HW9 Report

인대영 (2022311154)
2022-05-12

1. After defining functions for finding the mean vector, covariance matrix, and sample correlation matrix, respectively, we read 'stock.dat' and save it as 'stock'.

(a) We first obtain the sample correlation matrix of 'stock' and save it as 'data'.

In [4]:
```
#(a) Perform a factor analysis using principal component method on the sample correlation matrix.
#Correlation matrix
data = sample_corr_matrix(stock)
data
```

Out[4]:

	AC	DP	UC	EX	TX
AC	1.000000	0.576924	0.508656	0.386721	0.462178
DP	0.576924	1.000000	0.598384	0.389519	0.321953
UC	0.508656	0.598384	1.000000	0.436101	0.425627
EX	0.386721	0.389519	0.436101	1.000000	0.523529
TX	0.462178	0.321953	0.425627	0.523529	1.000000

We use the spectral decomposition of Σ as

$$\Sigma = \underset{\sim}{e} \Lambda \underset{\sim}{e}' = \lambda_1 \underset{\sim}{e_1} \underset{\sim}{e_1}' + \lambda_2 \underset{\sim}{e_2} \underset{\sim}{e_2}' + \cdots + \lambda_p \underset{\sim}{e_p} \underset{\sim}{e_p}'$$

$$= \underset{\sim}{e} \Lambda^{1/2} \Lambda^{1/2} \underset{\sim}{e}'$$

$$= \left[\sqrt{\lambda_1}\, \underset{\sim}{e_1} \; \vdots \; \cdots \; \vdots \; \sqrt{\lambda_p}\, \underset{\sim}{e_p} \right] \begin{bmatrix} \sqrt{\lambda_1}\, \underset{\sim}{e_1}' \\ \vdots \\ \sqrt{\lambda_p}\, \underset{\sim}{e_p}' \end{bmatrix}_{p \times p}$$

By ignoring the contribution of eigenvalues that are small, we can obtain the approximation

$$\Sigma \simeq \left[\sqrt{\lambda_1}\, \underset{\sim}{e_1} \; \vdots \; \cdots \; \vdots \; \sqrt{\lambda_m}\, \underset{\sim}{e_m} \right] \begin{bmatrix} \sqrt{\lambda_1}\, \underset{\sim}{e_1}' \\ \vdots \\ \sqrt{\lambda_m}\, \underset{\sim}{e_m}' \end{bmatrix}_{m \times p} = L_{p \times m} L'_{m \times p} \qquad (m < p)$$

We obtain the eigenvalues and eigenvectors from the sample correlation matrix 'data' and perform a factor analysis using Principal Component method. The corresponding code and results are shown below.

In [8]:
```
#sorted eigenvalues
index=eigval.argsort()[::-1]
eigval=eigval[index]
eigval
```

Out[8]:
```
array([2.85648688, 0.8091185 , 0.54004398, 0.45134682, 0.34300382])
```

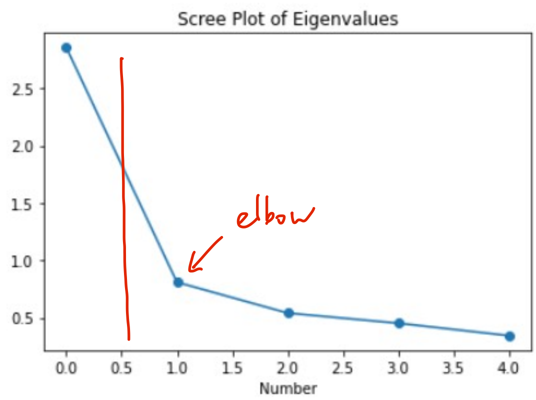
In [9]:
```
anal = pd.DataFrame()
anal['Eigenvalue'] = eigval
anal['Proportion'] = anal['Eigenvalue'] / len(anal)
anal['Cumulative'] = anal['Proportion'].cumsum(axis=0)
anal
```

Out[9]:

	Eigenvalue	Proportion	Cumulative
0	2.856487	0.571297	0.571297
1	0.809118	0.161824	0.733121
2	0.540044	0.108009	0.841130
3	0.451347	0.090269	0.931399
4	0.343004	0.068601	1.000000

(b) We draw a scree plot using the eigenvalues.

```
In [10]: #(b) How many factors are required to describe adequately the space in which these data actually fall?
#scree plot
plt.title('Scree Plot of Eigenvalues')
plt.xlabel('Number')
plt.plot(eigval, 'o-')
plt.show()
```



We can see that the elbow is at index = 1. Thus, only one factor is required to adequately describe the space in which the data actually falls. In addition, we may also make this selection based on the rule for choosing factors with eigenvalue ≥ 1.

(c) Since we choose Factor 1 only, the factor loading matrix can be computed as shown

```
In [11]: #(c) Obtain the factor loading matrix using Principal Component method.
loadings = pd.DataFrame()
loadings['Factor1'] = np.sqrt(eigval[0])*eigvec[:,0]
loadings.index = stock.columns
np.round(loadings,2)
```

Out[11]:

	Factor1
AC	0.78
DP	0.77
UC	0.79
EX	0.71
TX	0.71

(d) The communality is the sum of squares of the rows in the factor loading matrix.

$h_i^2 = l_{i1}^2 + \cdots + l_{im}^2$

```
In [12]: #(d) Obtain the communality of each variable
loadsquare = loadings**2
pd.DataFrame(loadsquare.sum(axis=1), columns = ['Communality'])
```

Out[12]:

	Communality
AC	0.613773
DP	0.596774
UC	0.630945
EX	0.507916
TX	0.507079

(e) From our results, we can say that the underlying characteristics of Factor 1 are all five variables. However, there is not a clear distinction. We may need to use other methods or rotation to get a better picture of the underlying characteristics of each factor.

(f) Assuming 'factor_analyzer' package is installed, we use the package to obtain the factor loading matrix using minimum residual method. Here we choose Factor 2 as well since its corresponding eigenvalue is close to 1. Also, we would like to check whether rotation can help us better identify the underlying characteristics. For this purpose, we choose n_factors = 2.

In [13]:
```
#(f) Obtain factor (before rotation) loading matrix using Minres method.
from factor_analyzer import FactorAnalyzer
```

In [14]:
```
fa = FactorAnalyzer(n_factors=2, rotation=None, method='minres', is_corr_matrix=True)
fa.fit(data)
```

Out[14]:
```
FactorAnalyzer(is_corr_matrix=True, n_factors=2, rotation=None,
               rotation_kwargs={})
```

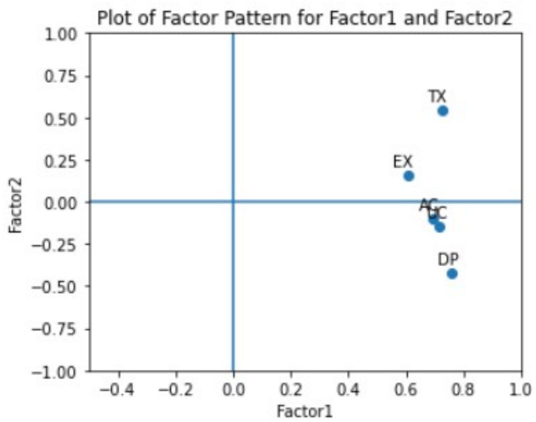
In [15]:
```
#Factor loadings before rotation
loadings = pd.DataFrame(fa.loadings_,
                        index = stock.columns,
                        columns = ['Factor1', 'Factor2'])
loadings
```

Out[15]:

	Factor1	Factor2
AC	0.696010	-0.096709
DP	0.758582	-0.418575
UC	0.715361	-0.146355
EX	0.606299	0.156562
TX	0.723501	0.545413

Plotting the factor pattern, we obtain the following.

In [16]:
```
#Preplot(Before Rotation)
x = loadings.Factor1 ; y = loadings.Factor2
plt.figure(figsize = (5,4))
plt.title('Plot of Factor Pattern for Factor1 and Factor2')
plt.xlabel('Factor1') ; plt.ylabel('Factor2')
plt.scatter(x,y)
for i in range(len(loadings)):
    plt.text(x[i]-0.05, y[i]+0.05, loadings.index[i])
plt.axvline(x = 0) ; plt.axhline(y = 0)
plt.xlim(-0.5,1); plt.ylim(-1,1)
plt.show()
```



Variables 'AC' and 'UC' are closer to Factor 1. However, it is difficult to distinguish which factor the rest of the variables belong to. Rotation may be useful.

(g) We apply Varimax rotation which finds the orthogonal transformation T that maximizes

$$V = \frac{1}{p}\sum_{j=1}^{m}\left[\sum_{i=1}^{p}\tilde{l}_{ij}^{*4} - \frac{\left(\sum_{i=1}^{p}\tilde{l}_{ij}^{*2}\right)^2}{p}\right]$$

Similarly, we obtain the factor loading matrix after rotation.

In [17]:
```
#(g) Obtain factor (after rotation) Loading matrix using Minres method.
# Rotation Method: Varimax
fa = FactorAnalyzer(n_factors=2, rotation='varimax', method='minres', is_corr_matrix=True)
fa.fit(data)
```

Out[17]:
```
FactorAnalyzer(is_corr_matrix=True, n_factors=2, rotation='varimax',
               rotation_kwargs={})
```

In [18]:
```
# Orthogonal Transformation Matrix
fa.rotation_matrix_
```

Out[18]:
```
array([[ 0.76844413,  0.63991689],
       [-0.63991689,  0.76844413]])
```

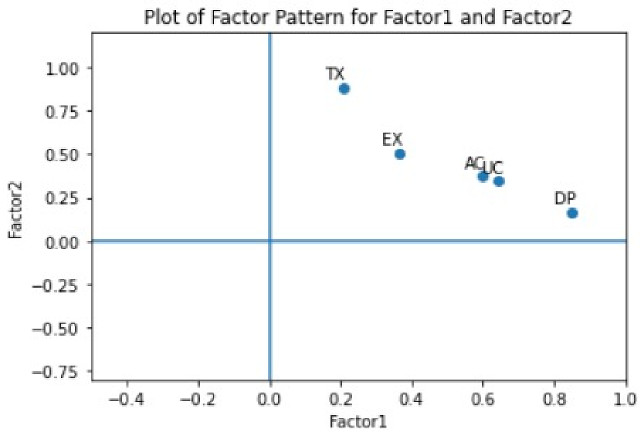
In [19]:
```
# Rotated Factor Pattern
Rloadings = pd.DataFrame(fa.loadings_,
            index = stock.columns,
            columns = ['Factor1', 'Factor2'])
Rloadings
```

Out[19]:

	Factor1	Factor2
AC	0.596731	0.371073
DP	0.850781	0.163778
UC	0.643369	0.345306
EX	0.365720	0.508290
TX	0.206951	0.882100

We plot the rotated factor pattern as below.

In [20]:
```
#Plot(After Rotation)
x = Rloadings.Factor1 ; y = Rloadings.Factor2
plt.figure(figsize = (6,4))
plt.title('Plot of Factor Pattern for Factor1 and Factor2')
plt.xlabel('Factor1') ; plt.ylabel('Factor2')
plt.scatter(x,y)
for i in range(len(loadings)):
    plt.text(x[i]-0.05, y[i]+0.05, Rloadings.index[i])
plt.axvline(x = 0) ; plt.axhline(y = 0)
plt.xlim(-0.5,1); plt.ylim(-0.8,1.2)
plt.show()
```



(h) We obtain the communalities (after rotation) using the following code.

In [21]:
```python
#(h) Obtain the communalities (after rotation) of each variable.
pd.DataFrame(fa.get_communalities(),
             index=stock.columns,
             columns=['Communality'])
```

Out[21]:

	Communality
AC	0.493783
DP	0.750651
UC	0.533161
EX	0.392110
TX	0.820929

(i) Based on the analysis using minimum residual method after rotation, it is clear that Factor 1 accounts for the variables 'AC', 'DP', and 'UC', whereas, Factor 2 accounts for the variables 'EX' and 'TX'. This demonstrates how rotation can be useful for interpreting factor analysis.