	0.902513
8	2.284351	0.064361	4.229275	0.957267
9	2.045978	0.062345	4.137344	0.998519
10	1.859652	0.059843	4.056482	1.013251
11	1.733284	0.068948	4.008922	0.997367
12	1.634825	0.065845	3.984649	1.000362
13	1.531494	0.073246	3.939119	0.996170
14	1.456438	0.060329	3.922042	0.994029
15	1.397617	0.070542	3.913014	0.993313
16	1.337569	0.068718	3.896540	0.989177
17	1.296587	0.074616	3.891016	0.983825
18	1.255413	0.081516	3.881532	0.993321
19	1.220525	0.074765	3.870574	0.980936
20	1.178313	0.083890	3.880990	0.968491
21	1.145032	0.086378	3.872632	0.971874
22	1.111806	0.088689	3.872202	0.974266
23	1.080893	0.094263	3.879527	0.973883
24	1.051113	0.088156	3.871909	0.960404
25	1.019080	0.086842	3.869594	0.957166
26	0.993135	0.090643	3.879844	0.951537
27	0.962149	0.086815	3.878045	0.946217
28	0.929707	0.078199	3.874896	0.942713
29	0.895529	0.084123	3.873548	0.940599
30	0.873665	0.084904	3.872613	0.943250
31	0.846827	0.075256	3.870729	0.944514
32	0.830217	0.073670	3.867038	0.943611
33	0.799343	0.065691	3.862166	0.931849
34	0.785035	0.067773	3.862822	0.930553
35	0.763113	0.061503	3.870465	0.930171
36	0.741708	0.059159	3.869296	0.927247
37	0.723735	0.058800	3.865906	0.922912
38	0.703929	0.053401	3.866582	0.915793
39	0.681665	0.048338	3.870768	0.914173
40	0.662913	0.049121	3.870494	0.910114
41	0.646489	0.048591	3.865731	0.911476
42	0.631817	0.049539	3.869065	0.912785
43	0.619386	0.051972	3.870595	0.910304
44	0.607160	0.054227	3.869764	0.911252
45	0.586053	0.051267	3.874236	0.908324
46	0.569952	0.048865	3.870530	0.907625
47	0.555036	0.051424	3.870704	0.910043
48	0.541221	0.052907	3.871662	0.912995
49	0.533422	0.051747	3.870832	0.912225

Final MSE test: 49 14.983343
Name: test-rmse-mean, dtype: float64

用test來看出來, 到第13層的時候mse的平方根數值就基本固定了

評估修正模型

```
# final muti-linear regression model
new_X_train = X_train[['NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT']]
new_X_test = X_test[['NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT']]
lm = LinearRegression()

# fit the model
lm.fit(new_X_train, y_train) #建立 model
print('intercept =', lm.intercept_ )
print('Coef =', lm.coef_)

# Find the R^2
r2_train = lm.score(new_X_train, y_train)
r2_test = lm.score(new_X_test, y_test)
print('The R-square for train is: ', r2_train )
print('The R-square for test is: ', r2_test )

r2_adj = 1 - ((1 - r2_test) * ((y_test.shape[0] - 1)/(y_test.shape[0] - 7-1)) )
print('The Adjusted R^2 for test is: ' + str(r2_adj))

#預測
Y_hat = lm.predict(new_X_test)
from sklearn.metrics import mean_squared_error
print('The mean square error of price and predicted value using multifit is: ', \
      mean_squared_error(y_test, Y_hat))
```

- 截距:45.6425
- train 的 R-square:0.7031
- test 的 R-square:0.7464
- 調整後test的 R-square:0.7271
- 使用multifit的價格和預測值的均方誤差(MSE)為:25.02792

對test data進行預測

```
import sklearn.metrics as sklm
import math
def print_metrics(y_true, y_predicted, n_parameters):
    ## First compute R^2 and the adjusted R^2
    r2 = sklm.r2_score(y_true, y_predicted)
    r2_adj = 1 - ((1 - r2) * ((y_true.shape[0] - 1)/(y_true.shape[0] - n_parameters -1)) )

    ## Print the usual metrics and the R^2 values
    print('Mean Square Error      = ' + str(sklm.mean_squared_error(y_true, y_predicted)))
    print('Root Mean Square Error = ' + str(math.sqrt(sklm.mean_squared_error(y_true, y_predicted))))
    print('Mean Absolute Error    = ' + str(sklm.mean_absolute_error(y_true, y_predicted)))
    print('Median Absolute Error  = ' + str(sklm.median_absolute_error(y_true, y_predicted)))
    print('R^2                    = ' + str(r2))
    print('Adjusted R^2           = ' + str(r2_adj))

# prediction for valid data
y_hat = xgb_mod.predict(X_valid3)
print_metrics(y_valid3, y_hat, 12)
```

```
Mean Square Error      = 1.4108700132548766
Root Mean Square Error = 1.1878004938771816
Mean Absolute Error    = 0.8647940682201849
Median Absolute Error  = 0.6576441764831547
R^2                    = 0.9794469184338032
Adjusted R^2           = 0.9758724694657689
```

使用train拆分出來的20%資料來算出mse...等

對test data進行預測

```
print_metrics(AW_answer.MEDVprice, y_hat_test, 12)
```

```
Mean Square Error      = 1.3444889189018034
Root Mean Square Error = 1.1595209868311154
Mean Absolute Error    = 0.8063692665100097
Median Absolute Error  = 0.5593993186950685
R^2                    = 0.9863807434696236
Adjusted R^2           = 0.9845022253275026
```

評估test資料的MSE,R-square...

用test data進行預測

	Unnamed: 0	MEDVprice	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	y_hat_test
0	0	28.2	0.04932	33.0	2.18	0.0	0.472	6.849	70.3	3.1827	7.0	222.0	18.4	396.90	7.53	30.240660
1	1	23.9	0.02543	55.0	3.78	0.0	0.484	6.696	56.4	5.7321	5.0	370.0	17.6	396.90	7.18	24.838207
2	2	16.6	0.22927	0.0	6.91	0.0	0.448	6.030	85.5	5.6894	3.0	233.0	17.9	392.74	18.80	18.287056
3	3	22.0	0.05789	12.5	6.07	0.0	0.409	5.878	21.4	6.4980	4.0	345.0	18.9	396.21	8.10	21.392822
4	4	20.8	3.67822	0.0	18.10	0.0	0.770	5.362	96.2	2.1036	24.0	666.0	20.2	380.79	10.19	19.048323
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	95	18.9	0.11747	12.5	7.87	0.0	0.524	6.009	82.9	6.2267	5.0	311.0	15.2	396.90	13.27	18.895651
96	96	22.4	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	21.635185
97	97	22.9	0.04203	28.0	15.04	0.0	0.464	6.442	53.6	3.6659	4.0	270.0	18.2	395.01	8.16	23.218956
98	98	44.8	0.31533	0.0	6.20	0.0	0.504	8.266	78.3	2.8944	8.0	307.0	17.4	385.05	4.14	45.837471
99	99	21.7	0.10793	0.0	8.56	0.0	0.520	6.195	54.4	2.7778	5.0	384.0	20.9	393.49	13.00	21.297445

100 rows × 16 columns

標準化殘差散佈圖

預測值與實際值的殘差大多集中在2.5~-2.5之間

```
# Instantiate the XGBRegressor: xg_reg
xg_reg = xgb.XGBRegressor(objective = "reg:squarederror"
                          ,n_estimators=10
                          , seed = 123
                          , booster="gbtree"
#                           , max_depth =6
                         )

# Fit the regressor to the training set
xg_reg.fit(X_train, y_train)

# Predict the labels of the test set: preds
y_train_pred = xg_reg.predict(X_train)
y_test_pred = xg_reg.predict(X_test)

# Compute the mse & R2
print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
print(xg_reg)
```

MSE train: 2.808, test: 9.084
R^2 train: 0.965, test: 0.908

心得、未來方向