SSE-DP-2025-2

統計的多変量解析
―Python による実装―

岡本雅子

統計数理研究所

2025 年 3 月

SSE-DP（ディスカッションペーパー・シリーズ）は以下のサイトから無料で入手可能です。
https://stat-expert.ism.ac.jp/training/discussionpaper/

SSE-DP-2025-2

A Multivariate Statistical Analysis —An implementation with Python—

Masako Okamoto

The Institute of Statistical Mathematics

March 2025

This paper provides Python implementations of various methods for statistical multivariate analysis for beginners in statistics and computer programming. You can learn how to implement the methods in Python by examining the Python programs and results. In addition, some comments on the Python output results are included when there are points to be noted, which should be used as a reference when interpreting the results of the analysis.

多変量解析

―Python による実装―

統計的多変量解析は、非常に複雑な計算が必要なため、重回帰分析を除いて手計算や Excel を用いて計算することは非常に困難です。そのため、一般的に R や Python、有料の統計解析ソフトを用いることが多い。本稿では、統計学およびプログラミングの初学者を対象として、統計的多変量解析に係る各手法について、Python で実装を紹介している。以下に紹介している Python のプログラムと実行結果を確認しながら、Python での実装方法を学習することができる。また、Python の出力結果について、留意すべき点がある場合はコメントを入れており、分析結果を解釈する際の参考にしていただきたい。

目次

● 回帰分析

(1)賃貸マンションデータセットを用いた回帰分析の前処理

* Python のスクリプト（ファイルの読み込み）

```
# 文字化け防止のため、フォントの表示を支援するライブラリを読み込む
import japanize_matplotlib
%matplotlib inline

# pandas（パンダス）を読み込んで、pd という名前で使用する
import pandas as pd

# excel に変換したデータの読み込み
file_path = 'room.xlsx'

# Pandas の ExcelFile の関数で[room.xlsx]にあるシート名を読み込む
sheet_names = pd.ExcelFile(file_path).sheet_names

# Excel ファイルにある各シートの番号と名前を出力する
print("Excel ファイルにあるシート:")
for i, sheet_name in enumerate(sheet_names):
    print(i, sheet_name)
```

※賃貸マンションのデータセット「room.xlsx」は、「日本統計学会公式認定　統計検定２級対応『統計学基礎』」のダウンロード用データから入手可能である。

http://www.tokyo-tosho.co.jp/books/ISBN978-4-489-02122-0.html

（ダウンロード用データ→「第１章」→「本文」→「room.csv」）

* 出力結果

Excel ファイルにあるシート：

0 Sheet1

【出力結果】Excel ファイルにあるシートの番号と名前
※Python では番号は「0」から開始される

* Python のスクリプト（単回帰）

シート番号の指定

```
# データフレームの読み込み
data = pd.read_excel(file_path, sheet_name=sheet_names[1])
data
```

* 出力結果

	近さ	家賃	大きさ	築年数
0	B	68000	19	12
1	B	68000	19	12
2	B	69000	19	14
3	B	70000	19	14
4	B	72000	15	9
...	...	...	...	...
135	B	145000	40	8
136	B	145000	54	28
137	A	148000	42	13
138	A	148000	42	13
139	B	150000	41	5

140 rows × 4 columns

* Python のスクリプト（ダミー変数への変換）

ここでは、比較的近い（A）を「0」、比較的遠い(B）を「1」と数値化する。

```
# scikit-learn の LabelEncoder を用いる
from sklearn.preprocessing import LabelEncoder

#  「近さ」の数値化（ダミー変数への変換）
a = LabelEncoder()
data['近さ'] = a.fit_transform(data['近さ'])
data
```

* 出力結果

	近さ	家賃	大きさ	築年数
0	1	68000	19	12
1	1	68000	19	12
2	1	69000	19	14
3	1	70000	19	14
4	1	72000	15	9
...	...	...	...	...
135	1	145000	40	8
136	1	145000	54	28
137	0	148000	42	13
138	0	148000	42	13
139	1	150000	41	5

140 rows × 4 columns

(2) 賃貸マンションデータセットの相関と散布図の確認

* Python のスクリプト（相関）

```
data.corr()
```

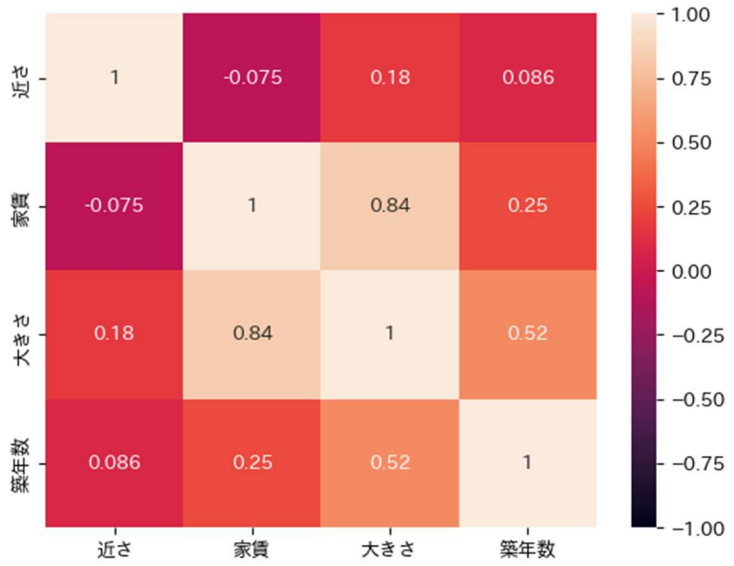
* 出力結果

	近さ	家賃	大きさ	築年数
近さ	1.000000	-0.075279	0.178161	0.086390
家賃	-0.075279	1.000000	0.841193	0.245452
大きさ	0.178161	0.841193	1.000000	0.515945
築年数	0.086390	0.245452	0.515945	1.000000

* Python のスクリプト（ヒートマップ）

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.heatmap(data.corr(), vmin=-1, vmax=1, annot=True)
plt.show()
```

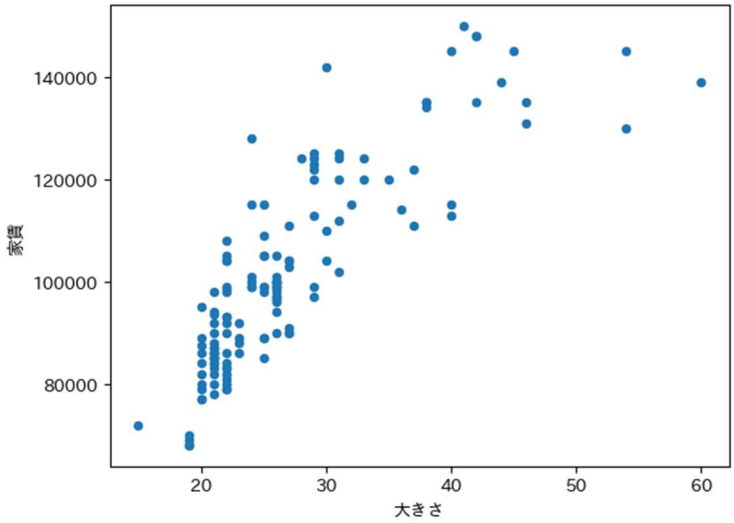
* 出力結果



* Python のスクリプト（散布図）

```
data.plot.scatter(x='大きさ', y='家賃')
```

* 出力結果



(3) 賃貸マンションデータセットの回帰分析の実施

* Python のスクリプト（単回帰分析：説明変数：部屋の大きさ）

```
# statsmodel をインポート
import statsmodels.formula.api as smf

formula = '家賃 ~ 大きさ'
model1 = smf.ols(formula, data=data)
result1 = model1.fit()

print(result1.summary())
```

回帰式の指定「'被説明変数 ~ 定数項以外の説明変数」

最小二乗法を指定

結果の表示

* 出力結果

表 1：OLS Regression Results

目的変数

最小二乗法

Dep. Variable:	家賃	R-squared:	0.708
Model:	OLS	Adj. R-squared:	0.705
Method:	Least Squares	F-statistic:	334.0
Date:	Fri, 16 Feb 2024	Prob (F-statistic):	1.14e-38
Time:	08:00:18	Log-Likelihood:	-1493.4
No. Observations:	140	AIC:	2991.
Df Residuals:	138	BIC:	2997.
Df Model:	1		
Covariance Type:	nonrobust		

【出力結果】OLS 推定の基本的な情報

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4.579e+04	3179.976	14.400	0.000	3.95e+04	5.21e+04
大きさ	2075.1455	113.553	18.275	0.000	1850.617	2299.674

【出力結果】主な推定結果

Omnibus:	6.374	Durbin-Watson:	1.308
Prob(Omnibus):	0.041	Jarque-Bera (JB):	6.691
Skew:	0.352	Prob(JB):	0.0352
Kurtosis:	3.808	Cond. No.	101.

【出力結果】様々な検定などに関する数値

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

今回は、目的変数を「家賃」、説明変数を「大きさ」として、Python の「Statsmodels」の OLS（Ordinary Least Squares）を使用して単回帰分析を行った。その結果、次のような回帰式が得られた（表 1 参照）。

$y = 4.579e+04 + 2075.15 * 大きさ$

回帰係数は、切片が 4.579e+04、傾きが 2075.15 であった。また、決定係数（R-squared）

は 0.71 であった。加えて、F 検定（F-stat: 334.0, Prob: 1.14e-38）が十分に大きく、有意確率(p-value)も十分に小さいので、モデル全体として概ね妥当であると判断できる。

* Python のスクリプト（単回帰分析：説明変数：築年数）

```
formula = '家賃 ~ 築年数'
model2 = smf.ols(formula, data=data)
result2 = model2.fit()

print(result2.summary())
```

* 出力結果

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   家賃   R-squared:                       0.060
Model:                            OLS   Adj. R-squared:                  0.053
Method:                 Least Squares   F-statistic:                     8.847
Date:                Tue, 19 Dec 2023   Prob (F-statistic):            0.00347
Time:                        01:47:48   Log-Likelihood:                -1575.1
No. Observations:                 140   AIC:                             3154.
Df Residuals:                     138   BIC:                             3160.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   9.347e+04   3164.758     29.533      0.000    8.72e+04    9.97e+04
築年数          852.4835    286.608      2.974      0.003     285.773    1419.194
==============================================================================
Omnibus:                        4.156   Durbin-Watson:                   0.093
Prob(Omnibus):                  0.125   Jarque-Bera (JB):                4.187
Skew:                           0.415   Prob(JB):                        0.123
Kurtosis:                       2.829   Cond. No.                         22.2
==============================================================================
```

今回は、目的変数を「家賃」、説明変数を「築年数」として、単回帰分析を行った結果、

$y = 9.347e+04 + 852.5 *$ 築年数

という回帰式が得られた。

* Python のスクリプト（単回帰分析：説明変数：近さ）

駅からの近さが 2 値の質的変数であるということである。比較的近い(A)を 0 とし、比較的遠い(B)を 1 と数値化し分析する。

```
formula = '家賃 ~ 築年数'
model3 = smf.ols(formula, data=data)
result3 = model3.fit()

print(result3.summary())
```

* 出力結果

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     家賃   R-squared:                       0.006
Model:                            OLS   Adj. R-squared:                 -0.002
Method:                 Least Squares   F-statistic:                    0.7865
Date:                Tue, 19 Dec 2023   Prob (F-statistic):              0.377
Time:                        01:50:02   Log-Likelihood:                -1579.1
No. Observations:                 140   AIC:                             3162.
Df Residuals:                     138   BIC:                             3168.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   1.031e+05   2322.592     44.381      0.000    9.85e+04    1.08e+05
近さ         -2892.3862  3261.428     -0.887      0.377   -9341.220    3556.447
==============================================================================
Omnibus:                       11.367   Durbin-Watson:                   0.010
Prob(Omnibus):                  0.003   Jarque-Bera (JB):               12.606
Skew:                           0.730   Prob(JB):                      0.00183
Kurtosis:                       2.824   Cond. No.                         2.64
==============================================================================
```

今回は、目的変数を「家賃」、説明変数を「近さ」として、単回帰分析を行った結果、

$y = 1.031e+045 + (-2892.4) * 近さ$

という回帰式が得られた。

* Python のスクリプト（重回帰分析：大きさと築年数）

部屋の大きさと築年数を説明変数として、重回帰分析する。

```
formula = '家賃 ~ 大きさ + 築年数'
model4 = smf.ols(formula, data=data)
result4 = model4.fit()

print(result4.summary())
```

* 出力結果

OLS Regression Results

Dep. Variable:	家賃	R-squared:	0.756
Model:	OLS	Adj. R-squared:	0.752
Method:	Least Squares	F-statistic:	212.3
Date:	Tue, 19 Dec 2023	Prob (F-statistic):	1.07e-42
Time:	02:00:50	Log-Likelihood:	-1480.7
No. Observations:	140	AIC:	2967.
Df Residuals:	137	BIC:	2976.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4.552e+04	2915.615	15.613	0.000	3.98e+04	5.13e+04
大きさ	2402.2011	121.520	19.768	0.000	2161.903	2642.499
築年数	-892.4534	171.086	-5.216	0.000	-1230.765	-554.142

Omnibus:	6.066	Durbin-Watson:	1.469
Prob(Omnibus):	0.048	Jarque-Bera (JB):	8.993
Skew:	0.137	Prob(JB):	0.0111
Kurtosis:	4.211	Cond. No.	107.

今回は、目的変数を「家賃」、説明変数を「大きさ」と「築年数」として、重回帰分析を行った結果、

$y = 4.552e+04 + 2402.2 * 大きさ + (-892.5) * 築年数$

という回帰式が得られた。

部屋の大きさに対する回帰係数は築年数の影響で単回帰分析のモデルとは異なる。築年数の回帰係数が負であることから、同じ大きさなら古いほど家賃が安くなることがわかる。また、いずれの t 値の絶対値も大きく十分に有意である。決定係数は 0.752 とモデル 1 より大きい。Y の変動のおおよそ 75%がこのモデルで説明できていることがわかる。

なお、説明変数の数が異なる場合は、決定係数ではなく、自由度調整済決定係数で比較することが望ましい。

* 【補足 Scikit-learn の場合】Python のスクリプト（単回帰）

```
# scikit-learn をインポート
from sklearn.linear_model import LinearRegression

# 説明変数と目的変数の設定
x = data[['大きさ']]
y = data[['家賃']]

model = LinearRegression()
model.fit (x,y)

# 説明変数の係数を出力
print('coefficient = ', model.coef_)

# 切片を出力
print('intercept = ', model.intercept_)
```

結果の表示
（必要な情報を指定する）

* 出力結果

coefficient = [[2075.14548924]]

intercept = [45791.44348218]

(4) 回帰直線の出力（散布図に回帰直線を引くには？）

* Python のスクリプト

```
# 散布図に回帰直線を引くライブラリ
import seaborn as sns

sns.lmplot(x='大きさ', y='家賃', data=data, fit_reg=True, height=7,
line_kws={'linewidth':1, 'color':'red'})
```

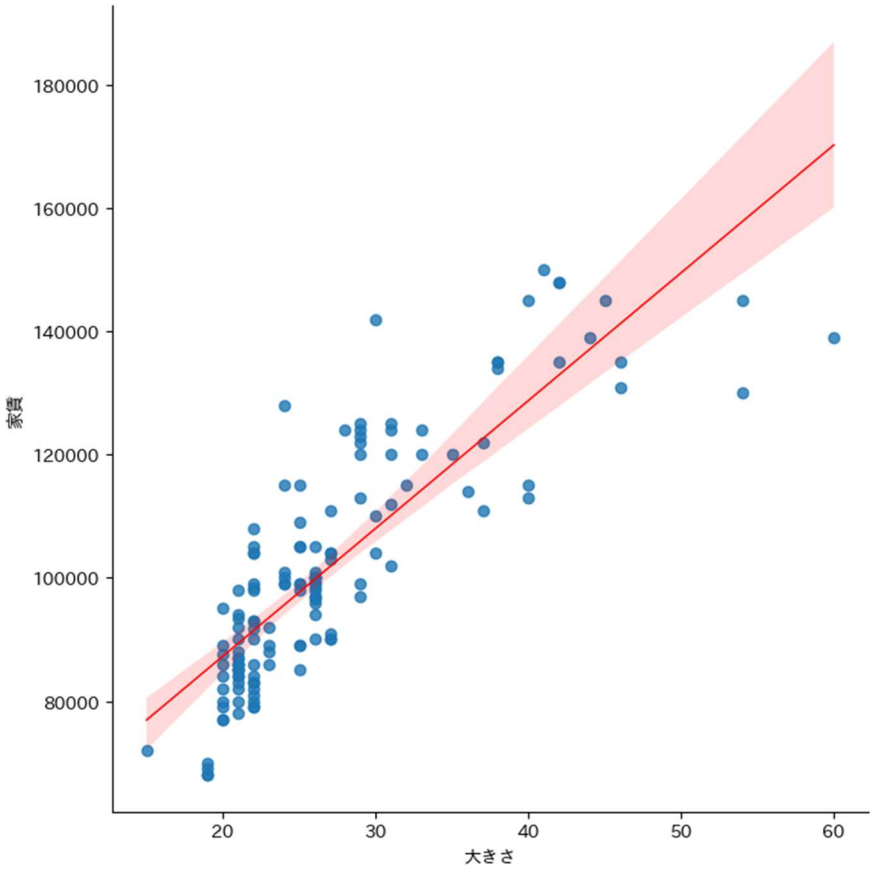
1 行で続けて入力する

回帰直線の設定

色の指定

次に、散布図と回帰直線を出力した。横軸は「大きさ」、縦軸は「家賃」であり、赤の直線が回帰直線で、薄い赤色は信頼区間を示している。大きさ（面積）が大きくなるとともに、家賃が上がっている。

* 出力結果



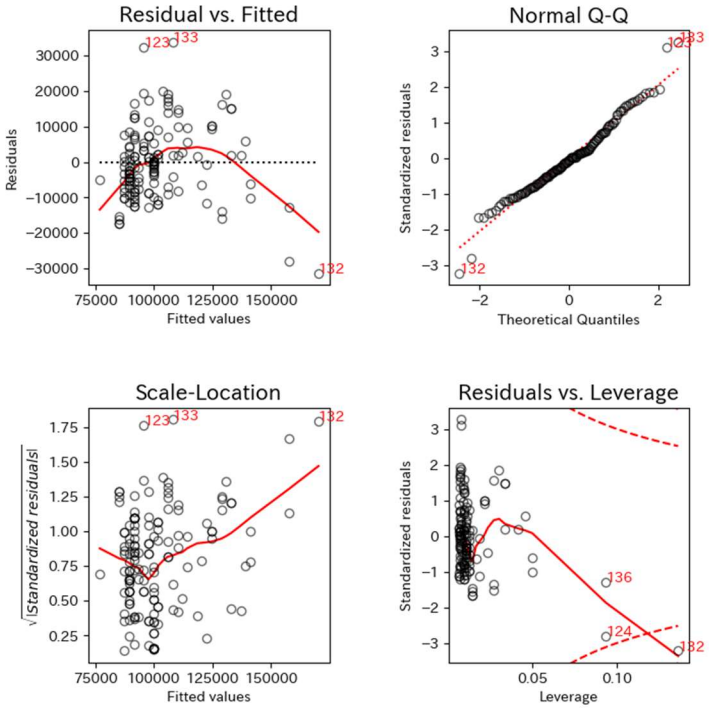
(4)回帰診断出力

* Python のスクリプト

```
# lmdiag で回帰診断
import lmdiag

plt.figure(figsize=(7, 7))
lmdiag.plot(result1)
```

最後に、回帰診断を実施した（出力結果参照）。右上の「残差の Q-Q プロット」から、今回の結果は概ね正規分布であると考えられる。他方、右下の「L-R プロット」では、Cook の距離が 0.5 を超えるものがあるため（132 番）、外れ値として再度分析してもよいかもしれない。

* 出力結果：回帰診断



- **判別分析**

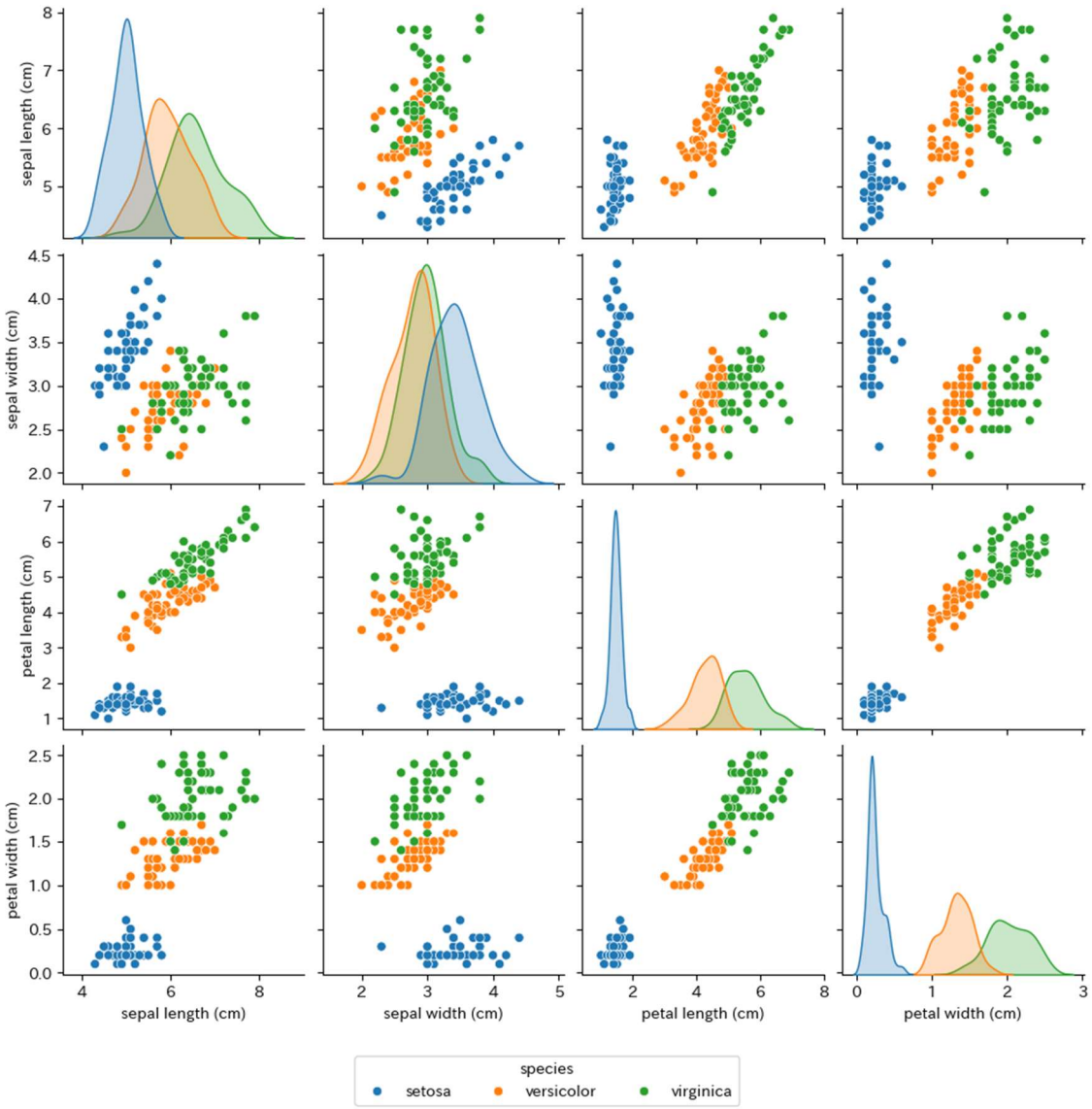
(1) iris データセットを用いた散布図行列（3 種類で色を変える）

（本分析や以下の分析で用いる iris データセットは、Python の「scikit-learn」に用意されている。）

* Python のスクリプト

```
import pandas as pd
import seaborn as sns

# iris データセットの読み込み
from sklearn.datasets import load_iris

iris = load_iris()

# 散布図行列の描画
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target_names[iris.target]

# 3 種類で色を変える
g = sns.pairplot(df, hue = 'species')
g._legend.remove()
g.fig.legend(
      handles = g._legend_data.values(),
      labels = g._legend_data.keys(),
      loc = 'lower center', ncol = 3,
      title = 'species')
g.fig.subplots_adjust(bottom = 0.12)
```

* 出力結果：iris データを用いた散布図行列



iris データセットは、

- sepal length（ガクの長さ）
- sepal width（ガクの幅）
- petal length（花弁の長さ）
- petal width（花の幅）

という 4 つの属性を持つ 3 種類のアヤメの花（Setosa、Versicolour、Virginica）から構成されている。それぞれの属性（特徴量）の分布がどのように関係しているのかを可視化するため、Python の「seaborn」ライブラリを使用し、散布図行列を作成した（出力結果参照）。なお、出力結果の「iris データを用いた散布図行列」では、iris データセットの品種を区別するため、species column で色分けをしている。setosa（青）は他の種と比べてガク

も花弁の形質も大きく異なることがわかり、分離が良いことがわかる。一方、versicolor（橙）と virginica（緑）の分離はやや曖昧になっている。

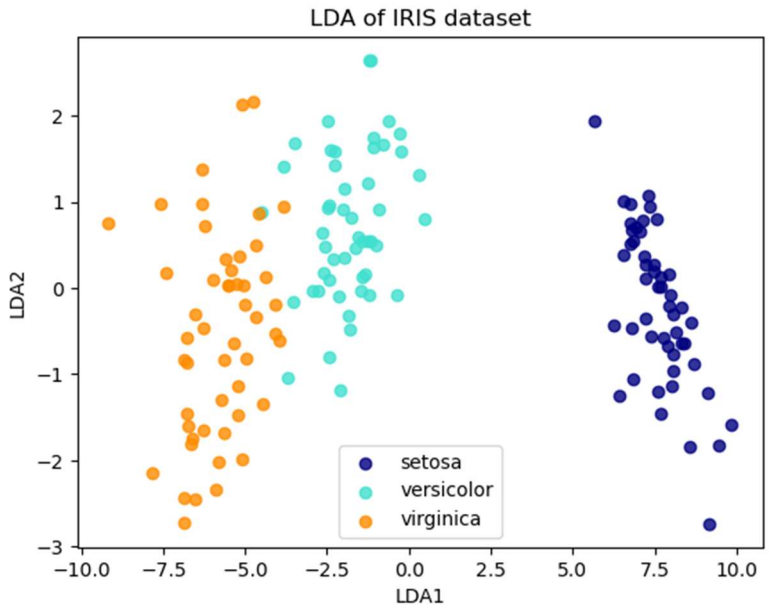
(2) iris データセットを用いた線形判別分析（Linear Discriminant Analysis）

* Python のスクリプト

```
import matplotlib.pyplot as plt

# 組み込みデータ iris 読み込み
from sklearn.datasets import load_iris

# 線形判別分析（Linear Discriminant Analysis）を行う
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

iris = datasets.load_iris()

X = iris.data
y = iris.target
target_names = iris.target_names

lda = LinearDiscriminantAnalysis(n_components=2)
X_r = lda.fit(X, y).transform(X)

plt.figure()
colors = ["navy", "turquoise", "darkorange"]
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(
        X_r[y == i, 0], X_r[y == i, 1], alpha=0.8, color=color, label=target_name
    )
plt.legend(loc="best", shadow=False, scatterpoints=1)
plt.xlabel('LDA1')
plt.ylabel('LDA2')
plt.title("LDA of IRIS dataset")

plt.show()
```

ここでは、iris データセットを Python の「scikit-learn」の「LinearDiscriminantAnalsyis」関数を用いて判別分析した。出力結果から、iris データセット内の 3 つの異なる種がどの程度、分離されているかを視覚化することができた。しかしながら、「versicolor」と「virginica」は距離が近いのでご判別される可能性がある。

* 出力結果：線形判別分析（Linear Discriminant Analysis）



なお、Python は R と異なり、線形判別分析結果を自動で表示する機能がないため、必要に応じて以下のように手動でスクリプトを作成することが求められる。今回の線形判別式は以下の通りである。

$$z = 0.83x_1 + 1.53x_2 - 2.20x_3 - 2.81x_4 - 4.38$$

z の正負により判別することが可能となる（今回は省略）。

* Python のスクリプト

```
import numpy as np

print('係数', lda.scalings_)

# グループの平均と係数の線形結合の平均が定数値
cv = np.mean(np.dot(lda.means_, lda.scalings_))
print('定数値', cv)
```

* 出力結果

```
係数 [[ 0.82937764 -0.02410215]
 [ 1.53447307 -2.16452123]
 [-2.20121166  0.93192121]
 [-2.81046031 -2.83918785]]
定数値 -4.383289492902386
```

- **主成分分析(PCA)**

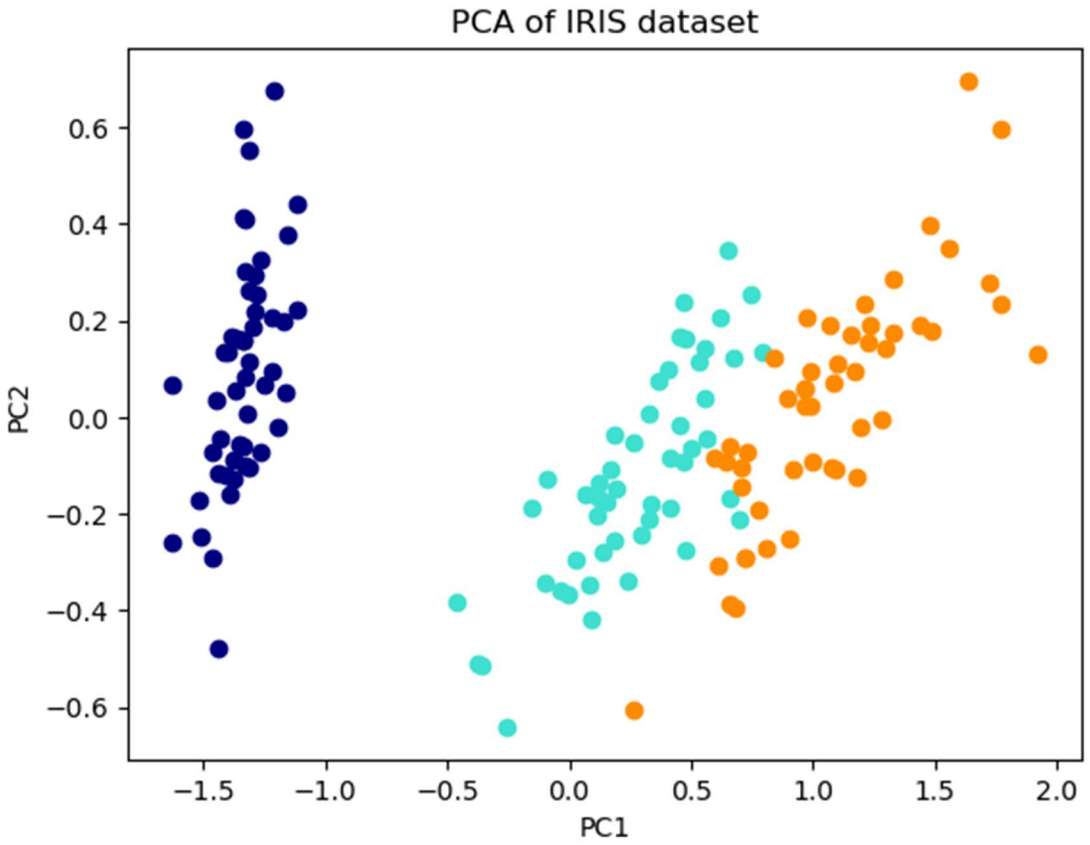
(1)iris データセットを用いた主成分分析

* Python のスクリプト

```
from __future__ import division
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

# 組み込みデータ iris 読み込み
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
Y = iris.target

# 正規化
X_scaled = (X-X.mean())/X.std()

# PCA
pca = PCA()
pca.fit(X_scaled)
reduced_iris = pca.fit_transform(X_scaled)

colors = ["navy", "turquoise", "darkorange"]

# 重複したクラスを削除
uniqueY = pd.unique(Y)
for i in range(len(uniqueY)):
    Yi = uniqueY[i]
    color = colors[i]
    plt.scatter(reduced_iris[np.where(Y == Yi),0], reduced_iris[np.where(Y == Yi),1], c =
color)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title("PCA of IRIS dataset")

plt.show()
```

iris データセットについて、Python の「sklearn.decomposition」を用いて主成分分析（PCA）を試みた。出力結果は、横軸が第一主成分、縦軸が第二主成分である。

* 出力結果：主成分分析結果



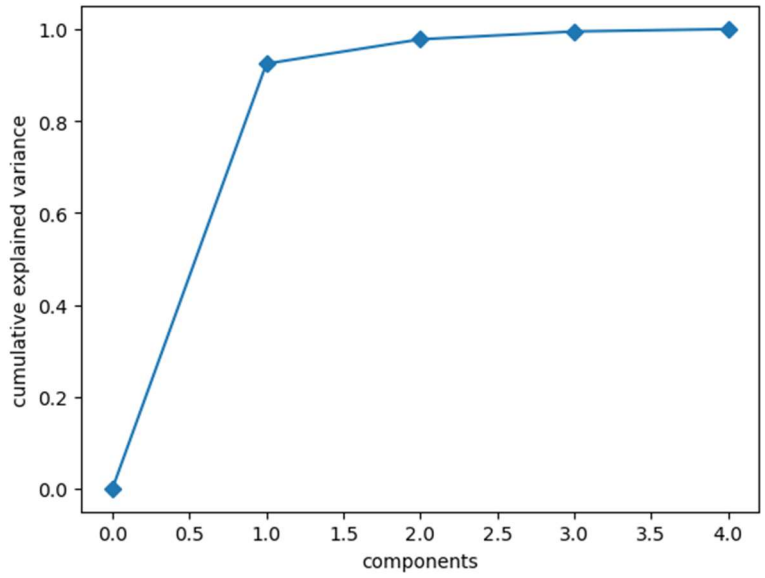
次に、各主成分への寄与率と累積寄与率を求めるため、以下のように追加のスクリプトを実行した。

* Python のスクリプト

```
# NumPy の ndarray オブジェクトを pandas の DataFrame オブジェクトに変換
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
X_scaled_pd = pd.DataFrame(X_scaled)

# 寄与率を求める
pca_col = ["PC{}".format(x + 1) for x in range(len(X_scaled_pd.columns))]
df_con_ratio = pd.DataFrame([pca.explained_variance_ratio_], columns = pca_col)
print(df_con_ratio.head())

# 累積寄与率を図示する
cum_con_ratio = np.hstack([0, pca.explained_variance_ratio_]).cumsum()
plt.plot(cum_con_ratio, 'D-')
plt.xlabel('components')
plt.ylabel('cumulative explained variance')
plt.show()
```

* 出力結果：累積寄与率

	PC1	PC2	PC3	PC4
0	0.924619	0.053066	0.017103	0.005212



上述の出力結果から、第一主成分は 92.5%、第二主成分は 5.3%であった。つまり、第一主成分と第二主成分までで 97.8%の情報が説明できている。さらに累積寄与率をプロットして確認した（図 6 参照)。一般的には，累積寄与率が 80%以上になる主成分数を採用して分析結果に用いることが多いため、第三主成分および第四主成分は削除しても影響は小さいと言える。

続いて、主成分負荷量（各主成分に対する各変数の影響度合い）を求め、第 1 主成分と第 2 主成分の主成分負荷量をプロットした散布図を作成した。

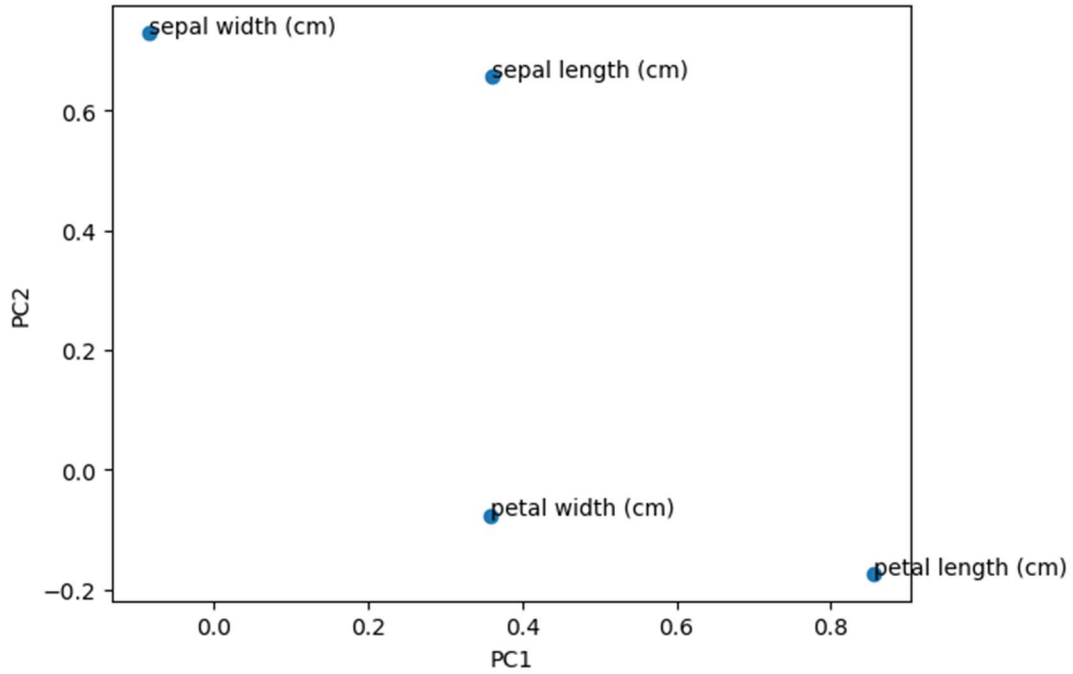
* Python のスクリプト

```
# 主成分負荷量を求める
df_pca = pd.DataFrame(reduced_iris, columns = pca_col)
df_pca_vec = pd.DataFrame(pca.components_, columns=iris_df.columns,
                          index=["PC{}".format(x + 1) for x in
range(len(df_pca.columns))])
print(df_pca_vec)

# 主成分負荷量を図示する
plt.figure()
for x, y, name in zip(pca.components_[0], pca.components_[1], iris_df.columns[0:]):
    plt.text(x, y, name)
plt.scatter(pca.components_[0], pca.components_[1])
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

* 出力結果

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
PC1	0.361387	-0.084523	0.856671	0.358289
PC2	0.656589	0.730161	-0.173373	-0.075481
PC3	-0.582030	0.597911	0.076236	0.545831
PC4	-0.315487	0.319723	0.479839	-0.753657

* 出力結果：主成分負荷量の散布図



出力結果から、第一主成分に対して主成分負荷量が大きい項目は「petal length (cm)」であることがわかった。第二主成分に対して主成分負荷量が大きい項目は、「sepal width (cm)」と「sepal length (cm)」である。

- **因子分析**

(1)iris データセットについて、因子数の決定（固有値の計算）

* Python のスクリプト

```
# ライブラリのインポート
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import japanize_matplotlib
%matplotlib inline
from factor_analyzer import FactorAnalyzer

# データセット読み込み
iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)

# 標準化
df_workers_std = iris_df.apply(lambda x: (x-x.mean())/x.std(), axis=0)

# 固有値を求める
ei = np.linalg.eigvals(iris_df.corr())
print(ei)
```

* 出力結果

[2.91849782 0.91403047 0.14675688 0.02071484]

ここでは、因子数を決定するために、読み込んだ説明変数の相関行列の固有値を求めた。今回は因子数を 2 とする。

(2) 因子分析（オブリミン回転）の実施

Python の iris データセットについて、Python の「FactorAnalyzer」を用いて因子分析（オブリミン回転）を試みた。今回は、因子数は 2、軸の回転方法はオブリミンに設定した。

* Python のスクリプト

```
# 因子分析（oblimin）の実行
fa = FactorAnalyzer(n_factors=2, rotation="oblimin")
fa.fit(df_workers_std)

# 因子負荷量，共通性
loadings_df = pd.DataFrame(fa.loadings_, columns=["第 1 因子", "第 2 因子"])
loadings_df.index = iris_df.columns
loadings_df["共通性"] = fa.get_communalities()
print(loadings_df)

# 因子負荷量の二乗和，寄与率，累積寄与率
var = fa.get_factor_variance()
df_var = pd.DataFrame(list(zip(var[0], var[1], var[2])),
                              index=["第 1 因子", "第 2 因子"],
                              columns=["因子負荷量の二乗和", "寄与率", "累積寄与率"])
print(df_var.T)

# バイプロットの作図
score = fa.transform(df_workers_std)
coeff = fa.loadings_.T
fa1 = 0
fa2 = 1
labels = iris_df.columns
annotations = iris_df.index
xs = score[:, fa1]
ys = score[:, fa2]
n = score.shape[1]
scalex = 1.0 / (xs.max() - xs.min())
scaley = 1.0 / (ys.max() - ys.min())
X = xs * scalex
Y = ys * scaley
for i, label in enumerate(annotations):
      plt.annotate(label, (X[i], Y[i]))
for j in range(coeff.shape[1]):
      plt.arrow(0, 0, coeff[fa1, j], coeff[fa2, j], color='r', alpha=0.5,
                  head_width=0.03, head_length=0.015)
      plt.text(coeff[fa1, j] * 1.15, coeff[fa2, j] * 1.15, labels[j], color='r',
                 ha='center', va='center')
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.xlabel("第 1 因子")
plt.ylabel("第 2 因子")
plt.grid()
plt.show()
```

* 出力結果

	第１因子	第２因子	共通性
sepal length (cm)	0.939580	0.181840	0.915876
sepal width (cm)	-0.018438	0.991866	0.984139
petal length (cm)	0.962714	-0.119362	0.941066
petal width (cm)	0.925447	-0.074021	0.861932

	第１因子	第２因子
因子負荷量の二乗和	2.666422	1.036591
寄与率	0.666605	0.259148
累積寄与率	0.666605	0.925753

* 出力結果：バイプロット

因子分析の結果として、因子負荷量と共通性を計算し、出力した（出力結果参照）。因子負荷量は、各因子に関係性が深い変数ほど値が大きくなり-1 から 1 の間の値を取る。第一因子は、「sepal length (cm)」「petal length (cm)」「petal width (cm)」の情報を多く含んでいる。第二因子は「sepal width (cm)」の因子負荷量が最も大きいため、「sepal width (cm)」と関連性が深い因子と考えられる。

また、共通性は各変数の持つ情報が因子モデルにどれだけ反映されているかを示しており、今回は各共通性の値が高いため、因子モデルに情報が反映されていると推察される。

次に、因子分析で計算した各因子に対して、寄与率と累積寄与率を求めた。第一因子の虚率は 66.66%、第ニ因子の寄与率は 25.91%であり、合わせて 92.57%であった。

最後に、第一因子を横軸に第ニ因子を縦軸にしたバイプロットを作成した。バイプロットは、各因子の因子負荷量のベクトル表記と因子得点をプロットした図である。バイプロットから、第一因子に強く相関がある変数は「sepal length (cm)」「petal length (cm)」「petal width (cm)」であることがわかる。

- **クラスター分析**

(1) iris データセットについて、因子数の決定（固有値の計算）

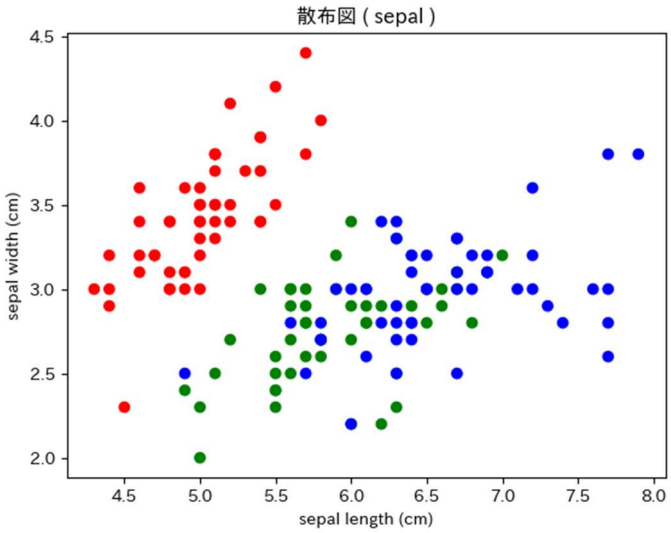
今回は、iris データセットについて、Python による非階層型クラスター分析（k-means 法）を試みた。

iris データセットは、4 つの説明変数を持っている。まずはデータの状態を知るために、「sepal length (cm)」を横軸、「sepal width (cm)」を縦軸において散布図を作成した。

* Python のスクリプト

```
# ライブラリのインポート
import pandas as pd
from sklearn.datasets import load_iris

# データを読み pandas の DataFrame 型に変換
iris = load_iris()

# DataFrame に変換
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['label'] = iris.target_names[iris.target]

rgb = np.array(['r','g','b'])
# がく片の長さ，幅で散布図をプロット
plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'],c= rgb[iris.target])
plt.title('散布図（sepal）')
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.show()
```

* 出力結果：散布図（sepal）

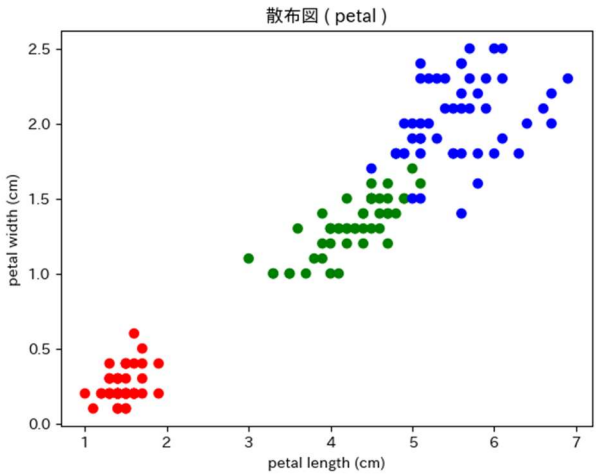


続いて、「petal length (cm)」を横軸、「petal width (cm)」を縦軸に散布図を作成した。

* Python のスクリプト

```
# 花びらの長さ，幅で散布図をプロット
plt.scatter(df['petal length (cm)'], df['petal width (cm)'],c= rgb[iris.target])
plt.title('散布図（ petal )')
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)')
plt.show()
```

* 出力結果：散布図（ petal)

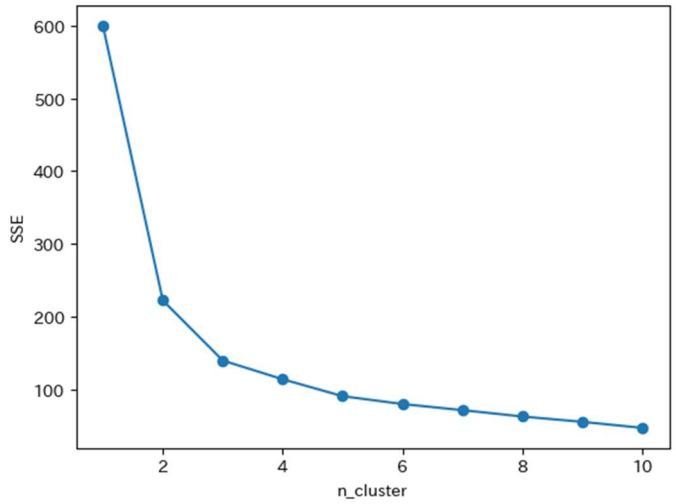


ここからクラスタリングをするが、k-means 法は最適なクラスタ数を先に決定する必要があるため、今回はエルボー法を使用して最適なクラスタ数を決定することにした。

* Python のスクリプト

```
# エルボー法による最適なクラスタ数の決定
from sklearn.cluster import KMeans
from sklearn import preprocessing
import matplotlib.pyplot as plt

# データを標準化
df_ss = df.copy()
ss = preprocessing.StandardScaler()
df_ss.iloc[:, :4] = ss.fit_transform(df_ss.iloc[:, :4].values)

sse_dict = {}
for n_cluster in range(1, 11):
    # パラメータ設定＆学習
    kmeans = KMeans(n_clusters=n_cluster, random_state=0)
    kmeans.fit(df_ss.iloc[:, :4].values)
    # sse を取得
    sse_dict[n_cluster] = kmeans.inertia_

# sse をグラフに描画
fig, ax = plt.subplots(1, 1)
ax.plot(sse_dict.keys(), sse_dict.values(), marker='o')
ax.set_xlabel("n_cluster")
ax.set_ylabel("SSE")
fig.show()
```

* 出力結果：クラスタ数ごとの SSE（クラスタ内誤差平方和）



出力結果：クラスタ数ごとの SSE（クラスタ内誤差平方和）から、n_clouster（クラスタ数）が 3 以降、SSE（クラスタ内誤差平方和）の減少が緩やかになっており、今回の k-means 法で使用する最適なクラスタ数は 3 と判断した。

続いて、クラスタ数を 3 と設定し、k-means 法を用いてクラスター分析を実行した。

* Python のスクリプト

```
# k-means
n_cluster = 3
kmeans = KMeans(n_clusters=n_cluster, random_state=0)
kmeans.fit(df_ss.iloc[:, :4].values)
# 標準化前の df に cluster ラベルを追加
df['cluster'] = kmeans.labels_
```

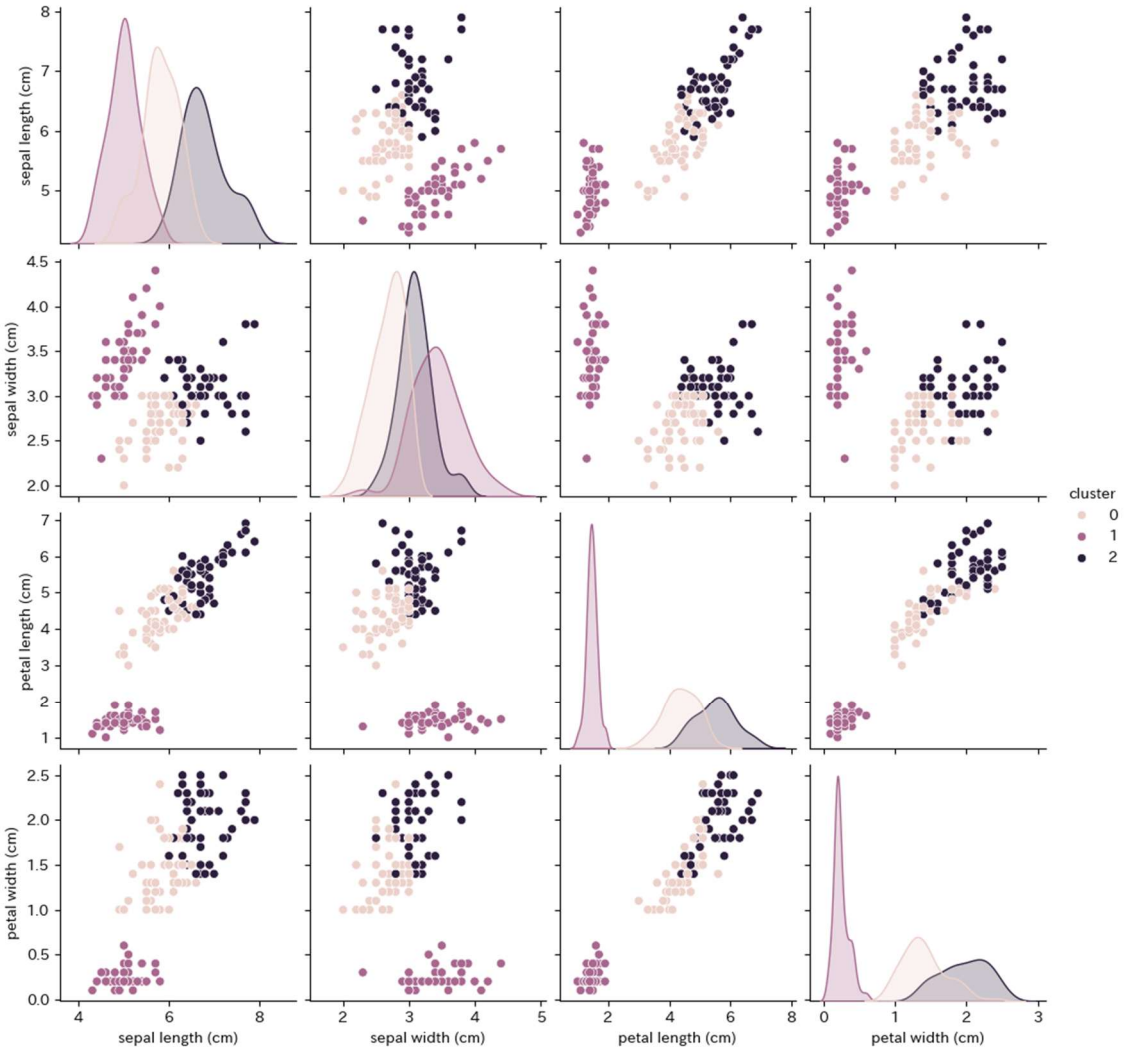
* 出力結果　→　なし

クラスタリング後、クラスタリングに使用したデータの特徴量（説明変数）の分布をクラスタごとに確認するため、ペアプロットを作成した。

* Python のスクリプト

```
import seaborn as sns

# 各クラスタの特徴量の分布を確認
sns.pairplot(df, hue='cluster').savefig('pairplot.png')
```

* 出力結果：各クラスタの特徴量の分布

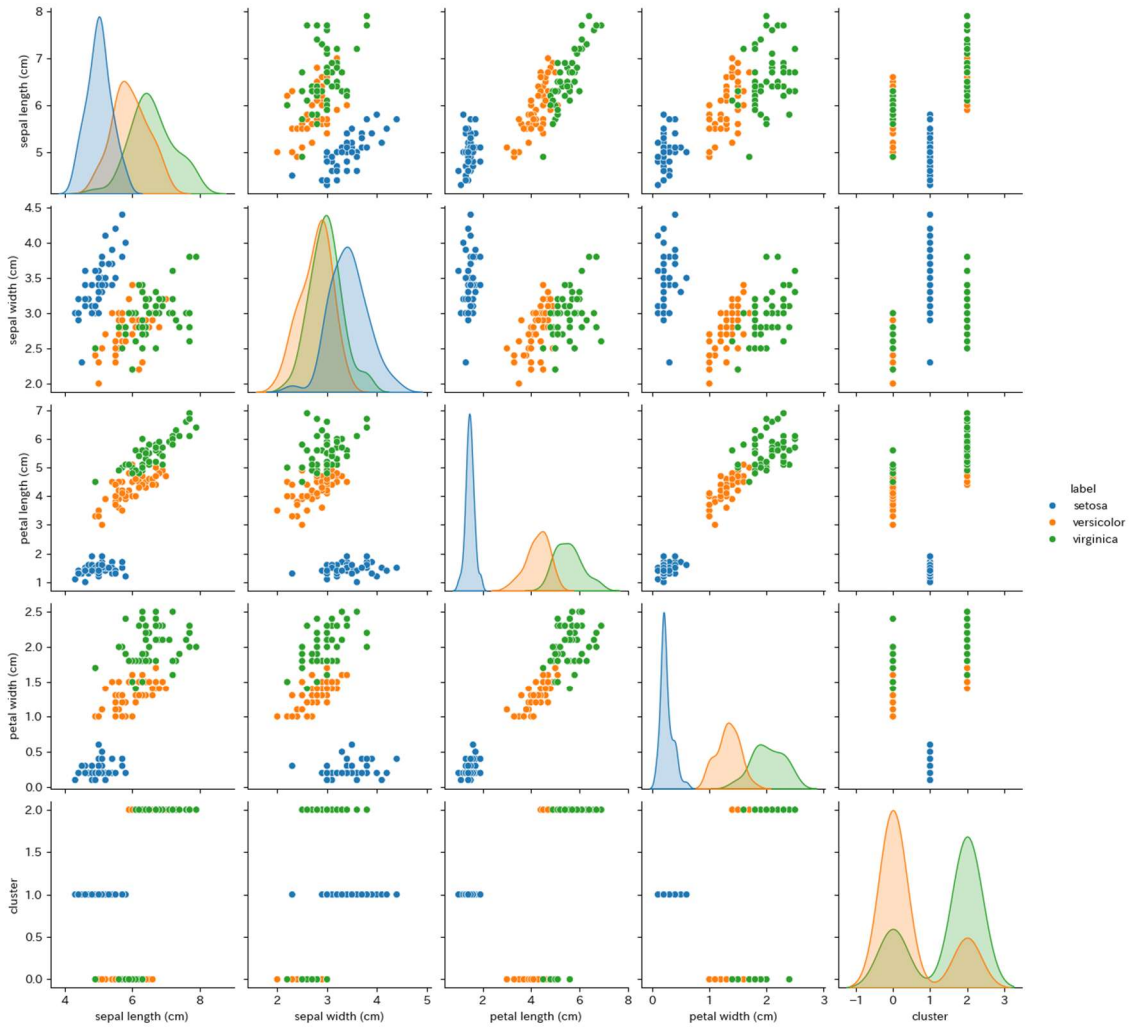


出力結果：各クラスタの特徴量の分布から、クラスタ 1 は、petal length と petal width が小さいアヤメであることがわかる。また、種族ごとにペアプロットを作成した。

* Python のスクリプト

```
# 各クラスタの特徴量の分布を確認
sns.pairplot(df, hue='label').savefig('pairplot_label.png')
```

* 出力結果：種族ごとのペアプロット



出力結果を確認すると、クラスタの分布と同じような結果が得られた。クラスタと種族には、クラスタ 0：versicolor、クラスタ 1：setosa、クラスタ 2：virginica という関係がみられ、クラスタは種族ごとに形成されていた。setosa のクラスタは完全に独立しているが、versicolor と virginica は分布が重なっており、versicolor と virginica は似た特徴をもっていることやミスラベリングが考えられる。

- **数量化理論**

(1) iris データセットについて、因子数の決定（固有値の計算）

今回は、得意科目のデータ（10 人の児童）を使用して、Python の「mca（Multiple Correspondence Analysis)」ライブラリを用いたコレスポンデンス分析を試みた。なお、csv ファイルの読み込みの際、様々な文字コードに対応できるようにした。

※csv ファイルの内容は、以下の「出力結果」を参照してください。

* Python のスクリプト

```
# ライブラリのインポート
import mca
import matplotlib.pyplot as plt
import pandas as pd
import chardet
import japanize_matplotlib
%matplotlib inline

# CSV ファイルの読み込み
with open("like.csv", 'rb') as f:
      binary = f.read()
d = chardet.detect(binary)
if d["encoding"] == "utf-8":
      enco = "utf-8"
elif d["encoding"] == "UTF-8-SIG":
      enco = "utf_8_sig"
else:
      enco = "SHIFT-JIS"
df = pd.read_table("like.csv", encoding=enco, sep=',', index_col=0, header=0)

df
```

* 出力結果

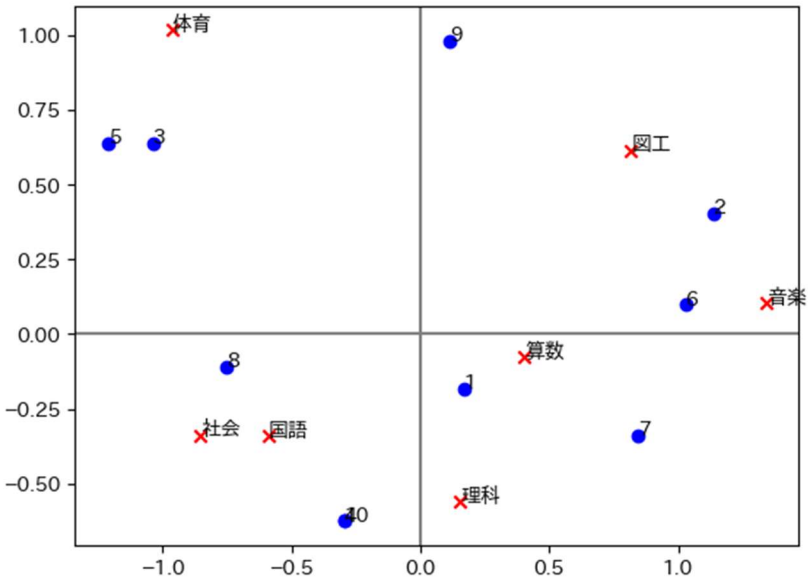
	国語	社会	算数	理科	音楽	図工	体育
生徒							
1	1	0	0	1	0	1	0
2	0	0	1	0	1	1	0
3	1	0	0	0	0	0	1
4	1	1	1	1	0	0	0
5	0	1	0	0	0	0	1
6	0	0	0	1	1	1	0
7	0	0	1	1	1	0	0
8	1	1	0	1	0	0	1
9	0	0	1	0	0	1	1
10	1	1	1	1	0	0	0

上述の出力結果から csv ファイルの内容を正しく読み取れていることができたので、次に mac ライブラリの MAC 関数を使用してコレスポンデンス分析を実施した。

* Python のスクリプト

```
# コレスポンデンス分析の実行
mca_counts = mca.MCA(df, benzecri=False)
rows = mca_counts.fs_r(N=2)
cols = mca_counts.fs_c(N=2)

# 散布図の作図設定
plt.scatter(rows[:, 0], rows[:, 1], c='b', marker='o')
labels = df.index
for label, x, y in zip(labels, rows[:, 0], rows[:, 1]):
    plt.annotate(label, xy=(x, y))

plt.scatter(cols[:, 0], cols[:, 1], c='r', marker='x')
labels = df.columns
for label, x, y in zip(labels, cols[:, 0], cols[:, 1]):
    plt.annotate(label, xy=(x, y))

# xy 軸の表示
plt.axhline(0, color='gray')
plt.axvline(0, color='gray')

# 散布図の描画
plt.show()
```

* 出力結果：コレスポンデンス分析結果



出力結果では、青が生徒番号で赤が科目名となっている。グラフの中央に横軸と縦軸の 0 を意味する十字の線をいれた。分布は、中央から遠ざかるほど特徴的で、プロットされた位置が近いほど、項目間の関連性が強いことを意味している。

最後に固有値と寄与率を確認した。

* Python のスクリプト

```
# 固有値の確認
print(mca_counts.L)

# 2 軸での寄与率
print(mca_counts.L[0]+mca_counts.L[1])
```

* 出力結果

[0.56125992 0.27930632 0.19680168 0.12538374 0.09114991 0.02943177]
0.8405662356601173

2 軸での寄与率は 84.1%と 2 つの軸の寄与率を合計した累積寄与率が 80%を超えているので、元データの反映率は比較的高いと言える。

参考文献リスト

[1] Sebastian Raschka ほか. Python 機械学習プログラミング PyTorch & scikit-learn 編，インプレス（2022）

[2] A.C. Muller, S. Guido. “Python ではじめる機械学習”，オライリージャパン（2017）