プログラミング演習３（Python）④ データ処理ライブラリ

１．ライブラリのインストール方法

Python では pip というパッケージ管理ツールを使って、ライブラリをインストール・管理できる。　Anaconda を使用している場合は conda というコマンドを使ってインストールできる。

例 1-1　ターミナルを開いてコマンドを実行
pip install <ライブラリ名>

例 1-2　Google Colaboratory の Notebook の場合

!pip install <ライブラリ名>

例 1-3　Anaconda を使用している場合

conda install

ライブラリー覧は

pip list

で確認できる

２．Numpy　～データの計算で使われるライブラリ～

Notebook や Python ファイルに、下記のようにライブラリを読み込む

```
import numpy as np
```

この宣言後は、as で指定する名前で numpy モジュールを使用することができる

例２－１　配列を使用する場合

```
arr = np.array([1,2,3,4])
arr
```

実行結果
```
array([1,2,3,4])
```

type()で型を確認する

実行結果

```
<class 'numpy.ndarray'>
```

多次元配列は次のように
リストの中にリストを持つように定義する

※２行４列の配列の場合
```
arr2 = np.array([[1,2,3,4][5,6,7,8]])
arr2
```

実行結果
```
array([1,2,3,4][5,6,7,8])
```

shape を使うと、どのような配列構造か確認できる

```
arr2.shape
```

実行結果
```
(2,4)
```

■要素へのアクセス

array の各要素はインデックスを指定して、
取り出せる。
先ほどの２行４列の配列は０を指定すると

１行目が取り出せる

```
print(arr2[0])
```

実行結果
[1,2,3,4]

同様に１を指定すると２行目を選択できる

```
print(arr2[1])
```

実行結果
[5,6,7,8]

行の後に列を指定して各要素を選択できる

```
print(arr2[1][2])
```

実行結果
7

スライスと言う：で範囲指定することもできる

```
print(arr2[1][1:3])
```

実行結果
[6,7]

■演算

numpy で作成したベクトルは、
次のように要素同士の演算を行うことができる

```
arr = np.array([1,2,3,4])
arr + arr
```

実行結果
array([2,4,6,8])

```
arr * arr
```

実行結果

```
array([1,4,9,16])

5 + arr
```

実行結果
```
array([6,7,8,9])

4 * arr
```

実行結果
```
array([4,8,12,16])
```

---
array 内の要素に対して演算を行うメソッド

```
arr = np.array([1,2,3,4])
```

和を計算する
```
arr.sum()
```

実行結果
```
10
```

平均を計算する

```
arr.mean()
```

実行結果
```
2.5
```

最大値を求める

```
arr.max()
```

実行結果
```
4
```

いろいろな初期化

```
arr=np.arange(10)
arr
```

実行結果
```
array([0,1,2,3,4,5,6,7,8,9])
```

reshape()を使うと多次元配列に変更できる

```
arr.reshape(2,5)
```

実行結果
```
array([[0,1,2,3,4],
[5,6,7,8,9]])
```

データ分析をするうえで
numpy を使う機会は非常に多い

■３．Pandas　～データ操作に用いられるライブラリ～

　Pandas は Python でデータ分析を行うためのライブラリで、主にデータの前処理を行うために使われることが多く、この Pandas と scikit-learn という機械学習を行うためのライブラリはシームレスにつながっているため、分析を行う上で必須のライブラリとなっている。

Pandas は pd という名前を付けて import することが多い

```
import pandas as pd
```

■Series

pandas は、Series と DataFrame という２つのデータ構造で
処理をして、
Series は一次元のデータ構造を持つ

```
a = pd.Series([2,3,4,5])
a
```

実行結果

```
0  2
1  3
2  4
3  5

dtype: int64
```

インデックスが左側、データの値が右側に出力される
numpy と同様に index を指定する

```
a[0]
```

実行結果

```
2
```

```
a[1:3]
```

実行結果

```
1  3
2  4
dtype: int64
```

算術メソッドを使って、平均や合計を求めることができる

```
a.sum()
```

実行結果

```
3.5
```

■DataFrame

DataFrame は多次元データをテーブル形式で表現できる
実際のデータ分析では、多次元データを扱うことがほとんど
のため、Series 型よりもこちらの DataFrame 形式で
Pandas を扱うことの方が多い。

```
col_1 = np.array(['A','B','A','D','E'])
col_2 = np.array([1,2,3,4,5])
col_3 = np.array([6,7,8,9,10])

df = pd.DataFrame({'col_1':col_1,'col_2':col_2,'col_3':col_3})
df.head()
```

実行結果

```
  col_1     col_2 col_3
0     A     1     6
1     B     2     7
2     A     3     8
3     D     4     9
4     E     5     10
```

```
df.shape
```

(5,3)

データフレームの各列の取り出しは
df['col_name'] または df.col_name

```
df['col_1']
```

実行結果

```
0  A
1  B
2  A
3  D
4  E
Name: col_1, dtype: object
```

番号を指定して抽出するときは iloc を使う

```
df.iroc[2]
```

実行結果

```
col_1   A
col_2   3
col_3   7
Name: 2, dtype:object
```

２以降の場合

```
df.iloc[2:]
```

実行結果

```
  col_1  col_2  col_3
2     A      3      7
3     D      4      8
4     E      5      9
```

３行２列の要素を取り出す

```
df.iloc[2,1]
```

実行

```
3
```

col_1 が A のものだけ取り出す

```
df[df['col_1'] == 'A']
```

実行結果

```
 col_1 col_2 col_3
0    A     1     5
2    A     3     7
```

query()を使ってもでる

```
df.query('col_1 == "B"')
```

実行結果

```
 col_1 col_2 col_3
1    B     2     6
```

■便利なメソッド

decsripe()を使用することで、
各カラムの平均値や標準偏差、四分位数
といった記述統計量を求めることができる。

```
df.describe()
```

実行結果

```
       col_2 col_3
count 5.000000    5.000000
mean  3.000000    8.000000
std   1.581139    1.581139
min   1.000000    6.000000
25%   2.000000    7.000000
50%   3.000000    8.000000
75%   4.000000    9.000000
max   5.000000    10.000000
```

```
df.info()
```

実行結果

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   col_1   5 non-null      object
 1   col_2   5 non-null      int64
 2   col_3   5 non-null      int64
dtypes: int64(2), object(1)
memory usage: 248.0+ bytes
```

各カラムに対して、sum()やmean()で
合計や平均が求められる

```
print(df['col_2'].sum())
print(df['col_2'].mean())
```

実行結果

```
15
3.0
```

sort_values()でデータの並び替えをする

ascending でデフォルトが昇順
fault で降順でソートできる。

```
df.sort_values('col_2',ascending=False)
```

```
      col_1 col_2 col_3
4     E     5     10
3     D     4     9
2     A     3     8
1     B     2     7
0     A     1     6
```

欠損値を外したいときは dropna()
欠損値を置換したいときは fillna()を
使用する

A を NaN に置き換える

```
df = df.replace('A', np.nan)
print(df)
```

実行結果

```
  col_1  col_2  col_3
0   NaN      1      6
1     B      2      7
2   NaN      3      8
3     D      4      9
4     E      5     10
```

```
df.dropna()
```

実行結果

```
      col_1 col_2 col_3
1     B     2     7
3     D     4     9
4     E     5     10
```

```
df.fillna(0)
```

実行結果

```
      col_1 col_2 col_3
0     0     1     6
1     B     2     7
2     0     3     8
3     D     4     9
4     E     5     10
```

データの読み込み

一般的に CSV 形式を Pandas 形式に取り込む
事が多い。

CSV 形式の読み込み　read_csv()
sep で区切り文字の指定
names でカラム名

```
df=pd.read_csv('filename',sep='¥t',names=[col1,col2,..])
```

CSV 形式の書き出し
index でデータフレームの index を含めるかの設定

```
df=to_csv('filename',sep='¥t',index=False)
```

―――――

４．可視化ライブラリ

代表的な可視化ライブラリである Matplotlib と seaborn の
可視化ライブラリを紹介する

```
from matplotlib import pyplot as plt
import seaborn as sns
plt.style.use('ggplot')
%matplotlib inline

from sklearn.datasets import load_iris
import pandas as pd
```

ここでは iris データを使って描画する

iris データについて
setosa「セトーサ」と読む、日本名：ヒオウギアヤメ
versicolor「ヴァーシカラー」と読む、日本名：ブルーフラッグ
virginica「ヴァージニカ」と読む、日本名：ヴァージニカ
の３種類のあやめのデータに
がく片長、がく片幅、花びら長、花びら幅
という４つの情報が用意されている

データセット iris の参考 URL

@IT メディア「Iris Dataset：あやめ（花びら／がく片の長さと幅の 4 項目）
の表形式データセット」
https://atmarkit.itmedia.co.jp/ait/articles/2206/13/news032.html
みんなのデータサイエンス「Iris データセット」
https://minnanods.com/iris-dataset/

iris データを読み込む

```
iris = load_iris()

df=pd.DataFrame(iris.data,columns=iris.feature_names)
df['target']=iris.target
df.loc[df['target']==0,'target']="setosa"
```

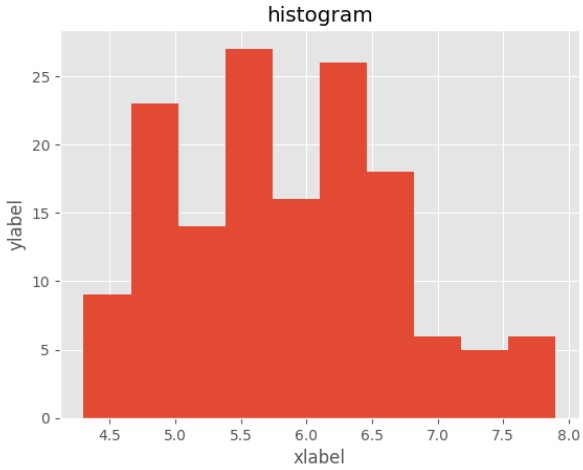
```
df.loc[df['target']==1,'target']="versicolor"
df.loc[df['target']==2,'target']="virginica"
```

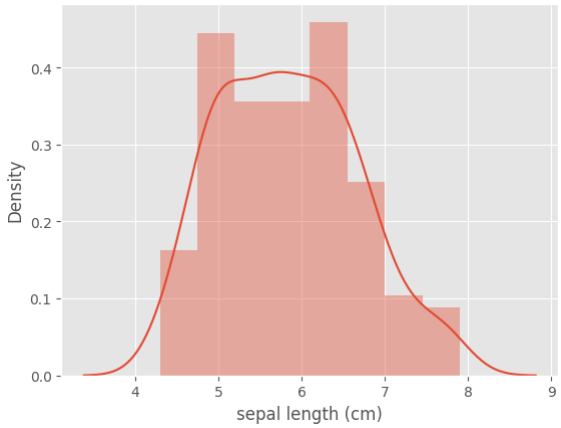
ヒストグラムを描画する

matplotlib でヒストグラムを作成するには plt.hist()を使う
タイトルやｘ軸、ｙ軸のラベル名を追加することもできる
seaborn では distplot()を使う

```
plt.title('histogram')
plt.xlabel('xlabel')
plt.ylabel('ylabel')
plt.hist(df['sepal length (cm)'])
plt.show()
```

```
sns.distplot(df['sepal length (cm)'])
```

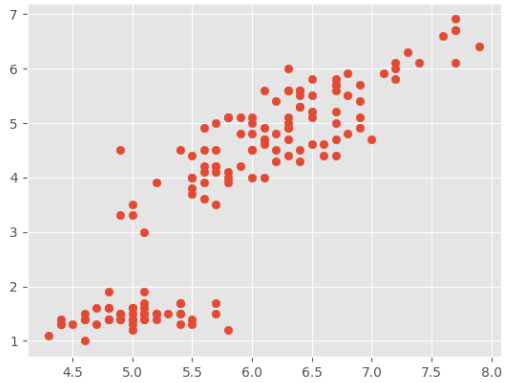




散布図を描画する

matplolib では scatter()
を使用する

```
plt.scatter(df['sepal length (cm)'],df['petal length (cm)'])
plt.show()
```



折れ線グラフを描画する

```
x=[1,2,3,4]
y=[2,4,3,8]

plt.plot(x,y)
plt.show()
```