2021年度 東京成徳大学 特別講座 資料
東京工科大学コンピュータサイエンス学部
福西広晃

第04回 Pythonによるデータ可視化

---

matplotlibについて

概要

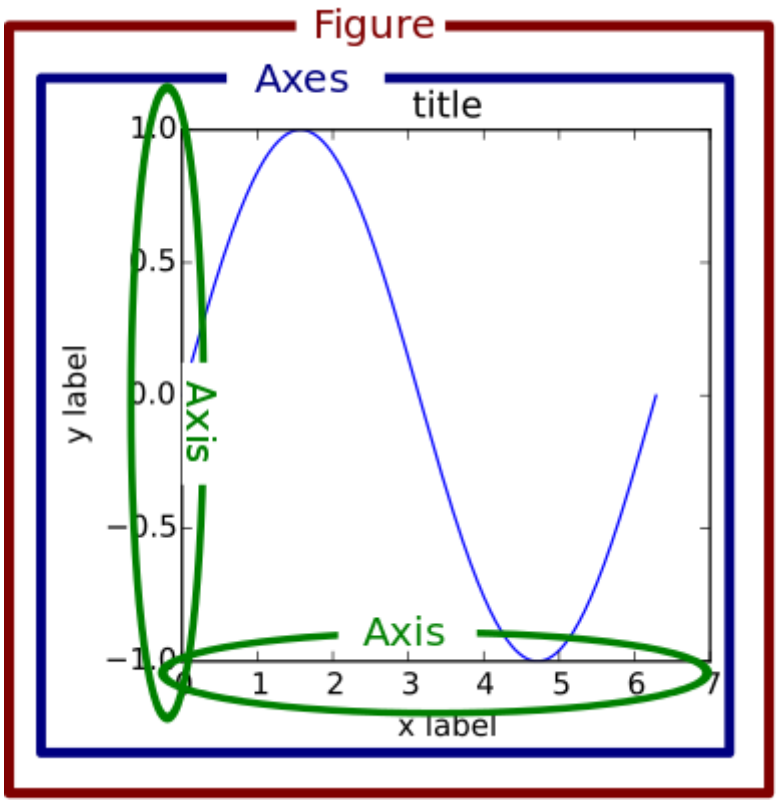
- matplotlibはPythonにおけるデータ可視化のための代表的なライブラリ
- Excelなどを使うよりも自由度の高く美しいグラフを書くことできる

matplotlib.pyplotのインポート

- グラフ作成のためにmatplotlibを利用する場合は、matplotlib.pyplotをインポートする。その際、 `plt` という名前を付けて使用するのが慣習である
  ```
  import matplotlib.pyplot as plt
  ```
- jupyterの中でnotebookの中にグラフを出力させるために `matplotlib inline` も同時に記述

matplotlibにおけるグラフの構造

- matplotlibのグラフは、FigureオブジェクトとAxesオブジェクトによって構成される
  - Figureオブジェクト：グラフを描く領域を表すオブジェクト(=描画領域)
  - Axesオブジェクト：グラフ自体を表すオブジェクト
- 下図のようにFigureオブジェクトにAxesオブジェクトを配置し、それぞれを修飾することでグラフを作成する。AxesオブジェクトはFigureオブジェクトに複数配置することも可能。



グラフの書き方の２つの方法

- pyplotインターフェース による方法 (1つのグラフを簡易に作成する方法)
  - Figureオブジェクト、Axesオブジェクトを意識せずにメソッド(作図機能)を用いてグラフを作成

- オブジェクト指向インターフェース による方法 (複数のグラフを並べて描くなど凝った方法)
  - Figureオブジェクト、Axesオブジェクトを段階的に作成して、グラフを作成
  - これにより、Figureオブジェクトの中に、複数のAxesオブジェクトを作成すること（つまり複数グラフを並べる）も可能になる

準備

- matplotlibではグラフに日本語が使用できない。日本語を使えるようにするためには `japanize-matplotlib` をインストールして、グラフ作成の際にインポートする必要がある。
- インストール方法 (Google Colaboraotyでは初期インストールされていないのでインストールが必要)
  `pip install japanize-matplotlib`

- インストールしていない場合は、以下のコメントアウトを外して実行

In [ ]:
```
pip install japanize-matplotlib
```

- Google Colaboratoryにgoogleドライブ上のファイルを読み込みたい場合（またはデータを

出力したい場合）は以下を実行しておく必要がある（毎回の作業前）

- Go to this URL in a browserの後に記載されているリンク先をクリックして、指示通りに許可やログインをすると、コードが表示される
- コードをコピーして、Enter your authorization codeに張り付けてEnterキーを押す
- Colab Notebooksフォルダにdatasetフォルダを作成し、AAA.csvを置いたとする。これをpandasで読み込む場合の例は、
  `df = pd.read_csv("drive/My Drive/Colab Notebooks/dataset/AAA.csv", encoding="shift_jis")`

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

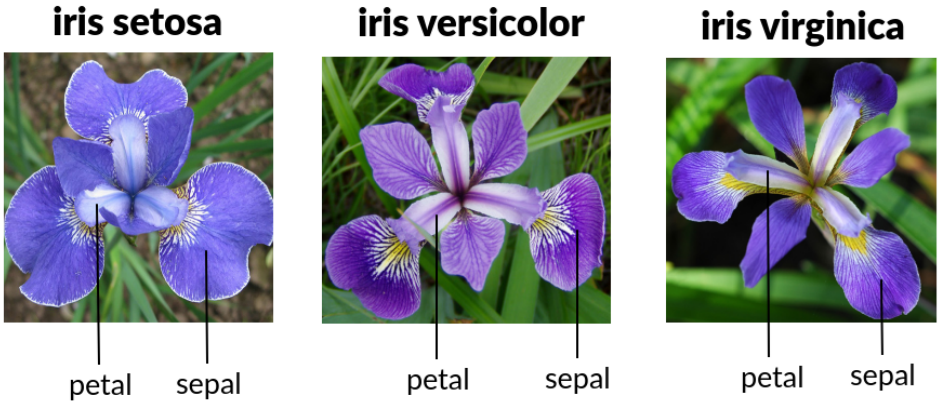
---

pyplotインターフェースによるグラフ作成

- pyplotインターフェースによる方法 (1つのグラフを簡易に作成する方法)
  - Figureオブジェクト、Axesオブジェクトを意識せずにメソッドを用いてグラフを作成

グラフ作成用のデータの読み込み

- データ：iris（あやめ）データ
  - sepal_length: がくの長さ
  - sepal_width : がくの幅
  - petal_length: 花弁の長さ
  - petal_width : 花弁の幅
  - species : 品種 (setosa,versicolor,virginica)



データの可視化に用いるデータは Pandas （前回学習）に格納します

In [ ]:
```
import pandas as pd # pandasを利用するためにインポート

# ファイルの指定
# Google Colaboratoryの場合
file = "drive/My Drive/Colab Notebooks/dataset/iris.xlsx"

# Anacondaの場合(Google Colaboratoryの場合は削除してください)
file = "dataset/iris.xlsx"

# データの読み込み
df_iris = pd.read_excel(file, sheet_name='iris')
```

```
# データの表示
display(df_iris)
```

品種ごとのサンプル数(行数)確認(ユニークな値の出現頻度)

- データフレームから対象の１列を抽出（シリーズ型になる）
- シリーズ名.value_counts()
- 出力もシリーズ型

In [ ]:
```
df_iris["species"].value_counts()
```

品種ごとにデータ抽出

In [ ]:
```
TrueFalse = df_iris["species"] == "setosa"          # setosaの行の真偽を判定
df_setosa = df_iris[TrueFalse]                      # True（真）の行を抽出
display(df_setosa.head(3))                          # 先頭3行を表示

TrueFalse     = df_iris["species"] == "versicolor" # versicolorの行の真偽を判定
df_versicolor = df_iris[TrueFalse]                  # True（真）の行を抽出
display(df_versicolor.head(3))                      # 先頭3行を表示

TrueFalse    = df_iris["species"] == "virginica"  # virginicaの行の真偽を判定
df_virginica = df_iris[TrueFalse]                 # True（真）の行を抽出
display(df_virginica.head(3))                     # 先頭3行を表示
```

散布図の作成

- 散布図を例にmatplotlibの使い方を学習する

装飾なしのグラフの生成

- 散布図は2次元データ(pandasの2列)が入力となる
- 今回は がくの長さ (sepal_length)と がくの幅 (sepal_width)の関係を表す散布図を作成
- 散布図は scatter メソッドを利用

  plt.scatter(x,y)

In [ ]:
```
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline

# 使用する列（項目）を抽出
x1 = df_setosa["sepal_length"] # がくの長さ（シリーズ型のデータ）
y1 = df_setosa["sepal_width"]  # がくの幅   （シリーズ型のデータ）

# グラフ作成
plt.scatter(x1,y1)

plt.show()        # 付けなくても表示できるが、グラフ以外の情報も出力されるので記述した
```

質問

グラフをレポートに載せたい場合、上記のグラフは適切ではありません。上記のグラフを見て足りない要素を挙げてみよう。

タイトルや軸ラベルを追加

- タイトルの追加: `plt.title("タイトル名")`
- x軸のラベルの追加: `plt.xlabel("タイトル名")`
- y軸のラベルの追加: `plt.ylabel("タイトル名")`

- **レポート用にグラフを作成する場合は、グラフタイトル、軸のラベルを付けるのがお作法です**

In [ ]:
```
# グラフ作成
plt.scatter(x1,y1)

# 追加オプション
plt.title("がくの長さと幅の関係") # タイトルを追加
plt.xlabel("がくの長さ(cm)")      # x軸のラベルを追加
plt.ylabel("がくの幅(cm)")        # y軸のラベルを追加

plt.show()
```

【注意】追加オプションは、保存されないので、グラフを描くたびに設定する必要がある

質問
上記のグラフは見やすいですか。理由とともに答えてください。

フォントサイズを変更

- タイトル: `plt.title("タイトル名", fontsize=値)`

- x軸のラベル: `plt.xlabel("ラベル名", fontsize=値)`
- y軸のラベル: `plt.ylabel("ラベル名", fontsize=値)`

- x軸の目盛: `plt.xticks(fontsize=値)`
- y軸の目盛: `plt.xticks(fontsize=値)`

- **レポート用にグラフを作成する場合は、文字を大きくして見やすくしましょう**

In [ ]:
```
# グラフ作成
plt.scatter(x1,y1)

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更

plt.show()
```

補助線の追加

- 補助線の追加: `plt.grid()`

- 横の補助線のみでよい場合は引数に axis="y" を追加: plt.grid(axis="y")
- 縦の補助線のみでよい場合は引数に axis="x" を追加: plt.grid(axis="x")

- **レポート用にグラフを作成する場合は、補助線を付けると値が読み取りやすくなります**

In [ ]:
```
# グラフ作成
plt.scatter(x1,y1)

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid() # 補助線を追加

plt.show()
```

複数のデータの重ね合わせ

- データごとに scatter メソッドを利用
- plt.scatter(x,y) を続けて記述すると複数のデータが重なる

In [ ]:
```
x2 = df_versicolor["sepal_length"] # がくの長さ（シリーズ型のデータ）
y2 = df_versicolor["sepal_width"]  # がくの幅　（シリーズ型のデータ）

x3 = df_virginica["sepal_length"] # がくの長さ（シリーズ型のデータ）
y3 = df_virginica["sepal_width"]  # がくの幅　（シリーズ型のデータ）

# グラフ作成
plt.scatter(x1,y1) # 1つ目のデータ
plt.scatter(x2,y2) # 2つ目のデータ
plt.scatter(x3,y3) # 3つ目のデータ

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.grid() # 補助線を追加
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更

plt.show()
```

質問
上記のグラフを読み取るときに困ることはありませんか

凡例を追加

- まず凡例に表示するにレベルを付けるために、 scatter メソッドの引数に label=ラベル名 を追加
  plt.scatter(x,y,label="ラベル名")
- その後、凡例を追加： plt.legend()
- フォントサイズを指定する場合： plt.legend(fontsize=値)

- 位置を指定する場合: `plt.legend(loc="位置")
  - 位置の選択肢: `best`, `upper right`, `upper left`, `lower left`, `lower right`, `right`, `center left`, `center right`, `lower center`, `upper center`, `center`
  - `best` は適切な場所を選んでくれる

- **レポート用にグラフを作成する場合、複数のデータが重なっていれば凡例が必要です**

In [ ]:
```
# グラフ作成
plt.scatter(x1,y1, label="setosa")     # 1つ目のデータに凡例用ラベルを追加
plt.scatter(x2,y2, label="versicolor") # 2つ目のデータに凡例用ラベルを追加
plt.scatter(x3,y3, label="virginica")  # 3つ目のデータに凡例用ラベルを追加

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.grid() # 補助線を追加
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.legend(fontsize=10, loc="best") # 凡例を追加

plt.show()
```

質問 上記のグラフを見てがくの長さと幅の関係性が強い品種はどれでしょうか

相関係数の計算

- 2つの項目の関係性の強さを定量的に表す指標として 相関係数 がある
- 相関係数を計算するライブラリ: scipy.stats.pearsonrのインポート

  `from scipy.stats import pearsonr`
- 相関係数（r）の計算

  `相関係数, p値 = pearsonr(x, y)`
- 相関係数（r）の意味付け
  - 0.7 < r < 1.0 強い正の相関
  - 0.4 < r < 0.7 適度な正の相関
  - 0.2 < r < 0.4 弱い正の相関
  - -0.2 < r < 0.2 相関なし
  - -0.4 < r < -0.2 弱い負の相関
  - -0.7 < r < -0.4 適度な負の相関
  - -1.0 < r < -0.7 強い負の相関

In [ ]:
```
from scipy.stats import pearsonr

# 相関係数の計算
corr1, p1 = pearsonr(x1, y1) # setosa
corr2, p2 = pearsonr(x2, y2) # versicolor
corr3, p3 = pearsonr(x3, y3) # virginica

# 相関係数の表示
print("setosa:", corr1)
print("versicolor:", corr2)
print("virginica:", corr3)
```

上記の相関係数を比較すると、がくの長さと幅の関係が最も強い品種はsetosa。
つまり、がくの長さが長いと幅も広がる。

色の変更

- scatterメソッドの引数にcolor="色"を追加
  `plt.scatter(x,y, color="色")`
- 代表的な色の種類
  - 赤: red
  - 青: blue
  - 緑: green
  - 黒: black

In [ ]:
```
# グラフ作成
plt.scatter(x1,y1, label="setosa", color="red")      # 1つ目のデータの色を変更
plt.scatter(x2,y2, label="versicolor", color="blue") # 2つ目のデータの色を変更
plt.scatter(x3,y3, label="virginica", color="green") # 3つ目のデータの色を変更

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid() # 補助線を追加
plt.legend(fontsize=10, loc="best") # 凡例を追加

plt.show()
```

グラフのサイズの変更

- `plt.figure(figsize=(横のインチ, 縦のインチ))` をグラフ作成
  （ `plt.scatter(x,y)` ）前に記述

In [ ]:
```
plt.figure(figsize=(8, 6)) # グラフサイズの変更

# グラフ作成
plt.scatter(x1,y1, label="setosa", color="red") # 1つ目のデータ
plt.scatter(x2,y2, label="versicolor", color="blue") # 2つ目のデータ
plt.scatter(x3,y3, label="virginica", color="green") # 3つ目のデータ

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid() # 補助線を追加
plt.legend(fontsize=10, loc="best") # 凡例を追加

plt.show()
```

目盛の調整

- x軸の表示範囲を調整
  `plt.xlim([最小値, 最大値])` 最小値はグラフの左隅にマージンを取るために設定した

い値より0.1程度引いた値がよい
- x軸の目盛位置を調整
  `plt.xticks(目盛位置のリスト)`

- y軸の表示範囲を調整
  `plt.xlim([最小値, 最大値])` 最小値はグラフの左隅にマージンを取るために設定したい値より0.1程度引いた値がよい
- y軸の目盛位置を調整
  `plt.xticks(目盛位置のリスト)`

In [ ]:
```
# グラフ作成
plt.scatter(x1,y1, label="setosa", color="red") # 1つ目のデータ
plt.scatter(x2,y2, label="versicolor", color="blue") # 2つ目のデータ
plt.scatter(x3,y3, label="virginica", color="green") # 3つ目のデータ

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid() # 補助線を追加
plt.legend(fontsize=10, loc="best") # 凡例を追加

plt.xlim([3.9, 8]) # x軸の表示範囲を変更
plt.xticks([4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0], fontsize=15) # x軸の目盛位置
plt.ylim([1.9, 5]) # y軸の表示範囲を変更
plt.yticks([2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0], fontsize=15) # y軸の目盛位置を変更

plt.show()
```

【参考】以下では目盛の間隔を内包表記で指定

In [ ]:
```
# グラフ作成
plt.scatter(x1,y1, label="setosa", color="red") # 1つ目のデータ
plt.scatter(x2,y2, label="versicolor", color="blue") # 2つ目のデータ
plt.scatter(x3,y3, label="virginica", color="green") # 3つ目のデータ

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid() # 補助線を追加
plt.legend(fontsize=10, loc="best") # 凡例を追加

plt.xlim([3.9, 8]) # x軸の目盛範囲を変更
plt.xticks([4 + 0.5*i for i in range(9)], fontsize=15) # x軸の間隔を変更
plt.ylim([1.9, 4.5]) # y軸の目盛範囲を変更
plt.yticks([2 + 0.5*i for i in range(7)], fontsize=15) # y軸の間隔を変更

plt.show()
```

グラフの画像を保存

- 保存形式の種類: `emf`, `eps`, `jpeg`, `jpg`, `pdf`, `png`, `ps`, `raw`, `rgba`, `svg`, `svgz`, `tif`, `tiff`

    - png 形式で保存することが多い
- plt.savefig("画像ファイル名.png")
    - このプログラムファイルと同じフォルダに保存される
    - 別のフォルダを指定したい場合はパスを記述
        - (例) plt.savefig("グラフ/画像ファイル名.png")
    - 画像のx軸ラベルやy軸ラベルが切れていることが、 bbox_inches="tight" を引数に入れておくと切れない
        - plt.savefig("画像ファイル名.png", bbox_inches="tight")

In [ ]:
```
plt.figure(figsize=(8, 6)) # グラフサイズの変更

# グラフ作成
plt.scatter(x1,y1, label="setosa", color="red") # 1つ目のデータ
plt.scatter(x2,y2, label="versicolor", color="blue") # 2つ目のデータ
plt.scatter(x3,y3, label="virginica", color="green") # 3つ目のデータ

# 追加オプション
plt.title("がくの長さと幅の関係", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("がくの幅(cm)",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid() # 補助線を追加
plt.legend(fontsize=10) # 凡例を追加

plt.xlim([3.9, 8]) # x軸の表示範囲を変更
plt.xticks([4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0], fontsize=15) # x軸の目盛位置
plt.ylim([1.9, 5]) # y軸の表示範囲を変更
plt.yticks([2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0], fontsize=15) # y軸の目盛位置を変更

# 出力ファイル名を指定（Googleドライブの場合）
outfile = "drive/My Drive/Colab Notebooks/グラフ_散布図_がくの長さと幅の関係.png"

# # 出力ファイル名を指定（Anacondaの場合, Google Colaboratoryの場合は削除してください）
outfile = "グラフ_散布図_がくの長さと幅の関係.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

plt.show()
```

練習

(1) 花びら（petal）の長さと幅の関係を表す散布図を以下の条件で作成してみよう。品種ごとに使用する列（項目）は記述しているので、それ以外を追加するものとする。

- setosa, versicolor, virginicaの3品種のデータを重ねる
- タイトルと軸のラベルを入れる
- グラフのフォントサイズを調整する（見やすいように調整してください）
- 補助線を入れる
- 凡例を入れる
- グラフの画像を「グラフ_散布図_花びらの長さと幅の関係.png」という名前で保存する

In [ ]:
```
# 品種ごとに使用する列（項目）を抽出
x11 = df_setosa["petal_length"] # 花びらの長さ（Series型のデータ）
y11 = df_setosa["petal_width"]  # 花びらの幅　（Series型のデータ）

x12 = df_versicolor["petal_length"] # 花びらの長さ（Series型のデータ）
```

```
y12 = df_versicolor["petal_width"]  # 花びらの幅   (Series型のデータ)

x13 = df_virginica["petal_length"] # 花びらの長さ (Series型のデータ)
y13 = df_virginica["petal_width"]  # 花びらの幅   (Series型のデータ)
```

(2) 花びらの長さと幅の関係性が強い品種はどれかを相関係数を計算して判定してみよう

In [ ]:

解答例

In [ ]:

```
#(1)
# 品種ごとに使用する列 (項目) を抽出
x11 = df_setosa["petal_length"] # 花びらの長さ (Series型のデータ)
y11 = df_setosa["petal_width"]  # 花びらの幅   (Series型のデータ)

x12 = df_versicolor["petal_length"] # 花びらの長さ (Series型のデータ)
y12 = df_versicolor["petal_width"]  # 花びらの幅   (Series型のデータ)

x13 = df_virginica["petal_length"] # 花びらの長さ (Series型のデータ)
y13 = df_virginica["petal_width"]  # 花びらの幅   (Series型のデータ)

# グラフ作成
plt.scatter(x11,y11, label="setosa", color="red") # 1つ目のデータ
plt.scatter(x12,y12, label="versicolor", color="blue") # 2つ目のデータ
plt.scatter(x13,y13, label="virginica", color="green") # 3つ目のデータ

# 追加オプション
plt.title("花びらの長さと幅の関係", fontsize=20) # タイトルを追加 (フォントサイズ指定)
plt.xlabel("花びらの長さ(cm)", fontsize=15) # x軸のラベルを追加 (フォントサイズ指定)
plt.ylabel("花びらの幅(cm)",  fontsize=15)  # y軸のラベルを追加 (フォントサイズ指定)
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid() # 補助線を追加
plt.legend(fontsize=15, loc="best") # 凡例を追加

# 出力ファイル名を指定 (Googleドライブの場合)
outfile = "drive/My Drive/Colab Notebooks/グラフ_散布図_花びらの長さと幅の関係.png"

# # 出力ファイル名を指定 (Anacondaの場合, Google Colaboratoryの場合は削除してください)
outfile = "グラフ_散布図_花びらの長さと幅の関係.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

plt.show()
```

In [ ]:

```
# (2)
from scipy.stats import pearsonr

# 相関係数の計算
corr1, p1 = pearsonr(x11, y11) # setosa
corr2, p2 = pearsonr(x12, y12) # versicolor
corr3, p3 = pearsonr(x13, y13) # virginica

# 相関係数の表示
print("setosa:", corr1)
print("versicolor:", corr2)
print("virginica:", corr3)
```

ヒストグラムの作成

- ヒストグラムはデータの分布(ばらつき)を確認するための可視化方法

- ヒストグラムの作成おいて、階級数は分析者が任意に定めることができるが、その値を決めるのは簡単ではない。階級数を多くても少なくてもデータの分布を正しく表現できない。そこで、階級数の目安となる公式に **スタージェスの公式** や **フリードマン＝ダイアコニスの法則の公式** がある。 **スタージェスの公式** は、データ数200未満ではよく機能するが、それをお超えると正確性が落ちる。

  データXのデータ数がN個のときの階級数の目安は以下の式で表される

  スタージェスの公式：階級数 $= 1 + \log_2 N$

  フリードマン＝ダイアコニスの法則の公式：階級数 $= 2 \times X$の四分位範囲 $+ N^{-\frac{1}{3}}$

5 10 **15** 25 30 **40** 60 70 **89** 90 98 第1四分位　第2四分位（中央値）　第3四分位	四分位範囲(IQR) = 第3四分位数 － 第1四分位数=89－15=64

データを読み込みと品種ごとのデータ作成（散布図で行った作業の同一）

In [ ]:
```
import pandas as pd # pandasを利用するためにインポート

# ファイルの指定
# Google Colaboratoryの場合
file = "drive/My Drive/Colab Notebooks/dataset/iris.xlsx"

# Anacondaの場合(Google Colaboratoryの場合は削除してください)
file = "dataset/iris.xlsx"

# データの読み込み
df_iris = pd.read_excel(file, sheet_name='iris')

TrueFalse = df_iris["species"] == "setosa"          # setosaの行の真偽を判定
df_setosa = df_iris[TrueFalse]                      # True（真）の行を抽出
display(df_setosa.head(3))                          # 先頭3行を表示

TrueFalse     = df_iris["species"] == "versicolor" # versicolorの行の真偽を判定
df_versicolor = df_iris[TrueFalse]                 # True（真）の行を抽出
display(df_versicolor.head(3))                     # 先頭3行を表示

TrueFalse    = df_iris["species"] == "virginica"  # virginicaの行の真偽を判定
df_virginica = df_iris[TrueFalse]                 # True（真）の行を抽出
display(df_virginica.head(3))                     # 先頭3行を表示
```

装飾なしのグラフの生成

- ヒストグラムは1次元データ(pandasの1列)が入力となる
- ここでは **がくの長さ** (sepal_length)のヒストグラムの作成
- ヒストグラムは `hist` メソッドを利用。デフォルトでは階級数10のヒストグラムが生成される
  `plt.hist(x)`

In [ ]:
```
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline
```

```
x1 = df_setosa["sepal_length"] # がくの長さ（シリーズ型のデータ）

plt.hist(x1)      # グラフ生成

plt.show()        # 付けなくても表示できるが、グラフ以外の情報も出力されるので記述した
```

上の図では、バー（bar）の区切りがないため見づらい

バー（bar）に区切りを入れる

- パターン①: 枠を入れる
  引数に ec = "色 を設定（色は黒(black)が見やすい）
  `plt.hist(x, ec="色")`
- パターン②: 棒の間に隙間を入れる場合
  引数に rwidth = 0から1の値 を設定(0.9くらいでよい)
  `plt.hist(x, rwidth=0.9)`

バー（bar）に黒枠を入れる場合

In [ ]:
```
plt.hist(x1, ec="black") # グラフ生成

plt.show()
```

バー(bar)の間に隙間を入れる場合

In [ ]:
```
plt.hist(x1, rwidth=0.9) # グラフ生成

plt.show()        # 付けなくても表示できるが、グラフ以外の情報も出力されるので記述した
```

階級数の設定

- 階級数または階級を指定したい場合は, bins=階級数(int型) または bins=階級(リスト型) を追加する

階級数20の場合(デフォルト10)

In [ ]:
```
plt.hist(x1, ec="black", bins=20) # グラフ生成
plt.show()
```

階級数3の場合(デフォルト10)

In [ ]:
```
plt.hist(x1, ec="black", bins=3) # グラフ生成
plt.show()
```

階級数の自動設定

- 引数で bins="auto" を指定すると、目安となる階級数を自動で設定してくれる。階級数は次のように設定される。 スタージェスの公式 と フリードマン＝ダイアコニスの法則（FDの法則）の公式の計算結果の内、両者ともに0より大きい場合は階級幅が小さいほう(階級数が多い)の階級数が設定され、 FDの法則 でゼロの場合は スタージェスの公式 の階級数が設定される
  `plt.hist(x, bins="auto")`

In [ ]:
```
plt.hist(x1, ec="black", bins="auto") # グラフ生成
plt.show()
```

上記のヒストグラムでは階級数が7と設定されている

【参考】階級数がスタージェスの公式かFDの公式のどちらが使われたかを確認

In [ ]:
```
import numpy as np

# データ数
num = len(df_setosa)
print("データ数:", num)

# スタージェスの公式による階級数
bins_Sturges = 1 + np.log2(num)
print("階級数(スタージェスの公式):", round(bins_Sturges))

# フリードマン＝ダイアコニスの法則の公式
q75, q25 = np.percentile(x1, [75 ,25])
iqr = q75 - q25
#print("IQR:", iqr)
bins_FD = 2 * iqr * num**(-1.0/3.0)
print("階級数(フリードマン＝ダイアコニスの法則の公式):", round(bins_FD))
```

今回はFDの法則がゼロなので、スタージェスの公式で計算される7が階級数となっている

階級（データ区間）で設定

- 特定の階級で集計したい場合、引数の bins で境界の値をリストで設定する
- 例えば、4.0-4.5, 4.5-5.0, 5.0-5.5, 5.5-6.0の階級をリストで記述する場合

  bins = [4.0, 4.5, 5.0, 5.4, 6.0]

In [ ]:
```
# 階級をリストで設定
list_bins = [4.0, 4.5, 5.0, 5.4, 6.0]

plt.hist(x1, ec="black", bins=list_bins) # グラフ生成
plt.show()
```

ヒストグラムの集計結果の取得

- 作成されたヒストグラムの集計値は、 plt.hist() の戻り値で取得する
- 次のように３つの変数名で戻り値を受け取ることが多い
  - n, bins, patches = plt.hist(x)
  - n : 各区間のデータの個数（度数）
  - bins : データ区間の境界値のリスト（階級）
  - patches : ヒストグラムのバーを表すオブジェクトのリスト（特に使用しない）

In [ ]:
```
n, bins, patches = plt.hist(x1, ec="black", bins="auto") # グラフ生成
plt.show()

print("階級", bins)
print("頻度", n)
```

質問

- グラフをレポートに載せたい場合、上記のグラフは適切ではありません。上記のグラフを見て足りない要素を挙げてみよう。散布図の作成で行った作業を思い出してください。

レポート用のグラフ作成のためのオプション設定

- 今回は以下を設定(散布図と同様)
  - グラフサイズの変更: `plt.figure(figure=(横インチ, 縦インチ))` グラフ作成前に記述
  - タイトル名追加とフォントサイズ変更: `plt.title(タイトル名, fontsize=値)`
  - x軸のラベル名追加とフォントサイズ変更: `plt.xlabel(ラベル名, fontsize=値)`
  - y軸のラベル名追加とフォントサイズ変更: `plt.ylabel(ラベル名, fontsize=値)`
  - x軸の目盛のフォントサイズ変更: `plt.xticks(fontsize=値)`
  - y軸の目盛フォントサイズ変更: `plt.yticks(fontsize=値)`
  - 横方向の補助線追加: `plt.grid(axis="y")`
    - 補助線はデフォルトではグラフの前面に引かれるので見た目に違和感を感じることがある。そこで、補助線を背面に移動させたい場合はグラフの作成前（ `plt.hist(x)` の実行前 ）に以下の記述を追加すればよい。
      `plt.rcParams["axes.axisbelow"] = True`
    - `plt.rcParam["パラメータ"]` はグラフ作成の基本設定の変更に使われるもので、一度実行するとプログラムファイルを閉じるまで有効となる
  - グラフの画像を保存: `plt.savefig("ファイル名", , bbox_inches="tight")`

In [ ]:
```
# グラフ作成前に設定するオプション
plt.rcParams["axes.axisbelow"] = True # 補助線を背面に表示
plt.figure(figsize=(8,4)) # グラフのサイズを横長に変更

# グラフ作成
plt.hist(x1, ec="black", bins="auto")

# 追加オプション
plt.title("がくの長さのヒストグラム", fontsize=20) # タイトルを追加（フォントサイズ指
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("頻度",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid(axis="y")

# 出力ファイル名を指定（Googleドライブの場合）
outfile = "drive/My Drive/Colab Notebooks/グラフ_ヒストグラム_がくの長さ.png"

# # 出力ファイル名を指定（Anacondaの場合, Google Colaboratoryの場合は削除してください）
outfile = "グラフ_ヒストグラム_がくの長さ.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

plt.show()
```

練習

上記の「レポート用のグラフ作成のためのオプション設定」の内容を参考にして、品種setosaのがくの幅（petal_width）のヒストグラムを作成してみよう。使用する列（項目）は記述しているので、それ以外を追加するものとする。

In [ ]:
```
# 使用する列（項目）の抽出
x = df_setosa["sepal_width"] # がくの幅（シリーズ型のデータ）
```

解答例

In [ ]:
```
# 使用する列（項目）の抽出
x = df_setosa["sepal_width"] # がくの幅（シリーズ型のデータ）

# グラフ作成前に設定するオプション
plt.rcParams["axes.axisbelow"] = True # 補助線を背面に表示
plt.figure(figsize=(8,4)) # グラフのサイズを横長に変更

# グラフ作成
plt.hist(x, ec="black", bins="auto")

# 追加オプション
plt.title("がくの幅のヒストグラム", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("がくの幅(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("頻度",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid(axis="y")

# 出力ファイル名を指定（Googleドライブの場合）
outfile = "drive/My Drive/Colab Notebooks/グラフ_ヒストグラム_がくの幅.png"

# # 出力ファイル名を指定（Anacondaの場合，Google Colaboratoryの場合は削除してください）
outfile = "グラフ_ヒストグラム_がくの幅.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

plt.show()
```

複数のデータの重ね合わせ

- `plt.hist(...)` を続けて記述すると重ね合わされる
- `plt.hist(...)` の引数に以下を追加
  - 凡例を入れるための `label=ラベル名`
  - 重ね合わせたときに色を透過するための `alpha=透過率`
- 凡例を入れるために `plt.legend(...)` を追加
- グラフの画像を保存するために `plt.savefig(...)` を追加

In [ ]:
```
# 使用する列(項目)の抽出
x2 = df_versicolor["sepal_length"] # がくの長さ（Series型のデータ）
x3 = df_virginica["sepal_length"]  # がくの長さ（Series型のデータ）

# グラフ作成前に設定するオプション
plt.rcParams["axes.axisbelow"] = True # 補助線を背面に表示
plt.figure(figsize=(8,4)) # グラフのサイズを横長に変更

# グラフ作成
plt.hist(x1, ec="black", bins="auto", label="setosa", alpha=0.5) # グラフ作成
plt.hist(x2, ec="black", bins="auto", label="versicolor", alpha=0.5) # グラフ作成(追加
plt.hist(x3, ec="black", bins="auto", label="virginica", alpha=0.5) # グラフ作成(追加

# 追加オプション
plt.title("がくの長さのヒストグラム", fontsize=20) # タイトルを追加（フォントサイズ指定
plt.xlabel("がくの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
```

```
plt.ylabel("頻度",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid(axis="y")
plt.legend(fontsize=15, loc="best") # 凡例を追加

# 出力ファイル名を指定（Googleドライブの場合）
outfile = "drive/My Drive/Colab Notebooks/グラフ_ヒストグラム_がくの長さ_3品種.png"

# # 出力ファイル名を指定（Anacondaの場合，Google Colaboratoryの場合は削除してください）
outfile = "グラフ_ヒストグラム_がくの長さ_3品種.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

plt.show()
```

練習

花びらの長さ（petal_length）のヒストグラムを以下の条件で作成してみよう。使用する列（項目）は記述しているので、それ以外を追加するものとする。

- setosa, versicolor, virginicaの3品種のデータを重ねる
- グラフの透過率は0.5とする
- タイトルと軸のラベルを入れる
- グラフのフォントサイズを調整する（見やすいように調整してください）
- 階級幅は自動設定する(bins="auto")
- 横方向の補助線を入れる(背面)
- 凡例を入れる
- グラフの画像を「グラフ_ヒストグラム_花びらの長さ_3品種.png」という名前を保存する

In [ ]:
```
# 使用する列(項目)の抽出
x11 = df_setosa["petal_length"] # 花びらの長さ（Series型のデータ）
x12 = df_versicolor["petal_length"] #花びらの長さ（Series型のデータ）
x13 = df_virginica["petal_length"]  # 花びらの長さ（Series型のデータ）
```

解答例

In [ ]:
```
# 使用する列(項目)の抽出
x11 = df_setosa["petal_length"] # 花びらの長さ（Series型のデータ）
x12 = df_versicolor["petal_length"] #花びらの長さ（Series型のデータ）
x13 = df_virginica["petal_length"]  # 花びらの長さ（Series型のデータ）

# グラフ作成前に設定するオプション
plt.rcParams["axes.axisbelow"] = True # 補助線を背面に表示
plt.figure(figsize=(8,4)) # グラフのサイズを横長に変更

# グラフ作成
plt.hist(x11, ec="black", bins="auto", label="setosa", alpha=0.5) # グラフ作成
plt.hist(x12, ec="black", bins="auto", label="versicolor", alpha=0.5) # グラフ作成(追
plt.hist(x13, ec="black", bins="auto", label="virginica", alpha=0.5) # グラフ作成(追加

# 追加オプション
plt.title("花びらの長さのヒストグラム", fontsize=20) # タイトルを追加（フォントサイズ
plt.xlabel("花びらの長さ(cm)", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("頻度",  fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid(axis="y")
```

```
plt.legend(fontsize=10, loc="best") # 凡例を追加

# 出力ファイル名を指定（Googleドライブの場合）
outfile = "drive/My Drive/Colab Notebooks/グラフ_ヒストグラム_花びらの長さ_3品種.png"

# # 出力ファイル名を指定（Anacondaの場合，Google Colaboratoryの場合は削除してください）
outfile = "グラフ_ヒストグラム_花びらの長さ_3品種.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

plt.show()
```

様々なグラフのサンプル

- グラフ作成で使用頻度が高い 折れ線グラフ 、 棒グラフ 、 円グラフ 、 箱ひげ図 の作成例を示す
- レポートの挿入するグラフとして最低限必要なオプションを付けて記載している

折れ線グラフ

- グラフ作成:
  `plt.bar([横軸データ(時系列)], [縦軸データ(値)])`
- マーカーの種類

In [ ]:
```
# 作図に必要なライブラリのインポート
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline

# データ準備
t = [1, 2, 3, 4, 5]   # 時系列データ
x = [1, 4, 9,16,25]   # 値

# グラフ作成
plt.plot(t,x)

# マーカーを付けたい場合は//（コメントアウト）を外す
#plt.plot(t,x, marker="o")

# 追加オプション
plt.title("タイトル", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("xラベル", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("yラベル", fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid()

# グラフ表示
plt.show()
```

複数の折れ線のグラフの重ね合わせ

In [ ]:
```
# 作図に必要なライブラリのインポート
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline
```

```
# データ準備
t = [1, 2, 3, 4, 5]   # 時系列データ
x = [1, 4, 9,16,25]   # 値1
y = [1, 2, 3, 4, 5]   # 値2
z = [2, 5, 8,10,20]   # 値3

# グラフ作成
plt.plot(t,x, label="x") # 凡例を付けるためにラベル名を設定
plt.plot(t,y, label="y") # 凡例を付けるためにラベル名を設定
plt.plot(t,z, label="z") # 凡例を付けるためにラベル名を設定

# 追加オプション
plt.title("タイトル", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("xラベル", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("yラベル", fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.legend(fontsize=15, loc="best")
plt.grid()

# グラフ表示
plt.show()
```

棒グラフ

- グラフ作成: `plt.bar([カテゴリ], [データ])`

In [ ]:

```
# 作図に必要なライブラリのインポート
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline

# データ準備
x = ['A','B','C','D','E'] # x軸はカテゴリ
y = [10, 20, 30, 20, 10]  # y軸は値

# グラフ作成
plt.bar(x, y)

# 追加オプション
plt.title("タイトル", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("xラベル", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("yラベル", fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid(axis="y")

# グラフ表示
plt.show()
```

円グラフ

- グラフ作成: `plt.pie([データ], labels=[ラベル])`
- 詳細設定(引数に追加)
  - 時計回りに設定: `counterclock=False`
  - 90度（12時）からスタート: `startangle=90`
  - 円内にパーセント表示を使い: `autopct="%.1f%%"`

In [ ]:
```
# 作図に必要なライブラリのインポート
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline

# データ準備
x = [15,24,45,55] # 頻度
label = ['AAA', 'BBB', 'CCC', 'DDD'] # ラベル

# グラフ作成
plt.pie(x, labels=label, counterclock=False, startangle=90,  autopct="%.1f%%")

# 追加オプション
plt.title("タイトル", fontsize=20) # タイトルを追加（フォントサイズ指定）

# グラフ表示
plt.show()
```

箱ひげ図

- グラフ作成: plt.boxplot([データ], labels=[ラベル])

In [ ]:
```
# 作図に必要なライブラリのインポート
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline

# データ準備
x = [1, 4, 9,16,25, 28, 30, 40, 42, 47]   # 値1
y = [1, 2, 3, 4, 5,  9, 13, 16, 20, 22]   # 値2
z = [2, 5, 8,10,17, 18, 20, 25, 33, 39]   # 値3
label = ["x", "y", "z"] # ラベル

# グラフ作成
plt.boxplot([x, y, x], labels=label)

# 追加オプション
plt.title("タイトル", fontsize=20) # タイトルを追加（フォントサイズ指定）
plt.xlabel("xラベル", fontsize=15) # x軸のラベルを追加（フォントサイズ指定）
plt.ylabel("yラベル", fontsize=15)  # y軸のラベルを追加（フォントサイズ指定）
plt.xticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.yticks(fontsize=15)  # x軸の目盛のフォントサイズ変更
plt.grid(axis="y")

plt.show()
```

練習

上記の折れ線グラフ、棒グラフ、円グラフ、箱ひげ図のいずれかを作図してみよう。データは自由に作成してよいものとする。

In [ ]:
```

```

以降、付録

オブジェクト指向インターフェースによるグラフ作成

- オブジェクト指向インターフェース による方法 (複数のグラフを並べて描くなど凝った方法)
  - Figureオブジェクト、Axesオブジェクトを段階的に作成して、グラフを作成
  - これにより、Figureオブジェクトの中に、複数のAxesオブジェクトを作成すること（つまり複数グラフを並べる）も可能になる
  - 公式はこちらの書き方を推奨している

データ読み込み

In [ ]:
```
import pandas as pd # pandasを利用するためにインポート

# ファイルの指定
# Google Colaboratoryの場合
file = "drive/My Drive/Colab Notebooks/dataset/iris.xlsx"

# Anacondaの場合(Google Colaboratoryの場合は削除してください)
file = "dataset/iris.xlsx"

# データの読み込み
df_iris = pd.read_excel(file, sheet_name='iris')

# 品種ごとのデータ抽出
TrueFalse = df_iris["species"] == "setosa" # setosaの行の真偽を判定
df_setosa = df_iris[TrueFalse] # 真の行を抽出
#display(df_setosa.head(3)) # 先頭3行を表示

TrueFalse     = df_iris["species"] == "versicolor" # versicolorの行の真偽を判定
df_versicolor = df_iris[TrueFalse] # 真の行を抽出
#display(df_versicolor.head(3)) # 先頭3行を表示

TrueFalse    = df_iris["species"] == "virginica" # virginicaの行の真偽を判定
df_virginica = df_iris[TrueFalse] # 真の行を抽出
#display(df_virginica.head(3)) # 先頭3行を表示
```

品種ごとにデータの分割

In [ ]:
```
# データ取得
x1 = df_setosa["sepal_length"] # がくの長さ（Series型のデータ）
y1 = df_setosa["sepal_width"]  # がくの幅　（Series型のデータ）
x2 = df_versicolor["sepal_length"] # がくの長さ（Series型のデータ）
y2 = df_versicolor["sepal_width"]  # がくの幅　（Series型のデータ）
x3 = df_virginica["sepal_length"] # がくの長さ（Series型のデータ）
y3 = df_virginica["sepal_width"]  # がくの幅　（Series型のデータ）
```

複数グラフの作成（レポート用）

- Matplotlibのデフォルトのグラフのスタイル（線やグリッドの種類・太さ・色など）はmatplotlibrcという設定ファイルで規定

Figureオブジェクト作成後に、Axesオブジェクトを作成する方法

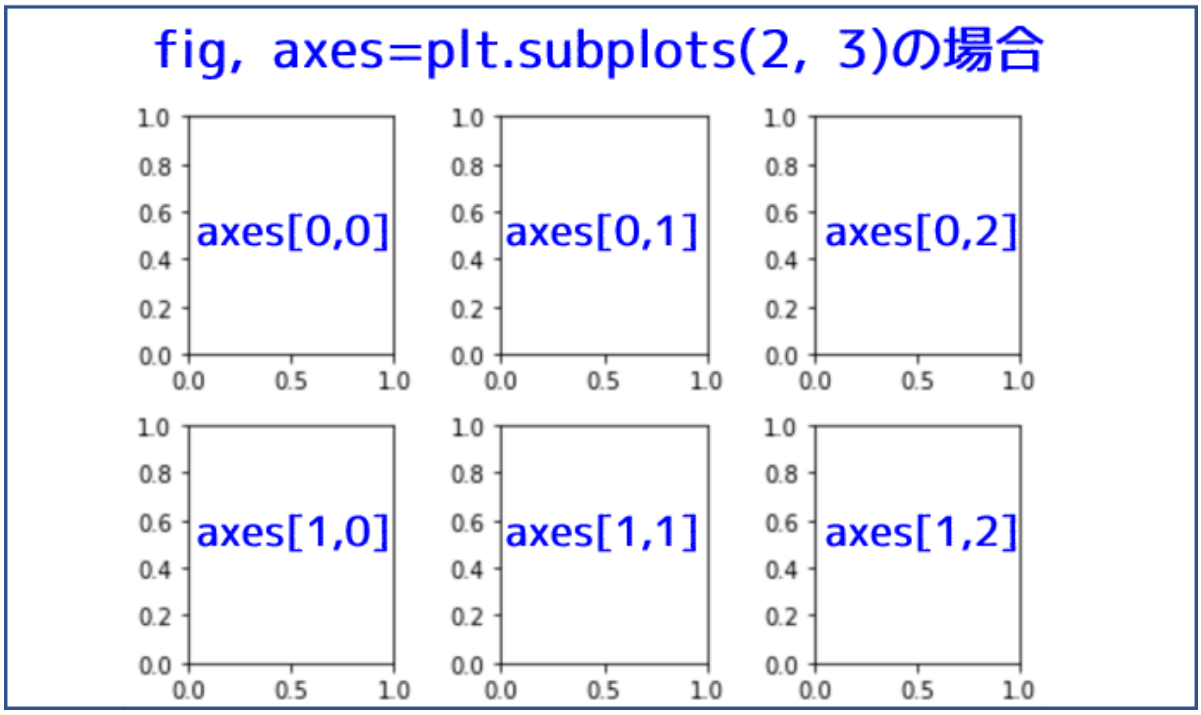
In [ ]:
```
# 作図に必要なライブラリのインポート
```

```
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline

# x軸とy軸の目盛のフォントサイズを変更
# pyplotインターフェースの場合は、plt.xticks(fontsize=インチ)で設定できたが、
# オブジェクト指向インターフェースの場合は、
# plt.rcParams["xtick.labelsize"] = インチ
# で変更するする必要あり
plt.rcParams["xtick.labelsize"] = 15
plt.rcParams["ytick.labelsize"] = 15

# Figureオブジェクトを作成
fig = plt.figure(figsize=(14, 5))

# Axiesオブジェクトを３つ生成
ax1 = fig.add_subplot(1,3,1) # ax1のグラフは1行3列に並べるうちの1番目に配置、(131)と記
ax2 = fig.add_subplot(1,3,2) # ax2のグラフは1行3列に並べるうちの1番目に配置、(132)と記
ax3 = fig.add_subplot(1,3,3) # ax3のグラフは1行3列に並べるうちの1番目に配置、(133)と記

# 各Axiesオブジェクトに３つのグラフを生成
ax1.scatter(x1, y1) # ax1のグラフの散布図を生成
ax2.scatter(x2, y2) # ax2のグラフの散布図を生成
ax3.scatter(x3, y3) # ax3のグラフの散布図を生成

# 追加オプション
# グラフタイトルの追加
ax1.set_title('setosa', fontsize=20) # ax1のグラフのタイトル
ax2.set_title('versicolor', fontsize=20) # ax2のグラフのタイトル
ax3.set_title('virginica', fontsize=20) # ax3のグラフのタイトル

# xラベルの追加
ax1.set_xlabel("がくの長さ(cm)", fontsize=15) # ax1のxラベル
ax2.set_xlabel("がくの長さ(cm)", fontsize=15) # ax2のxラベル
ax3.set_xlabel('がくの長さ(cm)', fontsize=15) # ax3のxラベル

# yラベルの追加
ax1.set_ylabel("がくの幅(cm)", fontsize=15)   # ax1のyラベル
ax2.set_ylabel("がくの幅(cm)", fontsize=15)   # ax2のyラベル
ax3.set_ylabel("がくの幅(cm)", fontsize=15)   # ax3のyラベル

# グリッド追加
ax1.grid() # ax1のグリッド
ax2.grid() # ax2のグリッド
ax3.grid() # ax3のグリッド

# tight_layout() メソッドはサブプロット間の最適な間隔を自動的に調整
# 複数のグラフを綺麗に整列させるために使用した方がよい
fig.tight_layout()

plt.show()
```

FigureオブジェクトとAxesオブジェクトを同時に作成する方法

上記では、Figureオブジェクト→Axesオブジェクトと段階的に作成したが、通常、Figureオブジェクトだけを作成することはないため、FigureオブジェクトとAxesオブジェクトを１ステップで作成する方が便利なこともある。例えば、Axesオブジェクトの配置をインデックスとして反

復処理（if文）を行う場合に記述しやすい。



In [ ]:

```
# 作図に必要なライブラリのインポート
import matplotlib.pyplot as plt # グラフ作成のためにインポート
import japanize_matplotlib      # グラフで日本語を使用するためにインポート
%matplotlib inline

# x軸とy軸の目盛のフォントサイズを変更
# pyplotインターフェースの場合は、plt.xticks(fontsize=インチ)で設定できたが、
# オブジェクト指向インターフェースの場合は、
# plt.rcParams["xtick.labelsize"] = インチ
# で変更するする必要あり
plt.rcParams["xtick.labelsize"] = 15
plt.rcParams["ytick.labelsize"] = 15

# FigureオブジェクトとAxesオブジェクトを同時に作成
# 引数ではグラフ配置を1行3列に設定
# 図のサイズ設定: figsize=(横インチ,縦インチ)
# サブプロット間の間隔を自動調整: tight_layout=True
# 1行でも２次元表記する設定: squeeze=False
fig, axes = plt.subplots(1, 3, figsize=(14,5), tight_layout=True, squeeze=False)

axes[0,0].scatter(x1, y1) # [0,0]の配置に散布図を生成
axes[0,1].scatter(x2, y2) # [0,1]の配置に散布図を生成
axes[0,2].scatter(x3, y3) # [0,2]の配置に散布図を生成

# 追加オプション
# グラフタイトルの追加
axes[0,0].set_title('setosa', fontsize=20) # グラフのタイトル
axes[0,1].set_title('versicolor', fontsize=20) # グラフのタイトル
axes[0,2].set_title('virginica', fontsize=20) # グラフのタイトル

# xラベル,yラベル, グリッドの追加
for i in range(3):
    axes[0,i].set_xlabel("がくの長さ(cm)", fontsize=15) # xラベルの追加
    axes[0,i].set_ylabel("がくの幅(cm)", fontsize=15) # yラベルの追加
    axes[0,i].grid() # グリッド追加

plt.show()
```