Ch 2

January 25, 2019

Table of Contents

In [10]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1 Conceptual

1.1 Q1.

a) Flexible. Since the sample size is large, it's possible that there are some noise data. A more flexible method can avoid high influence caused by these noise data, as well as overfitting.

b) Inflexible. Since the sample size is relatively small, a flexible method will be influenced by noise, and given another random sampling of data, the fit will be significantly different. Therefore, an inflexible method will be less likely to overfit.

c) Flexible. Given that the relationship is non-linear, an inflexible method cannot capture the non-linearity of the data, so that an overfit to this certain data may occur.

d) Inflexible. A high variance of error term means that the discrepancy between the values model captured and the real response values is relatively large, so we don't want that this noise captured by the model, which will cause overfit.

1.2 Q2.

a) Inference. Because we're interested in understanding the factors of response variable.

b) Classification. Because the response variable is categorical, success or failure.

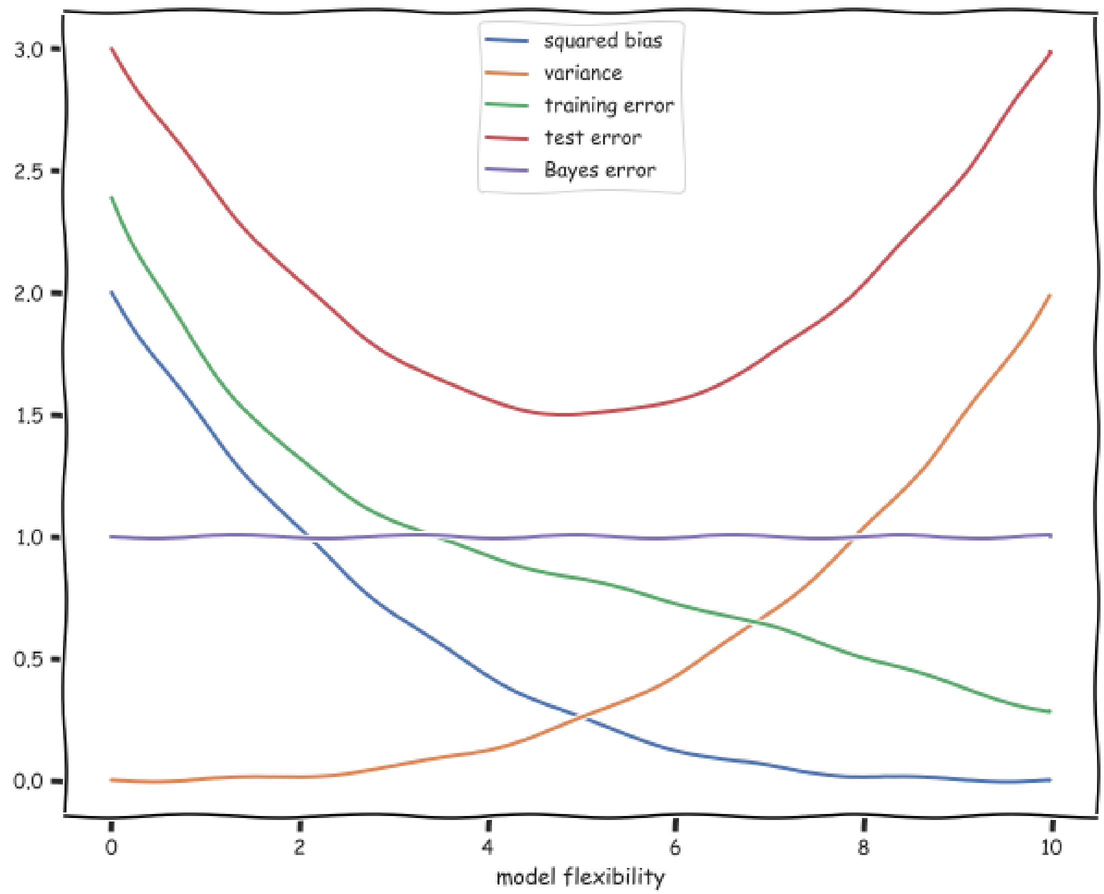
c) Prediction. Because we want to use the model to make prediction.

1.3 Q3

1.3.1 (a) Sketch of bias, variance, training, test and Bayes errors

In [136]:
```
x = np.arange(0.0, 10.0, 0.02)

def squared_bias(x):
    return .002*(-x+10)**3
def variance(x):
    return .002*x**3
def training_error(x):
    return 2.38936 - 0.825077*x + 0.176655*x**2 - 0.0182319*x**3 + 0.00067091*x**4
def test_error(x):
    return 3 - 0.6*x + .06*x**2
def bayes_error(x):
    return x + 1 - x

plt.xkcd()
#frame = plt.gca()
#frame.axes.xaxis.set_ticklabels([])
plt.figure(figsize=(10, 8))
plt.plot(x,squared_bias(x), label='squared bias')
plt.plot(x, variance(x), label='variance')
plt.plot(x, training_error(x), label='training error')
plt.plot(x, test_error(x), label='test error')
```

```
plt.plot(x, bayes_error(x), label='Bayes error')
plt.legend(loc='upper center')
plt.xlabel('model flexibility')
plt.show()
```



1.4 Q4.

a) gender-detection; mnist; package delivery point assignment

b) house price, stock price, revenue level

c) customer clustering, recommending system, social network

1.5 Q5.

Flexible models' advantages: less bias, given enough data we will have better results.
Flexible models' disadvantages: May be overfitting, hard and long to train, less interpretable.

1.6 Q6.

Parametric is the algorithms that make assumptions about the form and our goal is to get the coefficients for the function by training data; Non-parametric is the algorithms that don't have strong assumptions on the form of functions.

Parametric algorithm doesn't need a lot of observations, while non-parametric needs a lot of observations.

Non-parametric doesn't put stricts on the form so it can handle more kinds of data, while parametric can only be used in the kinds which are consistent to the stricts.

1.7 Q7.

In [137]:
```
import numpy as np
import pandas as pd

d = {'X1': pd.Series([0,2,0,0,-1,1]),
     'X2': pd.Series([3,0,1,1,0,1]),
     'X3': pd.Series([0,0,3,2,1,1]),
     'Y': pd.Series(['Red','Red','Red','Green','Green','Red'])}
df = pd.DataFrame(d)
df.index = np.arange(1, len(df) + 1)
df
```

Out[137]:

	X1	X2	X3	Y
1	0	3	0	Red
2	2	0	0	Red
3	0	1	3	Red
4	0	1	2	Green
5	-1	0	1	Green
6	1	1	1	Red

1.7.1 (a) Euclidian distance

In [138]:
```
from math import sqrt
df['distance']=np.sqrt(df['X1']**2+df['X2']**2+df['X3']**2)
df
```

Out[138]:

	X1	X2	X3	Y	distance
1	0	3	0	Red	3.000000
2	2	0	0	Red	2.000000
3	0	1	3	Red	3.162278
4	0	1	2	Green	2.236068
5	-1	0	1	Green	1.414214
6	1	1	1	Red	1.732051

In [139]:
```
# k =1
df.sort_values(['distance'])
```

Out[139]:

	X1	X2	X3	Y	distance
5	-1	0	1	Green	1.414214

```
6    1    1    1      Red   1.732051
2    2    0    0      Red   2.000000
4    0    1    2    Green   2.236068
1    0    3    0      Red   3.000000
3    0    1    3      Red   3.162278
```

when K=3, our prediction is Red, because that's the mode of the 3 nearest neigbours: Green, Red and Red (points 5, 6 and 2, respectively).

2 Applied

2.1 Q8 `college` dataset

In [41]:
```
college = pd.read_csv('College.csv',index_col =0)
college.head()
```

Out[41]:
```
                              Private  Apps  Accept  Enroll  Top10perc  \
Abilene Christian University      Yes  1660    1232     721         23
Adelphi University                Yes  2186    1924     512         16
Adrian College                    Yes  1428    1097     336         22
Agnes Scott College               Yes   417     349     137         60
Alaska Pacific University         Yes   193     146      55         16

                              Top25perc  F.Undergrad  P.Undergrad  Outstate  \
Abilene Christian University         52         2885          537      7440
Adelphi University                   29         2683         1227     12280
Adrian College                       50         1036           99     11250
Agnes Scott College                  89          510           63     12960
Alaska Pacific University            44          249          869      7560

                              Room.Board  Books  Personal  PhD  Terminal  \
Abilene Christian University        3300    450      2200   70        78
Adelphi University                  6450    750      1500   29        30
Adrian College                      3750    400      1165   53        66
Agnes Scott College                 5450    450       875   92        97
Alaska Pacific University           4120    800      1500   76        72

                              S.F.Ratio  perc.alumni  Expend  Grad.Rate
Abilene Christian University       18.1           12    7041         60
Adelphi University                 12.2           16   10527         56
Adrian College                     12.9           30    8735         54
Agnes Scott College                 7.7           37   19016         59
Alaska Pacific University          11.9            2   10922         15
```

2.1.1 summary

In [9]:
```
college.describe(include= 'all')
```

Out[9]:

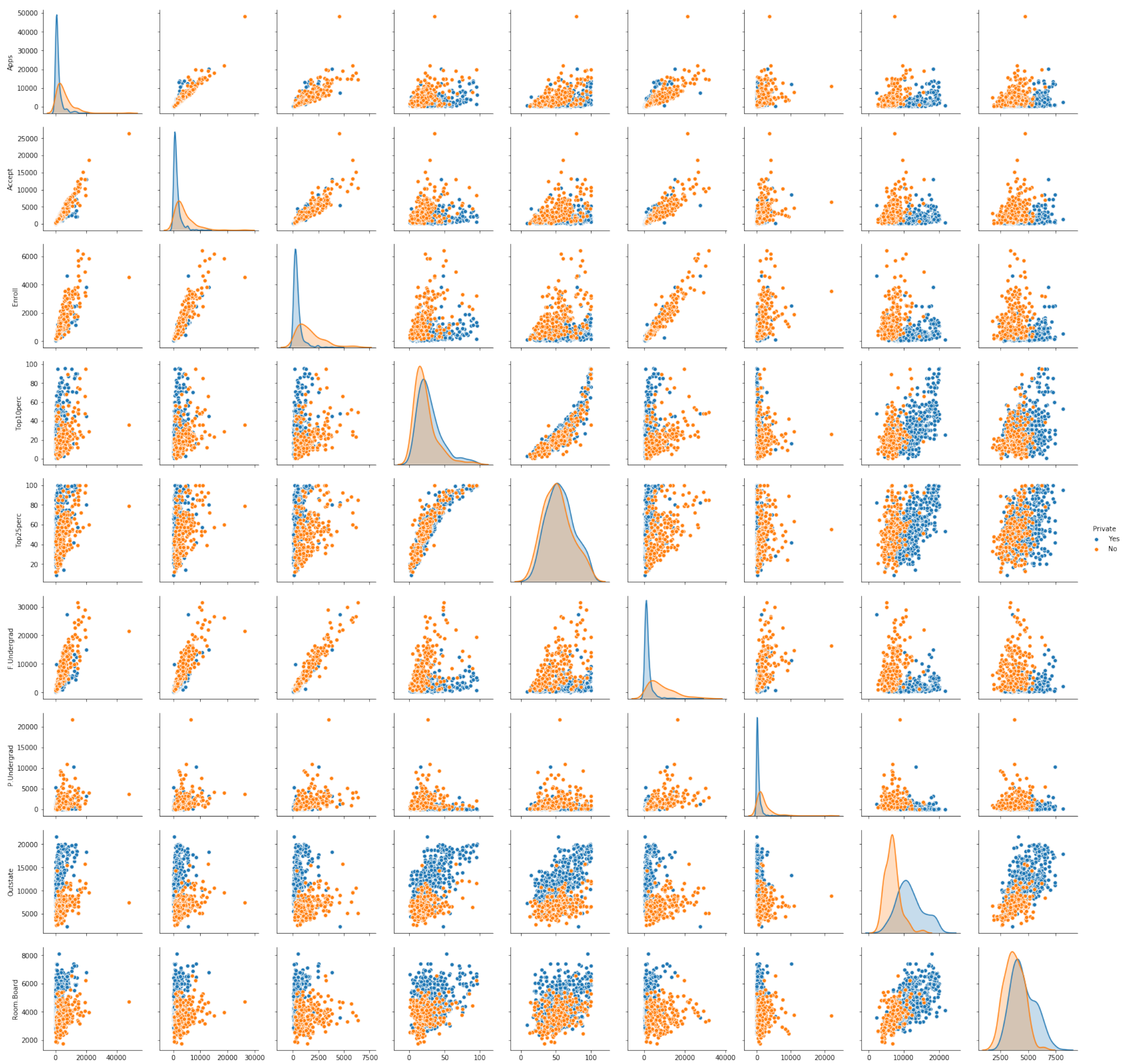
```
       Private          Apps        Accept       Enroll   Top10perc  \
count      777    777.000000    777.000000   777.000000  777.000000
unique       2           NaN           NaN          NaN         NaN
top        Yes           NaN           NaN          NaN         NaN
freq       565           NaN           NaN          NaN         NaN
mean       NaN   3001.638353   2018.804376   779.972973   27.558559
std        NaN   3870.201484   2451.113971   929.176190   17.640364
min        NaN     81.000000     72.000000    35.000000    1.000000
25%        NaN    776.000000    604.000000   242.000000   15.000000
50%        NaN   1558.000000   1110.000000   434.000000   23.000000
75%        NaN   3624.000000   2424.000000   902.000000   35.000000
max        NaN  48094.000000  26330.000000  6392.000000   96.000000

         Top25perc   F.Undergrad   P.Undergrad      Outstate   Room.Board  \
count   777.000000    777.000000    777.000000    777.000000   777.000000
unique         NaN           NaN           NaN           NaN          NaN
top            NaN           NaN           NaN           NaN          NaN
freq           NaN           NaN           NaN           NaN          NaN
mean     55.796654   3699.907336    855.298584  10440.669241  4357.526384
std      19.804778   4850.420531   1522.431887   4023.016484  1096.696416
min       9.000000    139.000000      1.000000   2340.000000  1780.000000
25%      41.000000    992.000000     95.000000   7320.000000  3597.000000
50%      54.000000   1707.000000    353.000000   9990.000000  4200.000000
75%      69.000000   4005.000000    967.000000  12925.000000  5050.000000
max     100.000000  31643.000000  21836.000000  21700.000000  8124.000000

              Books     Personal         PhD    Terminal   S.F.Ratio  \
count    777.000000   777.000000  777.000000  777.000000  777.000000
unique          NaN          NaN         NaN         NaN         NaN
top             NaN          NaN         NaN         NaN         NaN
freq            NaN          NaN         NaN         NaN         NaN
mean     549.380952  1340.642214   72.660232   79.702703   14.089704
std      165.105360   677.071454   16.328155   14.722359    3.958349
min       96.000000   250.000000    8.000000   24.000000    2.500000
25%      470.000000   850.000000   62.000000   71.000000   11.500000
50%      500.000000  1200.000000   75.000000   82.000000   13.600000
75%      600.000000  1700.000000   85.000000   92.000000   16.500000
max     2340.000000  6800.000000  103.000000  100.000000   39.800000

        perc.alumni        Expend   Grad.Rate
count    777.000000    777.000000   777.00000
unique          NaN           NaN         NaN
top             NaN           NaN         NaN
freq            NaN           NaN         NaN
mean      22.743887   9660.171171    65.46332
std       12.391801   5221.768440    17.17771
min        0.000000   3186.000000    10.00000
25%       13.000000   6751.000000    53.00000
```

```
50%      21.000000   8377.000000   65.00000
75%      31.000000  10830.000000   78.00000
max      64.000000  56233.000000  118.00000
```

2.1.2 pair plot

```
In [21]: sns.pairplot(data=college.iloc[:,0:10],hue = 'Private')
```

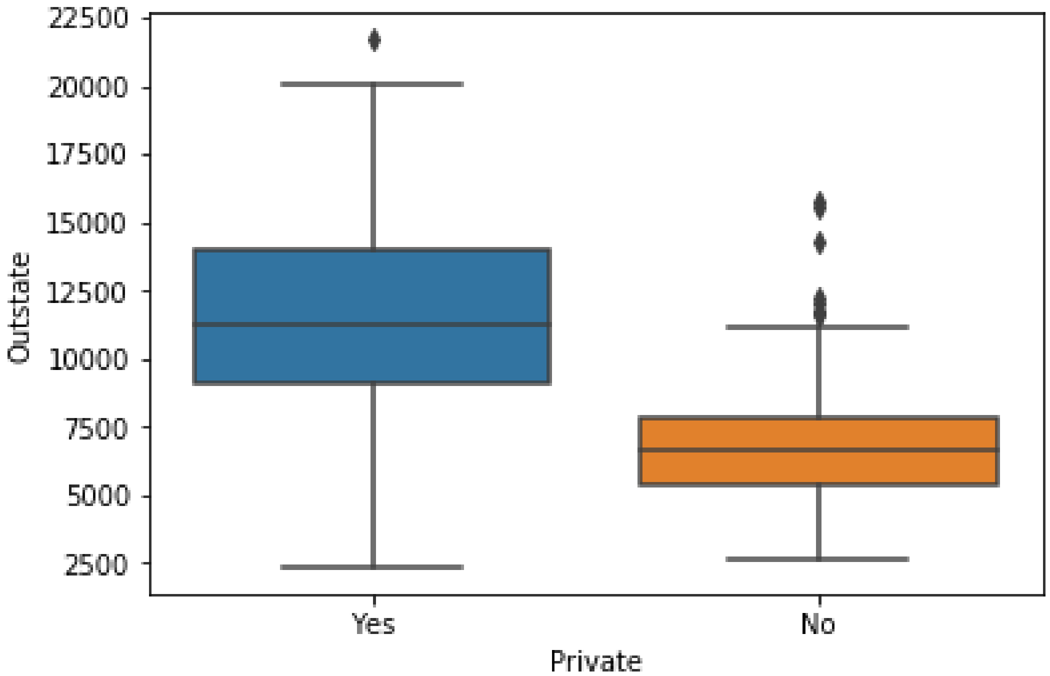
```
Out[21]: <seaborn.axisgrid.PairGrid at 0x210b4790>
```



2.1.3 boxplot - Outstate versus Private.

```
In [22]: sns.boxplot(x='Private', y = 'Outstate', data = college)
```

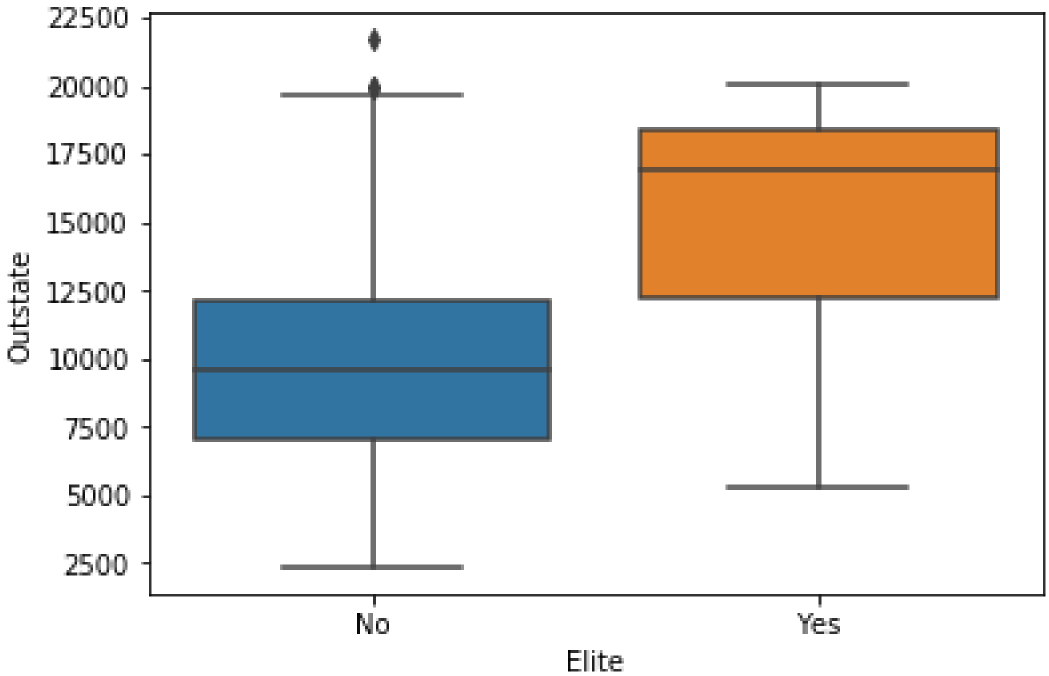
```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x20b4f670>
```



2.1.4 Elite variable

In [37]:
```
# create new variable 'Elite'
college['Elite'] = 'No'
college.loc[ college['Top10perc'] > 50 , 'Elite'] = 'Yes'
#college['Elite'].fillna('No', inplace = True)
#college.head(20)
sns.boxplot(x= 'Elite', y =  'Outstate',data = college)
```

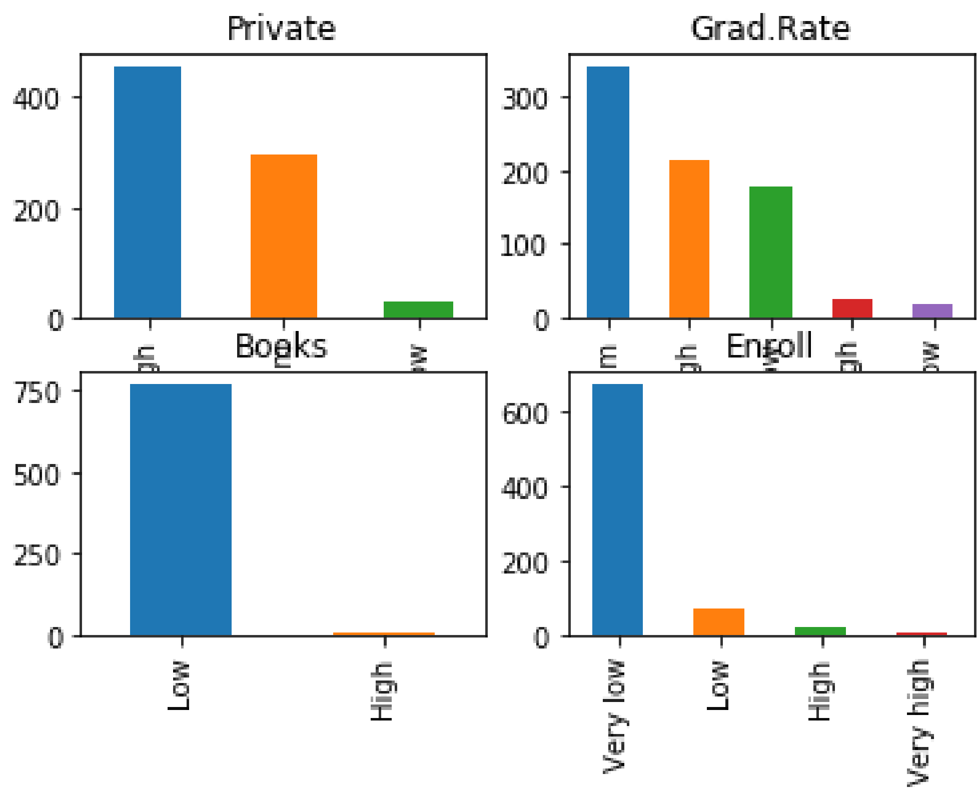
Out[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2930ed50>
```



2.1.5 Histogram

```
In [42]: #set bins
         college['Phd'] = pd.cut(college['PhD'], 3, labels=['low', 'medium','high'])
         college['Grad.Rate'] = pd.cut(college['Grad.Rate'], 5, labels=['Very low', 'Low', 'Me
         college['Books'] = pd.cut(college['Books'], 2, labels=['Low', 'High'])
         college['Enroll'] = pd.cut(college['Enroll'], 4, labels=['Very low', 'Low', 'High', '
```

```
In [44]: # plot histogram
         plt.subplot(221)
         college['Phd'].value_counts().plot(kind='bar', title = 'Private');
         plt.subplot(222)
         college['Grad.Rate'].value_counts().plot(kind='bar', title = 'Grad.Rate');
         plt.subplot(223)
         college['Books'].value_counts().plot(kind='bar', title = 'Books');
         plt.subplot(224)
         college['Enroll'].value_counts().plot(kind='bar', title = 'Enroll');

         fig.subplots_adjust(hspace=3) # To add space between subplots
```



2.2 Q9 Auto dataset

In [72]:
```
auto = pd.read_csv('Auto.csv')
auto.head()
```

Out[72]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year
0	18.0	8	307.0	130	3504	12.0	70
1	15.0	8	350.0	165	3693	11.5	70
2	18.0	8	318.0	150	3436	11.0	70
3	16.0	8	304.0	150	3433	12.0	70
4	17.0	8	302.0	140	3449	10.5	70

	origin	name
0	1	chevrolet chevelle malibu
1	1	buick skylark 320
2	1	plymouth satellite
3	1	amc rebel sst
4	1	ford torino

2.2.1 Data description

In [75]:
```
auto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 9 columns):
mpg             397 non-null float64
cylinders       397 non-null int64
displacement    397 non-null float64
horsepower      397 non-null object
weight          397 non-null int64
acceleration    397 non-null float64
year            397 non-null int64
origin          397 non-null int64
name            397 non-null object
dtypes: float64(3), int64(4), object(2)
memory usage: 24.9+ KB
```

In [76]:
```
print('quanlitative predictor: ',list(auto.select_dtypes(exclude=np.number).columns))
print('quantitative predictor: ',list(auto.select_dtypes(include=np.number).columns))
```

```
quanlitative predictor:  ['horsepower', 'name']
quantitative predictor:  ['mpg', 'cylinders', 'displacement', 'weight', 'acceleration', 'year'
```

In [77]:
```
des = auto.describe(include= np.number)
des
```

Out[77]:

	mpg	cylinders	displacement	weight	acceleration
count	397.000000	397.000000	397.000000	397.000000	397.000000
mean	23.515869	5.458438	193.532746	2970.261965	15.555668
std	7.825804	1.701577	104.379583	847.904119	2.749995
min	9.000000	3.000000	68.000000	1613.000000	8.000000
25%	17.500000	4.000000	104.000000	2223.000000	13.800000
50%	23.000000	4.000000	146.000000	2800.000000	15.500000
75%	29.000000	8.000000	262.000000	3609.000000	17.100000
max	46.600000	8.000000	455.000000	5140.000000	24.800000

	year	origin
count	397.000000	397.000000
mean	75.994962	1.574307
std	3.690005	0.802549
min	70.000000	1.000000
25%	73.000000	1.000000
50%	76.000000	1.000000
75%	79.000000	2.000000
max	82.000000	3.000000

In [78]:
```
des.loc['range'] = des.loc['max'] - des.loc['min']
des.loc['range']
```

```
Out[78]: mpg              37.6
         cylinders         5.0
         displacement    387.0
         weight         3527.0
         acceleration     16.8
         year             12.0
         origin            2.0
         Name: range, dtype: float64
```

```
In [79]: des.loc[['mean', 'std', 'range']]
```

```
Out[79]:          mpg  cylinders  displacement       weight  acceleration  \
         mean   23.515869  5.458438    193.532746  2970.261965     15.555668
         std     7.825804  1.701577    104.379583   847.904119      2.749995
         range  37.600000  5.000000    387.000000  3527.000000     16.800000

                     year    origin
         mean   75.994962  1.574307
         std     3.690005  0.802549
         range  12.000000  2.000000
```

```
In [82]: # remove the 10th through 85th observations
         auto_remv10 = auto.drop(auto.index[9:85])
         auto_remv10.head(20)
```

```
Out[82]:     mpg  cylinders  displacement horsepower  weight  acceleration  year  \
         0   18.0          8         307.0        130    3504          12.0    70
         1   15.0          8         350.0        165    3693          11.5    70
         2   18.0          8         318.0        150    3436          11.0    70
         3   16.0          8         304.0        150    3433          12.0    70
         4   17.0          8         302.0        140    3449          10.5    70
         5   15.0          8         429.0        198    4341          10.0    70
         6   14.0          8         454.0        220    4354           9.0    70
         7   14.0          8         440.0        215    4312           8.5    70
         8   14.0          8         455.0        225    4425          10.0    70
         85  13.0          8         350.0        175    4100          13.0    73
         86  14.0          8         304.0        150    3672          11.5    73
         87  13.0          8         350.0        145    3988          13.0    73
         88  14.0          8         302.0        137    4042          14.5    73
         89  15.0          8         318.0        150    3777          12.5    73
         90  12.0          8         429.0        198    4952          11.5    73
         91  13.0          8         400.0        150    4464          12.0    73
         92  13.0          8         351.0        158    4363          13.0    73
         93  14.0          8         318.0        150    4237          14.5    73
         94  13.0          8         440.0        215    4735          11.0    73
         95  12.0          8         455.0        225    4951          11.0    73

             origin                       name
         0        1     chevrolet chevelle malibu
```

```
1         1              buick skylark 320
2         1             plymouth satellite
3         1                  amc rebel sst
4         1                    ford torino
5         1               ford galaxie 500
6         1               chevrolet impala
7         1              plymouth fury iii
8         1               pontiac catalina
85        1              buick century 350
86        1                    amc matador
87        1               chevrolet malibu
88        1               ford gran torino
89        1           dodge coronet custom
90        1       mercury marquis brougham
91        1      chevrolet caprice classic
92        1                       ford ltd
93        1       plymouth fury gran sedan
94        1   chrysler new yorker brougham
95        1       buick electra 225 custom
```

In [83]:
```
des2 = auto_remv10.describe(include=np.number)
des2.loc['range'] = des2.loc['max'] - des2.loc['min']
des2.loc[['mean', 'std', 'range']]
```

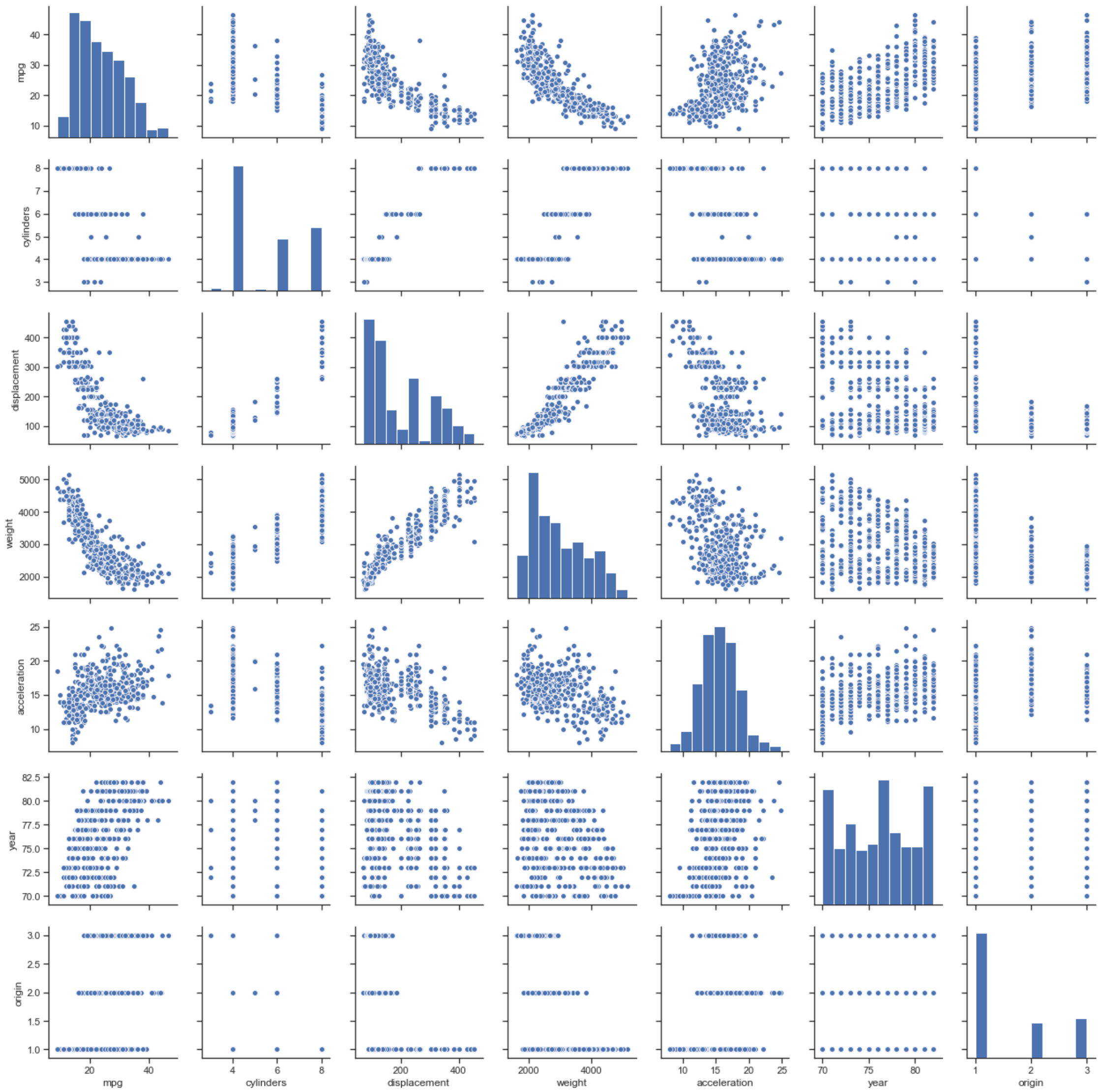
Out[83]:
```
             mpg  cylinders  displacement       weight  acceleration  \
mean   24.438629   5.370717    187.049844  2933.962617     15.723053
std     7.908184   1.653486     99.635385   810.642938      2.680514
range  35.600000   5.000000    387.000000  3348.000000     16.300000

            year    origin
mean   77.152648  1.598131
std     3.111230  0.816163
range  12.000000  2.000000
```

2.2.2 Visualization

scatter plot

In [85]:
```
sns.set(style="ticks")
sns.pairplot(data=auto)
```

Out[85]:
```
<seaborn.axisgrid.PairGrid at 0x2ff73c10>
```



- The histogram for `acceleration` resembles a normal distribution.
- `displacement` and `weight` have a strong linear relationship.
- `mpg` has a non-linear relationship with `weight`, `horsepower` and `displacement`.

2.3 Q10 `Boston` housing dataset

2.3.1 data import and description

In [124]:
```
from sklearn.datasets import load_boston

df = load_boston()
boston = pd.DataFrame(df.data ,columns= df.feature_names)
boston['target'] = df.target
boston.head()
```

```
Out[124]:       CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
          0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
          1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
          2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
          3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
          4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

             PTRATIO       B  LSTAT  target
          0     15.3  396.90   4.98    24.0
          1     17.8  396.90   9.14    21.6
          2     17.8  392.83   4.03    34.7
          3     18.7  394.63   2.94    33.4
          4     18.7  396.90   5.33    36.2
```

```
In [126]: print(df['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 us

    :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/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon Univers

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 address regress
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources 
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on t
```

```
In [100]: np.shape(boston)
```

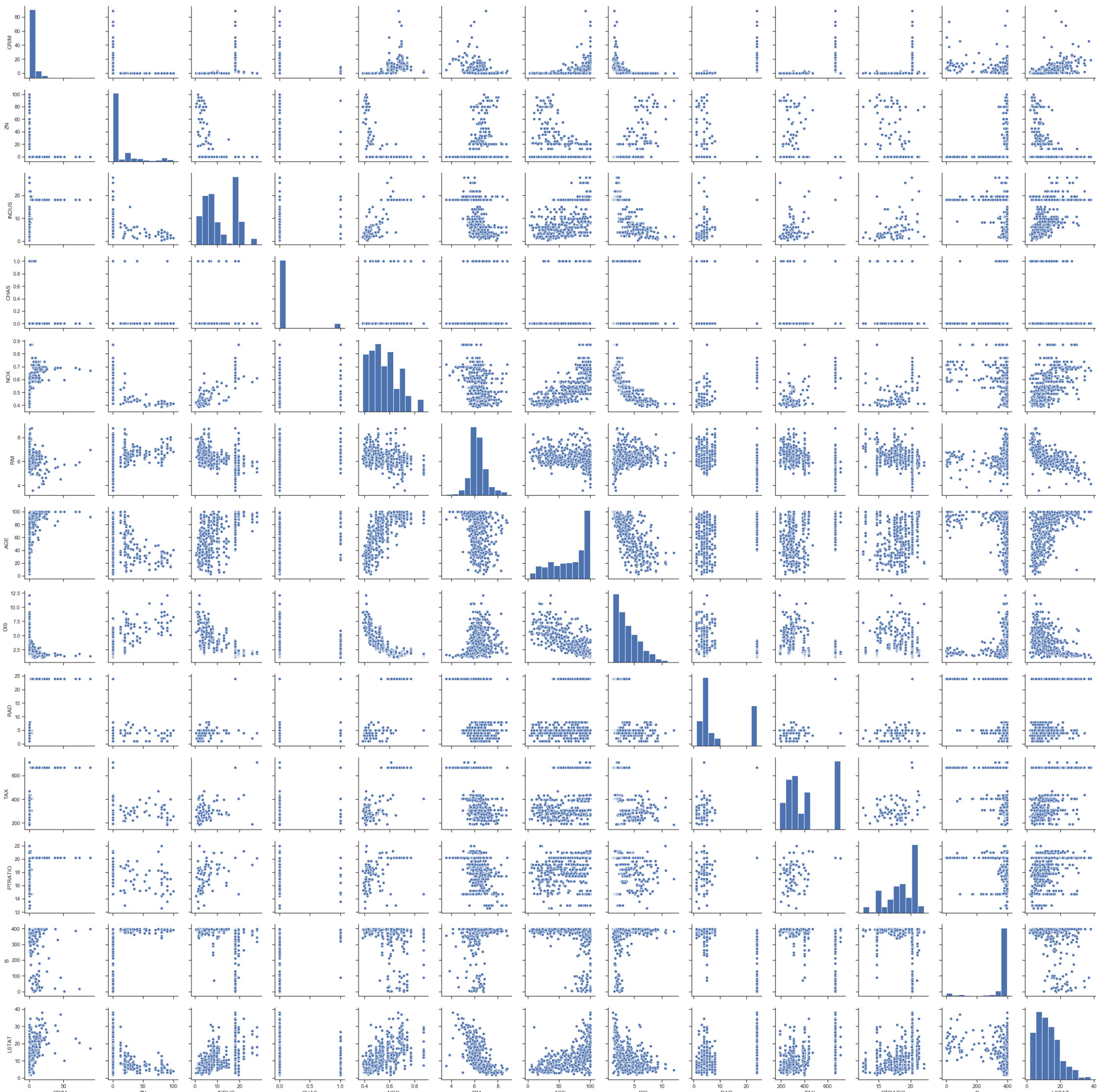
```
Out[100]: (506, 13)
```

- **Number of rows and columns :** 506 rows. 14 columns.
- **Rows and columns description :** Each rows is town in Boston area. Columns are features that can influence house price such as per capita crime rate by town ('CRIM').

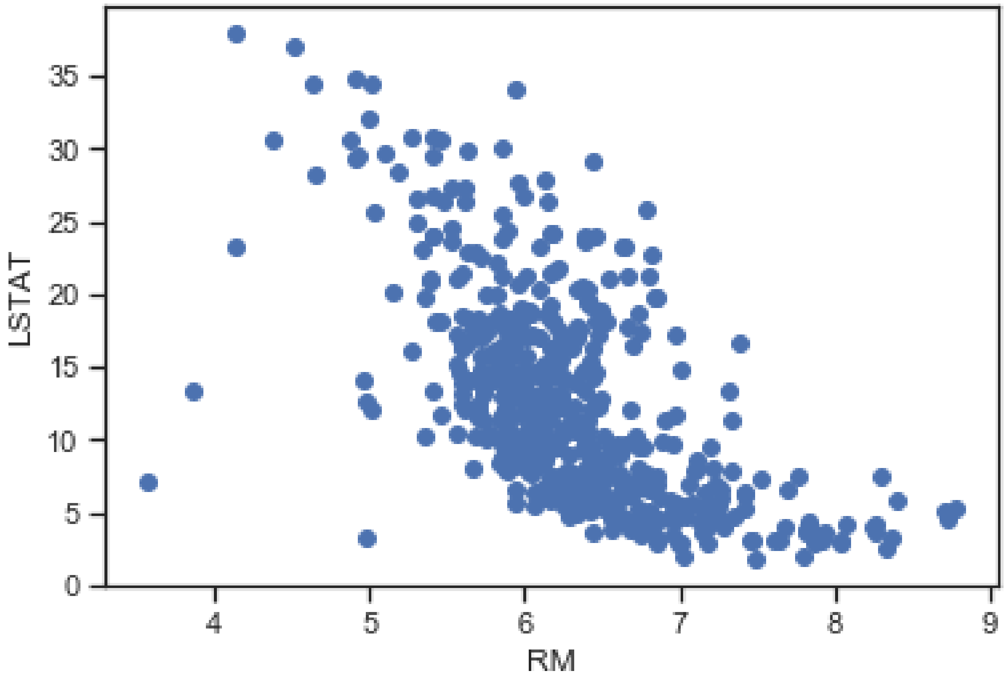
2.3.2 scatterplot

```
In [102]: sns.set(style = 'ticks')
          sns.pairplot(boston)
```

```
Out[102]: <seaborn.axisgrid.PairGrid at 0x35d344b0>
```




```
In [105]: plt.scatter(boston['RM'], boston['LSTAT'])
          plt.xlabel('RM')
          plt.ylabel('LSTAT')

Out[105]: Text(0, 0.5, 'LSTAT')
```



Findings : It seems to exist a negative non-linear relationship between LSTAT and RM It makes sense since people with less money (higher LSTAT) can't afford bigger houses (high RM)

2.3.3 Predictors associated with capita crime rate

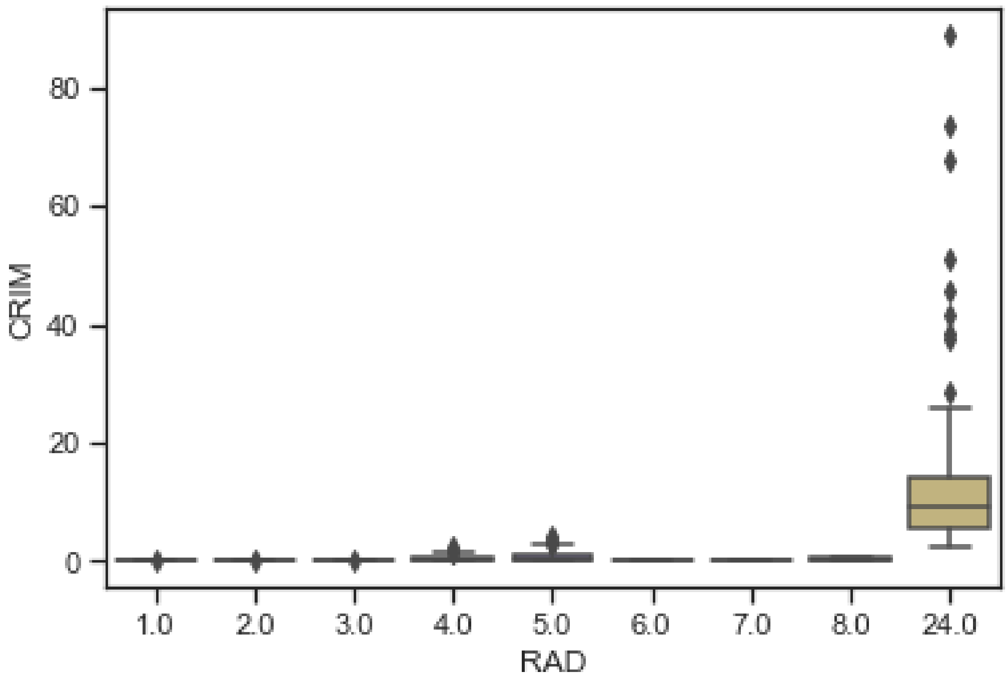
```
In [107]: boston.corrwith(boston['CRIM']).sort_values()
```

```
Out[107]: B          -0.385064
          DIS        -0.379670
          RM         -0.219247
          ZN         -0.200469
          CHAS       -0.055892
          PTRATIO     0.289946
          AGE         0.352734
          INDUS       0.406583
          NOX         0.420972
          LSTAT       0.455621
          TAX         0.582764
          RAD         0.625505
          CRIM        1.000000
          dtype: float64
```

Findings : Looking at the previous scatterplots and the correlation of each variable with 'CRIM', we will have a closer at the 3 with the largest correlation, namely: RAD, index of accessibility to radial highways, TAX, full-value property-tax rate (in dollars per $10,000), * LSTAT, percentage of lower status of the population.

```
In [109]: sns.boxplot(x="RAD", y="CRIM", data=boston)
```

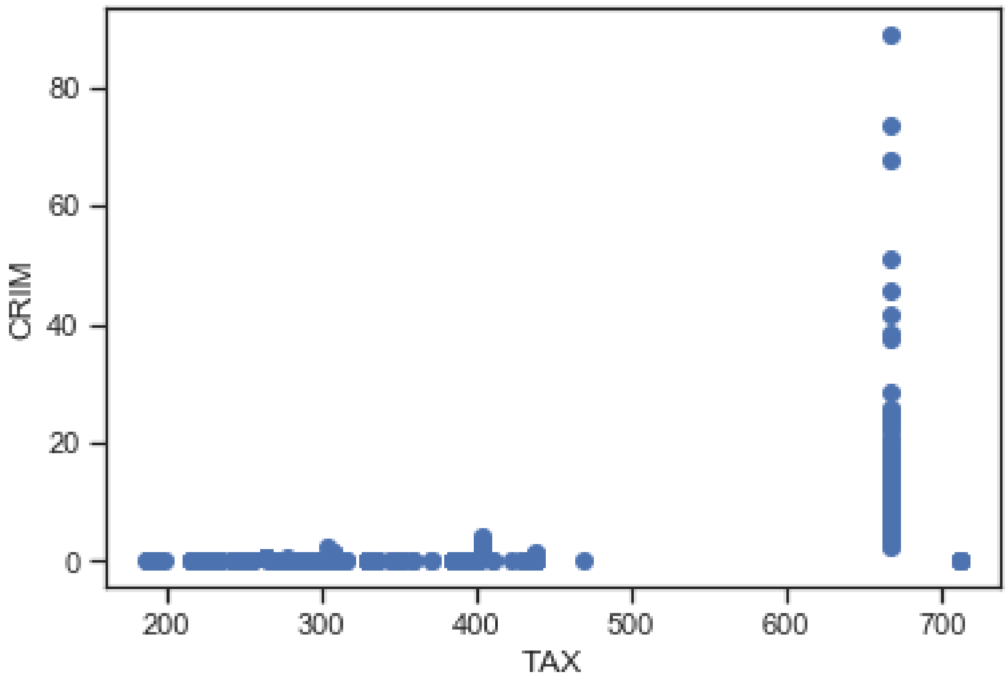
```
Out[109]: <matplotlib.axes._subplots.AxesSubplot at 0x3cf4f970>
```



Findings : When RAD is equal to 24 (its highest value), average CRIM is much higher and CRIM range is much larger.

```
In [110]: plt.scatter(boston['TAX'], boston['CRIM'])
          plt.xlabel('TAX')
          plt.ylabel('CRIM')
```

```
Out[110]: Text(0, 0.5, 'CRIM')
```



When TAX is equal to 666, average CRIM is much higher and CRIM range is much larger.

2.3.4 Crime rate, tax rate and pupil-teacher ratio in suburbs

```
In [113]: boston.iloc[boston['CRIM'].nlargest(5).index]

Out[113]:          CRIM   ZN  INDUS  CHAS    NOX     RM    AGE     DIS   RAD    TAX  \
          380  88.9762  0.0   18.1   0.0  0.671  6.968   91.9  1.4165  24.0  666.0
          418  73.5341  0.0   18.1   0.0  0.679  5.957  100.0  1.8026  24.0  666.0
          405  67.9208  0.0   18.1   0.0  0.693  5.683  100.0  1.4254  24.0  666.0
          410  51.1358  0.0   18.1   0.0  0.597  5.757  100.0  1.4130  24.0  666.0
          414  45.7461  0.0   18.1   0.0  0.693  4.519  100.0  1.6582  24.0  666.0

               PTRATIO       B  LSTAT
          380     20.2  396.90  17.21
          418     20.2   16.45  20.62
          405     20.2  384.97  22.98
          410     20.2    2.60  10.11
          414     20.2   88.27  36.98

In [115]: boston.iloc[boston['TAX'].nlargest(5).index]

Out[115]:          CRIM   ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
          488  0.15086  0.0  27.74   0.0  0.609  5.454  92.7  1.8209  4.0  711.0
          489  0.18337  0.0  27.74   0.0  0.609  5.414  98.3  1.7554  4.0  711.0
          490  0.20746  0.0  27.74   0.0  0.609  5.093  98.0  1.8226  4.0  711.0
```

```
491  0.10574  0.0  27.74   0.0  0.609  5.983  98.8  1.8681  4.0  711.0
492  0.11132  0.0  27.74   0.0  0.609  5.983  83.5  2.1099  4.0  711.0

     PTRATIO       B  LSTAT
488     20.1  395.09  18.06
489     20.1  344.05  23.97
490     20.1  318.43  29.68
491     20.1  390.11  18.07
492     20.1  396.90  13.35
```

In [116]: `boston.iloc[boston['PTRATIO'].nlargest(5).index]`

Out[116]:
```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE      DIS  RAD    TAX  \
354  0.04301  80.0   1.91   0.0  0.413  5.663  21.9  10.5857  4.0  334.0
355  0.10659  80.0   1.91   0.0  0.413  5.936  19.5  10.5857  4.0  334.0
127  0.25915   0.0  21.89   0.0  0.624  5.693  96.0   1.7883  4.0  437.0
128  0.32543   0.0  21.89   0.0  0.624  6.431  98.8   1.8125  4.0  437.0
129  0.88125   0.0  21.89   0.0  0.624  5.637  94.7   1.9799  4.0  437.0

     PTRATIO       B  LSTAT
354     22.0  382.80   8.05
355     22.0  376.04   5.57
127     21.2  392.11  17.19
128     21.2  396.90  15.39
129     21.2  396.90  18.34
```

In [117]: `boston.describe()`

Out[117]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  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
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT
```

```
count    506.000000
mean      12.653063
std        7.141062
min        1.730000
25%        6.950000
50%       11.360000
75%       16.955000
max       37.970000
```

Findings : The 5 towns shown in CRIM table are particularly high All the towns shown in the TAX table have maximum TAX level * PTRATIO table shows towns with high pupil-teacher ratios but not so uneven

2.3.5 Suburbs bounding the Charles river

```
In [118]: boston['CHAS'].value_counts()[1]

Out[118]: 35
```

2.3.6 Median pupil-teacher ratio

```
In [119]: boston['PTRATIO'].median()

Out[119]: 19.05
```

2.3.7 Suburb with lowest median value of owner occupied homes

```
In [127]: boston['target'].idxmin()

Out[127]: 398
```

```
In [130]: a = boston.describe()
          a.loc['range'] = a.loc['max'] - a.loc['min']
          a.loc[398] = boston.iloc[398]
          a
```

```
Out[130]:              CRIM          ZN       INDUS        CHAS         NOX          RM  \
          count  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
          std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
          min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
          25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
          50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
          75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
          max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000
          range   88.969880  100.000000   27.280000    1.000000    0.486000    5.219000
          398     38.351800    0.000000   18.100000    0.000000    0.693000    5.453000

                        AGE         DIS         RAD         TAX     PTRATIO           B  \
```

```
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000
range   97.100000   10.996900   23.000000  524.000000    9.400000  396.580000
398    100.000000    1.489600   24.000000  666.000000   20.200000  396.900000

            LSTAT      target
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
range   36.240000   45.000000
398     30.590000    5.000000
```

Findings : The suburb with the lowest median value is 398. Relative to the other towns, this suburb has high CRIM, ZN below quantile 75%, above mean INDUS, does not bound the Charles river, above mean NOX, RM below quantile 25%, maximum AGE, DIS near to the minimum value, maximum RAD, TAX in quantile 75%, PTRATIO as well, B maximum and LSTAT above quantile 75%.

2.3.8 Number of rooms per dwelling

```
In [133]: len(boston[boston['RM']>7])
```

```
Out[133]: 64
```

```
In [134]: len(boston[boston['RM']>8])
```

```
Out[134]: 13
```

```
In [135]: boston[boston['RM']>8].describe()
```

```
Out[135]:             CRIM         ZN      INDUS       CHAS        NOX         RM  \
           count  13.000000  13.000000  13.000000  13.000000  13.000000  13.000000
           mean    0.718795  13.615385   7.078462   0.153846   0.539238   8.348538
           std     0.901640  26.298094   5.392767   0.375534   0.092352   0.251261
           min     0.020090   0.000000   2.680000   0.000000   0.416100   8.034000
           25%     0.331470   0.000000   3.970000   0.000000   0.504000   8.247000
           50%     0.520140   0.000000   6.200000   0.000000   0.507000   8.297000
           75%     0.578340  20.000000   6.200000   0.000000   0.605000   8.398000
```

```
max     3.474280  95.000000  19.580000   1.000000   0.718000   8.780000

              AGE        DIS        RAD         TAX    PTRATIO           B  \
count  13.000000  13.000000  13.000000   13.000000  13.000000   13.000000
mean   71.538462   3.430192   7.461538  325.076923  16.361538  385.210769
std    24.608723   1.883955   5.332532  110.971063   2.410580   10.529359
min     8.400000   1.801000   2.000000  224.000000  13.000000  354.550000
25%    70.400000   2.288500   5.000000  264.000000  14.700000  384.540000
50%    78.300000   2.894400   7.000000  307.000000  17.400000  386.860000
75%    86.500000   3.651900   8.000000  307.000000  17.400000  389.700000
max    93.900000   8.906700  24.000000  666.000000  20.200000  396.900000

           LSTAT     target
count  13.000000  13.000000
mean    4.310000  44.200000
std     1.373566   8.092383
min     2.470000  21.900000
25%     3.320000  41.700000
50%     4.140000  48.300000
75%     5.120000  50.000000
max     7.440000  50.000000
```

```
In [ ]:
```