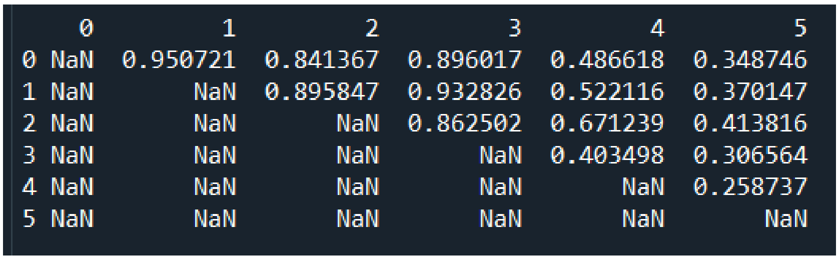
ML HW4 109070032 廖品睿

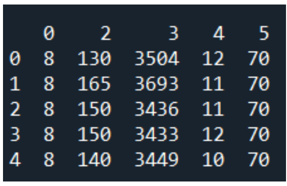
1.

(a) **#High correlation filter**

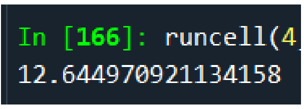
This is the covariance matrix of each feature and we get rid of the highest correlated feature first, which is feature 1 (displacement)



```
     0         1         2         3         4         5
0  NaN  0.950721  0.841367  0.896017  0.486618  0.348746
1  NaN       NaN  0.895847  0.932826  0.522116  0.370147
2  NaN       NaN       NaN  0.862502  0.671239  0.413816
3  NaN       NaN       NaN       NaN  0.403498  0.306564
4  NaN       NaN       NaN       NaN       NaN  0.258737
5  NaN       NaN       NaN       NaN       NaN       NaN
```



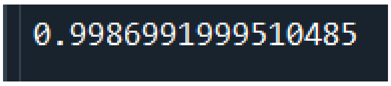
```
   0    2     3   4   5
0  8  130  3504  12  70
1  8  165  3693  11  70
2  8  150  3436  11  70
3  8  150  3433  12  70
4  8  140  3449  10  70
```



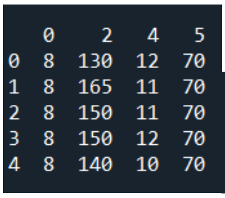
```
In [166]: runcell(4
12.644970921134158
```

the loss is 12.64

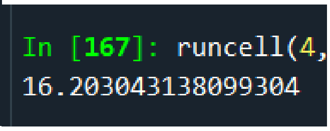
And the R-square is



```
0.9986991999510485
```



```
   0    2   4   5
0  8  130  12  70
1  8  165  11  70
2  8  150  11  70
3  8  150  12  70
4  8  140  10  70
```

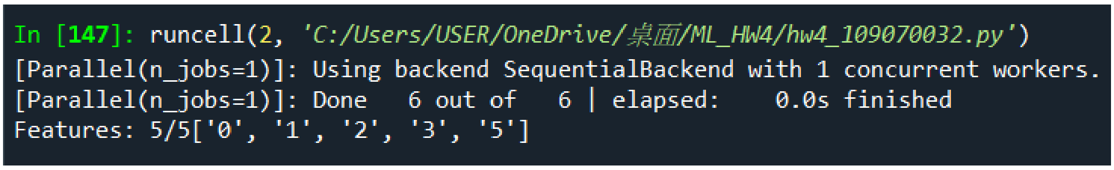


```
In [167]: runcell(4,
16.203043138099304
```

once we get rid of feature 3 (weight), the loss increases to 16.2, therefore we choose feature cylinders, horsepower, weight, acceleration, model year to predict results.

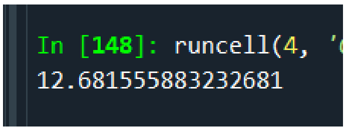
(b) **#Backward selection**

If we choose feature 0,1,2,3,5



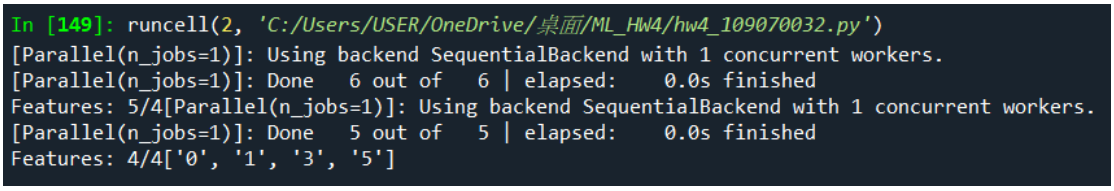
```
In [147]: runcell(2, 'C:/Users/USER/OneDrive/桌面/ML_HW4/hw4_109070032.py')
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished
Features: 5/5['0', '1', '2', '3', '5']
```

We get this much loss

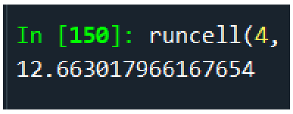


```
In [148]: runcell(4, '
12.681555883232681
```

If we choose feature 0,1,3,5



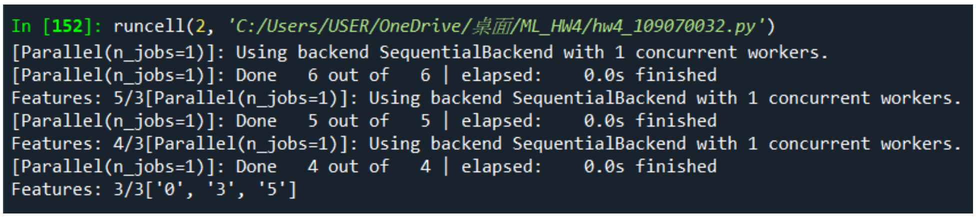
```
In [149]: runcell(2, 'C:/Users/USER/OneDrive/桌面/ML_HW4/hw4_109070032.py')
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished
Features: 5/4[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
Features: 4/4['0', '1', '3', '5']
```



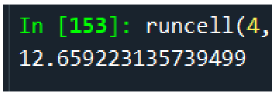
```
In [150]: runcell(4,
12.663017966167654
```

We get this much loss

If we choose feature 0,3,5



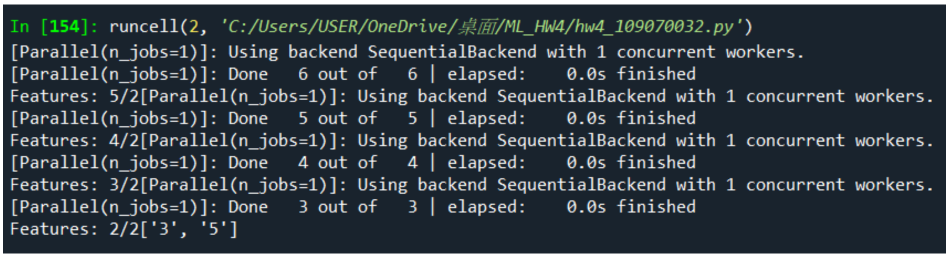
```
In [152]: runcell(2, 'C:/Users/USER/OneDrive/桌面/ML_HW4/hw4_109070032.py')
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished
Features: 5/3[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
Features: 4/3[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s finished
Features: 3/3['0', '3', '5']
```



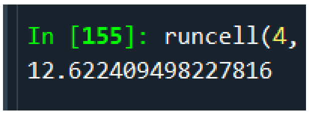
```
In [153]: runcell(4,
12.659223135739499
```

We get this much loss

If we choose feature 3,5

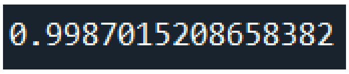


```
In [154]: runcell(2, 'C:/Users/USER/OneDrive/桌面/ML_HW4/hw4_109070032.py')
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished
Features: 5/2[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
Features: 4/2[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s finished
Features: 3/2[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s finished
Features: 2/2['3', '5']
```



```
In [155]: runcell(4,
12.622409498227816
```

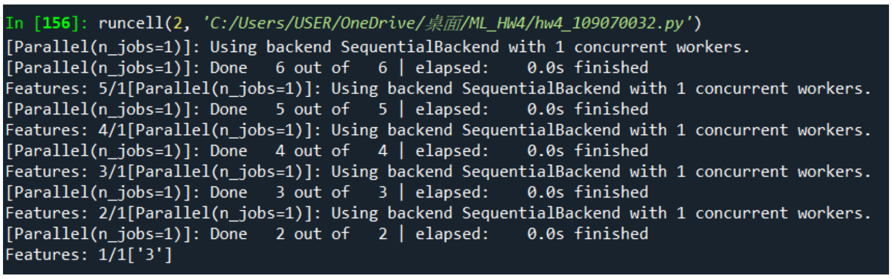
We get this much loss



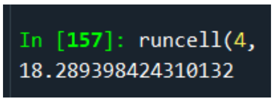
```
0.9987015208658382
```

And we get R-square close to

If we choose feature 3



```
In [156]: runcell(2, 'C:/Users/USER/OneDrive/桌面/ML_HW4/hw4_109070032.py')
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished
Features: 5/1[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
Features: 4/1[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s finished
Features: 3/1[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s finished
Features: 2/1[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s finished
Features: 1/1['3']
```



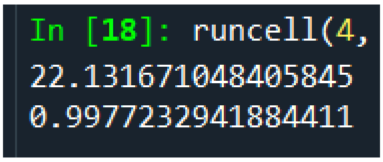
```
In [157]: runcell(4,
18.289398424310132
```

We get this much loss

The loss gets bigger, therefore we choose the feature 3 (weight) and feature 5 (model year) to build the model.

(c)**#PCA**

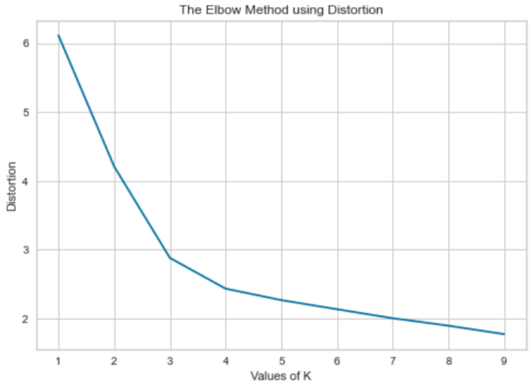
Using PCA method using only feature 1, we get 24.39 loss, and R-square equal to 0.9977, which is a bit lower than the first two methods because we only choose the features that > 95% variance. Apparently, keeping only one feature for prediction is somehow a bit inaccurate.

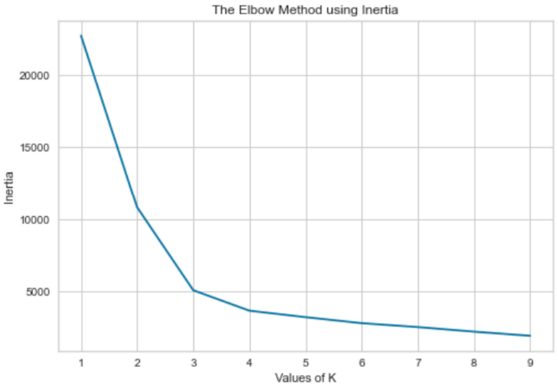


```
In [18]: runcell(4,
22.131671048405845
0.9977232941884411
```

2.

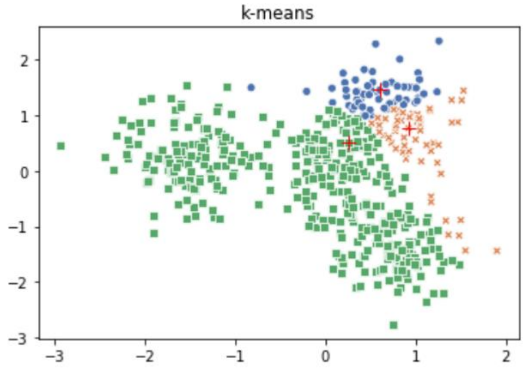
After fitting the dataset into the code below we can find the elbow point under the relatively less distortion score, and because distortion score and inertia score using elbow method is too large in n=2 and fit time is too long in n=4, therefore we choose the optimal n=3 to be the elbow point.

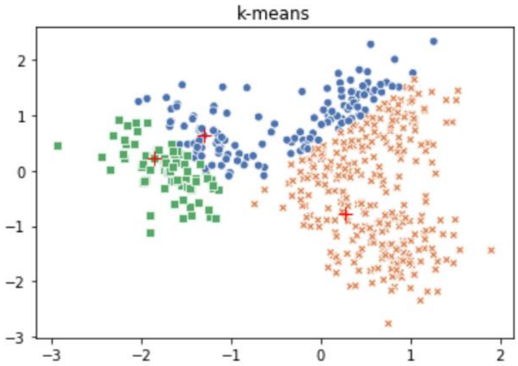


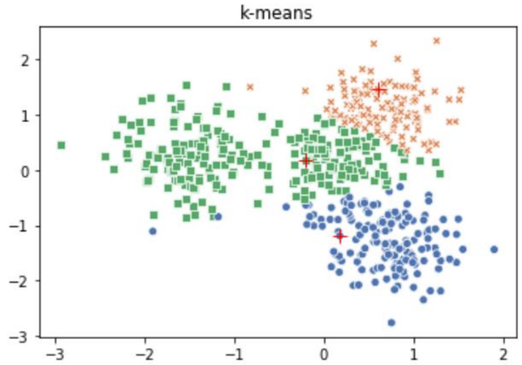


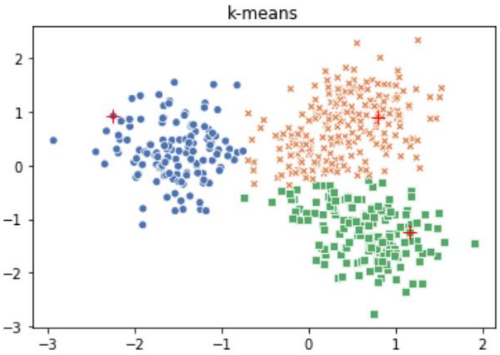
Using handcrafted k-means model to cluster data will have multiple kinds of outcomes because we randomize the center data to plot , therefore the size and the positions of clusters are relatively random, also we assume that we can converge the center in 300 iterations, unlike python k-means model, we get a relatively imprecise result because this method is unsupervised learning.

Note: the red cross represents the center of data, we approximately get 4 kinds of data like the 4 plots below.

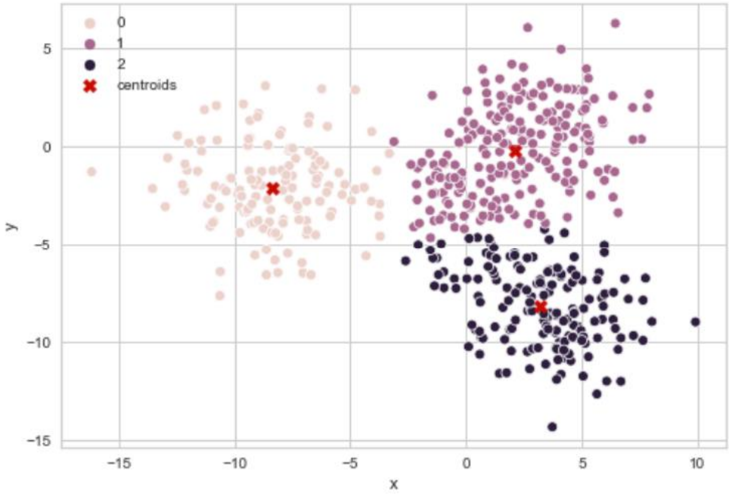








The plot below is created by sklearn model in python, which is only similar to the 4th plot above, and this model has the most accuracy and least distortion.


centroids
y
x