# PCA

August 27, 2021

## 1 Principal Component Analysis

[88]:
```python
import pandas as pd
import numpy as np
#Loading the data
df=pd.read_csv('cars.csv',sep=';',decimal=',')
display(df)
```

```
    Country                         Car   MPG  Weight  Drive_Ratio  Horsepower  \
0      U.S.          Buick Estate Wagon  16.9   4.360         2.73         155
1      U.S.  Ford Country Squire Wagon  15.5   4.054         2.26         142
2      U.S.          Chevy Malibu Wagon  19.2   3.605         2.56         125
3      U.S.     Chrysler LeBaron Wagon  18.5   3.940         2.45         150
4      U.S.                    Chevette  30.0   2.155         3.70          68
5     Japan               Toyota Corona  27.5   2.560         3.05          95
6     Japan                  Datsun 510  27.2   2.300         3.54          97
7      U.S.                  Dodge Omni  30.9   2.230         3.37          75
8   Germany                   Audi 5000  20.3   2.830         3.90         103
9    Sweden                Volvo 240 GL  17.0   3.140         3.50         125
10   Sweden                Saab 99 GLE  21.6   2.795         3.77         115
11   France              Peugeot 694 SL  16.2   3.410         3.58         133
12     U.S.        Buick Century Special  20.6   3.380         2.73         105
13     U.S.              Mercury Zephyr  20.8   3.070         3.08          85
14     U.S.                 Dodge Aspen  18.6   3.620         2.71         110
15     U.S.             AMC Concord D/L  18.1   3.410         2.73         120
16     U.S.        Chevy Caprice Classic  17.0   3.840         2.41         130
17     U.S.                    Ford LTD  17.6   3.725         2.26         129
18     U.S.        Mercury Grand Marquis  16.5   3.955         2.26         138
19     U.S.              Dodge St Regis  18.2   3.830         2.45         135
20     U.S.              Ford Mustang 4  26.5   2.585         3.08          88
21     U.S.           Ford Mustang Ghia  21.9   2.910         3.08         109
22    Japan                   Mazda GLC  34.1   1.975         3.73          65
23    Japan                  Dodge Colt  35.1   1.915         2.97          80
24     U.S.                  AMC Spirit  27.4   2.670         3.08          80
25  Germany                 VW Scirocco  31.5   1.990         3.78          71
26    Japan              Honda Accord LX  29.5   2.135         3.05          68
27     U.S.               Buick Skylark  28.4   2.670         2.53          90
28     U.S.              Chevy Citation  28.8   2.595         2.69         115
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 29 | U.S. | Olds Omega | 26.8 | 2.700 | 2.84 | 115 |
| 30 | U.S. | Pontiac Phoenix | 33.5 | 2.556 | 2.69 | 90 |
| 31 | U.S. | Plymouth Horizon | 34.2 | 2.200 | 3.37 | 70 |
| 32 | Japan | Datsun 210 | 31.8 | 2.020 | 3.70 | 65 |
| 33 | Italy | Fiat Strada | 37.3 | 2.130 | 3.10 | 69 |
| 34 | Germany | VW Dasher | 30.5 | 2.190 | 3.70 | 78 |
| 35 | Japan | Datsun 810 | 22.0 | 2.815 | 3.70 | 97 |
| 36 | Germany | BMW 320i | 21.5 | 2.600 | 3.64 | 110 |
| 37 | Germany | VW Rabbit | 31.9 | 1.925 | 3.78 | 71 |

| | Displacement | Cylinders |
|---|---|---|
| 0 | 350 | 8 |
| 1 | 351 | 8 |
| 2 | 267 | 8 |
| 3 | 360 | 8 |
| 4 | 98 | 4 |
| 5 | 134 | 4 |
| 6 | 119 | 4 |
| 7 | 105 | 4 |
| 8 | 131 | 5 |
| 9 | 163 | 6 |
| 10 | 121 | 4 |
| 11 | 163 | 6 |
| 12 | 231 | 6 |
| 13 | 200 | 6 |
| 14 | 225 | 6 |
| 15 | 258 | 6 |
| 16 | 305 | 8 |
| 17 | 302 | 8 |
| 18 | 351 | 8 |
| 19 | 318 | 8 |
| 20 | 140 | 4 |
| 21 | 171 | 6 |
| 22 | 86 | 4 |
| 23 | 98 | 4 |
| 24 | 121 | 4 |
| 25 | 89 | 4 |
| 26 | 98 | 4 |
| 27 | 151 | 4 |
| 28 | 173 | 6 |
| 29 | 173 | 6 |
| 30 | 151 | 4 |
| 31 | 105 | 4 |
| 32 | 85 | 4 |
| 33 | 91 | 4 |
| 34 | 97 | 4 |
| 35 | 146 | 6 |
| 36 | 121 | 4 |

```
37              89            4
```

[89]:
```
df.set_index(['Country','Car'],inplace=True)
```

### 1.0.1 Pour faire l'analys il faut scandardiser les donnees

[90]:
```
from sklearn.preprocessing import StandardScaler
```

[91]:
```
scaler = StandardScaler()
Z=(scaler.fit(df))
```

[92]:
```
res=scaler.transform(df)
```

### Checking that the data is prepared correctly

[29]:
```
import math
M=np.mean(res.round(3),axis=0)
print('The mean of the standerdised variables Xsc \n', M.round(3))
print('Standart deviation of variables Xsc \n', np.std(res,axis=0,ddof=0))
```

```
The mean of the standerdised variables Xsc
 [ 0. -0.  0. -0. -0. -0.]
Standart deviation of variables Xsc
 [1. 1. 1. 1. 1. 1.]
```

### 1.0.2 Doing the decomposition

[24]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=None)
pca.fit(res)
```

[24]:
```
PCA()
```

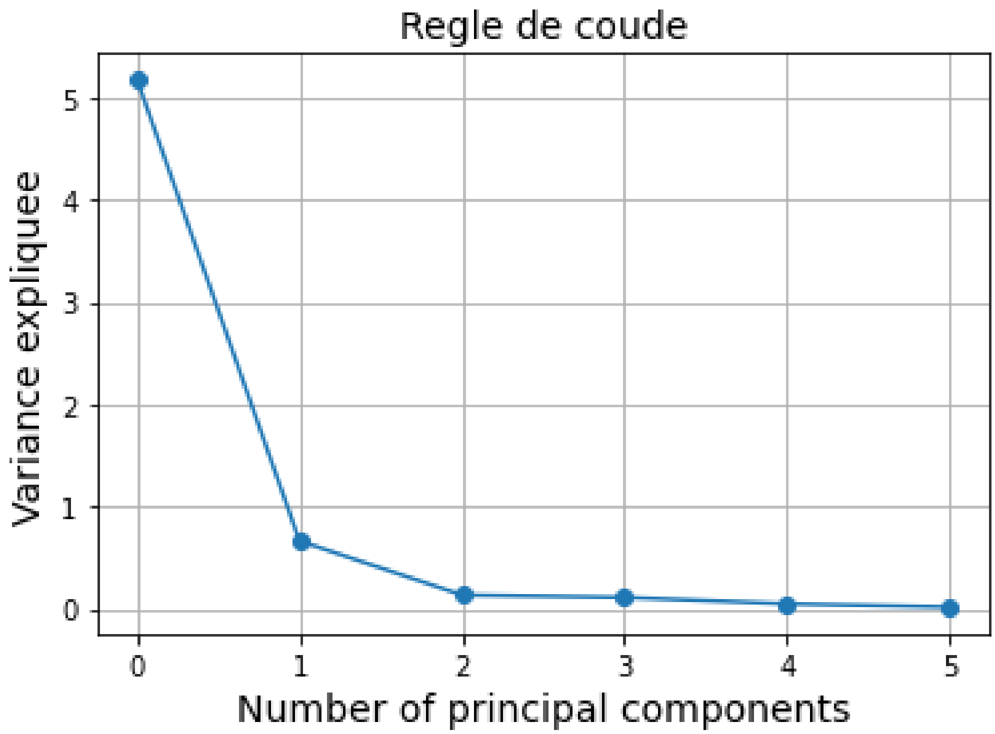
[42]:
```
print('Explained variance ratio: ',pca.explained_variance_ratio_)
coude=pca.explained_variance_
```

```
Explained variance ratio:  [0.83999614 0.10819696 0.0221887  0.01836637
0.00815936 0.00309248]
```

### 1.0.3 Regle de coude

[43]:
```
import matplotlib.pyplot as plt
plt.plot(np.arange(6),coude, marker='o')
plt.title('Regle de coude', fontsize=14)
plt.xlabel('Number of principal components', fontsize=14)
plt.ylabel('Variance expliquee', fontsize=14)
plt.grid(True)
```

```
plt.show()
None
```




### 1.0.4 Singular values

[44]:
```
print(pca.singular_values_)
```

```
[13.83904328  4.96678024  2.24922725  2.046346    1.3639407   0.83969387]
```

[93]:
```
print('number of components =',pca.n_components_)
```

```
number of components = 6
```

### 1.0.5 Matrice de correlation

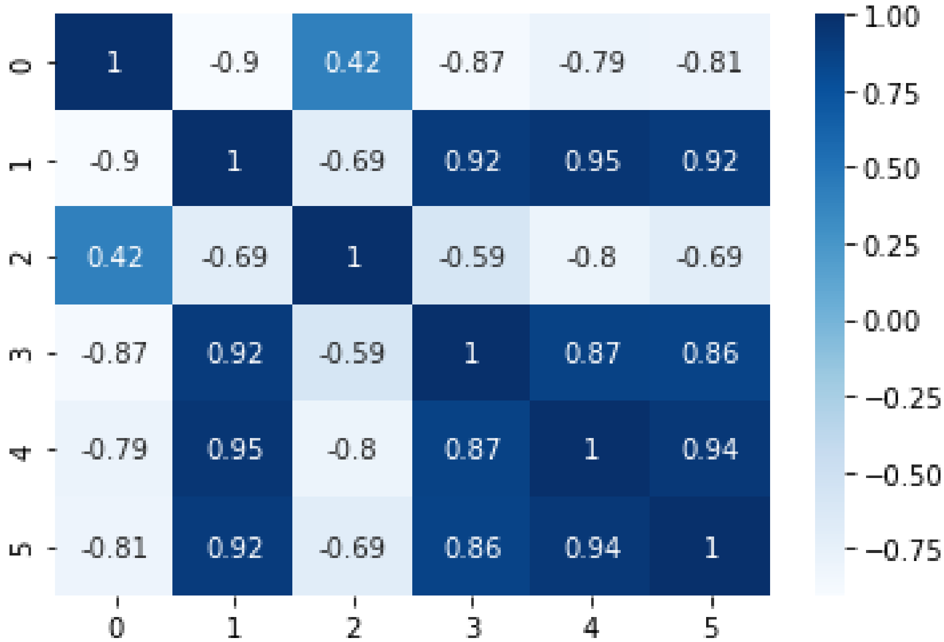
[50]:
```
M=pd.DataFrame(res) #Matrice de correlation des variables#
M.corr()
```

[50]:

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1.000000 | -0.903071 | 0.417225 | -0.871282 | -0.786048 | -0.805511 |
| 1 | -0.903071 | 1.000000 | -0.687880 | 0.917220 | 0.950765 | 0.916678 |
| 2 | 0.417225 | -0.687880 | 1.000000 | -0.588906 | -0.798273 | -0.692150 |
| 3 | -0.871282 | 0.917220 | -0.588906 | 1.000000 | 0.871799 | 0.863847 |

```
4 -0.786048  0.950765 -0.798273  0.871799  1.000000  0.940281
5 -0.805511  0.916678 -0.692150  0.863847  0.940281  1.000000
```

[76]:
```
import seaborn as sb
dataplot = sb.heatmap(M.corr(), cmap="Blues", annot=True)
plt.show()
```




### 1.0.6 Valuers propres (diagonalisation):

[72]:
```
eig_vals, eig_vecs = np.linalg.eig(M.corr())
eig_vals
```

[72]:
```
array([5.03997681, 0.64918174, 0.01855489, 0.04895616, 0.11019821,
       0.13313219])
```

[94]:
```
print('combien composants a garder (nombre des valuers qui sont > 1) = 1 \n ',␣
 ↪eig_vals*((df.shape[0]-1)/df.shape[0]))
```

```
combien composants a garder (nombre des valuers qui sont > 1) = 1
  [4.90734584 0.63209801 0.0180666  0.04766784 0.10729826 0.12962871]
```

## 1.1 Valuers propres a partir de singular values

[75]:
```
print(pca.singular_values_**2/df.shape[0])
```

```
[5.03997681 0.64918174 0.13313219 0.11019821 0.04895616 0.01855489]
```

[52]:

```python
Summa=sum(eig_vals)
print('La somee de valeurs propres de la matrice de correlation = ',Summa)
```

```
La somee de valeurs propres de la matrice de correlation =  5.999999999999997
```

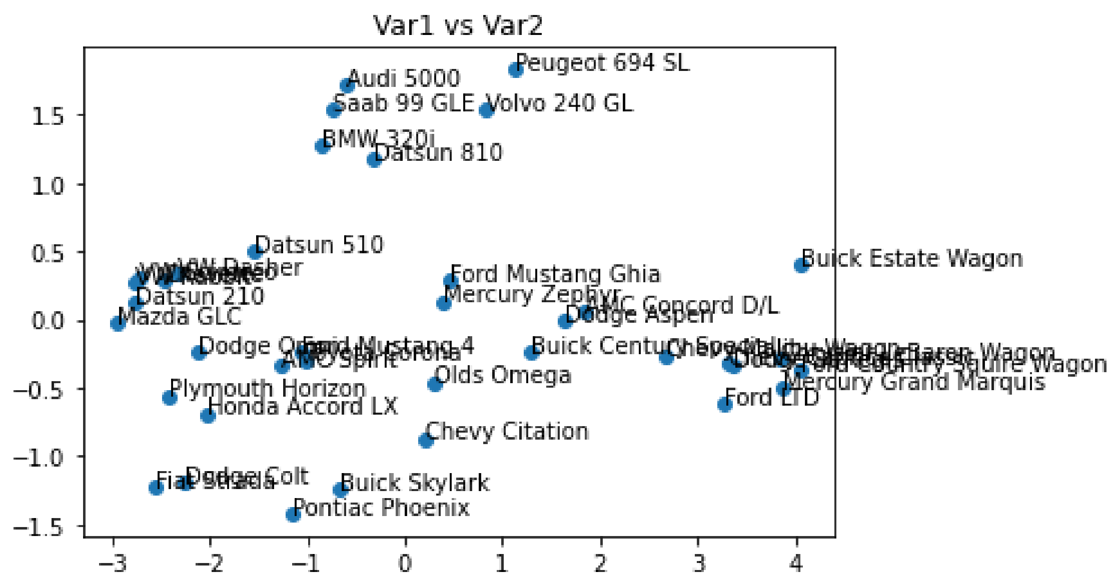
### 1.1.1 Projection des individus sur les 2 premieres composantes principales

[58]:

```python
coords=pca.fit_transform(res)
```

[116]:

```python
plt.scatter(coords[:,0],coords[:,1])
plt.title('Var1 vs Var2')
for i in range(df.shape[0]):
    plt.annotate(df.index[i][1],(coords[i,0],coords[i,1]))
None
```




[ ]: