データ科学基礎演習B（6）

データ科学科目部会

データの入出力と可視化

pandasを使ってみよう

pandasとは？

- データ解析に役立つ機能を提供するモジュール
  - 各種データの入出力（csv，json, Excel，SQLDB，他）
  - データに対する各種処理
    - アライメント
    - 欠損値処理
    - 結合
    - ピボット処理
    - …

本講義ではCSVデータの入力のみ紹介

- モジュールには別名「pd」を使う事が多い

```
import pandas as pd
```

CSVファイルって何？

- CSV = Comma Separated Value
  - カンマ／コンマ(,)をデータ区切りとする方法
  - ファイル拡張子は .csv

```
Year, Male, Female
1955, 63.6, 67.8
1960, 65.3, 70.2
1965, 67.7, 72.9
1970, 69.3, 74.7
```

【小ネタ】陸上競技で「コンマ1秒」という表現で小数点をコンマと呼ぶのは，
フランスやドイツでは小数点の記号にコンマが使われるため

CSVファイルの読み込み（read_csv関数）

- CSVファイルのパスもしくはURLを指定する
  - 戻り値はDataFrameという表形式のデータ

`pd.read_csv(`ファイルへのパス or URL`)`

```
[1]  1  import numpy as np
     2  import pandas as pd

[2]  1  pd.read_csv('https://bit.ly/36QRT2u')
```

	Year	Male	Female
0	1955	63.6	67.8
1	1960	65.3	70.2
2	1965	67.7	72.9
3	1970	69.3	74.7

※講義スライドで利用する平均寿命データ: https://bit.ly/36QRT2u

CSVファイルの読み込み（read_csv関数）

- CSVファイルのパスもしくはURLを指定する
  - 戻り値はDataFrameという表形式のデータ

```
pd.read_csv(ファイルへのパス or URL)
```

- 先頭行に見出しがないCSVファイルの場合
  - header=None を引数に追加する

※デフォルトでは1行目が見出しとなるため，データが1行分少なくなる

※講義スライドで利用する平均寿命データ：https://bit.ly/36QRT2u

DataFrame から numpy配列への変換

- DataFrameの変数（read_csvの戻り値）に対して「.values」を追加

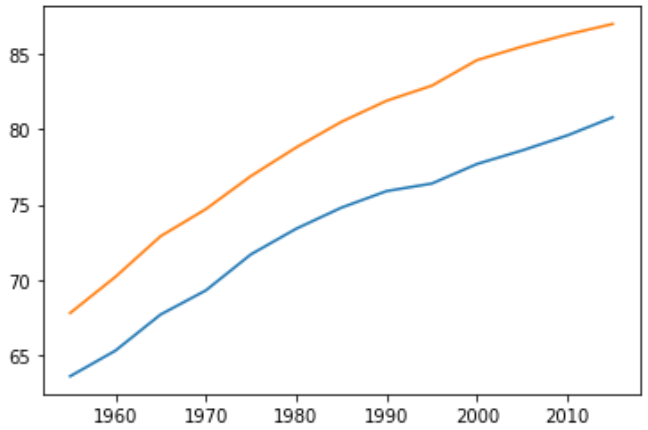
```
c = pd.read_csv(…)
a = c.values #aはnumpy配列
```

```
[1]  1  import numpy as np
     2  import pandas as pd

[2]  1  c = pd.read_csv('https://bit.ly/36QRT2u')

[3]  1  a = c.values

[4]  1  a

array([[1955. ,  63.6,  67.8],
       [1960. ,  65.3,  70.2],
       [1965. ,  67.7,  72.9],
       [1970. ,  69.3,  74.7],
       [1975. ,  71.7,  76.9],
       [1980. ,  73.4,  78.8],
       [1985. ,  74.8,  80.5],
       [1990. ,  75.9,  81.9],
       [1995. ,  76.4,  82.9],
       [2000. ,  77.7,  84.6],
       [2005. ,  78.6,  85.5],
       [2010. ,  79.6,  86.3],
       [2015. ,  80.8,  87. ]])
```

CSVで読み込んだ内容を描画してみよう

- CSVファイルを読み込んでグラフを描画する一連の流れを試してみましょう

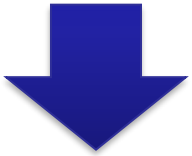
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

c  = pd.read_csv('https://bit.ly/36QRT2u')
a  = c.values
x  = a[:,0]
y1 = a[:,1]
y2 = a[:,2]

plt.plot(x, y1)
plt.plot(x, y2)
```



データ点のみをプロットしてみよう

- plot関数の標準はデータ点間を直線で結ぶ
  → データ点のマーカのみを表示するように変更しよう
  - marker, markersize, linewidthの引数を設定

```
plt.plot(x, y1)
plt.plot(x, y2)
```



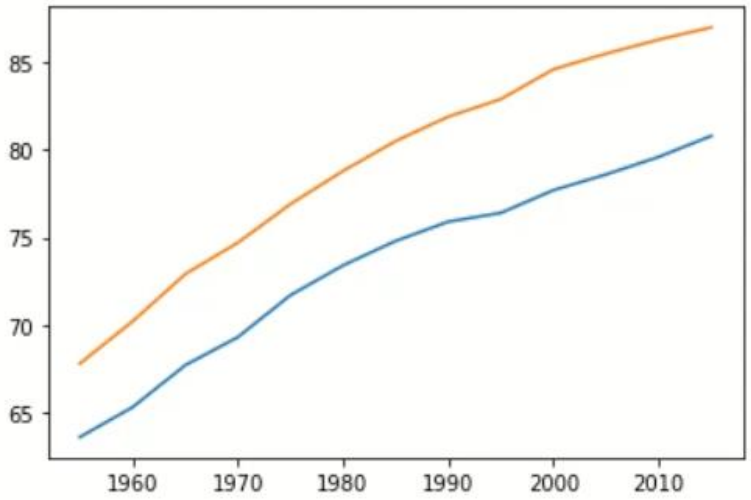
```
plt.plot(x, y1, linewidth=0, marker='o', markersize=5)
plt.plot(x, y2, linewidth=0, marker='x', markersize=5)
```

線を非表示　　マーカ記号　　マーカのサイズ

データ点のみをプロットしてみよう

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

c  = pd.read_csv('https://bit.ly/36QRT2u')
a  = c.values
x  = a[:,0]
y1 = a[:,1]
y2 = a[:,2]

plt.plot(x, y1)
plt.plot(x, y2)
```

```
[<matplotlib.lines.Line2D at 0x7fe33e4a3278>]
```

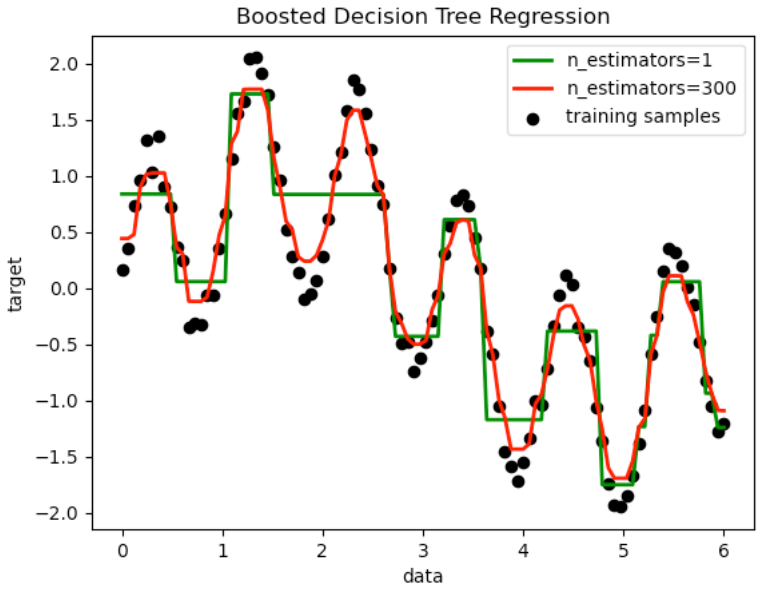


機械学習の基礎（回帰）

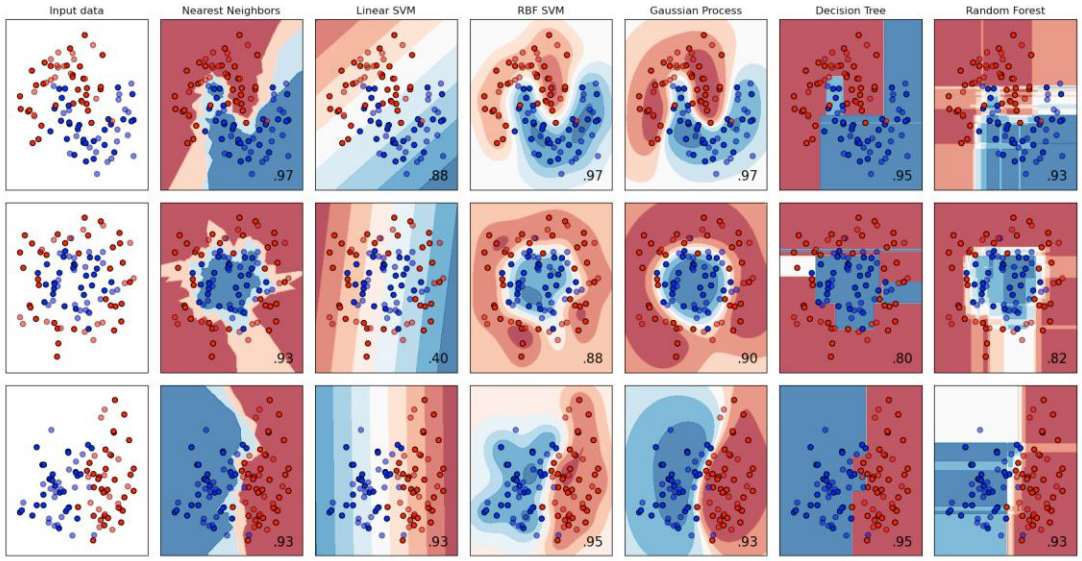
scikit-learnに触れてみよう

scikit-learn（https://scikit-learn.org/）とは？

- 様々な機械学習手法を網羅したライブラリ
  - 回帰分析，クラス分類，クラスタリング，etc.
- 異なる手法でも同じインタフェースで利用可能
  - 同じ使い方でいろいろな手法を試せる



回帰の例



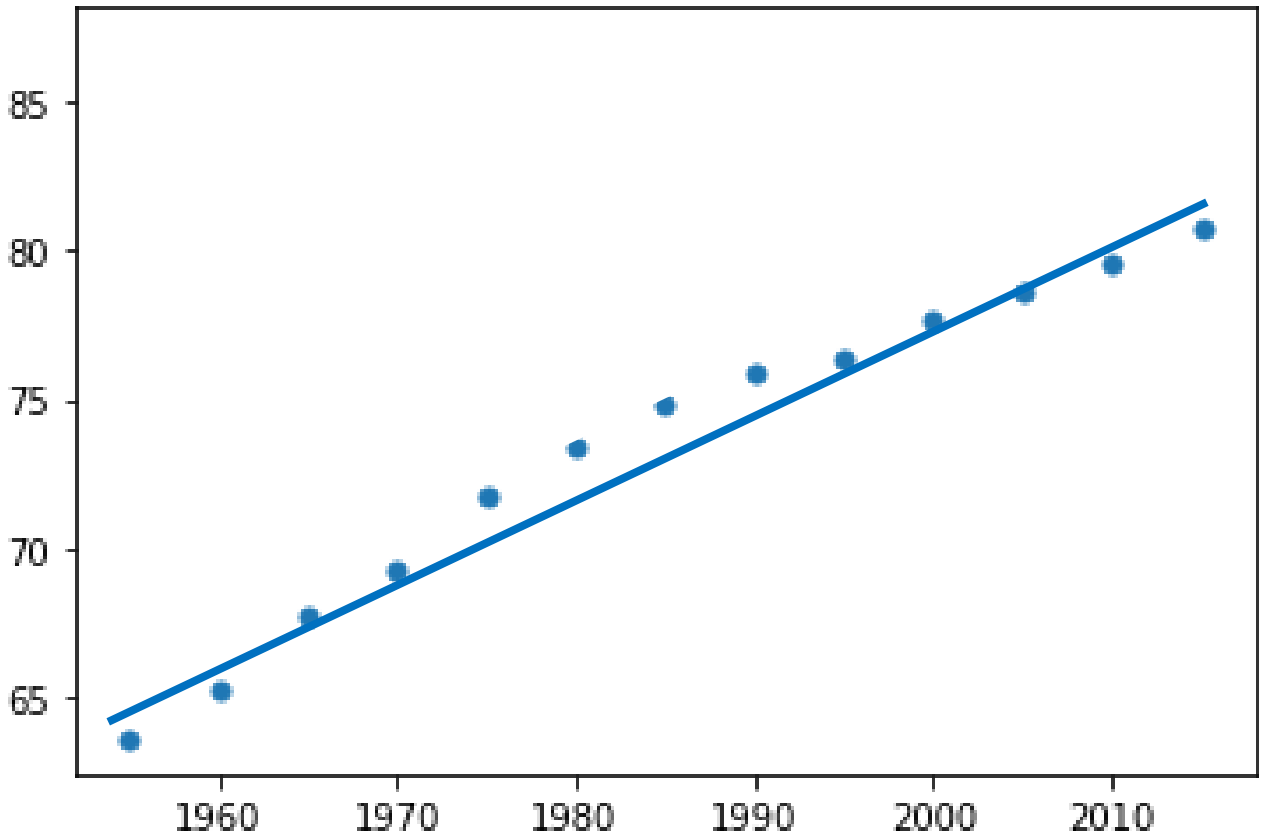
クラス分類の例

※図はsckit-learnホームページより引用

最小2乗法を用いた線形回帰(1)

- 入力 x と出力 y から係数 a と b を求める問題

$$y = ax + b$$



最小2乗法を用いた線形回帰(2)

- 入力 x と出力 y から係数 a と b を求める問題

$$y = ax + b$$

- 入力 x がM次元のベクトル $\boldsymbol{x}$ となった場合は？

$$y = \boldsymbol{a}^T\boldsymbol{x} + b$$

- 入力 $\boldsymbol{x}$ と出力 y のペアをN組用意する
  - $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), (\boldsymbol{x}_3, y_3), \ldots, (\boldsymbol{x}_N, y_N)$

最小2乗法を用いた線形回帰(3)

- 入力 $\boldsymbol{x}$ と出力 y のペアを N 組($N > M$)用意
  - $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), (\boldsymbol{x}_3, y_3), \ldots, (\boldsymbol{x}_N, y_N)$

$$\begin{cases} y_1 = \boldsymbol{a}^T \boldsymbol{x}_1 + b \\ y_2 = \boldsymbol{a}^T \boldsymbol{x}_2 + b \\ \vdots \\ y_N = \boldsymbol{a}^T \boldsymbol{x}_N + b \end{cases}$$

最小2乗法を用いた線形回帰(4)

- 入力 $\boldsymbol{x}$ と出力 y のペアを N 組($N > M$)用意
  - $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), (\boldsymbol{x}_3, y_3), \ldots, (\boldsymbol{x}_N, y_N)$
- ベクトルを要素に展開
  - $\boldsymbol{a} = (a_1, a_2, \ldots, a_M)^T$, $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{iM})^T$

$$
\begin{aligned}
y_1 &= a_1 x_{11} + a_2 x_{12} + \cdots + a_M x_{1M} + b \\
y_2 &= a_1 x_{21} + a_2 x_{22} + \cdots + a_M x_{2M} + b \\
&\vdots \\
y_N &= a_1 x_{N1} + a_2 x_{N2} + \cdots + a_M x_{NM} + b
\end{aligned}
$$

最小2乗法を用いた線形回帰(5)

- 行列の形で連立方程式を整理する

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}}_{Y} = \underbrace{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1M} & 1 \\ x_{21} & x_{22} & \cdots & x_{2M} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NM} & 1 \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_M \\ b \end{pmatrix}}_{\beta}$$

データとして与えられているので既知の値

求めたい未知変数

最小2乗法を用いた線形回帰(6)

- 行列の形で連立方程式を整理する

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta}$$

- 両辺に $\boldsymbol{X}^T$ を掛ける

$$\boldsymbol{X}^T\boldsymbol{Y} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta}$$

- 両辺に $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ を掛ける

$$(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{\beta}$$

- 最終的に $\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$ が求めたい係数

scikit-learnの線形モデルをインポート

- scikit-learnのモジュール名は sklearn
- scikit-learnは非常に大きなモジュールのため，必要な機能のみをインポートしよう

- 線形モデルをインポート
  – sklearn モジュールから linear_model をインポート

```
from sklearn import linear_model
```

最小2乗回帰手法を使うための準備

- linear_model の LinearRegression を初期化

```
from sklearn import linear_model
lr = linear_model.LinearRegression()
```

線形回帰に関する処理は
この変数に対して行う

scikit-learnを用いた線形回帰

- 覚える関数は2つ
  - fit 関数

データからモデルを学習

    - 入力 $\boldsymbol{x}$ と出力 y のペアを N 組与えて $\boldsymbol{\beta}$ を計算
  - predict 関数

モデルを使って出力を予測

    - 入力 $\boldsymbol{x}$ から出力 y を予測

      ※ fit関数で求めた $\boldsymbol{\beta}$ を用いて出力を計算

fit 関数の使い方

- 入力 x と出力 y のペアを引数に指定

```
lr.fit(入力X, 出力Y)
```

線形回帰モデルの変数
（2ページ前のlr）

入力 x と出力 y のペアの作り方(1)

- 入力 x と出力 y は別々の変数として用意する
  - N個の入力を $x_1, x_2, \cdots, x_N$, 出力を $y_1, y_2, \cdots, y_N$
    ※添字が同じであれば入力と出力が対応
- 入力データ X の作り方
  - 各行が $x_1, x_2 \cdots, x_N$ に対応するよう並べた行列を作る
    - i 個目のデータを $x_i = (x_{i1}, x_{i2}, \ldots, x_{iM})^T$ とすると

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & x_{22} & \cdots & x_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NM} \end{pmatrix}$$

x_1

x_N

最小2乗法の説明で出てきた最後の1のみの列は不要

入力 x と出力 y のペアの作り方(2)

- 入力 x と出力 y は別々の変数として用意する
  - N個の入力を $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N$, 出力を $y_1, y_2, \cdots, y_N$
    ※添字が同じであれば入力と出力が対応
- 出力データ $\boldsymbol{Y}$ の作り方
  - 各行もしくは各列が $y_1, y_2, \cdots, y_N$ となるベクトルを作る

$$\boldsymbol{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \quad \text{もしくは} \quad \boldsymbol{Y} = (y_1, y_2, \cdots, y_N)$$

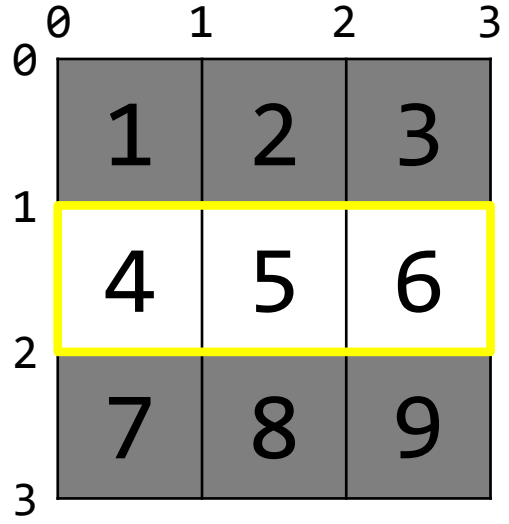
【再掲】numpy 配列の要素アクセス

- スライスを使って特定の行もしくは列を抽出

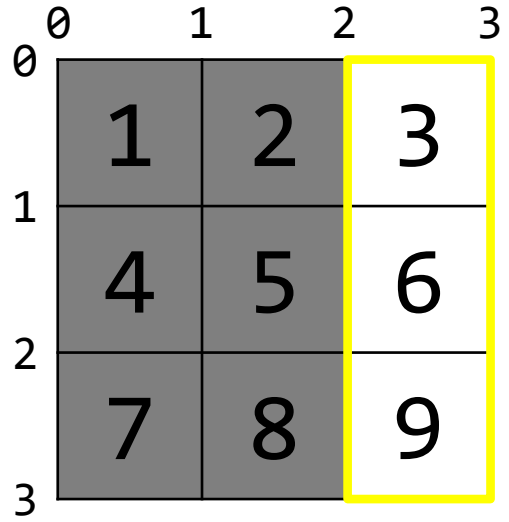
a =

1	2	3
4	5	6
7	8	9

a[1,:]



a[:,2]



このスライス指定では
抽出結果はいずれも
行ベクトル
になる

【再掲】numpy 配列からの列ベクトル抽出

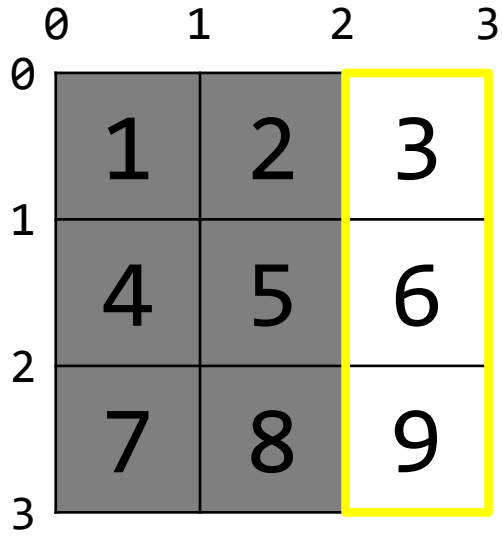
- スライスを使ってある列を列ベクトルとして抽出

a =

1	2	3
4	5	6
7	8	9

a[:,2:3]

列の範囲を明記



行ベクトルを列ベクトルに変換する方法

- reshape 関数で行ベクトルを列ベクトルに変換
  - 引数に -1 を指定するとどうなる？
    - 要素数を残りの次元の情報から自動計算してくれます

- 列ベクトルに変換 → a.reshape(-1,1)

行数は自動計算

列数は1に固定

- 行ベクトルに変換 → a.reshape(1,-1)

行数は1に固定

列数は自動計算

fit 関数の使い方

- 入力 x と出力 y のペアを引数に指定

lr.fit(入力X,出力Y)

```
[1]  1  %matplotlib inline
     2  import matplotlib.pyplot as plt
     3  import numpy as np
     4  import pandas as pd
     5  c  = pd.read_csv('https://bit.ly/36QRT2u')
     6  a  = c.values

[2]  1  from sklearn import linear_model
     2  lr = linear_model.LinearRegression()

[4]  1  x = a[:,0].reshape(-1,1)
     2  y = a[:,1].reshape(-1,1)
     3  lr.fit(x,y)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

     1
```

各行が一つ一つのデータになるよう注意

predict 関数の使い方

- 入力 x を引数に指定(複数の同時予測が可能)

```
lr.predict(入力X)
```

```
[3]  1  x = a[:,0].reshape(-1,1)
     2  y = a[:,1].reshape(-1,1)
     3  lr.fit(x,y)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

[4]  1  lr.predict(x)

array([[65.02637363],
       [66.42967033],
       [67.83296703],
       [69.23626374],
       [70.63956044],
       [72.04285714],
       [73.44615385],
       [74.84945055],
       [76.25274725],
       [77.65604396],
       [79.05934066],
       [80.46263736],
       [81.86593407]])
```

回帰結果の描画

```
[1]  1  %matplotlib inline
     2  import matplotlib.pyplot as plt
     3  import numpy as np
     4  import pandas as pd
     5  c  = pd.read_csv('https://bit.ly/36QRT2u')
     6  a  = c.values
```

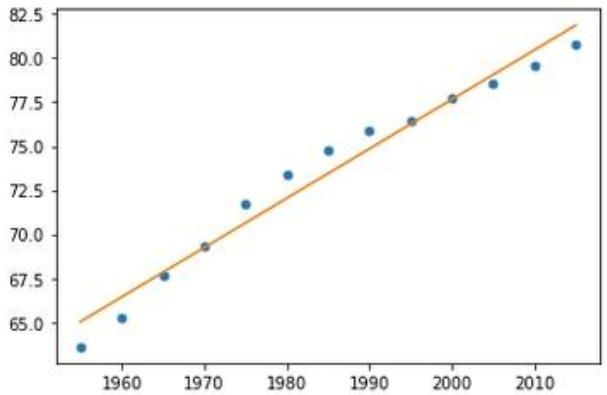
```
[2]  1  from sklearn import linear_model
     2  lr = linear_model.LinearRegression()
```

```
[3]  1  x = a[:,0].reshape(-1,1)
     2  y = a[:,1].reshape(-1,1)
     3  lr.fit(x,y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
[5]  1  plt.plot(x, y, linewidth=0, marker='o', markersize=5)
     2  yy = lr.predict(x)
     3  plt.plot(x, yy)
```

```
[<matplotlib.lines.Line2D at 0x7fa4eda7ada0>]
```

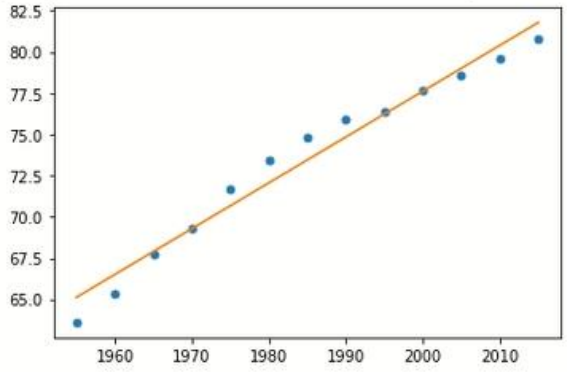


predict関数の結果をplot関数に渡して直線を描画

【発展】いろいろな回帰手法を試そう

- `linear_model.LinearRegression()` を変更するだけで様々な手法を試すことができる

```
[2]  1  from sklearn import linear_model
     2  lr = linear_model.Lasso()

[3]  1  x = a[:,0].reshape(-1,1)
     2  y = a[:,1].reshape(-1,1)
     3  lr.fit(x,y)

Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)

[4]  1  plt.plot(x, y, linewidth=0, marker='o', markersize=5)
     2  yy = lr.predict(x)
     3  plt.plot(x, yy)

[<matplotlib.lines.Line2D at 0x7f1997a64c18>]
```



Lasso回帰に変更した例

他の手法は下記URLを参照

https://bit.ly/3mZKbsq

課題

- CSVファイルを読み込んで線形回帰してみよう

- 下記URLから演習課題のノートブックを自身のGoogle Driveにコピーし，Google Collaboratoryを起動して各設問に回答すること
  - https://bit.ly/3yHGmNQ

- 演習課題を提出する際は，ノートブックのURLで共有し，そのURLを提出すること

課題提出時の注意点

- 課題提出の際には提出前に必ず以下2点を確認すること
  - Google Collaboratoryの共有設定の際には『リンクを知っている全員』にチェックを入れる
  - 課題提出時に貼り付けたURLの末尾が「?usp=sharing」となっている（共有設定を開き、「リンクのコピー」ボタンを押して共有用のURLをコピー＆ペーストする）