```
# Import required libraries
import pandas as pd
from sklearn.datasets import load_iris
from factor_analyzer import FactorAnalyzer
import matplotlib.pyplot as plt

# Loading Data
# Data Link:
df= pd.read_csv('bfi.csv')

# Preprocess Data
df.columns

# Dropping unnecessary columns
df.drop(['gender', 'education', 'age'],axis=1,inplace=True)

# Dropping missing values rows
df.dropna(inplace=True)

df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2436 entries, 0 to 2799
Data columns (total 26 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   rownames  2436 non-null   int64
 1   A1        2436 non-null   float64
 2   A2        2436 non-null   float64
 3   A3        2436 non-null   float64
```

4 A4 2436 non-null float64

5 A5 2436 non-null float64

6 C1 2436 non-null float64

7 C2 2436 non-null float64

8 C3 2436 non-null float64

9 C4 2436 non-null float64

10 C5 2436 non-null float64

11 E1 2436 non-null float64

12 E2 2436 non-null float64

13 E3 2436 non-null float64

14 E4 2436 non-null float64

15 E5 2436 non-null float64

16 N1 2436 non-null float64

17 N2 2436 non-null float64

18 N3 2436 non-null float64

19 N4 2436 non-null float64

...

24 O4 2436 non-null float64

25 O5 2436 non-null float64

dtypes: float64(24), int64(2)

memory usage: 513.8 KB

	rownames	A1	A2	A3	A4	A5	C1	C2	C3	C4	...	N1	N2	N3	N4	N5	O1	O2	O3	O4	O5
0	61617	2.0	4.0	3.0	4.0	4.0	2.0	3.0	3.0	4.0	...	3.0	4.0	2.0	2.0	3.0	3.0	6	3.0	4.0	3.0
1	61618	2.0	4.0	5.0	2.0	5.0	5.0	4.0	4.0	3.0	...	3.0	3.0	3.0	5.0	5.0	4.0	2	4.0	3.0	3.0
2	61620	5.0	4.0	5.0	4.0	4.0	4.0	5.0	4.0	2.0	...	4.0	5.0	4.0	2.0	3.0	4.0	2	5.0	5.0	2.0
3	61621	4.0	4.0	6.0	5.0	5.0	4.0	4.0	3.0	5.0	...	2.0	5.0	2.0	4.0	1.0	3.0	3	4.0	3.0	5.0
4	61622	2.0	3.0	3.0	4.0	5.0	4.0	4.0	5.0	3.0	...	2.0	3.0	4.0	4.0	3.0	3.0	3	4.0	3.0	3.0

5 rows × 26 columns

Adequacy Test

Before you perform factor analysis, you need to evaluate the “factorability” of our dataset. Factorability means "can we find the factors in the dataset?". There are two methods to check the factorability or sampling adequacy:

Bartlett’s Test

Kaiser-Meyer-Olkin Test

Bartlett’s test of sphericity checks whether or not the observed variables intercorrelate at all using the observed correlation matrix against the identity matrix. If the test found statistically insignificant, you should not employ a factor analysis.

In this Bartlett ’s test, the p-value is less than .05. The test is statistically significant, indicating that the observed correlation matrix is not an identity matrix.

Kaiser-Meyer-Olkin (KMO) Test measures the suitability of data for factor analysis. It determines the adequacy for each observed variable and for the complete model. KMO estimates the proportion of variance among all the observed variable. Lower proportion id more suitable for factor analysis. KMO values range between 0 and 1. Value of KMO less than 0.6 is considered inadequate.

from factor_analyzer.factor_analyzer import calculate_kmo

kmo_all,kmo_model=calculate_kmo(df)

kmo_model

Here, The overall KMO for our data is 0.84, which is excellent. This value indicates that you can proceed with your planned factor analysis.

Choosing the number of Factors

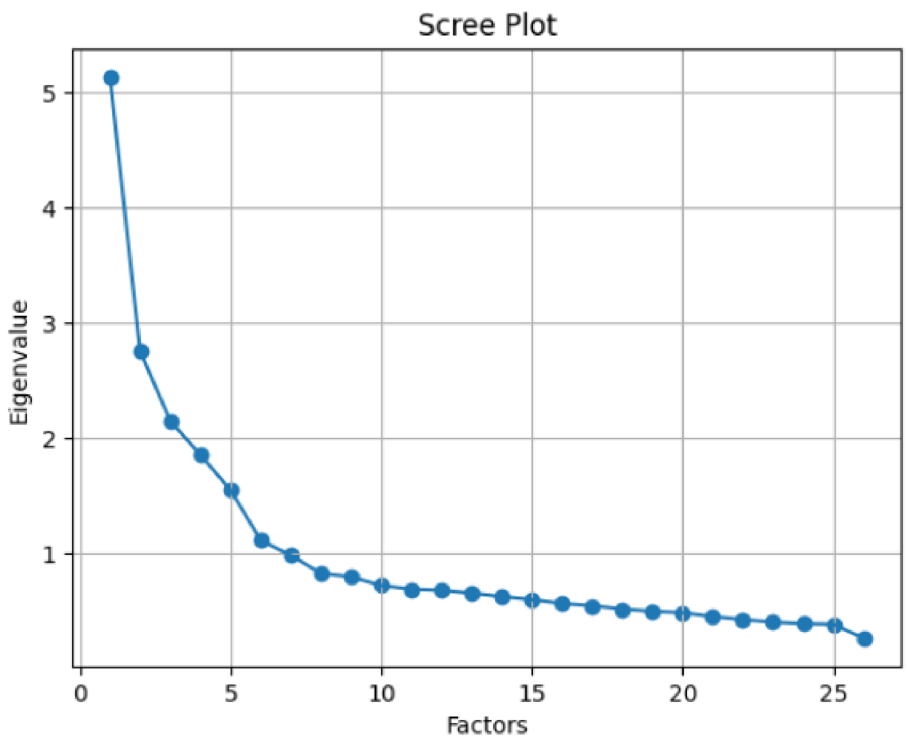
```
# Create factor analysis object and perform factor analysis
fa = FactorAnalyzer()
fa.fit(df, 25)
# Check Eigenvalues
ev, v = fa.get_eigenvalues()
ev
arr = fa.loadings_
arr
```

```
array([[-0.07131035, -0.01751926,  0.05405798],
       [-0.23219058,  0.08334477,  0.03458871],
       [ 0.5413533 ,  0.06211685,  0.04667964],
       [ 0.64586903,  0.04331579,  0.00192803],
       [ 0.3910012 , -0.0592843 ,  0.07552545],
       [ 0.64743433, -0.08504573, -0.01610074],
       [-0.012761  ,  0.08019221,  0.61189813],
       [-0.00668553,  0.14317342,  0.63664785],
       [-0.00354385, -0.00201999,  0.48602329],
       [ 0.05511348,  0.18756638, -0.63069868],
       [-0.05185813,  0.26189667, -0.46496583],
       [-0.53560024,  0.02486522,  0.09320605],
       [-0.57445533,  0.2160911 , -0.01194907],
       [ 0.6069387 ,  0.09092885,  0.0968013 ],
       [ 0.69550544, -0.10590156, -0.06884823],
       [ 0.4264341 ,  0.09512995,  0.30777364],
       [ 0.00397076,  0.74125646, -0.06354794],
       [-0.02255664,  0.74077167, -0.01052292],
```

```
[ 0.01980718,  0.74281971, -0.04895261],
[-0.1871232 ,  0.58791351, -0.08352484],
[-0.0122797 ,  0.50640962, -0.09478277],
[ 0.20715947,  0.08634594,  0.28898683],
[ 0.06626166,  0.09010407, -0.30154905],
[ 0.33053068,  0.12238437,  0.26646912],
[-0.01618707,  0.27125849,  0.14424164],
[-0.01671831, -0.00619755, -0.27851083]])
```

```
# Create Scree Plot using Matplotlib

plt.scatter(range(1,df.shape[1]+1),ev)
plt.plot(range(1,df.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
```



The scree plot method draws a straight line for each factor and its eigenvalues. Number eigenvalues greater than one considered as the number of factors.

Here, you can see only for 6-factors eigenvalues are greater than one. It means we need to choose only 6 factors (or unobserved variables).

 # Performing Factor Analysis

 # Create factor analysis object and perform factor analysis

fa = FactorAnalyzer(rotation='varimax')

fa.fit(df, 6)

FactorAnalyzer

```
▾ FactorAnalyzer
FactorAnalyzer(rotation='varimax', rotation_kwargs={})
```

```
import numpy as np
# Perform Factor Loadings
arr = fa.loadings_
arr
column_labels = np.array(["F1","F2","F3"])

# Creating array with column labels
arr_with_labels = np.vstack([column_labels, arr])
arr_with_labels
# Factor 1 has high factor loadings for E1,E2,E3,E4, and E5 (Extraversion)
```

```
array([['F1', 'F2', 'F3'],
       ['-0.05960933859466298', '-0.016013157910446676',
        '0.04276284605681199'],
       ['-0.22784874820728498', '0.09997988713870168',
        '-0.008324835759040543'],
       ['0.5363832566418344', '0.0111799850521527',
        '0.12820554734824247'],
       ['0.6326550228442914', '-0.01264580399164724',
        '0.10214863747542524'],
       ['0.4015212124152065', '-0.09928206918176838',
        '0.1401586239715859'],
       ['0.6395332627609799', '-0.1390389045700315',
        '0.09277602529736546'],
       ['0.08817428380928646', '0.027677865400728297',
        '0.5932123896884189'],
       ['0.09428496212052201', '0.08771919379740255',
        '0.6145042289254644'],
       ['0.0808510463269269', '-0.044026052882193384',
        '0.4763611260525772'],
```

```
       ['-0.06743230394551543', '0.23695830363782117',
        '-0.6217255430327657'],
       ['-0.14882021017417613', '0.3057598231851436',
        '-0.48086368099037313'],
       ['-0.5122247360781315', '0.06276498136297182',
        '0.004477865750030332'],
...
       ['0.3631559561838855', '0.0702125639731392', '0.3064234302970274'],
       ['-0.008812274988349923', '0.2589655685755964',
        '0.1218811911652352'],
       ['-0.06428469303282659', '0.019518596428927802',
        '-0.2755013209044167']], dtype='<U32')
```

The Matrix presents factor loadings for each variable (feature) on each factor (latent variable). Positive values represent positive relationship. Negative values represent negative relationship.

[7]:

 # Get variance of each factors

fa.get_factor_variance()

 # The get_factor_variance() method calculates the variance of each factor in the factor analysis model.

 # This can be useful for understanding how much of the total variance in the observed variables is explained by each factor.

 # The output of this method will likely be a table or array showing the variance of each factor.

[7]:

(array([3.27939442, 2.67445616, 2.24249104]),

, array([0.12613055, 0.1028637 , 0.08624966]),

, array([0.12613055, 0.22899425, 0.31524391]))