機械学習
教師なし学習１：
次元の削減

Pythonによる機械学習入門

第１４章

CONTENTS	
１４．１	次元削減の概要
１４．２	データの前処理
１４．３	主成分分析の実施
１４．４	結果の評価
１４．５	第１４章のまとめ
１４．６	練習問題
１４．７	練習問題の解答

P490

１４．１　次元削減の概要　P490～P498

教師あり学習

- 回帰
- 分類

→ 入力データをもとに何か別のデータ（正解データ）を予測する

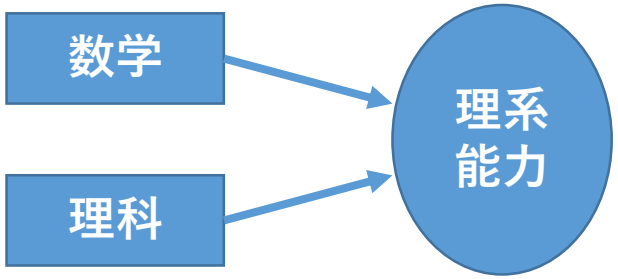
教師なし学習

- 次元削減 → たくさんある特徴量を少ない特徴量にまとめる
- クラスタリング → 似ているデータ同士をグループに分ける

１４．１．１　次元削減とは　P490～P493

次元削減 → 既存の列データを組み合わせて、新しい列を作ることができる

氏名	数学	理科
松田	80	70
浅木	85	90
工藤	70	85
国元	65	60

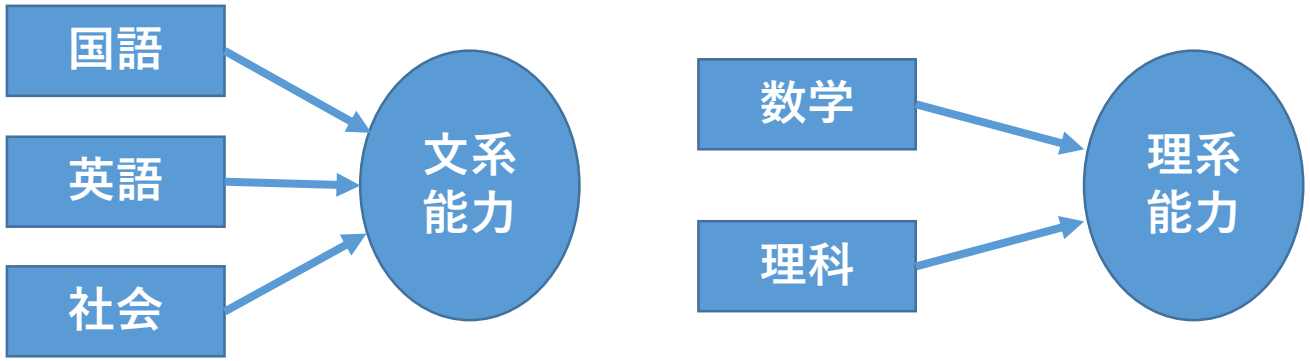


実際に測定できる構成要素を組み合わせることで実際には測定できない概念的な指標を作成できる

氏名	理系能力
松田	75
浅木	88
工藤	80
国元	62

国語　英語　社会　数学　理科

もともと５次元だったデータを２次元のデータにすることができる

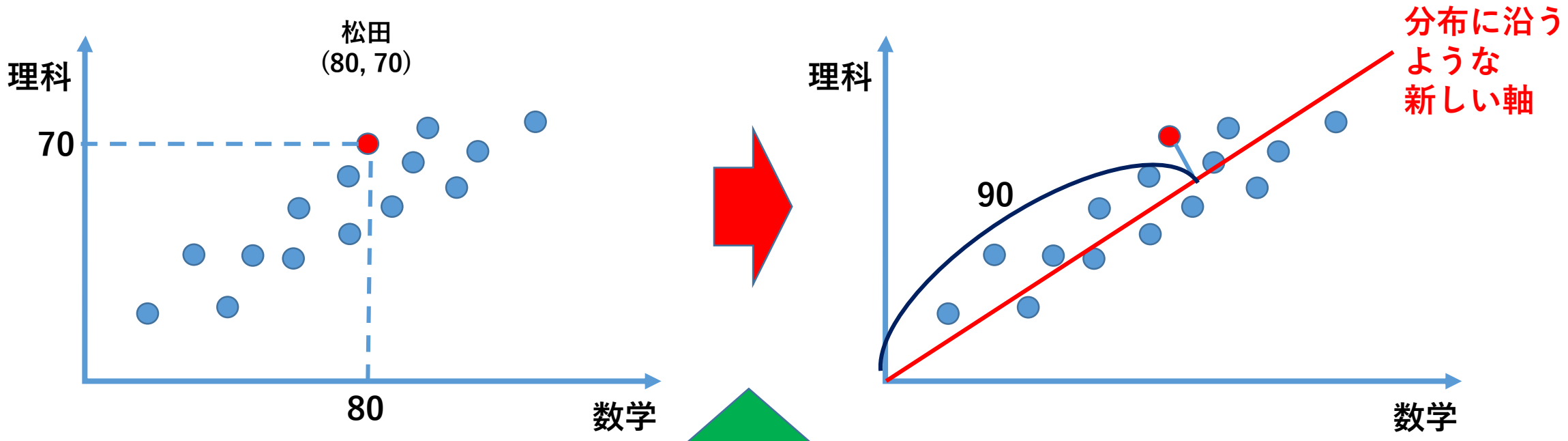


氏名	国語	数学	英語	理科	社会
松田	70	80	75	70	65
浅木	85	85	85	90	75
工藤	90	70	65	85	90
国元	65	65	56	60	90

次元削減モデル

文系能力	理系能力
70	82
80	84
84	69
72	81

主成分分析　最も代表的な次元削減の手法

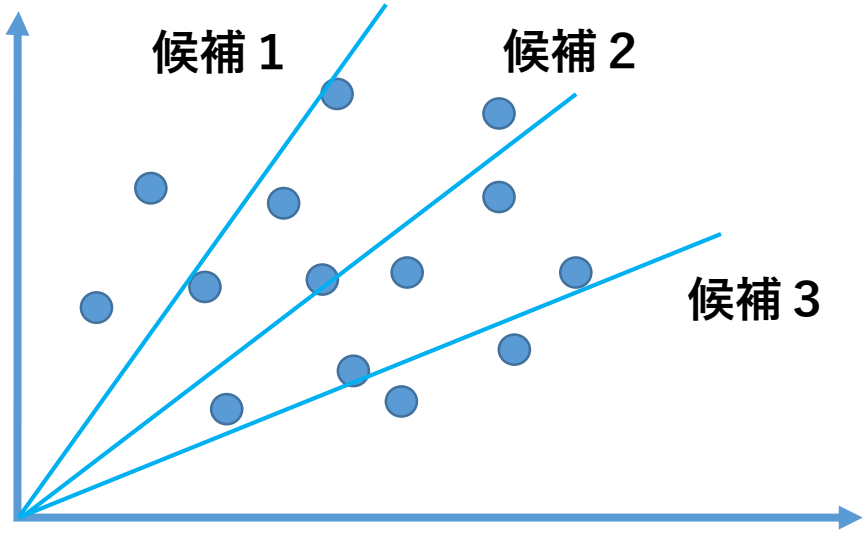


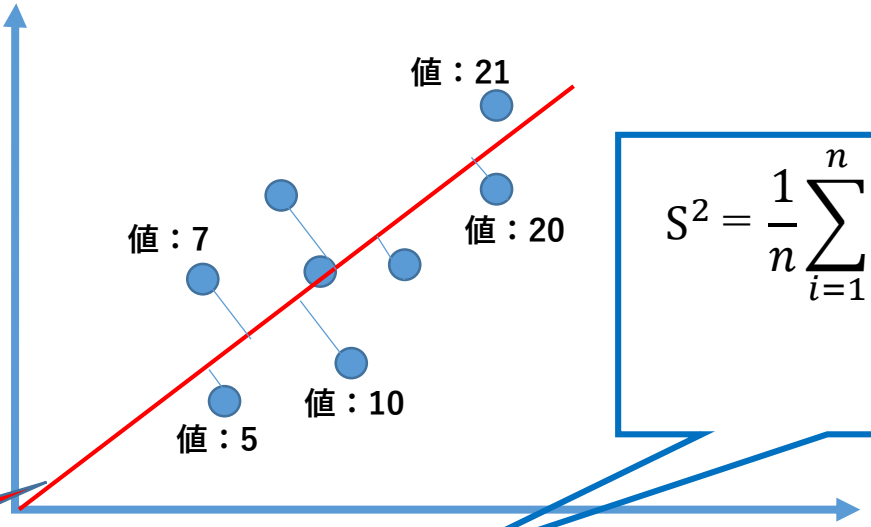
各列の相関が強い場合

各列の相関を考慮して
傾向が似ているかどうかを判断し
１列にまとめる

各列の相関がそんなに強くない場合

新しい軸にデータを移したとき、新しい軸上でのデータ分散値が最大になるような軸を選ぶ





$$S^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

n: データ総数
x_i: 個々の数値
$\overline{x}$:平均値

新しい軸での分散が大きいほど、元データの情報を反映している

新しいの候補の中で、データの分散値が最大になるような軸を選択

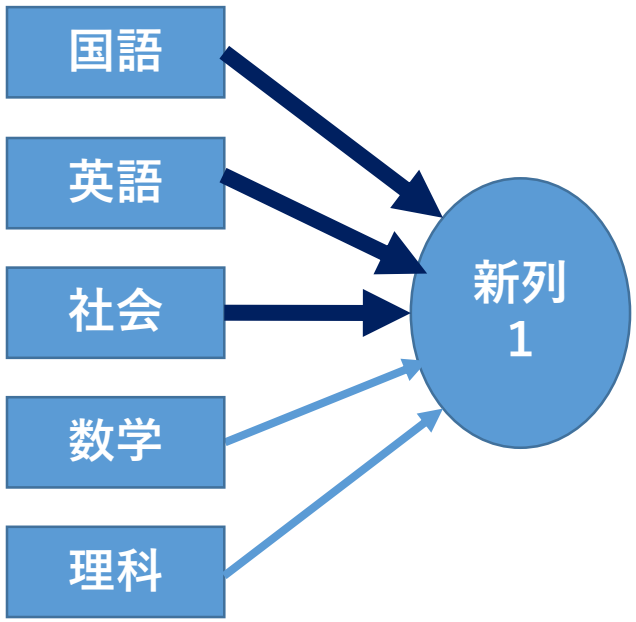
決定した新しい軸
固有ベクトル
または
主成分
と呼ぶ

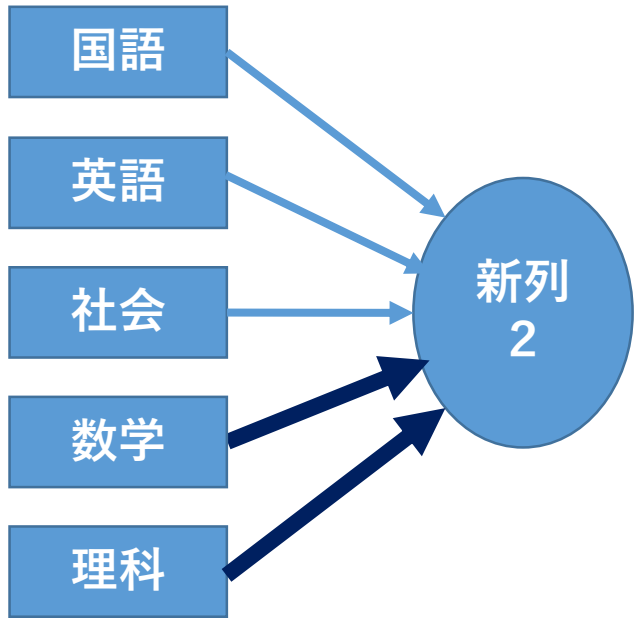
固有ベクトルは
２次元のデータから作成した場合　[0.7, 0.4]
のなどのように、２個の数値の列で表現できる

数学が60点、理科が50点の人の理系能力
理系能力 $= 0.7 * 60 + 0.4 * 50 = 42 + 20 = 62$

■ 新しい列（新しい軸）の名称は分析者が考える

■ 新しい列は、すべての既存列から大なり小なり影響を受けている





１４．２ データの前処理 P499～P501

１４．２．１ データの読み込み P499～P499

コード14-1 データの読み込み

```
# pandasのインポート
import pandas as pd
# Boston.csvの読み込み
df = pd.read_csv('Boston.csv')
# 先頭2行の表示
df.head(2)
```

実行結果

	CRIME	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	high	0.0	18.10	0	0.718	3.561	87.9	1.6132	24.0	666	20.2	354.7	7.12	27.5
1	low	0.0	8.14	0	0.538	5.950	82.0	3.9900	4.0	307	21.0	232.6	27.71	13.2

Boston.csv

CRIME	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
high	0	18.1	0	0.718	3.561	87.9	1.6132	24	666	20.2	354.7	7.12	27.5
low	0	8.14	0	0.538	5.95	82	3.99	4	307	21	232.6	27.71	13.2
very_low	82.5	2.03	0	0.415	6.162	38.4	6.27	2	348	14.7	393.77	7.43	24.1
low	0	21.89	0	0.624	6.151	97.9	1.6687	4	437	21.2	396.9	18.46	17.8
high	0	18.1	0	0.614	6.98	67.6	2.5329	24	666	20.2	374.68	11.66	29.8

データの各列の意味

列名	意味
CRIME	その地域の犯罪発生率（high, low, very_low）
ZN	25000平方フィート以上の住居区画の占める割合
INDUS	小売業以外の商業が占める面積の割合
CHAS	チャールズ川の付近かどうかによるダミー変数（１：川の周辺、０：それ以外）
NOX	窒素酸化物の濃度
RM	住居の平均部屋数
AGE	1940年より前に建てられて物件の割合
DIS	ボストン市内の５つの雇用施設からの距離

列名	意味
RAD	環状高速道路へのアクセスしやすさ
TAX	10000ドルあたりの不動産税率の総計
PTRATIO	町ごとの教員１人当たりの児童生徒数
B	町ごとの黒人の比率（Bk）を次の式で表したもの。1000（Bk - 0.68）^{2}
LSTAT	人口における低所得者の割合
PRICE	その地域の住宅平均価格

１４．２．２ 欠損値の確認 P499～P500

コード14-2 平均値で欠損値を穴埋めする

```
# 列ごとの平均値で欠損値の穴埋め
df2 = df.fillna(df.mean())
```

１４．２．３ ダミー変数化 P500～P500

コード14-3 CRIME列のダミー変数化

```
# CRIME列をダミー変数化
dummy = pd.get_dummies(df2['CRIME'], drop_first = True)
# df2とdummyを列方向に結合
df3 = df2.join(dummy)
# 元のCRIMEを削除
df3 = df3.drop(['CRIME'], axis = 1)
# データフレームを表示
df3.head(2)
```

実行結果

	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE	low	very_low
0	0.0	18.10	0	0.718	3.561	87.9	1.6132	24.0	666	20.2	354.7	7.12	27.5	0	0
1	0.0	8.14	0	0.538	5.950	82.0	3.9900	4.0	307	21.0	232.6	27.71	13.2	1	0

コード14-4 データの標準化

```
# 標準化モジュールをインストール
from sklearn.preprocessing import StandardScaler
# 中身が整数だと、fit_transformで警告になるので、
# float型に変換（省略可能）
df4 = df3.astype('float')
# 標準化
# StandardScalerインスタンスを生成
sc = StandardScaler()
# 標準化する
sc_df = sc.fit_transform(df4)
```

１４．３　主成分分析の実施　P502～P506

１４．３．１　モジュールのインポート　P502～P502

コード14-5 モジュールのインポート

```
# 主成分分析モジュール PCA をインポート
from sklearn.decomposition import PCA
```

１４．３．２　モデルの作成　P502～P506

コード14-6 モデルの作成

```
# モデル作成
model = PCA(n_components = 2, whiten = True)
```

白色化

固有ベクトル数

白色化　白色化を行うと各列の関係は無相関になり、各列の平均値が０、標準偏差が１になる

固有ベクトル　軸上のデータの分散値が最大になるような軸

固有ベクトル数　新たな軸をいくつ見つけるか

既存列をいくつの「新たな列」にまとめるか

モデルの作成

```
PCA( n_components=整数, whiten=真偽値 )

※ PCAは sklearn.decomposition からインポート済み
※ n_components は利用する新規の軸の個数
※ whiten=True で、次元削減の結果の白色化を行う。False だと行わない。
```

コード14-7 モデルに学習をさせる

```
# モデルに学習させる
model.fit(sc_df)
```

コード14-8 第１軸と第２軸の固有ベクトル

```
# 新規の第１軸（第１主成分とも呼ぶ）の固有ベクトル
print( model.components_[0] )
print('-----')
# 新規の第2軸（第2主成分とも呼ぶ）の固有ベクトル
print(model.components_[1])
```

実行結果

```
[-0.2258543   0.35923465  0.04220985  0.3499321  -0.19485285  0.29792086
 -0.29980115  0.30726517  0.32822012  0.16246983 -0.18251937  0.27543839
 -0.2018449   0.03831172 -0.31492126]
-----
[-0.1533893   0.02835867  0.19795373  0.13817925  0.4047141   0.20058802
 -0.29340246 -0.1027543  -0.11546952 -0.34046929  0.05661836 -0.17845386
  0.44390529  0.42253976 -0.27716437]
```

要素数１５＝sc_dfの列数

コード14-9 既存の sc_df を新しい２つの軸に当てはめる

```
# 既存の sc_df を新しい２つの軸に当てはめる
new = model.transform(sc_df)
# データフレーム化する
new_df = pd.DataFrame(new)
# 表示
new_df.head(3)
```

実行結果

	0	1
0	1.490417	-0.680415
1	0.703223	-0.252517
2	-1.403756	-0.613175

新しい２つの軸に当てはめたデータ

学習済みの新規の軸にデータを当てはめる（主成分得点の計算）

モデル変数.transform(表データ)
※ 表データはデータフレームや、numpy の ndarry
※ モデル変数は、事前に fit メソッドで学習
※ 戻り値は numpy であり、左列から順に最適な軸が選んでいる

14．4　結果の評価　P507～P521

14．4．1　主成分負荷量の確認　P502～P502

主成分負荷量　新しい列と既存の列との相関関係を考えるとき使用

コード14-10 新しい２列ともとの列を結合

```
# カラム名の設定
new_df.columns = ['PC1', 'PC2']
# 標準化済の既存データ（numpy)をデータフレーム化
df5 = pd.DataFrame(sc_df, columns = df4.columns)
# 2つのデータフレームを列方向に結合
df6 = pd.concat([df5, new_df], axis=1)
```

コード14-11 主成分負荷量の計算

```
df_corr = df6.corr() # 相関係数の計算
df_corr.loc[:'very_low', 'PC1':]
```

インデックスが very_low より上にある行(very_low含む)

PC1列より右の列(PC1含む)

実行結果

PC1ともとの列の相関係数

PC2ともとの列の相関係数

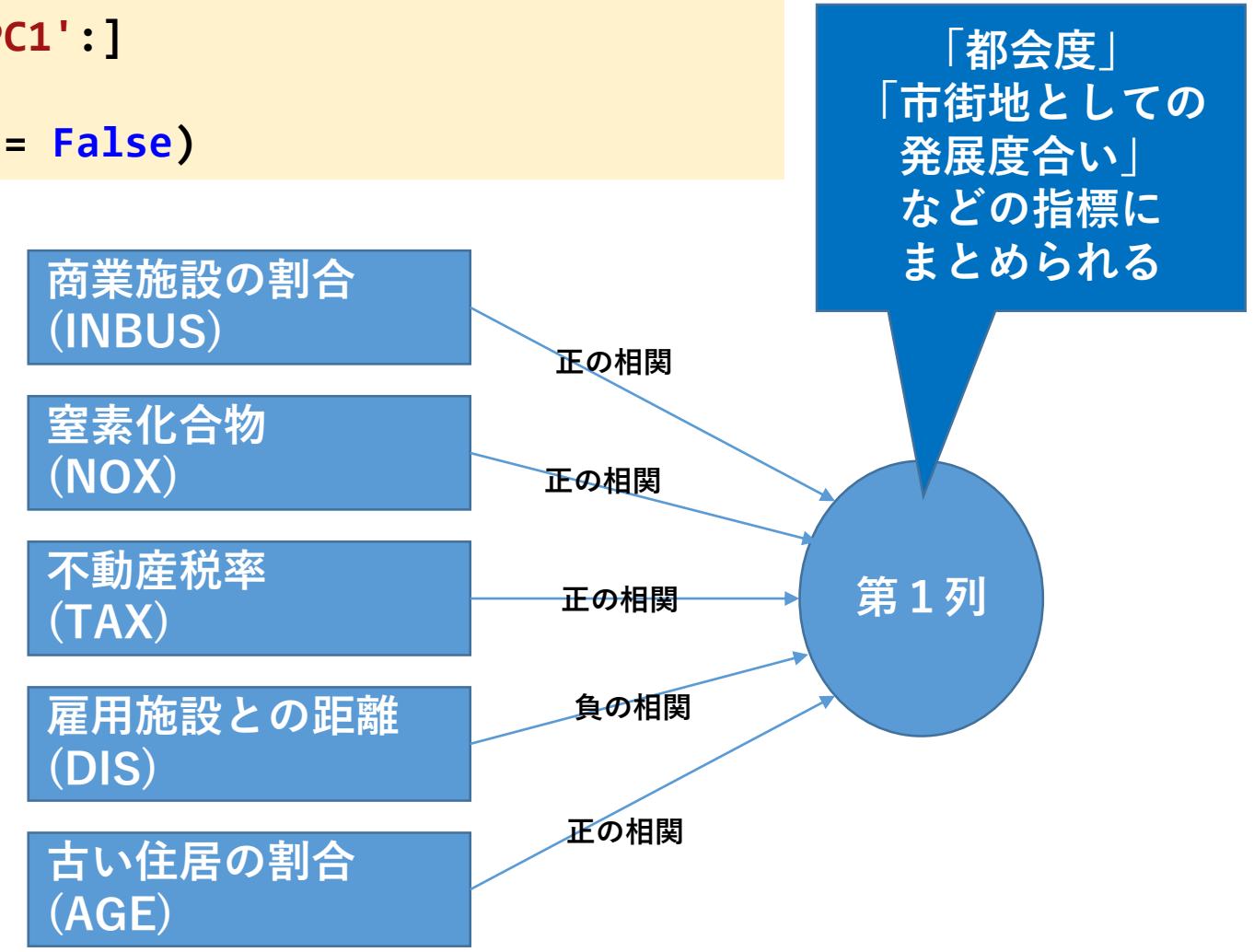
	PC1	PC2
ZN	-0.560802	-0.226097
INDUS	0.891989	0.041801
CHAS	0.104808	0.291786
NOX	0.868891	0.203678
RM	-0.483825	0.596553
AGE	0.739745	0.295669
DIS	-0.744414	-0.432478
RAD	0.762947	-0.151461
TAX	0.814979	-0.170203
PTRATIO	0.403417	-0.501855
B	-0.453200	0.083456
LSTAT	0.683921	-0.263043
PRICE	-0.501186	0.654321
low	0.095129	0.622828
very_low	-0.781958	-0.408543

コード14-12 相関係数を大きい順に並べ替える

```
# わかりやすいように変数に代入
pc_corr = df_corr.loc[:'very_low', 'PC1':]
# PC1列を降順にソート
pc_corr['PC1'].sort_values(ascending = False)
```

実行結果

```
INDUS       0.891989
NOX         0.868891
TAX         0.814979
RAD         0.762947
AGE         0.739745
LSTAT       0.683921
PTRATIO     0.403417
CHAS        0.104808
low         0.095129
B          -0.453200
RM         -0.483825
PRICE      -0.501186
ZN         -0.560802
DIS        -0.744414
very_low   -0.781958
Name: PC1, dtype: float64
```

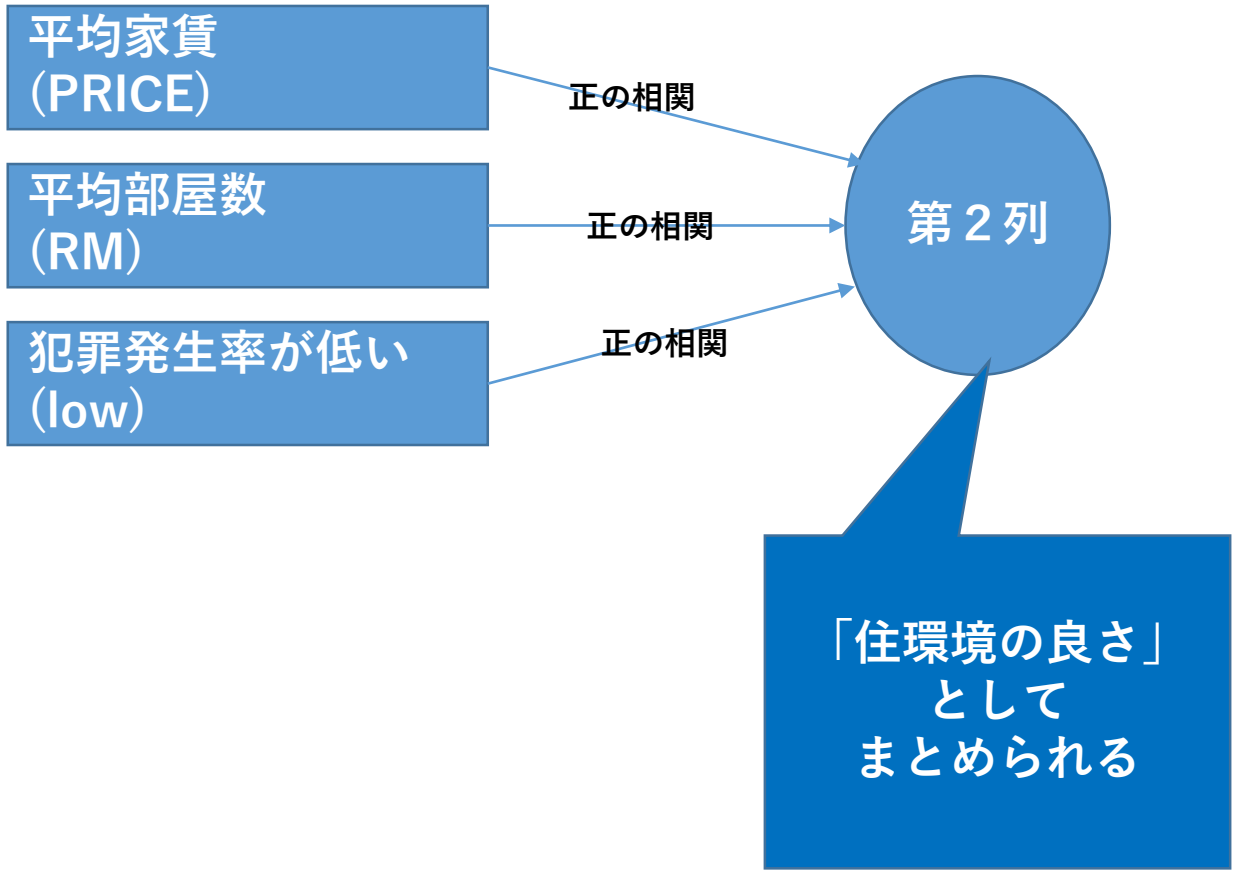


コード14-13 第２列の相関を確認

```
# PC2列を降順にソート
pc_corr['PC2'].sort_values(ascending = False)
```

実行結果

```
PRICE       0.654321
low         0.622828
RM          0.596553
AGE         0.295669
CHAS        0.291786
NOX         0.203678
B           0.083456
INDUS       0.041801
RAD        -0.151461
TAX        -0.170203
ZN         -0.226097
LSTAT      -0.263043
very_low   -0.408543
DIS        -0.432478
PTRATIO    -0.501855
Name: PC2, dtype: float64
```

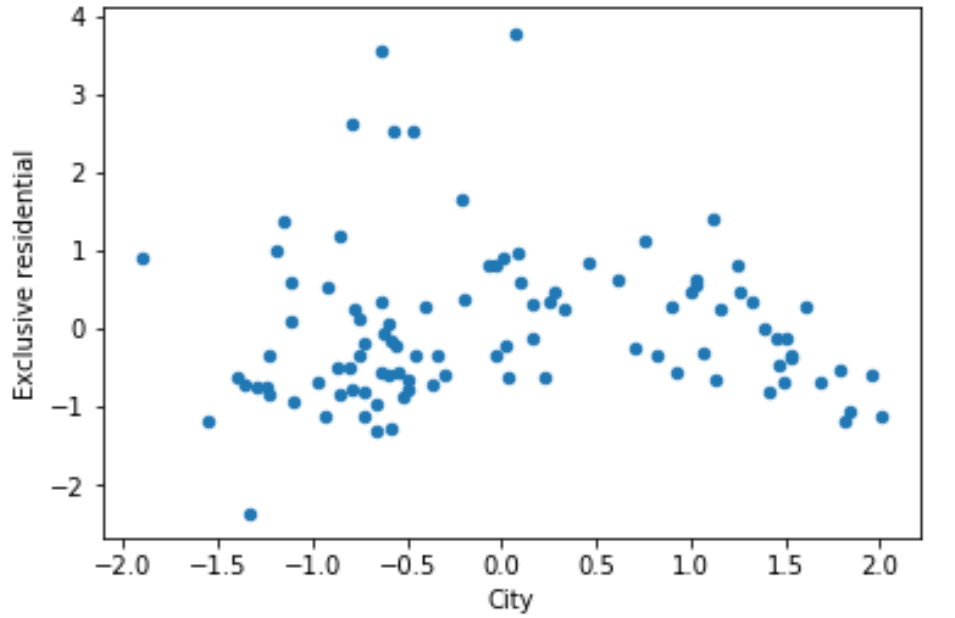


コード14-14 新しい列の散布図

```
# 新しい列名
#  「都市の発展度合い」と「住環境の良さ」
col = ['City', 'Exclusive residential']
# 列名の変更
new_df.columns = col
# 散布図
new_df.plot(kind = 'scatter', x = 'City',
                y = 'Exclusive residential')
```

実行結果

```
<matplotlib.axes._subplots.AxesSubplot at 0x177635236d8>
```



１４．４．２　最適な列の個数　―寄与率―　P514～P521

主成分分析の本来の目的　新規列の個数を少なくして簡潔にデータを表す

新規の列が元データの情報を反映すること

トレードオフの関係

最適な新規列の個数を考慮する必要がある

元のデータの情報量の70%～80%程度反映するように列の個数を選ぶ

コード14-15 新規の軸をすべて用意する

n_components を指定しないと、新規の軸はすべてデフォルトで作られる

```
# 主成分分析モデルを生成
model = PCA(whiten = True)
# 学習と新規軸へのデータの当てはめを一括で行う
tmp = model.fit_transform(sc_df)
# 次元の表示
tmp.shape
```

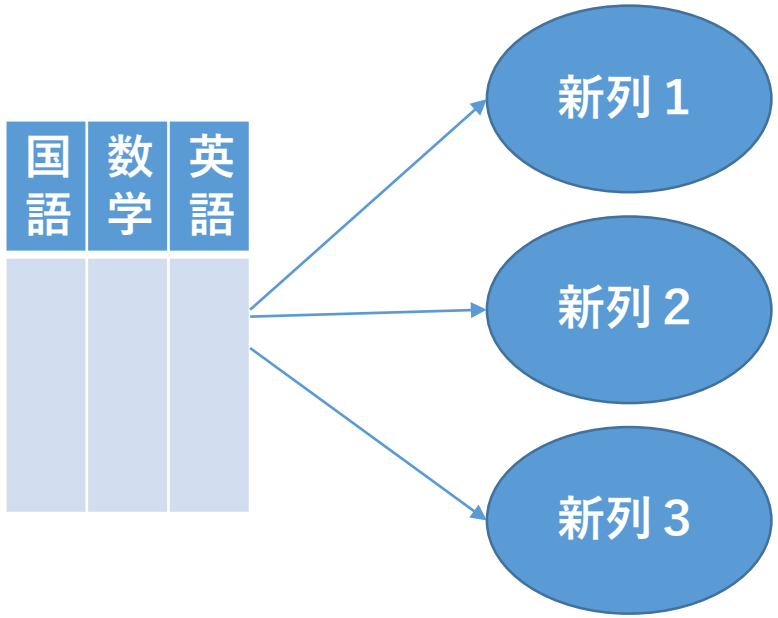
実行結果 (100, 15)

新しい列でのデータの分散値は、元データを反映している情報量と解釈することができる

元データの分散の全体（各列ごとの分散の合計値）と、新しい列の分散の比率を調べる

寄与率

新規の列がもとのデータの全情報量のうち何％ほど反映しているかを表す尺度



一番最適な列
分散値１００
寄与率：100/(100+80+50)

２番目に最適な列
分散値８０
寄与率：80/(100+80+50)

３番目に最適な列
分散値５０
寄与率：50/(100+80+50)

コード14-16 寄与率を表示する

```
# 寄与率
model.explained_variance_ratio_
```

実行結果

```
array([0.41102789, 0.14484698, 0.10192698, 0.06448954, 0.06233684,
       0.05810331, 0.04843711, 0.02885228, 0.02142431, 0.01831962,
       0.01572944, 0.01068611, 0.00918466, 0.00277548, 0.00185945])
```

第２列の寄与率

第１列の寄与率

元データの41%を
第１列だけで
反映することができる

元データの80%を反映させるように、新規の列を選びたい

最適な第１列から順々に寄与率を足していくと

0.41102789+0.14484698+0.10192698+0.06448954+0.06233684+0.05810331
=約0.85

第６列で、目標の80%を突破

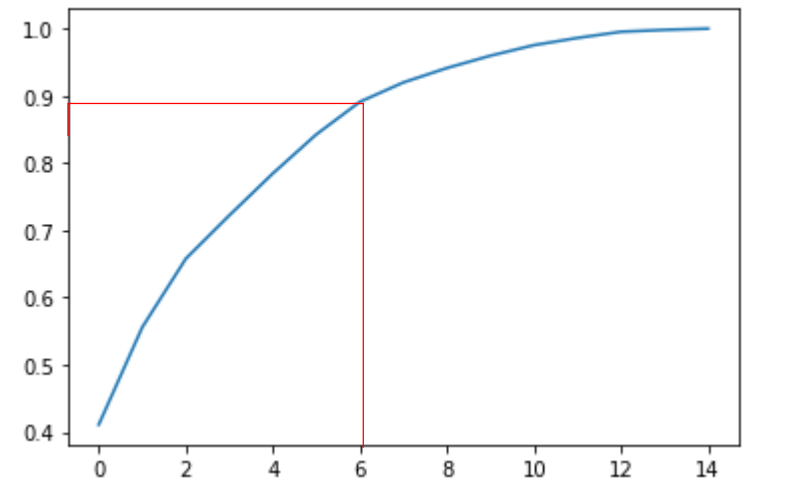
第N列の累積寄与率 ＞ 第１列～第N列の寄与率の合計

コード14-17 累積寄与率

```
# 寄与率のデータ集合
ratio = model.explained_variance_ratio_
# 第N列までの累積寄与率を格納するリスト
array = []
for i in range(len(ratio)):
  # 累積寄与率の計算
  ruiseki = sum(ratio[0:(i+1)])
  # 累積寄与率の格納
  array.append(ruiseki)
# 第N列の累積寄与率を折れ線グラフ化
pd.Series(array).plot(kind = 'line')
```

実行結果

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f46f950160>
```



コード14-18 情報量のしきい値を設定して必要な列数を求める

```
# 累積寄与率のしきい値
thred = 0.8
for i in range(len(array)):
  # 第(i + 1)列の累積寄与率がthredより大きいかチェック
  if array[i] >= thred:
    print(i + 1)
    break
```

実行結果　6

コード14-19 新規の列を６つ設定してモデルに学習させる

```
# もとのデータの全情報の80%を賄うために、新規の列を６つに設定
model = PCA(n_components=6, whiten = True)
# 学習
model.fit(sc_df)
# 元データを新規の列（6列）に当てはめる
new = model.transform(sc_df)
```

コード14-20　６列のデータをCSVファイルに保存

```
# 列名の指定
col = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6']
# 主成分分析の結果をデータフレームに変換
new_df2 = pd.DataFrame(new, columns = col)
# データフレームをcsvファイルとして保存
new_df2.to_csv('boston_pca.csv', index = False)
```

分散と寄与率

・新規の列（軸）での分散
モデル変数.explained_variance_

・寄与率
モデル変数.explained_variance_ratio_

※ 戻り値は numpy
※ ０番目から順に　新機軸１，新機軸２、・・・と続く

データフレームのcsv保存

df.to_csv(‘ファイル名’, index=ブール値)
※ index=Trueとすると、インデックスもCSVファイルに掃き出される