PS6_psy

February 19, 2019

1 PS6

1.1 Siyuan Peng

In [38]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
import statsmodels.api as sm
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

1.2 1

1.3 (a)

In [3]:
```
df = pd.read_csv('data/Auto.csv', na_values='?')
df.dropna(inplace=True)
df.head(10)
```

Out[3]:
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0       130.0    3504          12.0    70
1  15.0          8         350.0       165.0    3693          11.5    70
2  18.0          8         318.0       150.0    3436          11.0    70
3  16.0          8         304.0       150.0    3433          12.0    70
4  17.0          8         302.0       140.0    3449          10.5    70
5  15.0          8         429.0       198.0    4341          10.0    70
6  14.0          8         454.0       220.0    4354           9.0    70
7  14.0          8         440.0       215.0    4312           8.5    70
8  14.0          8         455.0       225.0    4425          10.0    70
9  15.0          8         390.0       190.0    3850           8.5    70

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

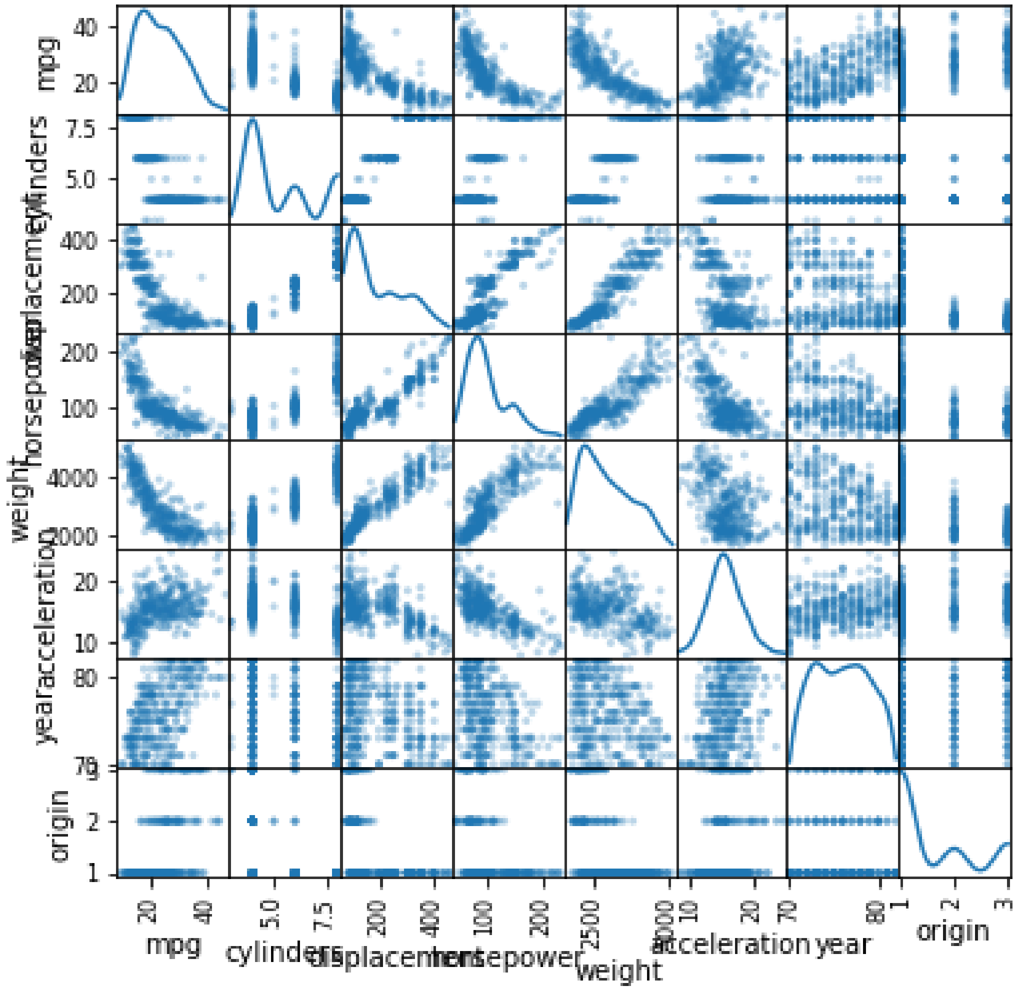
```
4       1                ford torino
5       1           ford galaxie 500
6       1           chevrolet impala
7       1          plymouth fury iii
8       1           pontiac catalina
9       1         amc ambassador dpl
```

1.4 (b)

```
In [4]: scatter_matrix(df, alpha=0.3, figsize=(6, 6),diagonal='kde')
```

```
Out[4]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001703B1EB630>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D249EB8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D27B470>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D2A29E8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D2CBF60>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D2FA518>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D322A90>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D353080>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x000001703D3530B8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D3A2B38>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D3D30F0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D3F9668>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D41EBE0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D454198>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D478710>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D4A0C88>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x000001703D4D0240>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D4F77B8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D520D30>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D5532E8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D57A860>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D5A0DD8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D5D0390>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D5F6908>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x000001703D61FE80>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D64F438>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D6779B0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D6A1F28>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D6D04E0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D6F7A58>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D71EFD0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D74F588>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x000001703D777B00>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D7A70B8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D7CD630>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D7F6BA8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D826160>,
```

```
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D84F6D8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D879C50>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D8A9208>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x000001703D8CF780>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D8F5CF8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D9262B0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D94E828>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D976DA0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D9A6358>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D9D08D0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703D9F6E48>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x000001703DA25400>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DA4D978>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DA75EF0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DAA54A8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DACCA20>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DAF7F98>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DB27550>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DB4BAC8>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x000001703DB7E080>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DBA55F8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DBCBB70>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DBFE128>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DC246A0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DC4BC18>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DC7C1D0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001703DCA2748>]],
dtype=object)
```



1.5 (c)

```
In [5]: df.corr()
```

```
Out[5]:                    mpg  cylinders  displacement  horsepower    weight  \
        mpg           1.000000  -0.777618     -0.805127   -0.778427 -0.832244
        cylinders    -0.777618   1.000000      0.950823    0.842983  0.897527
        displacement -0.805127   0.950823      1.000000    0.897257  0.932994
        horsepower   -0.778427   0.842983      0.897257    1.000000  0.864538
        weight       -0.832244   0.897527      0.932994    0.864538  1.000000
        acceleration  0.423329  -0.504683     -0.543800   -0.689196 -0.416839
        year          0.580541  -0.345647     -0.369855   -0.416361 -0.309120
        origin        0.565209  -0.568932     -0.614535   -0.455171 -0.585005

                      acceleration      year    origin
        mpg               0.423329  0.580541  0.565209
```

```
cylinders       -0.504683 -0.345647 -0.568932
displacement    -0.543800 -0.369855 -0.614535
horsepower      -0.689196 -0.416361 -0.455171
weight          -0.416839 -0.309120 -0.585005
acceleration     1.000000  0.290316  0.212746
year             0.290316  1.000000  0.181528
origin           0.212746  0.181528  1.000000
```

1.6 (d)

```
In [6]: y = df.mpg
        X = df[['cylinders', 'displacement','horsepower', 'weight', 'acceleration', 'year',
        'origin']]
        X = sm.add_constant(X)
        model = sm.OLS(y,X)
        results = model.fit()
        print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     252.4
Date:                Tue, 19 Feb 2019   Prob (F-statistic):         2.04e-139
Time:                        16:16:59   Log-Likelihood:                -1023.5
No. Observations:                 392   AIC:                             2063.
Df Residuals:                     384   BIC:                             2095.
Df Model:                           7
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const          -17.2184      4.644     -3.707      0.000     -26.350      -8.087
cylinders       -0.4934      0.323     -1.526      0.128      -1.129       0.142
displacement     0.0199      0.008      2.647      0.008       0.005       0.035
horsepower      -0.0170      0.014     -1.230      0.220      -0.044       0.010
weight          -0.0065      0.001     -9.929      0.000      -0.008      -0.005
acceleration     0.0806      0.099      0.815      0.415      -0.114       0.275
year             0.7508      0.051     14.729      0.000       0.651       0.851
origin           1.4261      0.278      5.127      0.000       0.879       1.973
==============================================================================
Omnibus:                       31.906   Durbin-Watson:                   1.309
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               53.100
Skew:                           0.529   Prob(JB):                     2.95e-12
Kurtosis:                       4.460   Cond. No.                     8.59e+04
==============================================================================

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

```
[2] The condition number is large, 8.59e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

From the above result, we could clearly see that:

i. Coefficients of constant, displacement, weight, year and origin are statistically significant at the 1% level.

ii. Coefficients of cylinders, horsepower and acceleration are not statistically significant at the 10% level.

iii. Given all other variables as the same, one more year of the vehicle will increase miles per gallon by about 0.7508.

1.7 (e)

From the above diagram, the three variables that most likely to be non-linearly related with mpg_i are $displacement_i$, $horsepower_i$, $weight_i$.

```
In [7]: df['displacement2'] = df.displacement ** 2
        df['horsepower2'] = df.horsepower ** 2
        df['weight2'] = df.weight ** 2
        df['acceleration2'] = df.acceleration ** 2
        y = df.mpg
        X2 = df[['cylinders', 'displacement','displacement2', 'horsepower',
                 'horsepower2', 'weight', 'weight2', 'acceleration', 'acceleration2', 'year',
        'origin']]
        X2 = sm.add_constant(X2)
        model2 = sm.OLS(y,X2)
        results2 = model2.fit()
        print(results2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.870
Model:                            OLS   Adj. R-squared:                  0.866
Method:                 Least Squares   F-statistic:                     230.2
Date:                Tue, 19 Feb 2019   Prob (F-statistic):         1.75e-160
Time:                        16:35:50   Log-Likelihood:                -962.02
No. Observations:                 392   AIC:                             1948.
Df Residuals:                     380   BIC:                             1996.
Df Model:                          11
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const            20.1084      6.696      3.003      0.003       6.943      33.274
cylinders         0.2519      0.326      0.773      0.440      -0.389       0.893
displacement     -0.0169      0.020     -0.828      0.408      -0.057       0.023
displacement2  2.257e-05   3.61e-05      0.626      0.532   -4.83e-05    9.35e-05
horsepower       -0.1635      0.041     -3.971      0.000      -0.244      -0.083
```

```
horsepower2        0.0004     0.000      2.943      0.003      0.000      0.001
weight            -0.0136     0.003     -5.069      0.000     -0.019     -0.008
weight2         1.514e-06  3.69e-07      4.105      0.000   7.89e-07   2.24e-06
acceleration      -2.0884     0.557     -3.752      0.000     -3.183     -0.994
acceleration2      0.0576     0.016      3.496      0.001      0.025      0.090
year               0.7810     0.045     17.512      0.000      0.693      0.869
origin             0.6104     0.263      2.320      0.021      0.093      1.128
==============================================================================
Omnibus:                       33.614   Durbin-Watson:                   1.576
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               77.985
Skew:                           0.438   Prob(JB):                     1.16e-17
Kurtosis:                       5.002   Cond. No.                     5.13e+08
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.13e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [8]:
```python
from statsmodels.iolib.summary2 import summary_col

info_dict={'R-squared' : lambda x: "{:.2f}".format(x.rsquared),
           'No. observations' : lambda x: "{0:d}".format(int(x.nobs))}

results_table = summary_col(results=[results,results2],
                            float_format='%0.2f',
                            stars = True,
                            model_names=['Model 1',
                                         'Model 2'],
                            info_dict=info_dict,
                            regressor_order=['cylinders',
'displacement','displacement2', 'horsepower',
                                             'horsepower2', 'weight', 'weight2',
'acceleration', 'acceleration2', 'year', 'origin'])

results_table.add_title('Table 2 - OLS Regressions')

print(results_table)
```

```
     Table 2 - OLS Regressions
===================================
                 Model 1  Model 2
-----------------------------------
cylinders        -0.49    0.25
                 (0.32)   (0.33)
displacement     0.02***  -0.02
                 (0.01)   (0.02)
displacement2             0.00
                          (0.00)
horsepower       -0.02    -0.16***
                 (0.01)   (0.04)
horsepower2               0.00***
```

```
                        (0.00)
weight           -0.01***  -0.01***
                 (0.00)    (0.00)
weight2                    0.00***
                           (0.00)
acceleration     0.08      -2.09***
                 (0.10)    (0.56)
acceleration2              0.06***
                           (0.02)
year             0.75***   0.78***
                 (0.05)    (0.04)
origin           1.43***   0.61**
                 (0.28)    (0.26)
const            -17.22*** 20.11***
                 (4.64)    (6.70)
R-squared        0.82      0.87
No. observations 392       392
===================================
Standard errors in parentheses.
* p<.1, ** p<.05, ***p<.01
```

i. From the above result, it is clear that the adjusted R-squared changes from 0.818 to 0.866. The new model is better to explain the variance.

ii. The coefficient of displacement drops from significant at 1% level to nonsignificant at 10% level. Its squared term is also nonsignificant at 10% level.

iii. The coefficient of cylinders remains unsignificant at 10% level.

1.8 (f)

```
In [9]: X_pre = [1, 6, 200, 200**2, 100, 100**2, 3100, 3100**2, 15.1, 15.1**2, 99, 1]
        pre_mpg = results2.predict(exog = X_pre)[0]
        print('The predicted miles per gallon is', pre_mpg)
```

```
The predicted miles per gallon is 38.73211109723756
```

1.9 2

1.10 (a)

```
In [15]: Points = np.array([[0,3,0],[2,0,0],[0,1,3],[0,1,2],[-1,0,1],[1,1,1]])
         Distances = np.zeros((6,1))
         for i in range(6):
             Distances[i,0] = np.sqrt((Points[i,0]-0) ** 2 + (Points[i,1]-0) ** 2 +
         (Points[i,2]-0) ** 2)
         for i in range(6):
             print("The distance from observation", i+1, "to point (0,0,0)
         is",round(Distances[i,0],4))
```

```
The distance from observation 1 to point (0,0,0) is 3.0
The distance from observation 2 to point (0,0,0) is 2.0
The distance from observation 3 to point (0,0,0) is 3.1623
The distance from observation 4 to point (0,0,0) is 2.2361
The distance from observation 5 to point (0,0,0) is 1.4142
The distance from observation 6 to point (0,0,0) is 1.7321
```

1.11 (b)

With K=1, we will choose observation 5 as the closest one and our predicition is green.

1.12 (c)

The three closest observations are 2,5 and 6. Considering that 2 of them are red and one of them is green, our predicition is red.

1.13 (d)

If the Bayes(optimal) decision boundary in this problem is highly non-linear, then we would expect the best value for K to be large. Considering that the boundary is highly non-linear, we want to make sure that K is big enough to capture enough points in all directions. If K is rather small, some directions might missing and lead us to wrong prediction.

1.14 (e)

```
In [35]: ys = np.array(['Red','Red','Red','Green','Green','Red']).reshape(-1,1)
         test_point = np.array([1,1,1]).reshape(1,-1)
         knn = KNeighborsClassifier(n_neighbors=2, weights='distance')
         cls = knn.fit(Points, ys)
         print('The KNN classifier of the test point is', cls.predict(test_point)[0])
```

```
The KNN classifier of the test point is Red
```

1.15 3

```
In [28]: df['mpg_high'] = 0
         df.mpg_high[df.mpg>df.mpg.median()] = 1
```

1.16 (a)

```
In [29]: y = df.mpg_high
         model3 = sm.Logit(y, X)
         results3 = model3.fit()
         print(results3.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.200944
         Iterations 9
                           Logit Regression Results
==============================================================================
```

```
Dep. Variable:               mpg_high   No. Observations:                  392
Model:                          Logit   Df Residuals:                      384
Method:                           MLE   Df Model:                            7
Date:                Tue, 19 Feb 2019   Pseudo R-squ.:                  0.7101
Time:                        17:14:40   Log-Likelihood:                -78.770
converged:                       True   LL-Null:                       -271.71
                                        LLR p-value:                 2.531e-79
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const          -17.1549      5.764     -2.976      0.003     -28.452      -5.858
cylinders       -0.1626      0.423     -0.384      0.701      -0.992       0.667
displacement     0.0021      0.012      0.174      0.862      -0.021       0.026
horsepower      -0.0410      0.024     -1.718      0.086      -0.088       0.006
weight          -0.0043      0.001     -3.784      0.000      -0.007      -0.002
acceleration     0.0161      0.141      0.114      0.910      -0.261       0.293
year             0.4295      0.075      5.709      0.000       0.282       0.577
origin           0.4773      0.362      1.319      0.187      -0.232       1.187
================================================================================

Possibly complete quasi-separation: A fraction 0.14 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

From the above result, it is clear that coefficients of constant, weight and year are statistically significant at the 5% level.

1.17 (b)

```
In [37]: y = df.mpg_high
         X = df[['cylinders', 'displacement','horsepower', 'weight', 'acceleration', 'year',
         'origin']]
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5,
         random_state=10)
```

1.18 (c)

```
In [48]: result = LogisticRegression(random_state=10, solver='lbfgs',multi_class='multinomial',
         max_iter=5000).fit(X_train, y_train)
         print(result.intercept_, result.coef_[0])
```

```
[-12.68077284] [-0.69837061  0.0105236   0.00759311 -0.0036614   0.06907609
0.29950794
  0.08758037]
```

1.19 (d)

```
In [60]: y_pred = result.predict(X_test)
         score = result.score(X_test, y_test)
         score
```

Out[60]: 0.8724489795918368

```
In [61]: print(confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [12 85]]
```

```
In [63]: print(classification_report(y_test, y_pred,target_names=['Low mpg', 'High mpg']))
```

```
              precision    recall  f1-score   support

     Low mpg       0.88      0.87      0.87        99
    High mpg       0.87      0.88      0.87        97

   micro avg       0.87      0.87      0.87       196
   macro avg       0.87      0.87      0.87       196
weighted avg       0.87      0.87      0.87       196
```

From the above result, with almost the same F1 score, precision and recall, this model predict high mpg and low mpg almost the same.