data_vis_kmean_asarker

October 20, 2019

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
```

0.1 Load and clean up automobile data

```
In [2]: cols= ["symboling","normalized-losses","make","fuel-type","aspiration","num-of-doors",'
            "drive-wheels","engine-location","wheel-base","length","width","height","curb-weigh
              "num-of-cylinders","engine-size","fuel-system","bore","stroke","compression-ratio
                "peak-rpm","city-mpg","highway-mpg","price"]

        car_data = pd.read_csv('imports-85.data', header=None,names=cols)
        car_data.shape
```

```
Out[2]: (205, 26)
```

```
In [3]: car_data.replace("?",np.nan,inplace=True)
        car_data.dropna(inplace=True)
        car_data.reset_index(drop=True, inplace=True)
```

0.2 Find out 6 continuous numerical attributes of the Automobile Data Set

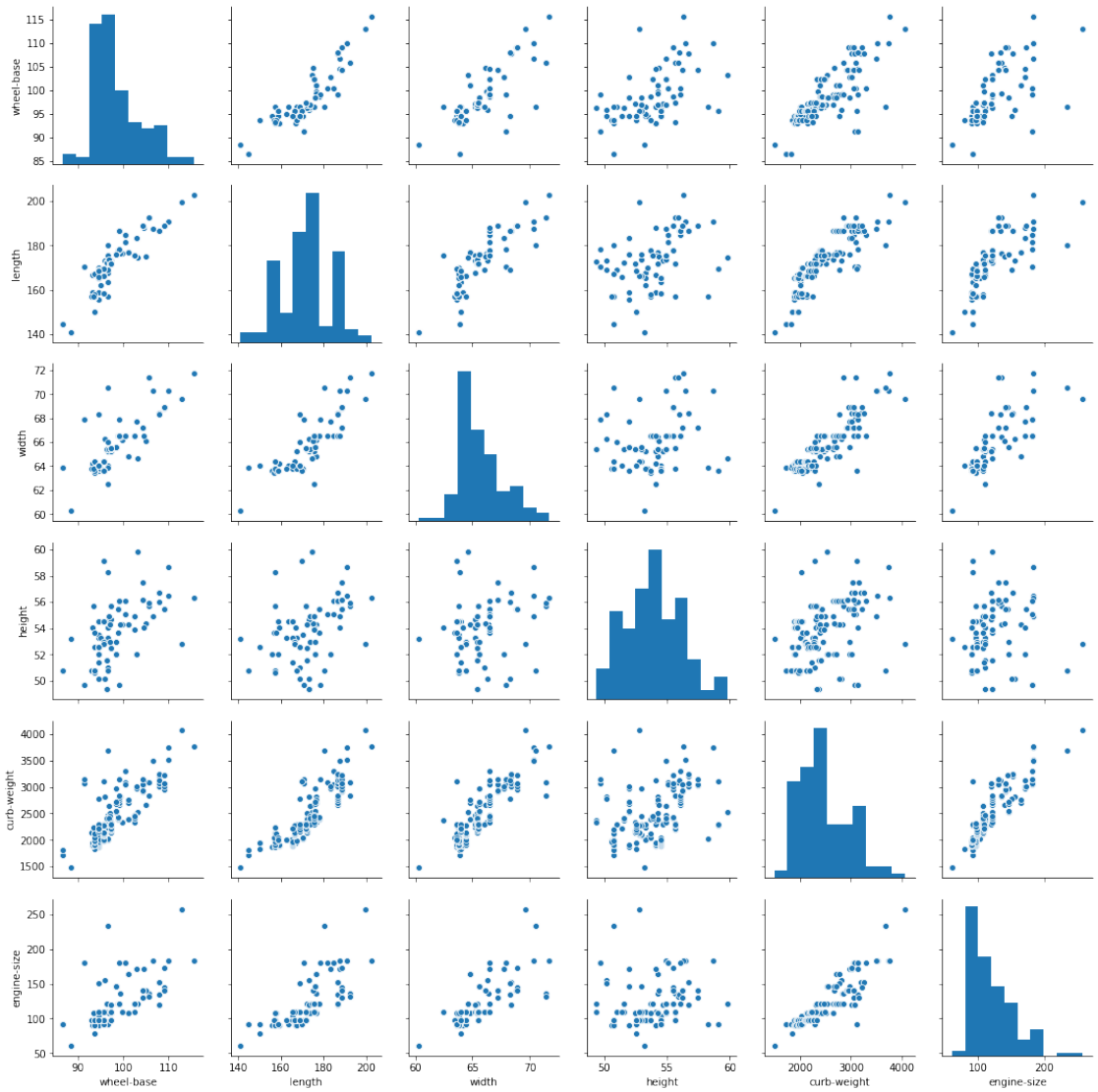
```
In [4]: car_data.corr()
```

```
Out[4]:                    symboling  wheel-base    length     width    height  \
        symboling           1.000000   -0.520591 -0.336257 -0.219186 -0.475185
        wheel-base         -0.520591    1.000000  0.871534  0.814991  0.555767
        length             -0.336257    0.871534  1.000000  0.838338  0.499251
        width              -0.219186    0.814991  0.838338  1.000000  0.292706
        height             -0.475185    0.555767  0.499251  0.292706  1.000000
        curb-weight        -0.251880    0.810181  0.871291  0.870595  0.367052
        engine-size        -0.109453    0.649206  0.725953  0.779253  0.111083
        compression-ratio  -0.138316    0.291431  0.184814  0.258752  0.233308
        city-mpg            0.089550   -0.580657 -0.724544 -0.666684 -0.199737
        highway-mpg         0.149830   -0.611750 -0.724599 -0.693339 -0.226136

                           curb-weight  engine-size  compression-ratio  city-mpg  \
        symboling            -0.251880    -0.109453          -0.138316  0.089550
```

```
wheel-base            0.810181     0.649206          0.291431 -0.580657
length                0.871291     0.725953          0.184814 -0.724544
width                 0.870595     0.779253          0.258752 -0.666684
height                0.367052     0.111083          0.233308 -0.199737
curb-weight           1.000000     0.888626          0.224724 -0.762155
engine-size           0.888626     1.000000          0.141097 -0.699139
compression-ratio     0.224724     0.141097          1.000000  0.278332
city-mpg             -0.762155    -0.699139          0.278332  1.000000
highway-mpg          -0.789338    -0.714095          0.221483  0.971999

                    highway-mpg
symboling              0.149830
wheel-base            -0.611750
length                -0.724599
width                 -0.693339
height                -0.226136
curb-weight           -0.789338
engine-size           -0.714095
compression-ratio      0.221483
city-mpg               0.971999
highway-mpg            1.000000
```

In [5]:
```
features = ["wheel-base", "length", "width", "height", "curb-weight", "engine-size"]
data =car_data[features]
sns.pairplot(data.dropna())
```

Out[5]:
```
<seaborn.axisgrid.PairGrid at 0x1a0e3a9390>
```



1 PCA Projection to 2D

In [6]:
```
from sklearn.preprocessing import StandardScaler

# Separating out the features
car_data_6D = car_data.loc[:, features].values

# Standardizing the features
car_data_6D = StandardScaler().fit_transform(car_data_6D)
```

In [7]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(car_data_6D)
```

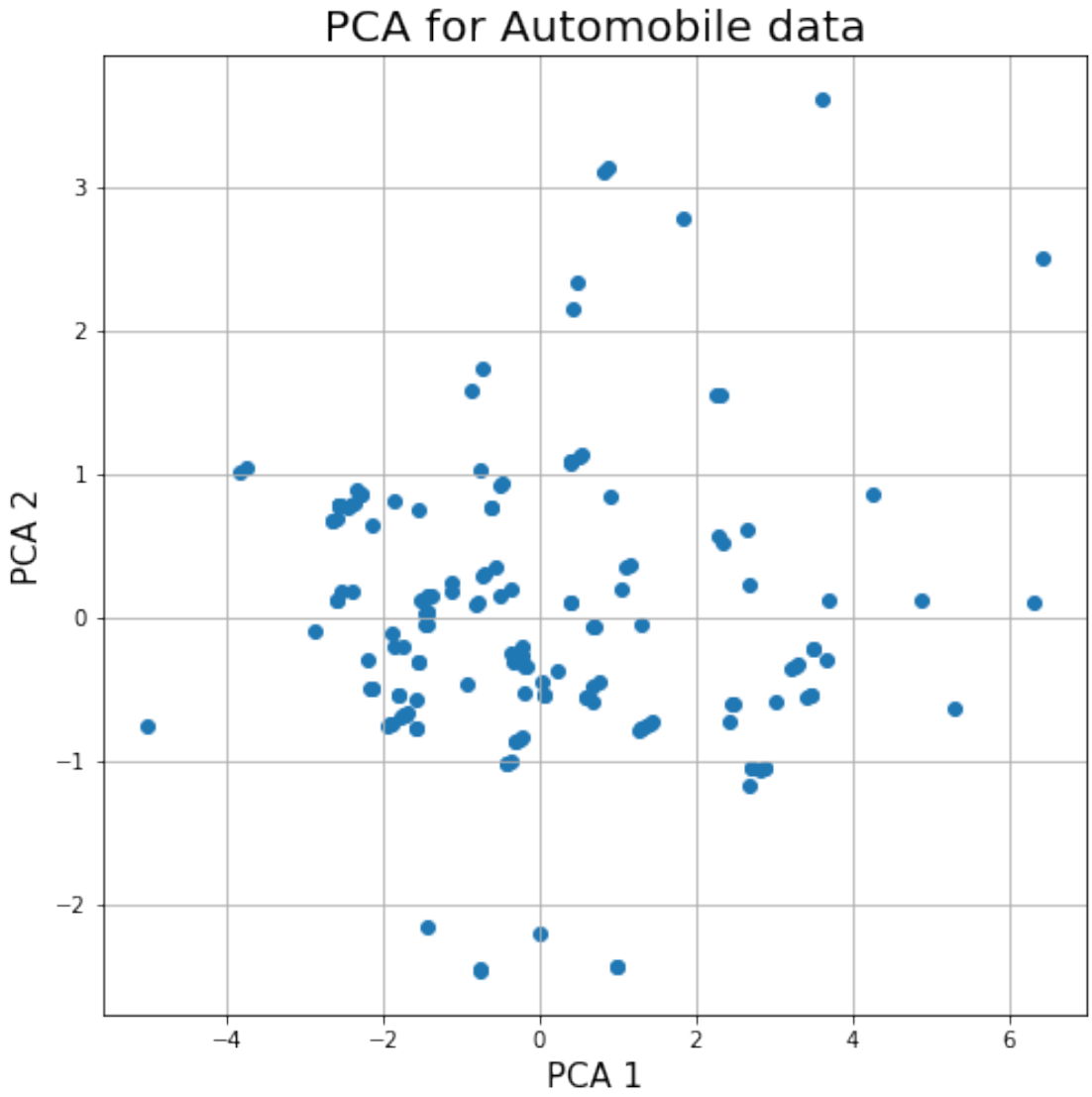
```
pcaDF = pd.DataFrame(data = principalComponents
             , columns = ['PCA 1', 'PCA 2'])
pcaDF.head()
```

Out[7]:

	PCA 1	PCA 2
0	0.216065	-0.366922
1	1.041547	0.201788
2	3.497362	-0.218345
3	3.680574	-0.287492
4	0.069300	-0.540125

1.1 Visualize 2D Projection

In [8]:
```
import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('PCA 1', fontsize = 15)
ax.set_ylabel('PCA 2', fontsize = 15)
ax.set_title('PCA for Automobile data', fontsize = 20)
ax.scatter(pcaDF['PCA 1'], pcaDF['PCA 2'])
ax.grid()
plt.show()
```




```
In [9]: pca.explained_variance_
```

```
Out[9]: array([4.4774846 , 0.99515306])
```

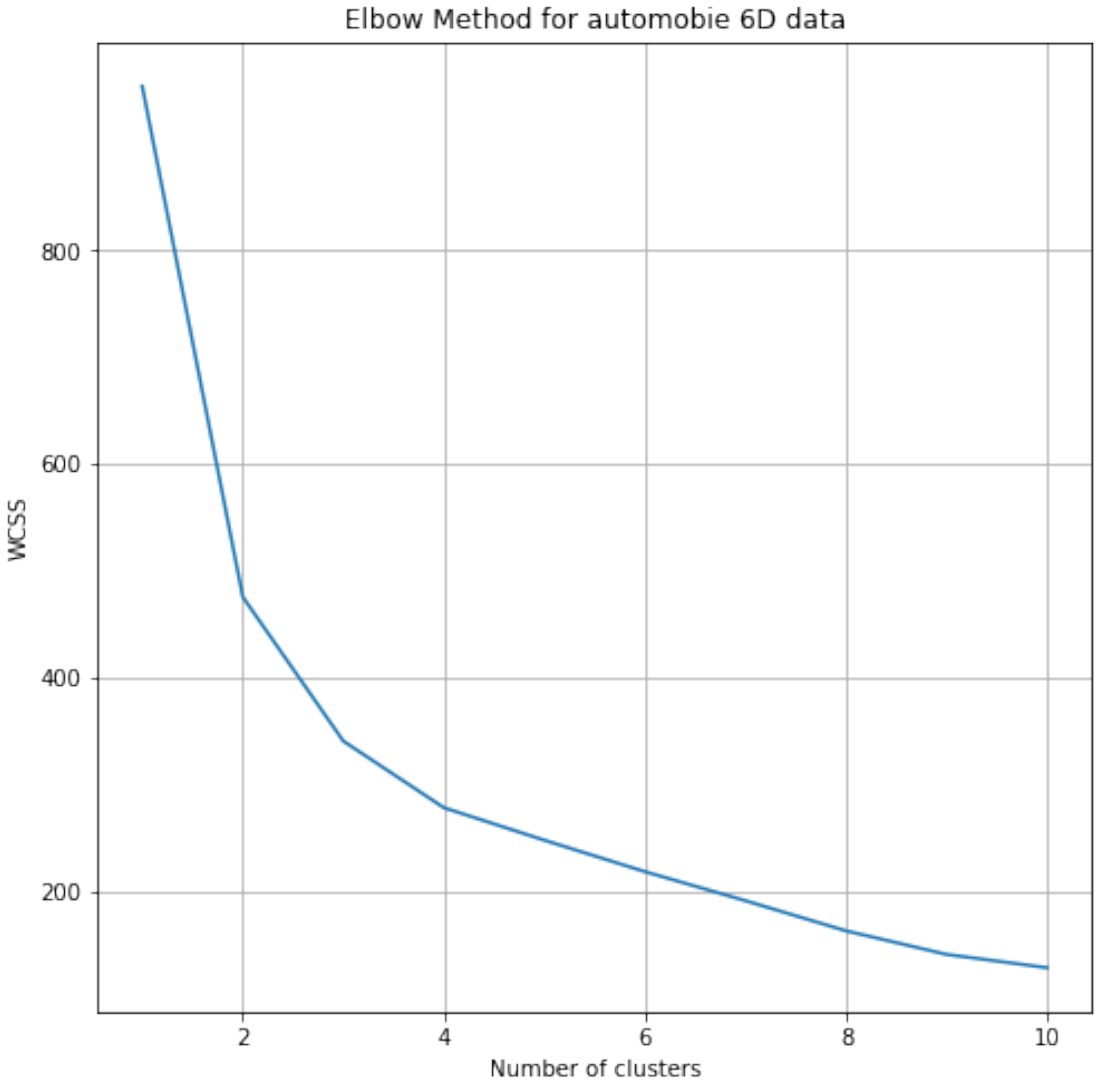
1.2 Discussion about PCA result

This two component now exlain the 6D of original data set. explained_variance_ values state that pca1 is 4.4774846 and pca2 is 0.99515306 which means 99% of data. we can use k-mean cluster as it is fast as we are computing the distances between points and group centers. it complexity is O(n).

```
In [10]: import numpy as np
         import pandas as pd
         from matplotlib import pyplot as plt
         from sklearn.cluster import KMeans
```

1.3 Find the value of k for 6D data

```
In [11]: wcss = []
         for i in range(1, 11):
             kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_st
             kmeans.fit(car_data_6D)
             wcss.append(kmeans.inertia_)
         plt.figure(figsize = (8,8))
         plt.plot(range(1, 11), wcss)
         plt.title('Elbow Method for automobie 6D data')
         plt.xlabel('Number of clusters')
         plt.ylabel('WCSS')
         plt.grid()
         plt.show()
```

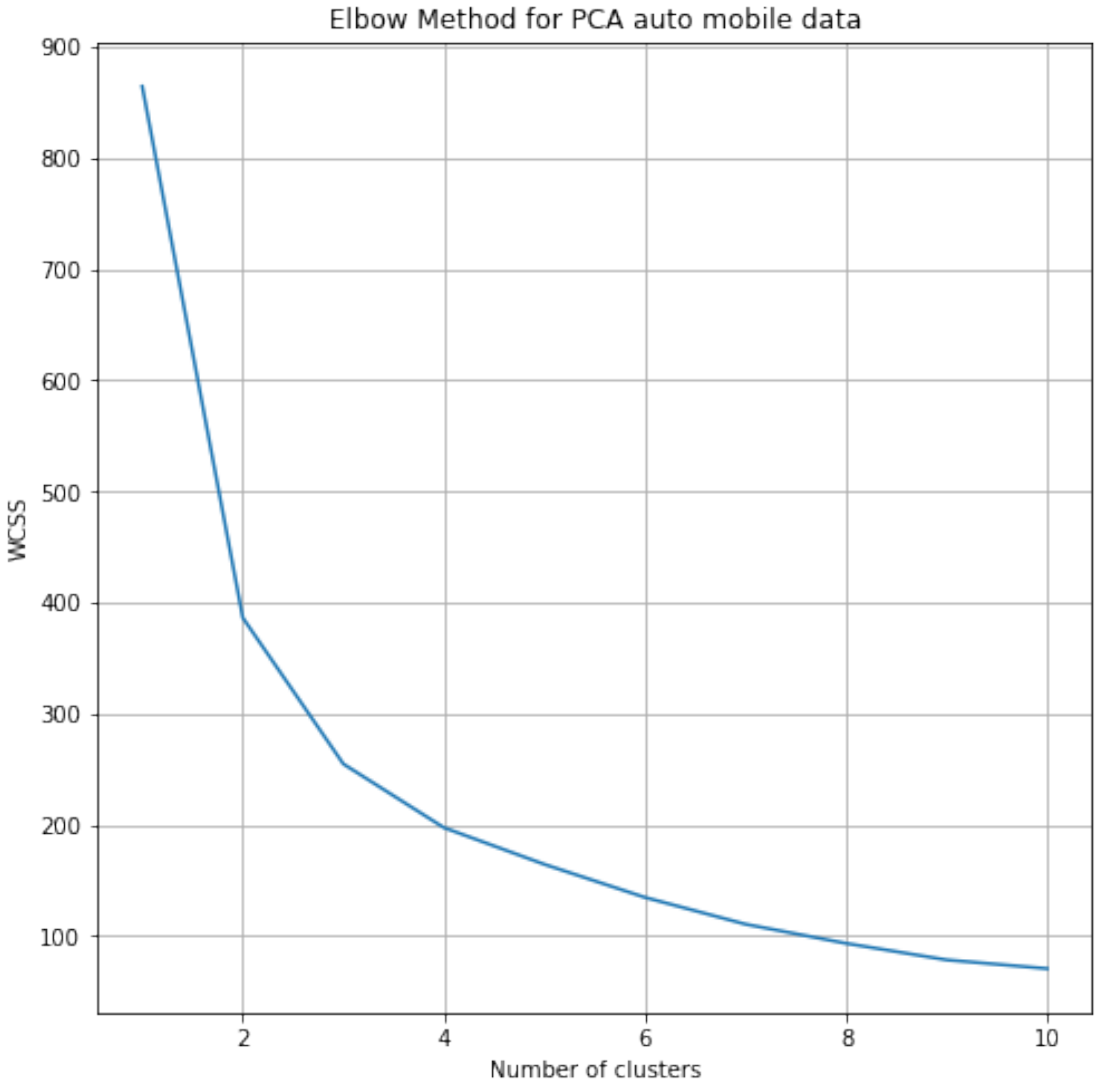


Looking at the above graph. it seems 4 is the right parameters for n_clusters KMeans 6D data.

```
In [12]: kmeans = KMeans(n_clusters=4, max_iter=300, n_init=10, random_state=20)
         pred_y = kmeans.fit_predict(car_data_6D)
         kmeans.inertia_

Out[12]: 277.8551035179315
```

1.4 Find the value of k for PCA data

```
In [13]: wcss = []
         for i in range(1, 11):
             kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_st
             kmeans.fit(pcaDF)
             wcss.append(kmeans.inertia_)
         plt.figure(figsize = (8,8))
         plt.plot(range(1, 11), wcss)
         plt.title('Elbow Method for PCA auto mobile data')
         plt.xlabel('Number of clusters')
         plt.ylabel('WCSS')
         plt.grid()
         plt.show()
```



Looking at the above graph. it seems 4 is the right parameters for n_clusters KMeans PCA data.

1.5 Visualize cluster for PCA 2D data

```
In [14]: kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=
         pred_y = kmeans.fit_predict(pcaDF)
         fig = plt.figure(figsize = (8,8))
         ax = fig.add_subplot(1,1,1)
         ax.set_xlabel('PCA 1', fontsize = 15)
         ax.set_ylabel('PCA 2', fontsize = 15)
         ax.set_title('PCA for Automobile data', fontsize = 20)
         ax.scatter(pcaDF['PCA 1'], pcaDF['PCA 2'])
         ax.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red
```

```
ax.grid()
plt.show()
```