Python プログラミング応用

1章: データ処理

CSV

CSVはComma Separated Valuesの略で、カンマ（`,`）で値を区切ることにより表形式の情報を表すデータ形式である。CSVは、様々なアプリケーションで使用される軽量なデータ形式であり、スプレッドシートやデータベースなどでデータをエクスポートやインポートする際によく使われる。

CSV形式のファイルはテキストデータであるため、メモ帳などのテキストエディタで開くことがでる。各行にはカンマで区切られた値が列として格納されており、行と行は改行コードで区切られている。以下は、CSV形式のデータの例である。

```
ID,Name,Email
1,Alice,alice@example.com
2,Bob,bob@example.com
3,Eve,eve@example.com
```

上記のCSVデータには4つの行があり、各行には3つの列が存在している。1行目は「ヘッダ」（header）と呼ばれる特別な行であり、各列の名前を定義する見出しを表す。なお、ヘッダはCSVにおいて必須ではなく省略可能である。各列の値を順番に見ていこう。1列目には各行の`ID`が格納されており、2行目には`1`が、3行目には`2`が、4行目には`3`が記述されている。2列目には`Name`が格納されており、2行目には`Alice`が、3行目には`Bob`が、4行目には`Eve`が記述されている。2列目には`Email`が格納されており、2行目には`alice@example.com`が、3行目には`bob@example.com`が、4行目には`eve@example.com`が記述されている。

上記のCSVデータは、以下のような行と列で構成される表であるとみなせる。

ID	Name	Email
1	Alice	alice@example.com
2	Bob	bob@example.com
3	Eve	eve@example.com

CSVデータでは、値を区切る文字（区切り文字）としてカンマを用いて、また、行を区切るために改行を用いるため、値の中にカンマや改行をそのまま含めることはできない。そのため、値の中にカンマや改行を含める場合は、値をダブルクオーテーション（`"`）で囲む必要がある。

たとえば以下のように、1列目が`ID`、2列目が`Message`で構成されるCSVデータにおいて、2列目の`Message`の値はいずれもカンマや改行を含んでいるため、ダブルクオーテーションで囲っている。

```
ID,Message
1,"この文字列はカンマ (,)を含む"
2,"この文字列は改行
を含む"
3,"この文字列はカンマ (,) と改行
を含む"
```

また、ダブルクオーテーションで囲まれた値の中で、ダブルクオーテーションを文字として使いたい場合がある。その場合は、ダブルクオーテーションを二つ続けることで、値の中の文字とみなすことができる。

たとえば以下のCSVデータでは、`ID`が`1`の行における`Message`の値は、`この文字列は "ダブルクオーテーション" を含む`と解釈される。

```
ID,Message
```

```
1,"この文字列は ""ダブルクオーテーション"" を含む"
2,"この文字列はカンマ (,)を含む"
3,"この文字列は改行
を含む"
```

CSVはカンマによりデータが区切られるが、カンマの代わりにタブ文字（ `\t` ）で区切られたTSV (Tab Separated Values) というデータ形式も存在する。

以下はTSV形式のデータの例である。なお、以下のデータにおいて区切り文字はタブ文字（ `\t` ）だが、タブ文字は表示上は空白で表されていることに注意せよ。

```
ID	Name	Email
1	Alice	alice@example.com
2	Bob	bob@example.com
3	Eve	eve@example.com
```

PythonでのCSVファイルの読み込み方法

Pythonでは、csvモジュール(https://docs.python.org/ja/3/library/csv.html) を使用することでCSVファイルを容易に取り扱える。

CSVファイルを読み込むには、csvモジュールで定義されている `reader()` 関数を使用する。以下は、 `reader()` 関数を使ってCSVファイルを読み込み、内容を出力するサンプルプログラムである。

```python
import csv

# CSVファイルを開く
with open("example1.csv", "r") as f:
    # CSVファイルを読み込む
    reader = csv.reader(f)
    # 1行ずつ読み込んで処理する
    for row in reader:
        print(row)
```

```
['名前', '動物', '年齢', '性別']
['ポチ', '犬', '3', 'オス']
['タマ', '猫', '2', 'メス']
['シロ', '犬', '4', 'オス']
```

`reader()` 関数を使用できるようにするため、まず、 `import csv` と記述し、 `csv` モジュールを読み込む。続いて、以下のように `with` と `open()` 関数を使い、CSVファイルを開いている。 `r` を指定することで、読み込み専用としてファイルを開いている。

```
with open("example1.csv", "r") as f:
```

続いて、以下のように `reader()` 関数を使いCSVファイルを読み込んでいる。 `csv.reader(f)` で、開いたCSVファイルを読み込んで、それを行単位のリストとして返すイテレータを生成する。

```
reader = csv.reader(f)
```

`reader` を作成した後に、以下のように `for` 文を使用して `reader` から一行ずつ内容を読み込んで、 `print` により行の内容を出力している。 `reader` はイテレータであるため、 `for` 文を使用して1行ずつ取り出すことができる。このイテレータは、CSVデータの1行につき、その行に含まれる値を要素とする1つのリストを返す。

```
for row in reader:
```

```
    print(row)
```

出力結果から、`example1.csv` の1行目には `['名前', '動物', '年齢', '性別']` とヘッダとして列名が格納されており、2行目以降には `['ポチ', '犬', '3', 'オス']` といったように各行のデータが格納されていることがわかる。

`csv` モジュールの `reader()` 関数には複数のオプションがあり、CSVファイルの読み込みをカスタマイズできる。以下に、主要なオプションを紹介する。

- `delimiter` ：区切り文字を指定できる。たとえば、TSVファイルを読み込む場合は、 `delimiter = "\t"` と指定する。デフォルトは、 `,` である。
- `quotechar` ：CSVファイルの中で、区切り文字や改行を含む文字列を一つの値として認識させたいときに囲むときの文字を指定できる。デフォルトは `"` である。
- `skipinitialspace` ：区切り文字の後に空白文字があった場合に、その空白文字を読み飛ばすかどうかを指定できる。デフォルトは `False` である。

上記のオプションを使って、次のTSVファイルを読み込む例を見てみよう。

```
ID  Name    Email
1    Alice   alice@example.com
2    Bob bob@example.com
3    Eve eve@example.com
```

以下は、このTSVファイルを読み込むプログラムの例である。TSVファイルを読み込むには、 `delimiter` オプションにより区切り文字として `\t` を指定する必要がある。

```
import csv

# CSVファイルを開く
with open("example2.tsv", "r") as f:
    # CSVファイルを読み込む
    reader = csv.reader(f, delimiter = "\t")
    # 1行ずつ読み込んで処理する
    for row in reader:
        print(row)
```

```
['ID  Name    Email']
['1', 'Alice', 'alice@example.com']
['2', 'Bob', 'bob@example.com']
['3', 'Eve', 'eve@example.com']
```

PythonでのCSVファイルの書き込み方法

csvモジュールで定義されている `writer()` 関数を使うことで、データをCSVファイルに書き込むことができる。

以下は、 `writer()` 関数を使いCSVファイルにデータを書き込むサンプルプログラムである。

```
import csv

# CSVファイルに書き込むデータ
fields = ["Name", "Age", "Gender"]
rows = [["John", 30, "Male"], ["Emily", 25, "Female"], ["Michael", 35, "Male"]]

# ファイル名
filename = "example3.csv"

# CSVファイルの書き込み処理
```

```
with open(filename, "w") as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)
```

上記のサンプルプログラムでは、`fields` にCSVファイルのヘッダとなるリストを格納している。`rows` にCSVファイルの各行を二次元リストで格納している。`filename` に書き出すCSVファイルのファイル名を指定している。

`with` と `open()` 関数を使い、`filename` に指定した名前のCSVファイルを、`w` を指定することで、書き込み専用のファイルとして開いている。

```
with open(filename, "w") as csvfile:
```

書き込み用のオブジェクトを生成して、`writer()` 関数を使い、`csvwriter` に格納する。

```
csvwriter = csv.writer(csvfile)
```

書き込み用のオブジェクト（writeオブジェクト）のメソッドである `writerow()` メソッドを使い、`fields` に格納されているリストをCSVファイルに一行書き込んでいる。

```
csvwriter.writerow(fields)
```

writeオブジェクトのメソッドである `writerows()` メソッドを使い、`rows` に格納されているリストを一行ずつCSVファイルに書き込んでいる。

```
csvwriter.writerows(rows)
```

上記のサンプルプログラムを実行することによって、書き出されたCSVファイルの中身が以下である。

```
Name,Age,Gender
John,30,Male
Emily,25,Female
Michael,35,Male
```

`writer()` 関数にも複数のオプションがあり、CSVファイルの書き込みをカスタマイズできる。以下に、主要なオプションを紹介する。

- `delimiter` ：区切り文字を指定できる。たとえば、TSVファイルを読み込む場合は、`delimiter = "\t"` と指定する。デフォルトは、`,` である。
- `quotechar` ：CSVファイルの中で、区切り文字や改行を含む文字列を一つの値として認識させたいときに囲むときの文字を指定できる。デフォルトは `"` である。
- `linetermin` ：CSVファイルの行末文字（改行コード）を指定できる。デフォルトは、`\r\n` である。改行コードはOSによって異なり、Windowsでは `\r\n` が、MacOSやLinuxでは `\n` が使われているため、CSVファイルをどのOSで使用するかで、改行コードを明示的に指定する必要がある。

上記のオプションを使って、TSVファイルを書き込む例を見てみよう。

以下がTSVファイルを書き込むプログラムの例である。

TSVファイルを書き込むには、`writer()` 関数を実行する際に、`delimiter` を使って区切り文字として `\t` を指定する必要がある。

```python
import csv

# CSVファイルに書き込むデータ
fields = ["Name", "Age", "Gender"]
rows = [["John", 30, "Male"], ["Emily", 25, "Female"], ["Michael", 35, "Male"]]

# ファイル名
filename = "example4.tsv"

# CSVファイルの書き込み処理
with open(filename, "w") as csvfile:
    csvwriter = csv.writer(csvfile, delimiter = "\t")
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)
```

以下が、上記のサンプルプログラムを実行して書き出されたTSVファイルである。

```
Name Age Gender
John 30  Male
Emily   25  Female
Michael 35  Male
```

PythonでのCSVファイルを使ったデータ処理

PythonでCSVファイルを扱ってデータ処理を行う方法には、pandasライブラリを使う方法や、リストや辞書などのコレクションを使う方法、データベースを用いる方法など様々な方法がある。

本章では、CSVファイルのデータを二次元リストに変換して、データを処理する方法について解説する。

解説に使用するCSVファイル `example5.csv` の中身は以下である。商品ごとの売上が各行に格納されている。

```
商品名,売上
ハンドソープ,134000
シャンプー,105000
リンス,74000
保湿クリーム,51000
```

CSVファイルのデータを二次元リストに変換する処理は以下である。 `csv_list` にCSVファイルのデータを変換した二次元リストを格納している。 `csv_list = [row for row in reader]` のようにリスト内包表記を使うことで、一行で二次元リストへの変換処理を記述できる。

```python
import csv

# CSVファイルを開く
with open("example5.csv", "r") as f:
    # CSVファイルを読み込む
    reader = csv.reader(f)
    # 1行ずつ読み込んで二次元リストに変換する
    csv_list = [row for row in reader]
    print(csv_list)
```

```
[['商品名', '売上'], ['ハンドソープ', '134000'], ['シャンプー', '105000'], ['リンス', '74000'], ['保湿クリーム', '51000']]
```

続いて、売上の合計を求める処理を考えてみよう。

`csv_list` に商品ごとの売上が二次元リストで格納されているため、二次元リスト内の売上の値のみを合計するような処理を考えればよい。

以下が `example5.csv` のデータを二次元リストに格納して、データ内の売上の合計を出力するプログラムである。

```
import csv

# CSVファイルを開く
with open("example5.csv", "r") as f:
    # CSVファイルを読み込む
    reader = csv.reader(f)
    # 1行ずつ読み込んで二次元リストに変換する
    csv_list = [row for row in reader]

# 売上の合計を求める処理
total_sales = 0
for row in csv_list[1:]:
    total_sales += int(row[1])

print("売上合計:", total_sales)
```

```
売上合計: 364000
```

`example5.csv` のデータを見ると、二行目以降の二列目の値が全て売上の値であるため、二行目以降の二列目の値を取り出して合計する処理を行えばよい。

上記のプログラムの以下の箇所で、売上の値の合計を行っている。 `csv_list[1:]` と指定することで `csv_list` のインデックスが1以降の要素（CSVファイル上の二行目以降）を取り出し、 `for` 文を使い二行目以降を一行ずつ `row` に格納している。 `row[1]` で取り出された行の二列目を指定して、 `total_sales` に加算している。また、 `csv_list` の要素は全て文字列として識別されているため、 `total_sales` に加算する際に、 `int()` 関数を使って、 `row[1]` の値を整数に変換している。

```
for row in csv_list[1:]:
    total_sales += int(row[1])
```

NumPy

NumPy（Numerical Python; https://numpy.org/ ）は、Pythonにおいて数値計算を容易かつ高速・効率的に行うための各種機能を提供するライブラリである。科学計算や統計学、機械学習などの分野で広く使用されており、pandas、SciPy、Matplotlib、scikit-learn、scikit-imageなどのデータサイエンスや科学計算に関連するPythonパッケージでも使用されている。このうち次章以降で詳しく解説するpandasでは内部的なデータの保持や数値計算を行うためにNumPyを使用しており、高速で効率的な計算を実現している。

NumPyでは、数値計算によく用いられる配列や行列などのデータ構造や、それらの操作や数学的な演算を高速に行うための関数・メソッドを提供している。NumPyにおける配列はPythonの標準機能で提供されるリストとは異なるデータオブジェクトであり、リストと比較して大幅に高速な動作を少ないメモリ使用量で実現している。なお、NumPyにおいて配列等のデータを高速に処理できるのは、内部的には最適化され事前にコンパイルされたC言語のプログラムが使用されており、特に大規模なデータをPythonのインタプリタで処理する際に生じるオーバーヘッドが低減されているためである。

ndarray

NumPyでは、多次元配列を扱うためのクラスとして `ndarray` を使用する。 `ndarray` は、NumPyにおける中心的なクラスであり、多次元配列を扱うための様々な機能を提供する。 `ndarray` は"N-dimentional array"の略であり、1次元以上の任意の次元数の配列を扱えるクラスである。

`ndarray` を生成するには、 `numpy` モジュールの `numpy.array()` 関数を使用する。以下は、 `numpy.array()` により `ndarray` クラスのオブジェクトを作成する例である。

```
import numpy as np

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

```
print(arr)
```

```
[1 2 3 4]
```

まず、NumPyを使用できるようにするため、`import numpy as np`により`numpy`モジュールを読み込む。一般的に、`numpy`モジュールを`import`する際は、`as np`として`np`という別名で読み込むことで、`numpy`モジュールに属するクラスやメソッドを使用する際に`np`と短く記述できるようにする。

`numpy.array()`関数では、引数としてPythonのリストを指定することで、リストのデータと同様の構造をした`ndarray`を生成できる。今回はこの方法で`ndarray`を作成するため、もととなるPythonのリストを`li = [1, 2, 3, 4]`により作成している。これにより作成したリストを`numpy.array()`の引数として`arr = np.array(li)`とすることで、Pythonのリスト`li`から`ndarray`を作成して`arr`に代入する。

出力結果を見ると、Pythonのリスト`li`と同様に、1から4の整数の要素が格納された配列として`[1 2 3 4]`というデータを持った`ndarray`が作成されたことが分かる。なお、以下のようにこのオブジェクトは`ndarray`クラスのインスタンスオブジェクトであることが確認できる。

```
print(type(arr))
```

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

`ndarray`は、Pythonのリストとは違い、要素の型は全て同一であるという前提で作成される。これは、NumPyがデータを高速かつ効率的に処理するために、内部的な実装の都合上必要なためである。`ndarray`の要素の型は、以下のように`dtype`プロパティにより参照できる。

```
print(arr.dtype)
```

```
int64
```

`arr`は整数値を要素に持つ`ndarray`のため、`dtype`は整数型を表す`int64`となっていることがわかる。なお、ここでは64ビット整数型の`int64`となっているが、実行環境によっては`int32`のように異なるビット数になることもある。

以下のように、浮動小数点数や文字列に対応する`dtype`も存在する。

```
print(np.array([1.5, 2.4, 3.6]).dtype)
print(np.array(["a", "b", "c"]).dtype)
```

```
float64
<U1
```

ここで、`float64`は浮動小数点数を表す`dtype`であり、`<U1`は1文字のUnicode文字列を表す`dtype`である。なお、以下のように複数の型のデータが混在しているリストから`ndarray`を作成すると、より一般的な型に自動的に変換が行われる。

```
print(np.array([1, 2, 3.6]).dtype)
print(np.array([1, 2.4, "c"]).dtype)
```

```
float64
<U32
```

上記の出力結果から、整数型と浮動小数点数型が混在する `np.array([1, 2, 3.6])` の `dtype` は浮動小数点数型の `float64` に変換され、整数型、浮動小数点数型、文字列型が混在する `np.array([1, 2.4, "c"])` の `dtype` は32文字のUnicode型の `<U32` に変換されていることが確認できる。実際に格納されている要素を出力すると、以下のように `dtype` で指定された型に変換されていることがわかる。

```
print(np.array([1, 2, 3.6]))
print(np.array([1, 2.4, "c"]))
```

```
[1.  2.  3.6]
['1' '2.4' 'c']
```

なお、NumPyの出力において、浮動小数点数の小数点以下が0のときは小数部が省略されるため、 `1.` と `2.` はそれぞれ `1.0` と `2.0` を表す。

また、以下のように `numpy.array()` 関数の引数に2次元リストを渡すことで、2次元配列の構造を持った `ndarray` を作成できる。

```
li = [
    [1, 2, 3],
    [4, 5, 6],
]
arr = np.array(li)
print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

```
print(arr.dtype)
```

```
int64
```

上記の出力結果から、 `arr` として2次元配列が作成されていることがわかる。また、 `dtype` は要素一つ分の型を表すため、整数型の `int64` となっている。

`ndarray` の生成には、 `numpy.array()` 関数を用いる方法の他にも、特定の要素を持つ配列を生成する関数として `numpy.zeros()` 関数や `numpy.ones()` 関数を用いる方法もある。 `numpy.zeros()` は引数で指定した形状の配列を全て0で初期化した `ndarray` を生成する関数であり、 `numpy.ones()` は指定した形状の配列を全て1で初期化した `ndarray` を生成する関数である。以下は、 `numpy.zeros()` および `numpy.ones()` を使用して、2×3の `ndarray` を生成する例である。

```
a_ndarray = np.zeros((2, 3))
print(a_ndarray)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
a_ndarray = np.ones((2, 3))
print(a_ndarray)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

上記の例のように、`numpy.zeros()` および `numpy.ones()` の引数として配列の形状を指定するには、タプルとして各次元の要素数を指定する。出力結果から、引数として指定した要素数の0または1で構成される2次元配列が生成されていることがわかる。

ndarrayによる演算

NumPyでは、`ndarray` を用いて様々な演算を行える。`ndarray` は配列であり、数学におけるベクトルや行列のように複数の要素によって構成されるため、演算もそれら複数の要素全てに対して行われる。

最も基本的な演算として、以下のように `+` 演算子を `ndarray` に対して適用することで、`ndarray` 同士の加算を行える。

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
print(a + b)
```

```
[5 7 9]
```

上記の例では、`ndarray` として作成した `a` と `b` に対して `a + b` とすることで、`a` と `b` に格納されている各要素同士を足し合わせている。その結果、1+4、2+5、3+6の結果を各要素に持つ `ndarray` として `[5 7 9]` が生成される。同様にして、以下のように減算、乗算、除算も行える。

```
print(a - b)
print(a * b)
print(a / b)
```

```
[-3 -3 -3]
[ 4 10 18]
[0.25 0.4  0.5 ]
```

上述した例では、演算を行う2つの `ndarray` が同一の次元数を持つ配列であったが、違う次元数の配列同士で演算を行いたい場合もある。その場合も同様に、以下のように `ndarray` に対して `+` 等の演算子を適用すればよい。

```
a = np.array([1, 2, 3])
b = np.array([[1, 1, 1], [2, 2, 2]])
print(a + b)
```

```
[[2 3 4]
 [3 4 5]]
```

上記の例では、`a` の次元数は1で `b` の次元数は2であるため、そのままでは加算ができないように見える。しかし、NumPyではブロードキャスティング（broadcasting）と呼ばれる機能により、次元数の異なる `ndarray` 同士での演算を可能としている。ブロードキャスティングは、`ndarray` 同士の演算を行うときに、次元数の小さい `ndarray` を次元数の大きい `ndarray` と同様の次元数に拡張して演算を行う機能

である。

上記の例において、 `a` は `[1 2 3]` という1次元の配列であるが、2次元の配列である `b` に合わせて、内部的に `[[1 2 3] [1 2 3]]` という2次元の配列に拡張されたうえで演算が行われる。その結果、 `[[1 2 3] [1 2 3]]` と `[[1 1 1] [2 2 2]]` の各要素同士で加算が行われ、最終的に `[[2 3 4] [3 4 5]]` が結果として得られる。

ブロードキャスティングにより、次元数の異なる `ndarray` 同士の演算の他にも、以下のように `ndarray` ではない通常の数値と `ndarray` の演算を行うこともできる。

```
a = np.array([1, 2, 3])
print(a * 2)
```

```
[2 4 6]
```

上記の例では、 `ndarray` である `a` に通常の整数である `2` を乗算している。ここでも同様にブロードキャスティングが働くことで、1次元配列である `a` に合わせて、 `2` が内部的に `[2 2 2]` という配列に拡張される。そのうえで `[1 2 3]` との乗算が行われるため、結果として `[2 4 6]` が得られる。

ndarrayとPythonのリストの比較

NumPyにおける `ndarray` が高速に動作することを確認するため、大規模な配列で演算を行い、Pythonの標準機能で提供されるリストと性能を比較する。

以下は、Pythonのリストと `ndarray` それぞれについて、リストまたは配列の各要素を2倍にする計算にかかる時間を比較するプログラムである。Pythonのリストの `li` を使用した場合と `ndarray` の `arr` を使用した場合それぞれについて、 `time` モジュールの `time()` 関数を用いて、計測に要する時間を計測する。

```
import numpy as np
import time


# Pythonのリストを使った場合
li = list(range(10000000))
# 計測開始
start_time = time.time()
# リストの要素を2倍にする
for i in range(len(li)):
    li[i] *= 2
# 計測終了
time_duration = time.time() - start_time
print(f"Pythonのリスト: {time_duration} 秒")

# ndarrayを使った場合
arr = np.array(range(10000000))
# 計測開始
start_time = time.time()
# 配列の要素を2倍にする
arr *= 2
# 計測終了
time_duration = time.time() - start_time
print(f"ndarray: {time_duration} 秒")
```

```
Pythonのリスト: 0.9335653781890869 秒
ndarray: 0.0061473846435546875 秒
```

Pythonのリストと `ndarray` で計算を行う箇所は `for i in range(len(li)): li[i] *= 2` および `arr *= 2` の部分である。Pythonのリストでは `li` の各要素を繰り返し処理により一つずつ処理することで各要素の値を2倍にしており、 `ndarray` ではブロードキャスティン

グを使用して `arr *= 2` とすることで各要素の値を2倍にしている。

上記の計算部分の前後では、 `time` モジュールを使用して時間の計測を行っている。 `time.time()` 関数は現在時刻を浮動小数点数として取得する関数であり、1970年1月1日（UTC）から現在時刻までの経過秒数を返す。前述のプログラムでは、 `start_time = time.time()` により計測開始時の時刻を `start_time` として記憶している。その後、計測終了時の `time_duration = time.time() - start_time` において、 `time.time()` で計算終了時の時刻を取得して `start_time` を減じることにより、計測開始時と計測終了時の時刻の差分、つまり計測開始時点からの経過時間を `time_duration` に代入している。

プログラムの出力結果から、Pythonのリストと `ndarray` で約150倍の差が出ており、 `ndarray` を使用した場合のほうが大幅に高速になっていることがわかる。なお、この結果は実行環境によって左右されるほか、行う計算処理の内容によっても差は大きく異なることに注意せよ。

このように、NumPyを使用することで高速に計算を行えるため、特に大規模なデータを対象に処理を行う際は、NumPyや、それを内部的に使用しているライブラリの活用が重要になる。なお、NumPyを内部的に使用しているライブラリの一つであり、データ分析を容易に行えるようにするパッケージであるpandasを次章以降で扱う。

問題1

問題文

CSVファイル `data.csv` を読み込み、 `data.csv` の内容を1行ずつリスト形式で出力するプログラムを記述せよ。

`data.csv` の内容は以下である。

```
1,2,3,4,5
6,7,8,9,10
```

本問では、読み込み対象のCSVファイルを問題のプログラム中で作成するため、プログラム中で指定の箇所は変更しないこと。

出力

```
['1', '2', '3', '4', '5']
['6', '7', '8', '9', '10']
```

解答の雛形

```python
import csv

# 以下はあらかじめCSVファイルを作成するためのコードのため、変更しないこと
data = "1,2,3,4,5\n6,7,8,9,10"
with open("data.csv", "w") as f:
    f.write(data)
####################

# ここに解答を入力
```

問題2

問題文

以下の内容のCSVファイル `data.csv` を作成するプログラムを記述せよ。

```
name,age
Alice,24
Bob,19
```

本問では、作成したCSVファイル `data.csv` が正しく書きこまれているか確認するため、 `data.csv` を読み込み出力する処理が問題のプログラム中に含まれている。プログラム中で指定の箇所は変更しないこと。

出力

```
name,age
Alice,24
Bob,19
```

解答の雛形

```python
import csv

# ここに解答を入力

# 以下は作成したCSVファイルの内容を出力するコードのため、変更しないこと
with open("data.csv", "r") as f:
    print(f.read())
####################
```

問題3

問題文

以下の内容のCSVファイル `data.csv` を読み込み、 `X県` の人口の合計を出力するプログラムを記述せよ。

`data.csv` の一列目には `県名` が、二列目は `都市名` が、三列目には `人口` が記述されている。

```
県名,都市名,人口
X県,A市,10000
X県,B市,25000
X県,C市,8800
```

本問では、読み込み対象のCSVファイルを問題のプログラム中で作成するため、プログラム中で指定の箇所は変更しないこと。

出力

```
43800
```

解答の雛形

```python
import csv

# 以下は読み込み対象のCSVファイルを作成するコードのため、変更しないこと
data = "県名,都市名,人口\nX県,A市,10000\nX県,B市,25000\nX県,C市,8800"
with open("data.csv", "w") as f:
```

```
    f.write(data)
####################

# ここに解答を入力
```

問題4

問題文

以下の内容のCSVファイル `data.csv` を読み込み、 `X県` の人口の合計を出力するプログラムを記述せよ。

`data.csv` の一列目には `県名` が、二列目は `都市名` が、三列目には `人口` が記述されている。

```
1  県名,都市名,人口
2  X県,A市,20000
3  X県,B市,14000
4  X県,C市,18000
5  Y県,D市,30000
6  Y県,E市,25000
7  Z県,F市,7700
```

本問では、読み込み対象のCSVファイルを問題のプログラム中で作成するため、プログラム中で指定の箇所は変更しないこと。

出力

```
1  X県の人口の合計: 52000
```

解答の雛形

```
import csv

# 以下は読み込み対象のCSVファイルを作成するコードのため、変更しないこと
data = "県名,都市名,人口\nX県,A市,20000\nX県,B市,14000\nX県,C市,18000\nY県,D市,30000\nY県,E市,25000\nZ県,F市,7700"
with open("data.csv", "w") as f:
    f.write(data)
####################

# ここに解答を入力
```

問題5

問題文

解答の雛形

```
import numpy as np

# ここに解答を入力
```

問題6

問題文

`numpy.ones()` 関数を使用して、出力例のように、配列の要素が全て `1` で初期化された4×3の `ndarray` を生成し、出力するプログラムを記述せよ。

出力

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

解答の雛形

```
import numpy as np

## ここに解答を入力
```

問題7

問題文

`ndarray` として定義された次元数の異なる配列 `a` と `b` について、 `a` と `b` に格納されている各要素同士を加算・減算・乗算した結果を、出力例のように順番に出力するプログラムを記述せよ。

出力

```
[[3 4 5]
 [6 7 8]]
[[-1  0  1]
 [ 2  3  4]]
[[ 2  4  6]
 [ 8 10 12]]
```

解答の雛形

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([2, 2, 2])

## ここに解答を入力
```

2章: pandas (1)

pandasの概要

Pythonでデータ分析を行う際、一般的にpandas (https://pandas.pydata.org/) と呼ばれるデータ解析用ライブラリがよく使われる。 pandasにはデータ分析のための機能が豊富に揃っており、pandasを使用することで、データの取得、整形、分析、可視化を容易に行うことができる。

前章でも触れたように、pandasでは内部的なデータの保持や数値計算を行うためにNumPyを使用しており、高速で効率的な計算を実現している。pandasのユーザが直接NumPyを意識する機会は少ないが、所々でNumPyのデータオブジェクトを用いることがある。

pandasは、行列を扱うためのデータ構造として「データフレーム」を提供しており、データフレームに対して加工・集計を行う機能も備わっている。そのため、データフレームを使用することで、ExcelやSQLなどと同様に、柔軟で多様なデータ操作が可能である。

また、pandasは、CSVやJSONなどのデータ形式を簡単に読み込むための機能や、各種統計量の算出に必要な機能も提供しており、データ分析の作業を効率的に行うことができる。

本章では、分析を行う際に必要となるpandasの基礎的な使用方法について解説する。

データフレーム

データフレームとは、pandasにおいてデータを扱うための行列構造である。データフレームを使用することで、行と列を持つ表形式のデータを簡単に扱うことができる。データ分析においてデータは表形式で扱うことが多く、pandasでも基本的にデータフレームを用いて分析を行う。

以下は、pandasでデータフレームを作成し、その内容を表示するサンプルプログラムである。

```python
import pandas as pd

# データを辞書型で定義
data = {
    "動物": ["犬", "猫", "犬"],
    "年齢": [3, 2, 4],
    "性別": ["メス", "オス", "オス"],
}

# データフレームを作成
df = pd.DataFrame(data)

# データフレームを表示
print(df)
```

```
   動物  年齢  性別
0  犬   3  メス
1  猫   2  オス
2  犬   4  オス
```

まず、pandasを使用できるようにするため、`import pandas as pd` によりpandasモジュールを読み込む。一般的に、pandasモジュールを `import` する際は、`as pd` として `pd` という別名で読み込むことで、pandasモジュールに属するクラスやメソッドを使用する際に `pd` と短く記述できるようにする。

次に、データフレームに格納するためのデータを、あらかじめ辞書型で `data` として定義する。データフレームの列として使用する属性を「動物」「年齢」「性別」のようにキーとし、各行に対応するデータを配列の各添え字の位置に保持する。

こうして定義した `data` を引数として `pd.DataFrame` クラスオブジェクトを生成することで、データフレームを作成する。最後に、`print(df)` により、作成したデータフレームを表示する。

上記のプログラムの実行結果から分かる通り、データフレームの列は各データを構成する属性を表し、行は一つ一つのデータを表す。例えば、

`0 犬 3 メス` は、1行目のデータとして、インデックスが `0` 、 `動物` の値が `犬` 、 `年齢` の値が `3` 、 `性別` の値が `メス` というデータが格納されていることを表す。

pandasによるファイル読み込み

pandasでは、ファイルに保存されたデータを読み込むためのメソッドがいくつか提供されている。その一例として、CSVファイルおよびJSONファイルを扱う方法について説明する。

CSVファイルの読み込み

CSVファイルとは、カンマで区切られたテキストファイルのことであり、表形式のデータを簡易的なフォーマットで表すことのできるファイル形式である。 CSVは様々なソフトウェアで広く使われているファイル形式であり、ExcelやGoogleスプレッドシートなど、多くの表計算ソフトウェアで作成や読み込みができる。

例えば、以下のような内容で、 `data.csv` としてCSVファイルが保存されているとする。

```
動物,年齢,性別
犬,猫,犬
3,2,4
メス,オス,オス
```

以下のサンプルプログラムにより、上記のCSVファイルを読み込んで、データフレームを作成できる。

```python
import pandas as pd

# CSVファイルを読み込む
df = pd.read_csv("data.csv")

# データフレームを表示
print(df)
```

```
   動物  年齢  性別
0   犬   猫   犬
1   3   2   4
2  メス  オス  オス
```

上記のプログラムでは、 `pd.read_csv()` メソッドを使用してCSVファイルを読み込んでいる。このとき、引数には読み込むCSVファイルのパス（ここでは `data.csv` ）を指定する。 `pd.read_csv()` メソッドにより、CSVファイルから読み込んだデータからデータフレームが生成され、変数 `df` に格納される。最後に、 `print(df)` により作成したデータフレームを表示する。

pandasのread_csvには複数のオプションがあり、CSVファイルの読み込みをカスタマイズすることができる。以下に、主要なオプションを紹介する。

- `sep` ：カンマ以外の文字で区切られたデータを読み込む場合に、区切り文字を指定できる。例えば、タブ文字（ `\t` ）で区切られたファイル（TSV: Tab Separated Values と呼ばれる）を読み込む場合は `sep="\t"` とする。
- `header` ：列の名前が書かれたヘッダが何行目にあるかを指定する。例えば、CSVファイルの1行目をヘッダとして使用する場合は、 `header=0` とする。また、 `header=None` とすると、ヘッダを使用せず、列の名前として番号が割り当てられる。 `header` を指定しなかった場合は、CSVファイルの1行目のデータなどから自動的に判別する。先に挙げたサンプルプログラムでは、1行目の `動物` 、 `年齢` 、 `性別` が自動的にヘッダとして使用されている。
- `usecols` ：どの列を読み込むかを指定できる。例えば、 `["動物", "年齢"]` や `[0, 1]` とすれば、 `動物` と `年齢` の列のみが読み込まれる。

上記のオプションを組み合わせてファイルを読み込む例を見てみよう。ここでは `sep` により区切り文字としてタブ文字を指定するため、CSVファイルの代わりにタブ文字で区切られたTSVファイルを使用する。読み込む対象の `data.tsv` は以下のようなファイルである。

```
動物  年齢    性別
```

```
犬   猫   犬
3    2   4
メス オス    オス
```

なお、区切り文字はタブ文字（ `\t` ）だが、タブ文字は表示上は空白で表されていることに注意せよ。

以下のプログラムにより、区切り文字をタブ文字（ `\t` ）として、ヘッダを使用せず、読み込む列の列番号を0番目と1番目のみとして `data.tsv` を読み込める。

```python
import pandas as pd

# CSVファイルを読み込む
df = pd.read_csv("data.tsv", sep="\t", header=None, usecols=[0, 1])

# データフレームを表示
print(df)
```

```
     0   1
0  動物  年齢
1   犬   猫
2   3   2
3  メス  オス
```

上記の例では、 `header=None` としているため、列の名前は `動物` 、 `年齢` ではなく、 `0` 、 `1` となっている。また、 `動物` 、 `年齢` はヘッダとして読み込まれず、データ1行分として読み込まれている。今回の例では、オプションの使い方を示すために敢えてこのような指定を行ったが、通常は初めに示した例のように1行目をヘッダとすることが多い。

JSONファイルの読み込み

JSONは、JavaScript Object Notationの略で、データを表すためのテキスト形式の一つである。JSONは、軽量なデータ形式でありながら、数値や文字列、真偽値などの様々なデータ型を扱えるほか、Pythonにおける辞書やリストのようなコレクションに相当するデータ構造も扱える。これにより、データを構造化して表現でき、複雑なデータも扱えるため、よくWeb APIなどで使用される。

pandasでは、JSON形式のデータを読み込むためのメソッドが提供されている。以下のように、CSVの読み込みで扱った例と同様の構造をしたJSON形式のファイル `data.json` があるとする。

```json
{
    "動物": ["犬", "猫", "犬"],
    "年齢": [3, 2, 4],
    "性別": ["メス", "オス", "オス"]
}
```

ここではJSON形式の詳細な解説は行わないが、簡単に上記のJSONファイルの内容を説明する。

`{}` で囲われた範囲は「オブジェクト」と呼ばれ、Pythonの辞書と同様に、キーと値の組み合わせのデータ構造を表している。上記のJSONファイルでは、「動物」、「年齢」、「性別」の3つのキーを持つ。それぞれのキーに対応する値として記述されている `[]` で囲われた範囲は「配列」と呼ばれ、Pythonのリストと同様のデータ構造で表される。そのため、このJSONファイルでは、3つの行、3つの列で構成される表形式のデータを表している。

上記のJSONファイルを読み込むには、以下のように `read_json()` メソッドを使用する。

```python
import pandas as pd

# JSONファイルを読み込む
df = pd.read_json("data.json")

# 読み込んだデータを表示
```

```
print(df)
```

```
   動物  年齢  性別
0  犬   3  メス
1  猫   2  オス
2  犬   4  オス
```

pandasによるデータフレームの操作（基礎）

pandasには、データフレームの操作に用いる様々な機能が存在する。これらの機能を使用することで、データを分析する際に使用しやすいようにデータを加工したり、特定のデータに対して分析を行ったりできる。

ここでは、pandasで提供されている基礎的な機能のうち、代表的なものをいくつか紹介する。

なお、本節の説明では、以下のようなデータフレーム `df` があらかじめ定義されているものとする。

```
import pandas as pd

data = {
    "列1": [3, 7, 5],
    "列2": ["A", "B", "C"],
}
df = pd.DataFrame(data)
print(df)
```

```
   列1 列2
0   3  A
1   7  B
2   5  C
```

行・列の選択

配列に対して添え字で値を参照するときのように、データフレームに対しても `[]` を使用することで行や列を選択できる。

特定の列を選択するには、以下のように `df[]` を使用して、選択する列の名前を `[]` の中に記述する。

```
# 列名が"列1"の列を選択
col = df["列1"]
print(col)
```

```
0    3
1    7
2    5
Name: 列1, dtype: int64
```

`[]` の中に列のリストを記述することで、複数の列を選択することもできる。

```
# 列名が"列1"と"列2"の列を選択
cols = df[["列1", "列2"]]
print(cols)
```

```
   列1 列2
0   3  A
```

```
3  1   7  B
4  2   5  C
```

また、特定の行を選択するには、以下のように `df.loc[]` を使用する。

```
1  # 行番号が0番目の行を選択
2  row = df.loc[0]
3  print(row)
```

```
1  列1    3
2  列2    A
3  Name: 0, dtype: object
```

```
1  # 行番号が0番目と1番目の行を選択
2  rows = df.loc[[0, 1]]
3  print(rows)
```

```
1     列1 列2
2  0   3  A
3  1   7  B
```

```
1  # 行番号が1番目以降の行を選択
2  rows = df.loc[1:]
3  print(rows)
```

```
1     列1 列2
2  1   7  B
3  2   5  C
```

フィルタリング

データフレームの行を選択する際に、 `[]` の中に抽出する行の条件を記述することで、特定の条件を満たす行だけを抽出するフィルタリングを行える。

以下は、フィルタリングを行うサンプルプログラムである。

```
1  # "列1"の値が5以上の行を抽出
2  filtered = df[df["列1"] >= 5]
3  print(filtered)
```

```
1     列1 列2
2  1   7  B
3  2   5  C
```

ここで、 `df["列1"] >= 5` は、データフレーム `df` の `"列1"` の値が5以上であるかを評価する条件式である。この評価結果は、行毎に `True` か `False` を返す。

例えば、今回扱うデータフレーム `df` は以下のようになっている。

```
print(df)
```

```
   列1 列2
0   3  A
1   7  B
2   5  C
```

このとき、`df["列1"] >= 5` の評価結果は、以下のようになる。

```
print(df["列1"] >= 5)
```

```
0    False
1     True
2     True
Name: 列1, dtype: bool
```

この条件式を `[]` の中に入れた `df[df["列1"] >= 5]` は、上記の評価結果が `True` の行だけを抽出することを意味する。したがって、初めに示したサンプルプログラムと同様に、フィルタリングの結果としては以下のデータフレームを返すことになる。

```
print(df[df["列1"] >= 5])
```

```
   列1 列2
1   7  B
2   5  C
```

また、以下のように複数の条件を組み合わせることで、複雑な条件を指定することもできる。条件を複数組み合わせるには、条件を括弧で囲み、論理演算子を使用する。論理演算子としては、`&`（かつ）や `|`（または）が使用できる。

```
# "列1"の値が5以上かつ、"列2"の値が"B"の行を抽出
filtered = df[(df["列1"] >= 5) & (df["列2"] == "B")]
print(filtered)
```

```
   列1 列2
1   7  B
```

```
# "列1"の値が5以上か、"列2"の値が"A"の行を抽出
filtered = df[(df["列1"] >= 5) | (df["列2"] == "A")]
print(filtered)
```

```
   列1 列2
0   3  A
1   7  B
2   5  C
```

ソート

`DataFrame` クラスの `sort_values()` メソッドを使用することで、データフレーム内のデータに対して、データを順番に並べ替えるソート操作を行える。 `sort_values()` メソッドでは、データフレームを指定した列で昇順や降順にソートできる。

以下は、データフレームをソートするサンプルプログラムである。

```
# "列1"で昇順にソート
sorted_df = df.sort_values(by="列1")
print(sorted_df)
```

```
   列1 列2
0   3  A
2   5  C
1   7  B
```

```
# "列2"で降順にソート
sorted_df = df.sort_values(by="列2", ascending=False)
print(sorted_df)
```

```
   列1 列2
2   5  C
1   7  B
0   3  A
```

上記の例では、 `sort_values()` メソッドのオプションとして `by` および `ascending` を使用して、並び替えの条件を指定している。 `by` にはソートの基準とする列を指定する。また、昇順に並び替えるなら `ascending=True` 、降順に並び替えるなら `ascending=False` を指定する。なお、 `ascending` に何も指定しない場合は、デフォルト値の `True` が使用されるため、昇順となる。

行・列の追加

行・列の選択では `df.loc[]` や `df[]` により行や列を参照していたが、 `df.loc[]` や `df[]` を使用して値を代入することで、データフレームに行や列を追加できる。

以下は、行を追加するサンプルプログラムである。なお、説明の都合上、元のデータフレーム `df` への変更を避けるため、 `df2 = df.copy()` として `df` をコピーした `df2` に対して操作を行っている。

```
# dfをコピーしてdf2を定義
df2 = df.copy()

# 新たに行を追加
new_row = [10, "D"]
df2.loc[len(df2)] = new_row

print(df2)
```

```
   列1 列2
0   3  A
1   7  B
2   5  C
3  10  D
```

上記のプログラムでは、 `df2.loc[] = new_row` の `[]` の中に追加する行の行番号を指定して、 `=` により新たな行 `new_row` を代入することで、 `df2` に行を追加している。なお、 `len(df2)` は、 `df2` の行数を取得する処理である。 `df2` の行数を行番号として指定することで、 `df2` の末尾に新たな行 `new_row` を追加できる。

また、以下は、列を追加するサンプルプログラムである。

```
# dfをコピーしてdf2を定義
df2 = df.copy()

# 新たに列を追加
new_col = [1, 2, 3]
df2["列3"] = new_col

print(df2)
```

```
   列1 列2  列3
0   3  A   1
1   7  B   2
2   5  C   3
```

列の追加は、以下ように既存の列から値を生成することで、より分析に適した値へ変換するのによく用いられる。

```
# dfをコピーしてdf2を定義
df2 = df.copy()

# 新たに列を追加
new_col = df2["列1"] * 2
df2["列3"] = new_col

print(df2)
```

```
   列1 列2  列3
0   3  A   6
1   7  B  14
2   5  C  10
```

上記の例では、 `列1` の値を2倍した値を持つ `列3` を新たに定義している。分析の内容によっては、既存の列の値を加工した値を使用したいケースも多いため、そういった場合にこのような変換が役に立つ。

値の変更

行・列の追加では、新たな行番号や列名を指定することで行や列の追加を行ったが、既に存在する行や列を指定することで、データフレームの値を変更できる。

以下は、データフレームの特定の行や列の値を一括で置き換えるサンプルプログラムである。

```
# dfをコピーしてdf2を定義
df2 = df.copy()

# 行番号0の行の値を10に変更
df2.loc[0] = 10
print(df2)
```

```
   列1  列2
0  10  10
1   7   B
2   5   C
```

```
# dfをコピーしてdf2を定義
df2 = df.copy()

# "列1"の値を10に変更
df2["列1"] = 10
print(df2)
```

```
   列1 列2
0  10  A
1  10  B
2  10  C
```

また、`loc[]`で行と列の両方を指定することで、特定のセルの値を変更することもできる。以下の例では、行番号が0、列名が`列1`のセル（一番左上のセル）の値のみを10に変更している。

```
# dfをコピーしてdf2を定義
df2 = df.copy()

# 行番号が0、列名が"列1"のセルを10に変更
df2.loc[0, "列1"] = 10
print(df2)
```

```
   列1 列2
0  10  A
1   7  B
2   5  C
```

`[]`の中に条件を記述することで、特定の条件を満たすセルの値を変更できる。以下は、"列1"の値が5以上のセルを10に変更するサンプルプログラムである。

```
# dfをコピーしてdf2を定義
df2 = df.copy()

# "列1"の値が5以上の行に対して、"列1"の値を10に変更
df2.loc[df["列1"] >= 5, "列1"] = 10
print(df2)
```

```
   列1 列2
0   3  A
1  10  B
2  10  C
```

さらに、`apply()`メソッドを使用することで、値を変更する際に関数やメソッドを適用することもできる。例えば、以下のように`apply()`メソッドの引数として`str.lower`メソッドを指定することで、アルファベットを小文字に変換できる。

```
# dfをコピーしてdf2を定義
df2 = df.copy()

# "列2"を小文字に変換
df2["列2"] = df["列2"].apply(str.lower)
print(df2)
```

```
   列1 列2
0   3  a
1   7  b
2   5  c
```

統計量の算出

データを分析する際、そのデータの特徴を大まかにつかむために、「統計量」と呼ばれる統計的な指標が用いられる。pandasには、データフレームに格納されたデータの統計量を簡単に算出するための仕組みが備わっている。今回は、代表的な統計量の算出方法と、それを用いてデータの特徴を把握する方法を例をもとに紹介する。

統計量

統計量とは、データに対して一定の計算を施すことで算出される、データの特徴を要約した統計的な指標を指す。統計量を用いることで、データの特徴をわかりやすく表すことができる。

代表的な統計量には、以下のようなものがある。

- 平均値：データの値をすべて足し合わせてデータ数で割った値を指す。
- 中央値：データを順番に並べたとき、中央に位置する値を指す。
- 最大値：データの中で最大の値を指す。
- 最小値：データの中で最小の値を指す。
- 標準偏差：データが平均からどの程度分散しているかを表す。

例えば、ある企業の年間売上をデータとして持っているとする。このデータから最大値や最小値を算出することで、その企業の年間売上がどの程度の範囲に収まっているかを知ることができる。

さらに、異なる企業の年間売上を比較するときにも統計量を活用できる。例えば、企業A、企業B、企業Cの年間売上の平均値を比較することで、企業間の売上を統一した基準で比較できる。

このように統計量を用いることで、データの特徴をわかりやすく表したり、異なるデータを比較しやすくしたりできる。

pandasによる統計量の算出

pandasには、データフレームの統計量を簡単に算出できるよう、`DataFrame`クラスや、後述するように`DataFrame`から取得できる`Series`クラスに、複数のメソッドが用意されている。以降では、pandasで実際に統計量を算出する方法を解説する。

例として、架空の企業の売り上げデータとして、以下のデータフレームを対象とする。

```python
import pandas as pd

# データフレームを作成
data = {
    "年度": [2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022],
    "売上": [1200, 1500, 1300, 1350, 1400, 1550, 1600, 1550],
}
df = pd.DataFrame(data)
```

以下のように、`df`に対して`mean()`メソッドを呼び出すことで、各列の平均値を算出できる。

```python
print(df.mean()) # 平均値
```

```
年度    2018.50
売上    1431.25
dtype: float64
```

上記の結果から分かるように、 `売上` 列だけでなく `年度` 列に対しても平均値が算出されている。しかし、今回のデータにおいて `年度` は各データのラベルとしての役割を持つものであり、平均値を算出する対象としては適切ではない。

そこで、 `df` に対して `mean()` メソッドを呼び出すのではなく、 `df["売上"]` に対して `mean()` メソッドを呼び出すことで、 `売上` 列に対してのみ平均値を算出できる。

```
print(df["売上"].mean()) # 平均値
```

```
1431.25
```

ここで、 `df["売上"]` は以下のように `Series` クラスのインスタンスオブジェクトであり、ラベル付けされた1次元のコレクション構造になっている。

```
print(type(df["売上"]))
```

```
<class 'pandas.core.series.Series'>
```

```
print(df["売上"])
```

```
0    1200
1    1500
2    1300
3    1350
4    1400
5    1550
6    1600
7    1550
Name: 売上, dtype: int64
```

上記の `Series` オブジェクトに対して `mean()` メソッドを呼び出すことで、 `売上` 列の平均値を求めることができる。

同様にして、以下のように `median()` 、 `max()` 、 `min()` 、 `std()` といったメソッドを呼び出すことで、各種統計量を計算できる。

```
print(df["売上"].median()) # 中央値
print(df["売上"].max()) # 最大値
print(df["売上"].min()) # 最小値
print(df["売上"].std()) # 標準偏差
```

```
1450.0
1600
1200
141.26343172547217
```

また、以下のように `describe()` メソッドを使用することで、各種統計量を一度に算出できる。

```
print(df["売上"].describe())
```

```
count       8.000000
mean     1431.250000
std       141.263432
min      1200.000000
25%      1337.500000
50%      1450.000000
75%      1550.000000
max      1600.000000
Name: 売上, dtype: float64
```

なお、`count` はデータの数を表し、`25%`、`50%`、`75%` はそれぞれ、データを昇順に並べたときに全体の `25%`、`50%`、`75%` に位置する値を表したものである。したがって、`50%` は中央値と同等である。`25%`、`75%` はそれぞれ、第1四分位数、第3四分位数と呼ばれる。これは、データ全体を4等分したときにそれぞれ1番目、3番目の区切り位置になっているためである。第1四分位数、中央値（第2四分位数とも呼ぶ）、第3四分位数をまとめて、四分位数と呼ぶ。

問題1

問題文

以下の内容のCSVファイル `data.csv` を読み込んで、pandasのデータフレームを作成し、作成したデータフレームを出力するプログラムを記述せよ。

```
県名,都市名,人口
X県,A市,12000
X県,B市,10000
X県,C市,24000
```

本問では、読み込み対象のCSVファイルを問題のプログラム中で作成するため、プログラム中で指定の箇所は変更しないこと。

出力

```
   県名 都市名     人口
0  X県   A市  12000
1  X県   B市  10000
2  X県   C市  24000
```

解答の雛形

```
import csv
import pandas as pd

# 以下はあらかじめCSVファイルを作成するためのコードのため、変更しないこと
data = "県名,都市名,人口\nX県,A市,12000\nX県,B市,10000\nX県,C市,24000"
with open("data.csv", "w") as f:
    f.write(data)
####################

# ここに解答を入力
```

問題2

問題文

以下のようなpandasのデータフレーム `df` の中から、 `都市名` の列と `人口` の列のみを出力するプログラムを記述せよ。

なお、出力例のように、左側に `都市名` の列が出力され、右側に `人口` の列が出力されるようにプログラムを記述すること。

```
   県名 都市名     人口
0  X県  A市  12000
1  X県  B市   8000
2  X県  C市  22000
3  Y県  D市  15000
4  Y県  E市  32000
5  Z県  F市   9000
```

出力

```
  都市名     人口
0  A市  12000
1  B市   8000
2  C市  22000
3  D市  15000
4  E市  32000
5  F市   9000
```

解答の雛形

```
import pandas as pd

data = {
    "県名": ["X県", "X県", "X県", "Y県", "Y県", "Z県"],
    "都市名": ["A市", "B市", "C市", "D市", "E市", "F市"],
    "人口": [12000, 8000, 22000, 15000, 32000, 9000],
}

df = pd.DataFrame(data)

## ここに解答を入力
```

問題3

問題文

以下のようなpandasのデータフレーム `df` の中から、インデックスが `3` の行とインデックスが `4` の行のみを出力するプログラムを記述せよ。

なお、出力例のように、上側にインデックス `3` の行が出力され、下側にインデックス `4` の行が出力されるようにプログラムを記述すること。

```
   県名 都市名     人口
0  X県  A市  12000
1  X県  B市   8000
2  X県  C市  22000
3  Y県  D市  15000
4  Y県  E市  32000
5  Z県  F市   9000
```

出力

```
   県名 都市名     人口
3  Y県  D市  15000
4  Y県  E市  32000
```

解答の雛形

```
import pandas as pd

data = {
    "県名": ["X県", "X県", "X県", "Y県", "Y県", "Z県"],
    "都市名": ["A市", "B市", "C市", "D市", "E市", "F市"],
    "人口": [12000, 8000, 22000, 15000, 32000, 9000],
}

df = pd.DataFrame(data)

## ここに解答を入力
```

問題4

問題文

以下のようなpandasのデータフレーム `df` の中から、 `県名` 列が `X県` であり、かつ `人口` 列の値が `10000` 以上の行のみを出力するプログラムを記述せよ。

```
   県名 都市名     人口
0  X県  A市  12000
1  X県  B市   8000
2  X県  C市  22000
3  Y県  D市  15000
4  Y県  E市  32000
5  Z県  F市   9000
```

出力

```
   県名 都市名     人口
0  X県  A市  12000
2  X県  C市  22000
```

解答の雛形

```
import pandas as pd

data = {
    "県名": ["X県", "X県", "X県", "Y県", "Y県", "Z県"],
    "都市名": ["A市", "B市", "C市", "D市", "E市", "F市"],
    "人口": [12000, 8000, 22000, 15000, 32000, 9000],
}

df = pd.DataFrame(data)

## ここに解答を入力
```

問題5

問題文

以下のようなpandasのデータフレーム `df` がある。出力例のように、 `人口` 列で降順に行をソートして出力するプログラムを記述せよ。

```
1     県名 都市名     人口
2  0  X県   A市  12000
3  1  X県   B市   8000
4  2  X県   C市  22000
5  3  Y県   D市  15000
6  4  Y県   E市  32000
7  5  Z県   F市   9000
```

出力

```
1     県名 都市名     人口
2  4  Y県   E市  32000
3  2  X県   C市  22000
4  3  Y県   D市  15000
5  0  X県   A市  12000
6  5  Z県   F市   9000
7  1  X県   B市   8000
```

解答の雛形

```
import pandas as pd

data = {
    "県名": ["X県", "X県", "X県", "Y県", "Y県", "Z県"],
    "都市名": ["A市", "B市", "C市", "D市", "E市", "F市"],
    "人口": [12000, 8000, 22000, 15000, 32000, 9000],
}

df = pd.DataFrame(data)

## ここに解答を入力
```

問題6

問題文

以下のようなpandasのデータフレーム `df` がある。データフレーム `df` に `名産` 列を追加して、その後、データフレームの末尾の行に `県名` が `Z県` 、 `都市名` が `G市` 、 `人口` が `10000` 、 `名産` が `スイカ` となる行を追加するプログラムを記述せよ。

出力結果のように、インデックス `0` から `5` の行の `名産` 列にはそれぞれ、 `ゴボウ` 、 `ネギ` 、 `ミカン` 、 `トマト` 、 `ミカン` 、 `メロン` を追加する。

```
    県名 都市名     人口
0  X県  A市  12000
1  X県  B市   8000
2  X県  C市  22000
3  Y県  D市  15000
4  Y県  E市  32000
5  Z県  F市   9000
```

出力

```
    県名 都市名     人口    名産
0  X県  A市  12000  ゴボウ
1  X県  B市   8000   ネギ
2  X県  C市  22000  ミカン
3  Y県  D市  15000  トマト
4  Y県  E市  32000  ミカン
5  Z県  F市   9000  メロン
6  Z県  G市  10000  スイカ
```

解答の雛形

```
import pandas as pd

data = {
    "県名": ["X県", "X県", "X県", "Y県", "Y県", "Z県"],
    "都市名": ["A市", "B市", "C市", "D市", "E市", "F市"],
    "人口": [12000, 8000, 22000, 15000, 32000, 9000],
}

df = pd.DataFrame(data)

## ここに解答を入力
```

問題7

問題文

以下のようなpandasのデータフレーム `df` がある。出力例のように、データフレーム `df` のデータの中で、値が `X県` のデータの値を、 `福岡県` に変更してデータフレーム `df` を出力するプログラムを記述せよ。

```
    県名 都市名     人口
0  X県  A市  12000
1  X県  B市   8000
2  X県  C市  22000
3  Y県  D市  15000
4  Y県  E市  32000
5  Z県  F市   9000
```

出力

```
    県名 都市名     人口
0  福岡県  A市  12000
1  福岡県  B市   8000
2  福岡県  C市  22000
3   Y県  D市  15000
4   Y県  E市  32000
5   Z県  F市   9000
```

解答の雛形

```
import pandas as pd

data = {
    "県名": ["X県", "X県", "X県", "Y県", "Y県", "Z県"],
    "都市名": ["A市", "B市", "C市", "D市", "E市", "F市"],
    "人口": [12000, 8000, 22000, 15000, 32000, 9000],
}

df = pd.DataFrame(data)

## ここに解答を入力
```

問題8

問題文

以下のようなpandasのデータフレーム `df` がある。 `人口` 列の値の平均値、中央値、最大値、最小値を1行ずつ出力するプログラムを記述せよ。

なお、出力例のように、平均値、中央値、最大値、最小値の順に出力すること。

```
   県名 都市名     人口
0  X県  A市  12000
1  X県  B市   8000
2  X県  C市  22000
3  Y県  D市  15000
4  Y県  E市  32000
```

出力

```
17800.0
15000.0
32000
8000
```

解答の雛形

```
import pandas as pd

data = {
    "県名": ["X県", "X県", "X県", "Y県", "Y県"],
    "都市名": ["A市", "B市", "C市", "D市", "E市"],
    "人口": [12000, 8000, 22000, 15000, 32000],
}
```

```
df = pd.DataFrame(data)

## ここに解答を入力
```

3章: pandas (2)

pandasによるデータフレームの操作（発展）

前章では、pandasを用いてデータフレームを操作する方法のなかでも、基礎的なものを解説した。本章では、より複雑な操作を行うための発展的な内容を解説する。

前章で解説した内容に加え、本章で解説する方法を併せて使用することで、データフレームに対してより高度な操作を行えるようになり、その後の分析の幅を広げることができる。

グループ集計

`DataFrame` クラスの `groupby()` メソッドを使用することで、データフレームを特定のキーに基づいてグループ分けして、各グループに対して集計を行うことができる。例えば、様々な商品の売り上げを格納したデータフレームに対して、商品の売り上げを都道府県ごとに集計するといったことができる。

例として、以下のようなデータフレームを対象にグループ集計を行うことを考える。

```python
import pandas as pd

# データフレームを作成するためのデータを作成
data = {
    "都道府県": ["東京都", "東京都", "大阪府", "大阪府", "愛知県"],
    "商品": ["商品A", "商品B", "商品A", "商品C", "商品B"],
    "売り上げ": [10000, 20000, 30000, 40000, 50000],
}

# データを元にデータフレームを作成
df = pd.DataFrame(data)
print(df)
```

```
  都道府県   商品   売り上げ
0  東京都  商品A  10000
1  東京都  商品B  20000
2  大阪府  商品A  30000
3  大阪府  商品C  40000
4  愛知県  商品B  50000
```

以下は、データフレームを都道府県ごとにグループ集計して、都道府県ごとの売り上げの合計を計算するサンプルプログラムである。

```python
# "都道府県"でグループ分けし、"売り上げ"を合計する
grouped_df = df.groupby("都道府県").sum()
print(grouped_df)
```

```
        売り上げ
都道府県
大阪府    70000
愛知県    50000
東京都    30000
```

上記のように、 `groupby()` メソッドにグループ分けするキーの列名として `"都道府県"` を指定することで、 `都道府県` 列の値が同一のデータを一つにまとめてグループ化できる。さらに、グループ化した結果に対して `sum()` メソッドを使用して `売り上げ` の合計を計算することで、最終的に都道府県ごとの売り上げの合計が格納されたデータフレームを作成している。

上記のプログラムで使用されている `groupby()` と `sum()` の挙動はやや特殊なため、より詳細な動作を解説する。 `df.groupby("都道府県")` の結果として、以下のような `DataFrameGroupBy` クラスのインスタンスオブジェクトが返却される。

```
print(df.groupby("都道府県"))
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f69731a70a0>
```

上記の `DataFrameGroupBy` オブジェクトは、 `groupby()` によって指定されたキーによってグループ分けした結果を格納したオブジェクトであり、各グループに対して集計などの操作を行うためのメソッド群を提供する。

今回使用している `sum()` メソッドはそのうちの一つであり、各グループ内の要素を足し合わせた結果を格納したデータフレームを返却する。 `sum()` メソッドは本来、グループ分けのキーとして指定した `都道府県` 以外の各列に対して合計を計算するメソッドのため、 `売り上げ` 列だけでなく `商品` 列に対しても合計を計算するはずだが、 `商品` 列は数値データではなく合計が計算できないため、 `売り上げ` 列の合計のみが結果として返却される。

したがって、 `df.groupby("都道府県").sum()` とすることで、 `都道府県` 列を基にグループ分けして、各 `都道府県` に対して `売り上げ` の合計を計算できる。

`DataFrameGroupBy` は様々なメソッドを提供しているが、中でも有用なものをいくつか紹介する。

- `sum()` ：各グループにおける和を求める。
- `mean()` ：各グループにおける平均値を求める。
- `count()` ：各グループにおける要素数を求める。
- `size()` ：各グループにおける要素数を求める。
- `max()` ：各グループにおける最大値を求める。
- `min()` ：各グループにおける最小値を求める。
- `median()` ：各グループにおける中央値を求める。

これらのメソッドを用いることで、グループごとに集計を容易に行える。

データフレームの結合

pandasでは、複数のデータフレームを結合して一つのデータフレームを新たに作成するための機能が提供されている。データフレームの結合には、横方向に結合する方法と縦方向に結合する方法があり、後述するようにそれぞれ異なる機能を使用する。

データフレームの結合の例として、以下のような2つのデータフレームを考える。

```
3  A
7  B
```

```
5  C
2  D
```

データフレームを横方向に結合すると、以下のように、各データフレームの行をつなぎ合わせたデータフレームが作成される。

```
3  A  5  C
7  B  2  D
```

一方、データフレームを縦方向に結合すると、以下のように、各データフレームの列をつなぎ合わせたデータフレームが作成される。

```
3  A
7  B
5  C
2  D
```

上記のように、結合の方向によって作成されるデータフレームの構造が異なるため、目的に応じて使い分ける必要がある。

横方向の結合

実際にpandasでデータフレームを横方向に結合する例として、以下の2つのデータフレームを結合する。

```python
# データフレーム(df1)を作成
data1 = {"ID": [1, 2, 3], "商品名": ["商品A", "商品B", "商品C"], "価格": [100, 200, 300]}
df1 = pd.DataFrame(data1)
print(df1)
```

```
   ID  商品名   価格
0   1  商品A  100
1   2  商品B  200
2   3  商品C  300
```

```python
# データフレーム(df2)を作成
data2 = {"ID": [1, 3, 4], "在庫": [1, 2, 3]}
df2 = pd.DataFrame(data2)
print(df2)
```

```
   ID  在庫
0   1   1
1   3   2
2   4   3
```

これらのデータフレームを横方向に結合するには、以下のように `pandas.merge()` 関数を使用する。

```python
# df1とdf2を横方向に結合
merged_df = pd.merge(df1, df2)
print(merged_df)
```

```
   ID  商品名   価格  在庫
0   1  商品A  100   1
1   3  商品C  300   2
```

上記のように、 `pandas.merge()` 関数に、結合する2つのデータフレーム `df1` と `df2` を指定することで、横方向に結合したデータフレームを作成できる。

実行結果を見ると、元々 `df1` と `df2` には3行ずつのデータが存在したにもかかわらず、結合後のデータフレームには2行のデータしか存在していないことがわかる。これは、データを横方向に結合するときのルールによるものである。

データフレームの結合を行う際、各データフレームに共通する列（今回の場合は `ID` ）を見て、同じ `ID` を持つ行をつなげて一つの行としている。このとき、いずれか片方にしか存在しない `ID` を持つ行は除外される。したがって、 `merged_df` は、両方のデータフレームに共通する `ID` である `1` と `3` を持つ行のみで構成されるデータフレームとなる。

このように、ある列を基準として行を対応付けることでデータフレーム同士を結合するとき、両方のデータフレームに共通する行のみを使用す

る方法を「内部結合」と呼ぶ。なお、結合の基準とする列を、これ以降の説明ではキー（key）と呼ぶ。前述のプログラムでは、キーは `ID` である。

`pandas.merge()` では、使用する行をどういったルールで選ぶかを `how` オプションで指定できる。使用可能な `how` オプションの値と、対応する結合ルールは以下のとおりである。なお、前述のプログラムでは、 `how` を指定していないため、デフォルト値として `inner` が使用されている。

- `inner` **（内部結合）**：キーの値が両方のデータフレームに共通する行のみを使用する。キーの値が同じ行同士を結合する。
- `left` **（左外部結合）**：左側のデータフレームの行をすべて使用する。左側のデータフレームの各行に対して、右側のデータフレームのうちキーの値が同じ行を結合する。左側のデータフレームの行のうち、右側のデータフレームに存在しないキーの値を持つものに関しては、右側のデータフレームの列の値がすべて `NaN` として処理される。
- `right` **（右外部結合）**：右側のデータフレームの行をすべて使用する。右側のデータフレームの各行に対して、左側のデータフレームのうちキーの値が同じ行を結合する。右側のデータフレームの行のうち、左側のデータフレームに存在しないキーの値を持つものに関しては、左側のデータフレームの列の値がすべて `NaN` として処理される。
- `outer` **（完全外部結合）**：両方のデータフレームの行をすべて使用する。キーの値が同じ行はそれら同士で結合し、キーの値が左右どちらかのデータフレームにしか存在しない行に関しては、もう一方のデータフレームの列の値がすべて `NaN` として処理される。
- `cross` **（交差結合）**：両方のデータフレームの行をすべて使用する。左側のデータフレームの行と、右側のデータフレームの行をそれぞれ選んでできるすべての組み合わせで結合する。

例えば、以下のように `how="left"` とすることで、左外部結合によりデータフレームを結合できる。

```python
# df1とdf2を左外部結合により結合
merged_df = pd.merge(df1, df2, how="left")
print(merged_df)
```

```
   ID  商品名   価格   在庫
0   1  商品A  100  1.0
1   2  商品B  200  NaN
2   3  商品C  300  2.0
```

出力結果から、左側のデータフレーム `df1` に存在する行がすべて含まれており、 `ID` が `2` の行は右側のデータフレームに存在しないため `在庫` が `NaN` となっていることがわかる。

縦方向の結合

次に、データフレームを縦方向に結合する例として、以下の2つのデータフレームを結合する。

```python
# データフレーム(df1)を作成
data1 = {"ID": [1, 2, 3], "商品名": ["商品A", "商品B", "商品C"], "価格": [100, 200, 300]}
df1 = pd.DataFrame(data1)
print(df1)
```

```
   ID  商品名   価格
0   1  商品A  100
1   2  商品B  200
2   3  商品C  300
```

```python
# データフレーム(df2)を作成
data2 = {"ID": [4, 5, 6], "商品名": ["商品D", "商品E", "商品F"], "価格": [400, 500, 600]}
df2 = pd.DataFrame(data2)
print(df2)
```

```
   ID  商品名   価格
0   4  商品D  400
1   5  商品E  500
2   6  商品F  600
```

これらのデータフレームを縦方向に結合するには、以下のように `pandas.concat()` 関数を使用する。

```
# df1とdf2を縦方向に結合
concat_df = pd.concat([df1, df2])
print(concat_df)
```

```
   ID  商品名   価格
0   1  商品A  100
1   2  商品B  200
2   3  商品C  300
0   4  商品D  400
1   5  商品E  500
2   6  商品F  600
```

`pandas.concat()` 関数では、第1引数として結合するデータフレームのコレクションを指定する。上記のプログラムでは、結合する2つのデータフレームとして `df1` と `df2` をリスト形式で渡している。

この例のように、結合するデータフレームの列名がすべて同一であれば、各データフレームを単純に縦に並べたのと同等の構造のデータフレームが得られる。

一方、異なる列名を持つデータフレームを縦に結合する場合は、 `join` オプションにより使用する列を制御できる。例えば、以下の2つのデータフレームを縦に結合することを考える。

```
# データフレーム(df1)を作成
data1 = {"ID": [1, 2, 3], "商品名": ["商品A", "商品B", "商品C"], "価格": [100, 200, 300]}
df1 = pd.DataFrame(data1)
print(df1)
```

```
   ID  商品名   価格
0   1  商品A  100
1   2  商品B  200
2   3  商品C  300
```

```
# データフレーム(df2)を作成
data2 = {"ID": [4, 5, 6], "商品名": ["商品D", "商品E", "商品F"], "在庫": [1, 2, 3]}
df2 = pd.DataFrame(data2)
print(df2)
```

```
   ID  商品名  在庫
0   4  商品D   1
1   5  商品E   2
2   6  商品F   3
```

この場合、 `join="outer"` とするか、 `join` オプションを指定せずデフォルト値を使うと、各データフレームに存在する列をすべて使用して結合を行う。一部のデータフレームにしか存在しない列がある場合は、その列が存在しないデータフレームのデータは `NaN` として処理される。

```
# df1とdf2を縦方向に結合
concat_df = pd.concat([df1, df2], join="outer")
print(concat_df)
```

```
   ID  商品名     価格   在庫
0   1  商品A  100.0  NaN
1   2  商品B  200.0  NaN
2   3  商品C  300.0  NaN
0   4  商品D    NaN  1.0
1   5  商品E    NaN  2.0
2   6  商品F    NaN  3.0
```

```
# df1とdf2を縦方向に結合
concat_df = pd.concat([df1, df2])
print(concat_df)
```

```
   ID  商品名     価格   在庫
0   1  商品A  100.0  NaN
1   2  商品B  200.0  NaN
2   3  商品C  300.0  NaN
0   4  商品D    NaN  1.0
1   5  商品E    NaN  2.0
2   6  商品F    NaN  3.0
```

また、`join="inner"`とすると、全データフレームに共通する列のみを使用して結合を行う。

```
# df1とdf2を縦方向に結合
concat_df = pd.concat([df1, df2], join="inner")
print(concat_df)
```

```
   ID  商品名
0   1  商品A
1   2  商品B
2   3  商品C
0   4  商品D
1   5  商品E
2   6  商品F
```

pandasによるデータ品質の向上

データ品質

データ品質とは、特定の目的でデータを使用する際に、使用するデータがどれだけニーズを満たしているかを指す。品質が悪いデータの例として、日本全国の合計の人口が知りたく47都道府県それぞれの人口のデータを集めた場合を考える。このとき、データの品質が悪く、東北や九州の県の人口が抜けていたり、北海道の人口が本来の10倍の値など間違えた値になっていたりすると、日本全国の合計の人口が分からず目的を達成することができない。

品質の悪いデータを利用する場合、そのまま使うと正しい結果を得ることができず目的を達成できないため、利用できるように、人力もしくは機械的に、データの状態を確認し、問題がある箇所を修正し、使えるように整える必要がある。

データの中の破損した部分や不正確な部分などを特定し、削除・修正して、データの品質を上げる作業をデータクレンジングという。

実際に、品質が良いデータと悪いデータを比べて、統計量がどのように変わるか比較してみよう。

まずは、品質が良いデータをみてみよう。

以下は四国の2019年度の人口である。以下のデータは、総務省統計局が整備し、独立行政法人統計センターが運用管理している政府統計の総合窓口e-Statから引用したデータである。

県	総人口（人）
愛媛県	1,339,000
香川県	956,000
徳島県	728,000
高知県	698,000

※出典：政府統計の総合窓口(e-Stat) | 都道府県・市区町村のすがた（社会・人口統計体系） https://www.e-stat.go.jp/regional-statistics/ssdsview

上記のデータを元に、四国の2019年度の人口の平均値、最大値、最小値を求める。

統計量	人
平均値	930,250
最大値	1,339,000
最小値	698,000

統計量とデータから、2019年度の四国の最大人口の県は愛媛県で、最小人口の県は高知県であることがわかる。

続いて、品質が悪いデータをみてみよう。今回は、ある県の人口が間違えていたパターンと、ある県の人口のデータが欠けていたパターンの二つのパターンを紹介する。

ある県の人口が間違えていたパターンとして、高知県の人口を間違えて本来の10倍の値が記述されているデータが以下である。

県	総人口（人）
愛媛県	1,339,000
香川県	956,000
徳島県	728,000
高知県	6,980,000

上記のデータを元に、四国の2019年度の人口の平均値、最大値、最小値を求めたものが以下である。

統計量	人
平均値	2,500,750
最大値	6,980,000
最小値	728,000

高知県の人口が本来の10倍の値のデータを使って統計量を出すと、平均値が本来の値の約2.7倍となり、最大値は高知県の人口（本来は愛媛県）で、最小値は徳島県の人口（本来は高知県）と、本来の統計量と大きく乖離する結果となり、正しい結果を得ることができない。

ある県の人口のデータが欠けていたパターンとして、愛媛県の人口の値が欠けていたデータが以下である。

県	総人口（人）
愛媛県	
香川県	956,000
徳島県	728,000
高知県	698,000

上記のデータを元に、四国の2019年度の人口の平均値、最大値、最小値を求めたものが以下である。平均値は、香川県・徳島県・高知県の三県で求めている。

統計量	人
平均値	794,000
最大値	956,000
最小値	698,000

愛媛県の人口の値が欠けていたデータのデータを使って統計量を出すと、平均値が本来の値の約0.9倍となり、最大値は香川県の人口（本来は愛媛県）と、本来の統計量と異なる結果となり、正しい結果を得ることができない。

以上のように、品質が悪いデータをそのまま使うと、本来求めたい結果を得ることができないため、使用する前にデータの状態を確認し、必要であれば修正する必要がある。

今回はデータ内に、本来あるべきだが得られていないデータ（欠損値）や他の値から大きく外れた値（外れ値）が含まれる場合、pandasを用いてデータクレンジングを行う方法について以下の流れで紹介する。

1. データの欠損値・外れ値の確認
2. 欠損値・外れ値の置き換え・削除

pandasを用いたデータの欠損値・外れ値の確認

pandasが提供する機能を用いることで、データフレームに含まれる欠損値や外れ値を確認できる。 pandasにおける欠損値や外れ値の確認には様々な方法があるが、ここではいくつかの方法を一例として紹介する。

欠損値の確認

データの欠損値を確認するには、 `DataFrame` クラスの `isnull()` メソッドを使用する。 `isnull()` メソッドは、データフレームの各要素について欠損値かどうか判定し、元のデータフレームの各要素をTrue（欠損値）またはFalse（欠損値ではない）に置き換えたデータフレームを返却する。

以下に、 `isnull()` メソッドを使用して欠損値を確認するサンプルプログラムを示す。

```
import pandas as pd
import numpy as np

# データフレームを作成
df = pd.DataFrame({
    "A": [1, None, 3, 4],
    "B": [5, 6, np.nan, 8],
    "C": [9, 10, 11, 12]
})
print(df)
```

```
     A    B   C
0  1.0  5.0   9
1  NaN  6.0  10
2  3.0  NaN  11
3  4.0  8.0  12
```

```
# 欠損値を確認
print(df.isnull())
```

```
       A      B      C
0  False  False  False
1   True  False  False
2  False   True  False
3  False  False  False
```

上記のプログラムでは、欠損値を含むデータフレーム `df` を作成した後に、 `df.isnull()` を呼び出すことで、データフレームの各要素について欠損値かどうかを判定している。

まずはじめにデータフレーム `df` を作成する際、 `A` 列の行番号 `1` の要素に `None` を、 `B` 列の行番号 `2` の要素に `np.nan` を欠損値として使用している。 `np.nan` はNumPyにおいて非数値 (NaN: Not a Number) を表す定数である。 `np.nan` 、 `None` のいずれも、データフレームの要素として指定した場合、欠損値として処理される。作成したデータフレーム `df` の出力結果を見ると、欠損値はいずれも `NaN` と表されていることがわかる。

次に、 `print(df.isnull())` の実行結果として出力されたデータフレームを見ると、元のデータフレームにおいて `None` と `np.nan` になっていた箇所に対応するセルは `True` となり、それ以外のセルは `False` となっていることがわかる。したがって、 `df.isnull()` の結果から、元のデータフレームの各セルについて欠損値の有無が確認できる。

上記の例では、一つ一つのセルについて欠損しているかどうかを出力しているが、以下のように `DataFrame` クラスの `any()` メソッドを使用することで、各列や各行、もしくはデータフレーム全体に欠損値があるかどうかを確認することもできる。

```
# 列ごとに欠損値の有無を確認
print(df.isnull().any(axis=0))
```

```
A     True
B     True
C    False
dtype: bool
```

```
# 行ごとに欠損値の有無を確認
print(df.isnull().any(axis=1))
```

```
0    False
1     True
2     True
3    False
dtype: bool
```

```
# データフレーム全体で欠損値の有無を確認
print(df.isnull().any(axis=None))
```

```
True
```

`any()` メソッドは、指定した範囲ごとに `True` が存在するかどうかを判定するメソッドである。判定を行う範囲は、引数の `axis` で以下のように指定する。

- `0` : 列ごとに判定する。
- `1` : 行ごとに判定する。
- `None` : データフレーム全体で判定する。

先ほど例に挙げたプログラムでは、まず `df.isnull()` で欠損値のセルに `True` が入ったデータフレームを作成し、そのデータフレームに対して `any()` メソッドを呼び出すことで、元のデータフレームに欠損値がある範囲については `True` を、そうでない範囲については `False` を出力している。

`any()` メソッドを使用することで、欠損値の有無をセルごとに一つ一つ確認する代わりに、指定した範囲で要約して確認できる。

外れ値の確認

データの外れ値を確認するには、第2章でも扱った、 `DataFrame` クラスの `describe()` メソッドや、 `Series` クラスの `mean()` メソッド等で各種統計量を算出して、それをもとに判定を行う。

はじめに、 `describe()` メソッドで統計量を一括して算出することで、データの大まかな分布を把握する。外れ値は他のデータと大きく異なる値であるため、データの分布から大きく逸脱した値を見つけることで、外れ値を特定できる。

外れ値を確認するために、 `describe()` メソッドを使用してデータフレームに格納されているデータの統計量を算出する例を示す。

```
# データフレームを作成
df = pd.DataFrame({
    "A": [1, 5, 2, 1, 4, 3, 4],
    "B": [3, 2, 100, 3, 2, 1, 2],
    "C": [2, 4, 2, 3, 1, 4, 2]
})
print(df)
```

```
   A    B  C
0  1    3  2
1  5    2  4
2  2  100  2
3  1    3  3
4  4    2  1
5  3    1  4
6  4    2  2
```

```
# 外れ値を確認するため統計量を計算
print(df.describe())
```

```
              A           B         C
count  7.000000    7.000000  7.000000
mean   2.857143   16.142857  2.571429
std    1.573592   36.983909  1.133893
min    1.000000    1.000000  1.000000
25%    1.500000    2.000000  2.000000
50%    3.000000    2.000000  2.000000
75%    4.000000    3.000000  3.500000
max    5.000000  100.000000  4.000000
```

上記のプログラムでは、外れ値を含むデータフレーム `df` を作成した後に、 `df.describe()` を呼び出すことで、 `df` の各種統計量を出力している。

作成したデータフレーム `df` の出力結果を見ると、ほとんどの要素は1〜5となっているが、 `B` 列の行番号 `2` の要素のみ100となっている。この値が特定したい外れ値である。なお、今回は分かりやすく極端な値を使用しているため、データフレーム `df` の出力結果を見れば100が外れ値であることは一目瞭然だが、実際にデータを分析する際は、データの量が膨大だったり、一見するだけでは外れ値かどうか判断しづらかったりするため、これから説明するように統計量から判断する必要がある。

外れ値を実際に特定する方法の前に、統計量の出力結果から、外れ値が存在するデータの特徴を読み取る例を解説する。 `df.describe()` の出力結果を見ると、外れ値が存在する `B` 列のいくつかの統計量が特徴的な値を示していることがわかる。

まず、平均値（ `mean` ）、標準偏差（ `std` ）、最大値（ `max` ）の値が、他の `A` および `C` 列の値よりかなり大きくなっている。このことから、 `B` 列は他の列と比べて大きな値を含み、かつ値のばらつきが大きいことが読み取れる。しかし、これだけでは `B` 列がたまたま他の列と異なる分布をしているに過ぎない可能性があり、外れ値があるとは断定できない。

次に、 `B` 列の平均値（ `mean` ）と中央値（ `50%` ）を比較すると、二者の値が大きく異なっていることがわかる。中央値はデータ全体の中間の順位の値を示すため、データの中に他の値と大幅に異なる値が存在したとしても、中央値は大多数の値に近い値になる。一方で、平均値はすべての値の和をデータ数で割って計算されるため、データの中に極端に大きな値や小さな値が存在すると、その値の影響を強く受けて、平均値

が大多数の値とは離れた値になる傾向がある。今回、平均値が中央値よりかなり大きいことから、一部の値が極端に大きな値になっている可能性が考えられる。

外れ値を実際に特定するには、どういった値を外れ値として扱うか定義する必要がある。これにはいくつかの方法があるが、ここでは簡易的な方法として、平均値と標準偏差を用いる方法と、四分位数を用いる方法を解説する。

平均値と標準偏差を用いる場合は、平均値をμ、標準偏差をσで表すと、μ ± 2σ もしくは μ ± 3σ の範囲外の値を外れ値として定義する。この式は、正規分布と呼ばれる一般的な形状をした分布において、大多数の値が含まれる範囲を表したもので、μ ± 2σ の範囲には約95%、μ ± 3σ の範囲には約99.7%の値が含まれる。前述した `df` の `B` 列の例では、平均値（ `mean` ）は約16.1、標準偏差（ `std` ）は約37.0のため、μ ± 2σ を用いる場合はおよそ-57.9〜90.1の範囲に入らない値が外れ値となり、μ ± 3σ を用いる場合はおよそ-94.9〜127.1の範囲に入らない値が外れ値となる。したがって、前者では100は外れ値となる一方、後者では100は外れ値としてみなされない。これは、データのサイズが小さいときに外れ値が存在すると、平均値や標準偏差の値が大幅に変動するため、μ ± 3σ の範囲も大きくなってしまったためである。

四分位数を用いる場合は、第1四分位数をQ1、第3四分位数をQ3、Q3-Q1をIQR（Interquartile Range: 四分位範囲）で表すと、Q1 - 1.5 × IQR 〜 Q3 + 1.5 × IQR の範囲外の値を外れ値として定義する。Q1、Q3はそれぞれ全体の25%、75%に位置する値を示したものであるため、そこからさらに範囲を拡大した Q1 - 1.5 × IQR 〜 Q3 + 1.5 × IQR には大多数の値が含まれることになる。なお、正規分布において、この範囲はおよそ μ ± 2.7σ の範囲に相当する。前述した `df` の `B` 列の例では、Q1(`25%`)は2、Q3(`75%`)は3のため、IQRは Q3 - Q1 = 1 となり、Q1 - 1.5 × IQR 〜 Q3 + 1.5 × IQR は 0.5 〜 4.5 の範囲となる。したがって、100は範囲外の値のため外れ値とみなされる。このように四分位数を用いて外れ値を定義すると、平均値や標準偏差と比べて値が変動しづらいため、データのサイズが小さい場合でも外れ値を検知しやすい傾向がある。

上記で行った計算をプログラム上で行う場合は、 `mean()` や `std()` などの個別に統計量を算出するメソッドを使用する。以下は、平均値と標準偏差を用いて、 `B` 列の外れ値を特定する例である。

```
# 平均値を算出
mean = df["B"].mean()
# 標準偏差を算出
std = df["B"].std()
# μ - 2σ を計算
lower_bound = mean - 2 * std
# μ + 2σ を計算
upper_bound = mean + 2 * std
# 外れ値を含む行をフィルタリング
outlier = df[(df["B"] < lower_bound) | (df["B"] > upper_bound)]
print(outlier)
```

```
    A    B  C
2  2  100  2
```

上記のプログラムでは、データフレーム `df` の `B` 列の平均値と標準偏差から外れ値を特定し、外れ値を含む行を出力している。

まず、 `df["B"]` で取得した `Series` オブジェクトの `mean()` メソッドにより `B` 列の平均値を算出する。同様に、 `std()` メソッドを使用して `B` 列の標準偏差を算出する。次に、 `lower_bound = mean - 2 * std` 、 `upper_bound = mean + 2 * std` とすることで、μ - 2σ と μ + 2σ をそれぞれ `lower_bound` 、 `upper_bound` に格納する。最後に、 `B` 列の値が μ ± 2σ の範囲外の場合に外れ値であると見なすため、 `df[(df["B"] < lower_bound) | (df["B"] > upper_bound)]` により、 `B` 列の値が `lower_bound` より小さいか、 `upper_bound` の値より大きい行をフィルタリングしている。最後に、フィルタリングされた行を出力することで、外れ値を特定する。

実行結果から、行番号 `2` の行が外れ値 `100` を含んでいることが確認できる。

続いて、四分位数を用いて `B` 列の外れ値を特定する例を以下に示す。

```
# 第1四分位数を算出
q1 = df["B"].quantile(0.25)
# 第3四分位数を算出
q3 = df["B"].quantile(0.75)
# 四分位範囲（IQR）を算出
iqr = q3 - q1
# Q1 - 1.5 × IQR を計算
lower_bound = q1 - 1.5 * iqr
```

```
# Q3 + 1.5 × IQR を計算
upper_bound = q3 + 1.5 * iqr
# 外れ値を含む行をフィルタリング
outlier = df[(df["B"] < lower_bound) | (df["B"] > upper_bound)]
print(outlier)
```

```
    A    B  C
2  2  100  2
```

上記のプログラムも、前述した平均値と標準偏差から外れ値を特定するプログラムと同様のことを行っている。

まず、第1四分位数と第3四分位数を計算するため、`quantile()`メソッドを用いている。`quantile()`メソッドは、データを昇順に並べたときに、引数で指定した割合の位置に当たる値を取得するメソッドである。したがって、`quantile(0.25)`では第1四分位数を、`quantile(0.75)`では第3四分位数を算出できる。次に、IQR（四分位範囲）を算出したうえで、Q1 - 1.5 × IQR と Q3 + 1.5 × IQR をそれぞれ算出して、`lower_bound`および`upper_bound`とする。最後に、`df[(df["B"] < lower_bound) | (df["B"] > upper_bound)]`により、`B`列に外れ値を含む行をフィルタリングする。

pandasによる欠損値や外れ値の処理

欠損値や異常値を処理するには、欠損値を他のデータで補間（置き換え）する、欠損値を含む行を削除する、などの方法がある。 pandasでは、欠損値や異常値を置き換え・削除するためのいくつかの機能が提供されている。

欠損値の置き換え・削除

欠損値を特定の値で置き換えるには`fillna()`メソッドを使用する。以下は、欠損値を0で置き換える例である。

```
# データフレームを作成
df = pd.DataFrame({
    "A": [1, 2, 3, 4, None],
    "B": [5, 6, 7, 8, 9],
    "C": [None, 10, 11, 12, 13]
})
print(df)
```

```
     A  B     C
0  1.0  5   NaN
1  2.0  6  10.0
2  3.0  7  11.0
3  4.0  8  12.0
4  NaN  9  13.0
```

```
# 欠損値を0で置き換え
df_replaced = df.fillna(0)
print(df_replaced)
```

```
     A  B     C
0  1.0  5   0.0
1  2.0  6  10.0
2  3.0  7  11.0
3  4.0  8  12.0
4  0.0  9  13.0
```

上記の例のように、`df.fillna()`メソッドの引数に値を指定することで、データフレーム内の欠損値（`NaN`）の箇所を指定した値で置き換えることができる。実行結果から、`A`列の行番号`4`および`C`列の行番号`0`の値が0に置き換わっていることがわかる。

また、0のような一定の値で置き換える代わりに、以下のように列ごとに算出した平均値で置き換えることもできる。

```
# 列ごとに平均値を算出
mean = df.mean()
print(mean)
```

```
A     2.5
B     7.0
C    11.5
dtype: float64
```

```
# 欠損値を平均値で置き換え
df_replaced = df.fillna(mean)
print(df_replaced)
```

```
     A  B     C
0  1.0  5  11.5
1  2.0  6  10.0
2  3.0  7  11.0
3  4.0  8  12.0
4  2.5  9  13.0
```

上記のプログラムでは、列ごとの平均値を計算するため、`DataFrame`オブジェクトに対して`mean()`メソッドを呼び出している。これにより、列ごとに平均値が格納された`Series`オブジェクトが得られる。この`Series`オブジェクトを用いて`df.fillna(mean)`とすることで、各列の欠損値をその列に対応する平均値で置き換えている。実行結果から、`A`列の行番号`4`の欠損値は`A`列の平均値である2.5に置き換わり、`C`列の行番号`0`の欠損値は`C`列の平均値である11.5に置き換わっていることがわかる。

欠損値を含む行を削除するには、以下の例のように`dropna()`メソッドを使用する。

```
# 欠損値を含む行を削除
df_dropped = df.dropna()
print(df_dropped)
```

```
     A  B     C
1  2.0  6  10.0
2  3.0  7  11.0
3  4.0  8  12.0
```

`df.dropna()`とすることで、データフレーム`df`から欠損値（`NaN`）を一つでも含む行を削除する。実行結果から、欠損値を含んでいた行番号`0`と`4`の行が削除されていることが確認できる。

外れ値の置き換え・削除

外れ値を置き換えるには、第2章で扱ったのと同様に、`loc[]`を用いて値の置き換えを行う。以下は、列`B`の値が10より大きい場合は外れ値として、10で置き換える例である。

```
# データフレームを作成
df = pd.DataFrame({
    "A": [1, 2, 3, 4, 5],
    "B": [3, 3, 4, 4, 15],
})
print(df)
```

```
   A   B
0  1   3
1  2   3
2  3   4
3  4   4
4  5  15
```

```
# 列"B"の値が10より大きい場合に、10で置き換える
df.loc[df["B"] > 10, "B"] = 10
print(df)
```

```
   A   B
0  1   3
1  2   3
2  3   4
3  4   4
4  5  10
```

上記の例では、`df.loc[]` により行と列を指定して、値を置き換えている。行の指定を `df["B"] > 10` とすることで、行をフィルタリングするときと同様に、列 `B` の値が10より大きい行のみを対象とする。したがって、`df.loc[df["B"] > 10, "B"] = 10` により、列 `B` の値が10より大きい行に対して、列 `B` の値が10に置き換えられるため、列 `B` の要素のうち10より大きい値をすべて10に置き換えることができる。

外れ値を含む行を削除するには、外れ値を含まない行のみを選択するようにフィルタリングを行えばよい。以下は、列 `B` の値が10より大きい行を削除する例である。

```
# データフレームを作成
df = pd.DataFrame({
    "A": [1, 2, 3, 4, 5],
    "B": [3, 3, 4, 4, 15],
})
print(df)
```

```
   A   B
0  1   3
1  2   3
2  3   4
3  4   4
4  5  15
```

```
# 列"B"の値が10より大きい行を削除する
df = df[df["B"] <= 10]
print(df)
```

```
   A  B
0  1  3
1  2  3
2  3  4
3  4  4
```

上記の例では、`df[]` で行を選択する条件を `df["B"] <= 10` とすることで、列 `B` の値が10以下の行のみをフィルタリングしている。これにより、列 `B` の値が10より大きい行が除外されるため、外れ値を含む行を削除したデータフレームを作成できる。

問題1

問題文

以下のようなpandasのデータフレーム `df` がある。出力例のように、 `施設` ごとの `月間利用者数` の合計の値を出力するプログラムを記述せよ。

```
   都道府県    施設  月間利用者数
0   X県  図書館   12000
1   X県  体育館    8000
2   X県  美術館   22000
3   Y県  図書館   15000
4   Y県  体育館   32000
5   Z県  体育館    9000
6   Z県  美術館   36000
```

出力

```
        月間利用者数
施設
体育館    49000
図書館    27000
美術館    58000
```

解答の雛形

```
import pandas as pd

data = {
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, 15000, 32000, 9000, 36000],
}

df = pd.DataFrame(data)

## ここに解答を入力
```

問題2

問題文

以下のようなpandasのデータフレーム `df1` と `df2` がある。出力例のように、データフレーム `df2` に存在しない `ID` を持つ行は `休館日` 列のデータの値が `NaN` になるように、 `df1` と `df2` を結合し出力するプログラムを記述せよ。

- データフレーム `df1`

```
   ID 都道府県    施設  月間利用者数
0   1   X県  図書館   12000
1   2   X県  体育館    8000
2   3   X県  美術館   22000
3   4   Y県  図書館   15000
4   5   Y県  体育館   32000
5   6   Z県  体育館    9000
```

```
8  6   7   Z県  美術館   36000
```

- データフレーム `df2`

```
1     ID  休館日
2  0   1  火曜日
3  1   2  月曜日
4  2   3  月曜日
5  3   6  水曜日
6  4   7  月曜日
```

出力

```
1     ID 都道府県    施設  月間利用者数  休館日
2  0   1   X県  図書館   12000  火曜日
3  1   2   X県  体育館    8000  月曜日
4  2   3   X県  美術館   22000  月曜日
5  3   4   Y県  図書館   15000  NaN
6  4   5   Y県  体育館   32000  NaN
7  5   6   Z県  体育館    9000  水曜日
8  6   7   Z県  美術館   36000  月曜日
```

解答の雛形

```
import pandas as pd

data1 = {
    "ID": [1, 2, 3, 4, 5, 6, 7],
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, 15000, 32000, 9000, 36000],
}

data2 = {
    "ID": [1, 2, 3, 6, 7],
    "休館日": ["火曜日", "月曜日", "月曜日", "水曜日", "月曜日"],
}


df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
## ここに解答を入力
```

問題3

問題文

以下のようなpandasのデータフレーム `df1` と `df2` がある。出力例のように、データフレーム `df1` に存在しない `ID` を持つ行は `月間利用者数` 列のデータの値が `NaN` になり、データフレーム `df2` に存在しない `ID` を持つ行は `休館日` 列のデータの値が `NaN` になるように、`df1` と `df2` を結合し出力するプログラムを記述せよ。

- データフレーム `df1`

```
1     ID 都道府県    施設  月間利用者数
2  0   1   X県  図書館   12000
```

```
3  1   2   X県  体育館    8000
4  2   3   X県  美術館   22000
5  3   4   Y県  図書館   15000
6  4   5   Y県  体育館   32000
7  5   6   Z県  体育館    9000
8  6   7   Z県  美術館   36000
```

- データフレーム `df2`

```
1     ID 都道府県     施設   休館日
2  0   8   Z県   図書館  火曜日
3  1   9   Z県  県民会館  水曜日
```

出力

```
1     ID 都道府県     施設    月間利用者数   休館日
2  0   1   X県   図書館  12000.0  NaN
3  1   2   X県   体育館   8000.0  NaN
4  2   3   X県   美術館  22000.0  NaN
5  3   4   Y県   図書館  15000.0  NaN
6  4   5   Y県   体育館  32000.0  NaN
7  5   6   Z県   体育館   9000.0  NaN
8  6   7   Z県   美術館  36000.0  NaN
9  0   8   Z県   図書館      NaN  火曜日
10  1   9   Z県  県民会館      NaN  水曜日
```

解答の雛形

```
import pandas as pd

data1 = {
    "ID": [1, 2, 3, 4, 5, 6, 7],
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, 15000, 32000, 9000, 36000],
}

data2 = {
    "ID": [8, 9],
    "都道府県": ["Z県", "Z県"],
    "施設": ["図書館", "県民会館"],
    "休館日": ["火曜日", "水曜日"],
}


df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
## ここに解答を入力
```

問題4

問題文

以下のようなpandasのデータフレーム `df` がある。出力例のように、 `df` に欠損値が存在するかどうかの真偽値で判定結果を出力するプログラムを記述せよ。 `df` に欠損値があれば `True` 、なければ `False` と出力すること。

```
    ID 都道府県   施設   月間利用者数
0   1   X県  図書館  12000.0
1   2   X県  体育館   8000.0
2   3   X県  美術館  22000.0
3   4   Y県  図書館      NaN
4   5   Y県  体育館      NaN
5   6   Z県  体育館   9000.0
6   7   Z県  美術館  36000.0
```

出力

```
True
```

解答の雛形

```
import pandas as pd

data = {
    "ID": [1, 2, 3, 4, 5, 6, 7],
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, None, None, 9000, 36000],
}


df = pd.DataFrame(data)
## ここに解答を入力
```

問題5

問題文

以下のようなpandasのデータフレーム `df` がある。平均値をμ、標準偏差をσで表し、$\mu \pm 2\sigma$の範囲外の値を外れ値とした場合、出力例のように、 `df` の `月間利用者数` の値が外れ値である行を出力するプログラムを記述せよ。

```
    ID 都道府県   施設   月間利用者数
0   1   X県  図書館    12000
1   2   X県  体育館     8000
2   3   X県  美術館    22000
3   4   Y県  図書館  2500000
4   5   Y県  体育館     5000
5   6   Z県  体育館     9000
6   7   Z県  美術館    36000
```

出力

```
    ID 都道府県   施設   月間利用者数
3   4   Y県  図書館  2500000
```

解答の雛形

```
import pandas as pd

data = {
    "ID": [1, 2, 3, 4, 5, 6, 7],
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, 2500000, 5000, 9000, 36000],
}


df = pd.DataFrame(data)
## ここに解答を入力
```

問題6

問題文

以下のようなpandasのデータフレーム `df` がある。 `df` の `月間利用者数` の平均値を求め、出力例のように、欠損値を `月間利用者数` の平均値で置き換えて出力するプログラムを記述せよ。なお、平均値を求める際、 `df` の列名を指定して平均値を求めること。

```
1     ID 都道府県    施設    月間利用者数
2  0   1   X県  図書館  12000.0
3  1   2   X県  体育館   8000.0
4  2   3   X県  美術館  22000.0
5  3   4   Y県  図書館      NaN
6  4   5   Y県  体育館      NaN
7  5   6   Z県  体育館   9000.0
8  6   7   Z県  美術館  36000.0
```

出力

```
1     ID 都道府県    施設    月間利用者数
2  0   1   X県  図書館  12000.0
3  1   2   X県  体育館   8000.0
4  2   3   X県  美術館  22000.0
5  3   4   Y県  図書館  17400.0
6  4   5   Y県  体育館  17400.0
7  5   6   Z県  体育館   9000.0
8  6   7   Z県  美術館  36000.0
```

解答の雛形

```
import pandas as pd

data = {
    "ID": [1, 2, 3, 4, 5, 6, 7],
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, None, None, 9000, 36000],
}


df = pd.DataFrame(data)
## ここに解答を入力
```

問題7

問題文

以下のようなpandasのデータフレーム `df` がある。出力例のように、欠損値がある行を除いて出力するプログラムを記述せよ。

```
    ID 都道府県    施設    月間利用者数
0   1   X県   図書館  12000.0
1   2   X県   体育館   8000.0
2   3   X県   美術館  22000.0
3   4   Y県   図書館      NaN
4   5   Y県   体育館      NaN
5   6   Z県   体育館   9000.0
6   7   Z県   美術館  36000.0
```

出力

```
    ID 都道府県    施設    月間利用者数
0   1   X県   図書館  12000.0
1   2   X県   体育館   8000.0
2   3   X県   美術館  22000.0
5   6   Z県   体育館   9000.0
6   7   Z県   美術館  36000.0
```

解答の雛形

```
import pandas as pd

data = {
    "ID": [1, 2, 3, 4, 5, 6, 7],
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, None, None, 9000, 36000],
}


df = pd.DataFrame(data)
## ここに解答を入力
```

問題8

問題文

以下のようなpandasのデータフレーム `df` がある。 `月間利用者数` の値が500000より大きい値は外れ値であると仮定し、出力例のように、 `月間利用者数` の値が500000より大きいデータは、値を500000に置き換える。その後、置き換えた後のデータフレームを出力するプログラムを記述せよ。

```
   ID 都道府県   施設   月間利用者数
0   1   X県  図書館    12000
1   2   X県  体育館     8000
2   3   X県  美術館    22000
3   4   Y県  図書館  2500000
4   5   Y県  体育館     5000
5   6   Z県  体育館     9000
6   7   Z県  美術館    36000
```

出力

```
   ID 都道府県   施設  月間利用者数
0   1   X県  図書館   12000
1   2   X県  体育館    8000
2   3   X県  美術館   22000
3   4   Y県  図書館  500000
4   5   Y県  体育館    5000
5   6   Z県  体育館    9000
6   7   Z県  美術館   36000
```

解答の雛形

```python
import pandas as pd

data = {
    "ID": [1, 2, 3, 4, 5, 6, 7],
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, 2500000, 5000, 9000, 36000],
}


df = pd.DataFrame(data)
## ここに解答を入力
```

問題9

問題文

以下のようなpandasのデータフレーム `df` がある。平均値をμ、標準偏差をσで表し、$\mu \pm 2\sigma$の範囲外の値を外れ値とした場合、出力例のように、 `df` の `月間利用者数` の値が外れ値でない行を出力するプログラムを記述せよ。

```
   ID 都道府県   施設   月間利用者数
0   1   X県  図書館    12000
1   2   X県  体育館     8000
2   3   X県  美術館    22000
3   4   Y県  図書館  2500000
4   5   Y県  体育館     5000
5   6   Z県  体育館     9000
6   7   Z県  美術館    36000
```

出力

```
   ID 都道府県   施設  月間利用者数
0   1   X県  図書館   12000
```

```
3  1   2   X県  体育館    8000
4  2   3   X県  美術館   22000
5  4   5   Y県  体育館    5000
6  5   6   Z県  体育館    9000
7  6   7   Z県  美術館   36000
```

解答の雛形

```
import pandas as pd

data = {
    "ID": [1, 2, 3, 4, 5, 6, 7],
    "都道府県": ["X県", "X県", "X県", "Y県", "Y県", "Z県", "Z県"],
    "施設": ["図書館", "体育館", "美術館", "図書館", "体育館", "体育館", "美術館"],
    "月間利用者数": [12000, 8000, 22000, 2500000, 5000, 9000, 36000],
}


df = pd.DataFrame(data)
## ここに解答を入力
```

4章: Webスクレイピング

Webスクレイピング

Webスクレイピングとは、WebサイトからHTML形式のデータを取得し、特定の情報を自動で抽出するコンピューターソフトウェア技術である。

Webスクレイピングを使うことで、たとえば、ニュースサイトの記事一覧を取得し、タイトルや本文などを抽出したり、eコマースサイトの商品情報を収集し、競合の価格分析などに利用したりすることが可能である。ただし、スクレイピングするWebサイトによっては、利用規約に違反する場合があるため、使用する前には必ず利用規約を確認することが必要である。

Webスクレイピングは以下の流れで行う。

1. Webサイトにアクセスし、HTMLのソースコードを取得する
2. 取得したHTMLのソースコードを解析し、望むデータを抽出する
3. 抽出したデータを加工し、利用する

Webスクレイピングの具体的な方法について解説する前に、HTMLやHTTPリクエスト、HTTPレスポンスといったWeb技術とそれらを用いてどのようにWebサイトが表示されているか理解する必要がある。そのため、まずは、Webスクレイピングに必要なWeb技術と、Web技術を用いてどのようにWebサイトが表示されるかについて解説する。

Webスクレイピングに関わるWeb技術

HTML

HTMLはHypertext Markup Languageの略称である。HTMLはWebページを表現するためのマークアップ言語であり、Webアプリケーションで必ず利用される。マークアップ言語とは、コンピュータによって処理されるコンピュータ言語の一種であり、データ中に特定の記法を用いて何らかの情報を埋め込むための言語である。HTMLで記述した内容は、ブラウザを通してWebサイトの画面として表示される。

ブラウザとは、インターネット上のWebページを表示するためのソフトウェアで、HTMLやCSS、JavaScriptなどを解析し、Webページを表示する。主要なブラウザとして、Google Chrome、Mozilla Firefox、Microsoft Edge、Safariがある。

HTMLでは `<` と `>` で囲まれたタグという特殊な記法を用いることで、文書の構造や装飾、ボタンなどを表現できる。タグは、開始タグと終了タグのセットで構成され、開始タグは `<タグ名>` 、終了タグは `</タグ名>` の形式で記述される。 例えば、以下のように `HTML について学習しよう` という文字列を `h1` タグで囲むことで、該当文字列を見出しとして表現できる。 `HTML について学習しよう` という文字列はコンテンツ、タグとコンテンツを含めた `<h1>HTMLについて学習しよう</h1>` を要素という。

```
1  <h1>HTMLについて学習しよう</h1>
```

ここからは、Webページを構成するためのHTML形式のデータの構造について解説する。

以下がWebページを構成するためのHTMLドキュメントの雛形である。

`<!DOCTYPE html>` は、文書型宣言といい、正式なHTMLドキュメントにおいては先頭に必ず記述しなければならない。

`<html>` タグで、Webページの要素全体を囲む必要がある。

Webページを構成する要素は、大きく `<head>` 要素と `<body>` 要素の二つに分けることできる。

`<body>` 要素に、テキストやボタン、画像などWebページで表示するコンテンツを記述する。

`<head>` 要素には、Webページで表示するコンテンツ以外に、HTML形式のデータとして持ちたい情報を記述する。例えば、ページのタイトルやメタ情報、Webページのレイアウトなどを記述するスタイルシートなどがある。

`<title>` 要素には、ページのタイトルを指定する。 `<title>` 要素で記述した内容はブラウザのタブに表示される。

`<meta>` 要素に、Webページが持つメタ情報を記述する。開始タグには、 `charset="utf-8"` や `name="viewport"` のように属性という

コンテンツとして表示させたくないが要素として持ちたい情報を記述することができ、 `<meta>` 要素では属性にメタ情報を記述する。また、 `<meta>` 要素には終了タグは不要である。このように、HTMLの要素には開始タグのみで記述できるものも存在する。

```
<!DOCTYPE html>
<html>

<head>
    <title>Example page</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
</head>

 <body>
 </body>

 </html>
```

上記の雛形に、Webページとして表示するコンテンツを記述したものが以下である。

`<head>` 要素に追加した要素からみていこう。

`<head>` 要素には、新しく、 `<style>` 要素を追加した。 `<style>` 要素では、属性に `type="text/css"` と指定することで、Webページのレイアウトや背景や文字の色などを指定するCascading Style Sheets (CSS) を記述できる。本節はHTMLの解説であるため、詳しい説明は割愛するが、以下の例では、CSSを記述して、 `<body>` 要素の背景色を灰色したり、 `<div>` 要素の背景色を `<body>` 要素よりも白くするなどの指定をしている。

続いて、 `<body>` 要素に追加した要素をみていこう。

`<h1>` 要素には、見出しを記述する。 `<h1>` から `<h6>` まであり、数字が小さいほど、表示される文字が大きくなる。

`<p>` 要素には、段落を示している。 `<p>` タグで囲まれたコンテンツが一つの段落であることを示す。

`<ul>` タグで囲むことで、順序なしのリストを表現できる。リストの中に入る要素は、 `<li>` タグで囲み記述する。順序ありのリストを表現したい場合は、 `<ul>` タグではなく、 `<ol>` を使用する。

`<table>` 要素を使うことで、表を表現できる。 `border="1"` で表を囲む枠の大きさを定義している。 `<table>` 要素は、表の見出しを記述する `<thead>` 要素と表の本体を記述する `<tbody>` 要素で構成される。 `<thead>` 要素や `<tbody>` 要素に `<tr>` 要素を記述することで、表の行を定義できる。 `<tr>` 要素に、各データのセルを示す `<td>` 要素および見出しセルを示す `<th>` 要素を記述することで、表のセルを定義できる。

`<a>` 要素では、 `href` 属性を用いて、別のWebサイトなど他のURLへのハイパーリンクを作成する。 `<a>` タグで囲まれたコンテンツ部分がハイパーリンクになる。

```
<!DOCTYPE html>
<html>

<head>
    <title>Example page</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <style type="text/css">
        body {
            background-color: #f0f0f2;
            margin: 0;
            padding: 0;
            font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", He
        }

        div {
            width: 600px;
            margin: 5em auto;
            padding: 2em;
```

```
            background-color: #fdfdff;
            border-radius: 0.5em;
            box-shadow: 2px 3px 7px 2px rgba(0, 0, 0, 0.02);
        }
    </style>
</head>

<body>
    <div>
        <h1>HTMLについて学習しよう</h1>
        <p>このサイトでは、以下の項目について学びます。</p>
        <ul>
            <li>HTMLについて</li>
            <li>ブラウザについて</li>
            <li>タグについて</li>
        </ul>
        <p>HTMLのタグには以下のような種類がある</p>
        <table border="1">
            <thead>
                <tr>
                    <th>タグ</th>
                    <th>説明</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>h1</td>
                    <td>見出しタグ</td>
                </tr>
                <tr>
                    <td>p</td>
                    <td>行タグ</td>
                </tr>
            </tbody>
        </table>

        <p>詳しくは内容は<a href="https://www.google.com/?hl=ja">Google</a>で検索しよう</p>
    </div>
</body>

</html>
```

上記のHTMLドキュメントをブラウザ（Google Chrome）で表示すると、以下のように表示される。

HTTPリクエスト / HTTPレスポンス

HTTP

HTTPとは、Hyper Text Transfer Protocolの略称であり、クライアントとサーバ間で、データをやりとりするために用いられる通信規約（プロトコル）である。HTTPを使うことで、HTMLの文書や画像、音声データなどをクライアントとサーバ間でやりとりできる。

HTTPリクエスト

HTTPリクエストは、クライアントがサーバにどのようなことをしてほしいか要求を伝えるメッセージである。HTTPリクエストは、ヘッダとボディでメッセージは構成される。ヘッダには、サーバに要求する動作を表すメソッドとパス、HTTPバージョン、要求の詳細で構成される。

主要なメソッドとして以下が挙げられる。

- GET：サーバに対してリソースを取得するために使用する。たとえば、ブラウザは、GETリクエストを使用してHTMLドキュメントを取得しのWebページを表示する。
- POST：サーバに対してデータを送信するために使用する。サーバにデータを送って、新しくアカウントなどのデータを作成するときなどに使用する。
- PUT：サーバに対してデータを送信し、データを更新するために使用する。
- DELETE：指定したデータを削除するために使用する。

パスは、クライアントが要求する資材のサーバ上での所在を表す。HTTPバージョンは、「HTTP/1.1」や「HTTP/2.0」などメッセージが準拠するHTTP規格を示す。

また、ボディにはリクエストに対して送信したいデータが入っている。通常、POSTやPUTメソッドを使用してサーバにデータを送信する場合に、リクエストボディにデータが入っている。

HTTPレスポンス

HTTPレスポンスは、クライアントからのHTTPリクエストを受けて、サーバがクライアントに返すメッセージである。HTTPレスポンスも、ヘッダとボディでメッセージは構成される。

ヘッダは、HTTPバージョン、HTTPステータスコード、ステータスメッセージ、コードの詳細で構成される。

HTTPステータスコードとは、要求に対する応答の種類を、三桁の数字であるコードである。また、HTTPステータスコードを人間にも分かりやすい表現にした文字列がステータスメッセージである。たとえば、`500` というステータスコードのステータスメッセージは `Internal Server Error` であり、ステータスメッセージから、サーバ内部のエラーが発生した場合のコードであることがわかる。

HTTPステータスコードは、百の位の数字で以下のように分類される。

- 100番台（Informational）：リクエストは受理され、処理は続行されることを表す。
- 200番台（Successful）：リクエストが成功したことを表す。
- 300番台（Redirection）：リクエストを完了させるために追加で処理を行う必要があることを表す。
- 400番台（Client Error）：リクエストにエラーがあることを表す。クライアントのリクエストに問題がある場合に返される。
- 500番台（Server Error）：サーバによるエラーであることを表す。サーバ側に問題がある場合に返される。

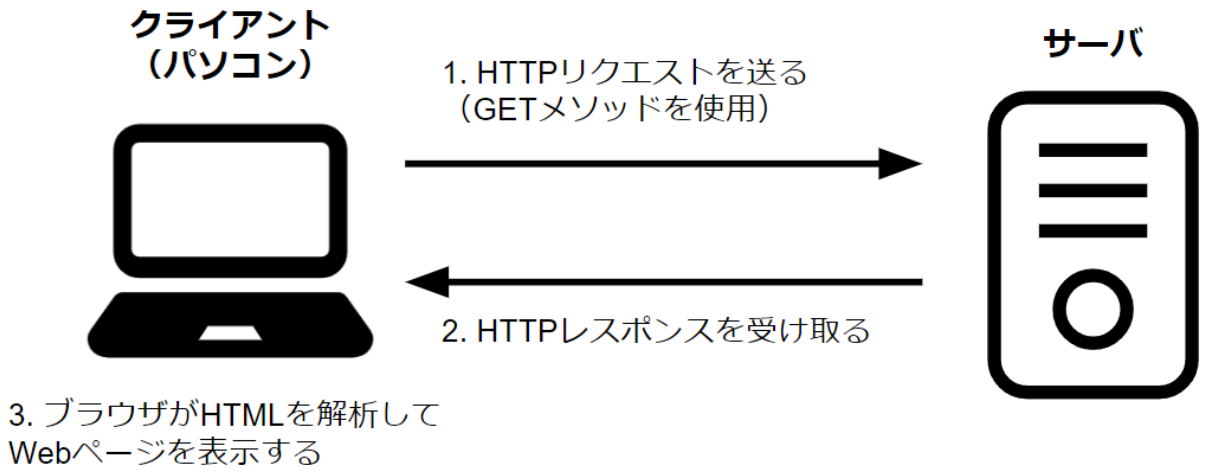
例えば、`200 OK` はリクエストが成功し、レスポンスにデータが含まれていることを示す。一方、`404 Not Found` はリクエストしたWebページやデータが存在しないことを示す。`500 Internal Server Error` はサーバ内部でエラーが発生したことを示す。

ボディに、HTTPリクエストで要求されたデータが入っている。たとえば、HTTPリクエストでWebページのデータを要求され、リクエストが成功すれば、HTTPレスポンスのボディにはHTML形式のデータが入っている。

Webページを表示する流れ

ここまでの説明を踏まえて、クライアントからリクエストを出してから、Webページが表示されるまでの流れについて解説する。

クライアントは以下の流れで、リクエストを出し、Webページを表示する。



1. HTTPリクエストを送る

- クライアントは、サーバに対して表示したページのHTMLドキュメントを送るようにHTTPリクエストを送る。クライアントはデータを取得したいため、GETメソッドを使用する。

2. HTTPリクエストを受け取る

- サーバはHTTPリクエスト受けて、処理を実行し、HTTPレスポンスを作成しクライアントに送り返す。HTTPリクエストでHTMLドキュメントを送るように要求を受けたため、HTTPレスポンスのボディには、要求に基づいたHTML形式のデータが含まれている。

3. ブラウザでHTMLを解析してWebページを表示する

- ブラウザが、HTTPレスポンスのボディに含まれるHTMLドキュメントを解析して、Webページとして表示する。

requestsライブラリによるHTTPリクエスト

requestsライブラリは、PythonでHTTPリクエストを送信するためのライブラリである。Webページのデータを取得する際にrequestsライブラリを使うことで、HTTPリクエストの送信や、レスポンスの取得が簡単に行える。

以下は、https://example.com にHTTPリクエストを送信し、得られたHTTPレスポンスの内容を出力する例である。

```
import requests

response = requests.get("https://example.com")
print(response.status_code)
```

```
200
```

```
print(response.text)
```

```
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvet

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

上記の例では、`requests.get("https://example.com")` により https://example.com に対してHTTP GETリクエストを送信し、そのレスポンスを取得している。`requests.get()` の返却値は、HTTPレスポンスの各種情報を含んだ `Response` オブジェクトである。`response.status_code` と `response.text` により、返却された `Response` クラスの `status_code` および `text` プロパティから、HTTPレスポンスのステータスコードと本文を取得し、その内容を出力している。

実行結果から、ステータスコードは200となっており、HTTP GETリクエストが正常に処理されたことがわかる。また、レスポンスの本文として、https://example.com のページの内容がHTML形式で取得できていることが確認できる。なお、exmaple.com は実在するドメインのため、実際にリクエストが正しく取得出来ていることに注意せよ。

このようにrequestsライブラリでは、前節で説明したようなHTTPリクエストの送信やHTTPレスポンスの受信、HTTPレスポンスを解析してHTMLドキュメントやステータスコードなどの情報を抽出するといった処理を自動的に行うため、ライブラリの利用者は簡単にWebページの情報を取得できる。

BeautifulSoupライブラリによるデータ抽出

HTMLのパースとタグの抽出

前節でHTTPリクエストにより取得したデータから情報を抽出するには、HTML形式のデータを解析し、pythonのデータ構造に変換することで、目的のデータを取得できるようにする必要がある。このように、文字列のデータを解析してプログラムで扱えるようなデータ構造に変換することを一般に「パースする」という。

ここでは、PythonにおいてHTML形式のデータをパースするためのライブラリとしてBeautifulSoupを使用する。BeautifulSoupは、HTMLを解析して、タグや属性などの情報を抽出するための機能を提供している。

以下は、BeautifulSoupを使って、HTML形式の文字列データをパースするサンプルプログラムである。

```python
from bs4 import BeautifulSoup

html = """
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <p>Hello World!</p>
        <p>Hello BeautifulSoup!</p>
        <a href="https://example.com">Hyper Link</a>
    </body>
</html>
"""
soup = BeautifulSoup(html)
print(soup.title)
```

```
<title>Example</title>
```

上記のプログラムでは、解析対象のHTMLドキュメントとして、あらかじめ変数 `html` として定義した文字列を使用している。実際にスクレイピングを行う際は、前節で扱ったようにHTTPリクエストにより取得したHTMLドキュメントを対象に解析を行うが、ここでは説明のためにソースコードに埋め込んだデータを対象としている。

まず `from bs4 import BeautifulSoup` により、BeautifulSoupライブラリのモジュール（`bs4`）から `BeautifulSoup` クラスをインポートしている。`BeautifulSoup` クラスは、HTML形式の文字列データを自動的に解析して、タグや文字列などのデータにアクセスするための各種機能を提供するクラスである。

次に、`soup = BeautifulSoup(html)` により、`html` に格納されている文字列から `BeautifulSoup` クラスのオブジェクトとして `soup` を生成している。`BeautifulSoup` クラスは、オブジェクト生成時の引数で指定した文字列を解析し、タグや属性などの情報を抽出できるようにする。

最後の `print(soup.title)` が、実際にHTML形式の文字列データから情報を抽出して出力する部分である。`BeautifulSoup` クラスでは、解析したデータの各タグの中身に簡単にアクセスできるよう、HTML形式のデータにおけるタグの階層構造に合わせてオブジェクトのデー

タ構造が定義される。解析対象のHTML形式のデータに存在する各タグは、 `BeautifulSoup` クラスのオブジェクト（今回の例では `soup` ）に、タグの名前と同じ名前で定義されたプロパティによりアクセスできる。例えば、 `soup` オブジェクトの `title` プロパティにより `<title>` タグの情報にアクセスできるため、上記の例では `print(soup.title)` とすることで `<title>` タグの中身を出力している。

なお、上記の例では `print(soup.title)` としているため、 `soup.title` の文字列表現として `<title>` タグおよびその中身が出力されているが、 `soup.title` は実際には以下のように `Tag` クラスのオブジェクトである。

```
print(type(soup.title))
```

```
<class 'bs4.element.Tag'>
```

`Tag` クラスは、解析したタグの情報を保持し、そのタグの名前や属性、子要素のタグや文字列などへアクセスするための各種プロパティやメソッドを提供する。例えば、以下のように `soup.title` の `name` プロパティにアクセスすることで、 `<title>` タグの名前である `title` という文字列を取得できる。

```
print(soup.title.name)
```

```
title
```

なお、前述の例では `print(soup.title)` とすることにより、 `Tag` クラスの `__str__()` メソッドが呼ばれたため、 `Tag` クラスのオブジェクトである `soup.title` の文字列表現として `<title>Example</title>` が出力されていた。これは、 `soup.title.__str__()` を明示的に呼び出すことでも確認できる。

```
print(soup.title.__str__())
```

```
<title>Example</title>
```

`BeautifulSoup` クラスのオブジェクトのプロパティによりタグを参照すると、HTML形式のデータ内に存在する同名のタグのうち、初めに出現するタグの情報を取得する。今回のように `<title>` タグが一つしか存在しない場合はそのタグが取得できるが、上記のプログラム例における `<p>` タグのように複数存在する場合は、以下のように初めに出現する `<p>` タグである `<p>Hello World!</p>` のみが取得されることに注意せよ。

```
print(soup.p)
```

```
<p>Hello World!</p>
```

複数存在するタグをすべて取得したい場合や、二番目以降に出現するタグを取得したい場合は、以下のように `BeautifulSoup` クラスの `find_all()` メソッドを使用する。

```
print(soup.find_all("p"))
```

```
[<p>Hello World!</p>, <p>Hello BeautifulSoup!</p>]
```

```
print(type(soup.find_all("p")))
```

```
<class 'bs4.element.ResultSet'>
```

```
print(soup.find_all("p")[1])
```

```
<p>Hello BeautifulSoup!</p>
```

`find_all()` メソッドは、引数で指定した条件のタグをすべて取得し、リストのように扱える `ResultSet` クラスのオブジェクトとして返すメソッドである。第一引数を指定した場合、指定した文字列と同一の名前を持つタグをすべて取得する。上記の例では、`soup.find_all("p")` により、解析対象のHTMLドキュメント内に存在する `<p>` タグをすべて取得している。出力結果から、二番目以降に出現する `<p>` タグとして `<p>Hello BeautifulSoup!</p>` が取得できていることが分かる。

タグに定義されている属性を取得するには、辞書オブジェクトに対して値を参照するのと同様に、 `Tag` クラスのオブジェクトに対して、取得したい属性の名前をキーにして `[]` で値を参照する。以下は、 `<a>` タグから `href` 属性を取得する例である。

```
print(soup.a["href"])
```

```
https://example.com
```

また、特定の属性を持つタグを取得するには、 `find_all()` メソッドの引数として、 `属性名=値` といった形式で属性名と値を指定する。以下は、 `id` 属性が `name` のタグを取得する例である。

```
from bs4 import BeautifulSoup

html = """
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <p id="name">Alice</p>
        <p id="score">100</p>
    </body>
</html>
"""
soup = BeautifulSoup(html)
print(soup.find_all(id="name"))
```

```
[<p id="name">Alice</p>]
```

上記の例において、 `html` として定義したHTMLドキュメントには、 `<p>` タグが2つ存在する。このうち、 `id` 属性の値として `name` を持つタグは `<p id="name">` のみである。上記のプログラムでは、 `soup.find_all(id="name")` により `id` 属性が `name` のタグを取得している。出力結果から、目的のタグの情報として `<p id="name">Alice</p>` が取得できていることがわかる。

`find_all()` メソッドと同様に属性の値を指定しつつ、条件に一致するタグのうち初めに出現するタグのみを取得するには、 `find()` メソッドを使用する。 `find()` メソッドは、返却する値が単一の値になっていること以外は、 `find_all()` と同等の機能を持つ。

```
print(soup.find(id="name"))
```

```
<p id="name">Alice</p>
```

`find_all()` メソッドではタグを要素とする `ResultSet` が返却されていたが、上記の `find()` メソッドの出力結果では単一のタグのみを取得できていることがわかる。

特定のタグの開始タグと終了タグの内部に存在する文字列を取得するには、 `Tag` クラスの `get_text()` メソッドを使用する。

```
tag = soup.find(id="name")
print(tag.get_text())
```

```
Alice
```

上記のプログラムでは、 `soup.find(id="name")` の結果として得られる `Tag` クラスのオブジェクトを変数 `tag` に格納したうえで、 `tag.get_text()` によりタグの内部に存在する文字列を取得し、出力している。出力結果から、 `soup.find(id="name")` により得られる `<p id="name">Alice</p>` のうち、タグの内部に存在する文字列として `Alice` を取得できていることがわかる。

なお、HTML形式のデータからタグを取得する際、 `BeautifulSoup` クラスのオブジェクトに対してプロパティを参照したり `find_all()` や `find()` を使用したりする方法の他に、 `Tag` クラスのオブジェクトに対してプロパティを参照したり `find_all()` や `find()` を使用したりする方法もある。 `Tag` オブジェクトに対してこれらの操作を行うと、解析対象のHTML形式のデータ全体を対象にタグを検索する代わりに、操作対象の `Tag` オブジェクトに対応するタグの内側に存在する子要素を対象にタグを検索して取得できる。以下は、特定の `id` の値を持つ `<div>` の子要素を対象に、タグを取得する例である。

```
from bs4 import BeautifulSoup

html = """
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <div id="first">
            <p class="name">Alice</p>
            <p class="score">100</p>
        </div>
        <div id="second">
            <p class="name">Bob</p>
            <p class="score">90</p>
        </div>
    </body>
</html>
"""
soup = BeautifulSoup(html)
tag = soup.find(id="first")
print(tag.find(class_="name"))
```

```
<p class="name">Alice</p>
```

上記のプログラムでは、まず `soup.find(id="first")` により `id` の値が `first` であるタグを取得し、 `tag` に格納している。該当するタグと、その子要素は以下の部分である。

```
<div id="first">
```

```
    <p class="name">Alice</p>
    <p class="score">100</p>
</div>
```

次に、`print(tag.find(class_="name"))`により、`tag`の子要素の中で`class`が`name`であるタグを取得し、出力している。なお、`class=`ではなく`class_=`となっているのは、`class`はPythonにおける予約語のため使用できないことから、`find()`メソッドでは`class_`をキーワード引数の名前として使用することになっているためである。

元のHTMLドキュメント全体では、`class`が`name`のタグは`<p class="name">Alice</p>`と`<p class="name">Bob</p>`の2つ存在するが、そのうち`tag`の子要素である`<p class="name">Alice</p>`のみが出力されていることが確認できる。

HTMLからのデータ抽出

ここまでに扱ったBeautifulSoupの各種機能を使って、HTML形式の文字列データから、集計や分析に使用するためのデータを実際に抽出する例を解説する。

今回扱うHTMLドキュメントは、以下のような商品一覧である。

```
html_text = """
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <div>
            <h3 class="name">商品1</h3>
            <p class="price">1000円</p>
        </div>
        <div>
            <h3 class="name">商品2</h3>
            <p class="price">2000円</p>
        </div>
        <div>
            <h3 class="name">商品3</h3>
            <p class="price">3000円</p>
        </div>
    </body>
</html>
"""
```

上記のHTMLドキュメントをブラウザで表示すると、以下のようになる。

商品1

1000円

商品2

2000円

商品3

3000円

このHTMLドキュメントから、各商品の名前と価格を抽出し、商品ごとに名前と価格のタプルを作成することを考える。

まず、HTMLドキュメントの構造を観察すると、各 `<div>` が一つの商品に対応していることがわかる。また、 `<div>` の内部には、名前と価格に対応するタグとして、 `class` が `name` および `price` であるタグがそれぞれ存在している。したがって、まずは商品ごとに情報を抽出するために、以下のように `find_all()` メソッドで `<div>` タグをすべて取得する。

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(html_text)

# 商品一覧を取得
products = soup.find_all("div")

print(products)
```

```
[<div>
<h3 class="name">商品1</h3>
<p class="price">1000円</p>
</div>, <div>
<h3 class="name">商品2</h3>
<p class="price">2000円</p>
</div>, <div>
<h3 class="name">商品3</h3>
<p class="price">3000円</p>
</div>]
```

```
for product in products:
    print(type(product))
```

```
<class 'bs4.element.Tag'>
<class 'bs4.element.Tag'>
<class 'bs4.element.Tag'>
```

上記のプログラムでは、 `soup.find_all("div")` により、HTMLドキュメント全体から `<div>` タグを検索して取得している。出力結果から、HTMLドキュメント中のすべての `<div>` タグに対応する `Tag` クラスのオブジェクトを取得できていることが確認できる。

次に、各 `<div>` タグから名前と価格を抽出するため、 `<div>` の子要素のうち `class` が `name` と `price` であるタグを `find()` メソッドによりそれぞれ取得し、取得した各タグの内部の文字列を `get_text()` メソッドにより抽出する。抽出した名前と価格を保持しておくため、あらかじめリストを定義しておき、このリストに抽出された名前と価格のタプルを追加する。以下は、名前と価格を抽出してリストに格納し、最終的に抽出された各商品の名前と価格を出力するプログラムである。

```
extracted_products = []

# 各商品の情報を抽出
for product in products:
    name = product.find(class_="name").get_text()
    price = product.find(class_="price").get_text()
    extracted_products.append((name, price))

print(extracted_products)
```

```
[('商品1', '1000円'), ('商品2', '2000円'), ('商品3', '3000円')]
```

上記のプログラムでは、まず抽出したタグの情報を格納するためのリストとして `extracted_products` を定義する。次に、各商品に対応す

る `<div>` を一つずつ処理するため、 `for product in products:` により `products` に格納されている各 `Tag` クラスのオブジェクトを変数 `product` に格納して繰り返し処理する。 `name = product.find(class_="name").get_text()` により、 `product` に格納されている `Tag` クラスに対応するタグ（現在、繰り返し処理の対象としている `<div>` タグ）の子要素のうち、 `class` が `name` であるタグを取得し、そのタグの内部にある文字列を取得して `name` に格納する。これと同様の処理を `price = product.find(class_="price").get_text()` により `price` についても行う。これにより抽出された `name` （商品の名前）および `price` （商品の価格）をもとに、 `(name, price)` によりタプルを作成し、 `extracted_products.append()` により `extracted_products` に格納する。最後に、 `print(extracted_products)` により、抽出された各商品の名前と価格のタプルのリストを出力する。

上記のプログラムの出力結果から、1つの商品の名前と価格を1つのタプルとして、各商品の情報を含んだタプルのリストが作成できていることが確認できる。

問題1

問題文

施設案内に関する以下のようなWebページのHTML形式のデータが格納されている変数 `html` がある。出力例のように、このHTML形式のデータに含まれる `h1` タグおよびその中身を出力するプログラムを記述せよ。

```
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
             <p id="explanation">このサイトでは、X県の施設に以下の施設について説明します。</p>
             <ul>
                 <li class="list">図書館</li>
                 <li class="list">美術館</li>
                 <li class="list">体育館</li>
             </ul>
         </div>
     </body>

</html>
```

出力

```
<h1 id="top">施設案内</h1>
```

解答の雛形

```
from bs4 import BeautifulSoup

html = """
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
```

```
            <p id="explanation">このサイトでは、X県の施設に以下の施設について説明します。</p>
            <ul>
                <li class="list">図書館</li>
                <li class="list">美術館</li>
                <li class="list">体育館</li>
            </ul>
        </div>
    </body>

</html>
"""

## ここに解答を入力
```

問題2

問題文

施設案内に関する以下のようなWebページのHTML形式のデータが格納されている変数 `html` がある。出力例のように、このHTML形式のデータに含まれる上から三番目の `li` タグおよびその中身を出力するプログラムを記述せよ。

```
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
             <p id="explanation">このサイトでは、X県の施設に以下の施設について説明します。</p>
             <ul>
                 <li class="list">図書館</li>
                 <li class="list">美術館</li>
                 <li class="list">体育館</li>
             </ul>
         </div>
     </body>

</html>
```

出力

```
<li class="list">体育館</li>
```

解答の雛形

```
from bs4 import BeautifulSoup

html = """
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
```

```
            <p id="explanation">このサイトでは、X県の施設に以下の施設について説明します。</p>
            <ul>
                <li class="list">図書館</li>
                <li class="list">美術館</li>
                <li class="list">体育館</li>
            </ul>
        </div>
    </body>

</html>
"""

## ここに解答を入力
```

問題3

問題文

施設案内に関する以下のようなWebページのHTML形式のデータが格納されている変数 `html` がある。出力例のように、このHTML形式のデータに含まれる `p` タグの `id` 属性の値を出力するプログラムを記述せよ。

```
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
             <p id="explanation">このサイトでは、X県の施設に以下の施設について説明します。</p>
             <ul>
                 <li class="list">図書館</li>
                 <li class="list">美術館</li>
                 <li class="list">体育館</li>
             </ul>
         </div>
     </body>

</html>
```

出力

```
explanation
```

解答の雛形

```
from bs4 import BeautifulSoup

html = """
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
```

```
            <p id="explanation">このサイトでは、X県の施設に以下の施設について説明します。</p>
            <ul>
                <li class="list">図書館</li>
                <li class="list">美術館</li>
                <li class="list">体育館</li>
            </ul>
        </div>
    </body>

</html>
"""

## ここに解答を入力
```

問題4

問題文

施設案内に関する以下のようなWebページのHTML形式のデータが格納されている変数 `html` がある。出力例のように、このHTML形式のデータに含まれる、 `id` 属性の値が `explanation_place` のタグとその中身を出力するプログラムを記述せよ。

```
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
             <p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
             <ul id="list_place">
                 <li class="list_item">図書館</li>
                 <li class="list_item">美術館</li>
                 <li class="list_item">体育館</li>
             </ul>
             <p id="explanation_item">施設については以下の項目について説明します。</p>
             <ul id="list_info">
                 <li class="list_item">利用料</li>
                 <li class="list_item">休館日</li>
             </ul>
         </div>
     </body>

</html>
```

出力

```
<p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
```

解答の雛形

```
from bs4 import BeautifulSoup

html = """
<html>

    <head>
        <title>X県施設情報</title>
```

```
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
            <p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
            <ul id="list_place">
                <li class="list_item">図書館</li>
                <li class="list_item">美術館</li>
                <li class="list_item">体育館</li>
            </ul>
            <p id="explanation_item">施設については以下の項目について説明します。</p>
            <ul id="list_info">
                <li class="list_item">利用料</li>
                <li class="list_item">休館日</li>
            </ul>
        </div>
    </body>

</html>
"""

## ここに解答を入力
```

問題5

問題文

施設案内に関する以下のようなWebページのHTML形式のデータが格納されている変数 `html` がある。出力例のように、このHTML形式のデータに含まれる、 `title` タグの開始タグと終了タグの間にある文字列を出力するプログラムを記述せよ。

```
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
             <p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
             <ul id="list_place">
                 <li class="list_item">図書館</li>
                 <li class="list_item">美術館</li>
                 <li class="list_item">体育館</li>
             </ul>
             <p id="explanation_item">施設については以下の項目について説明します。</p>
             <ul id="list_info">
                 <li class="list_item">利用料</li>
                 <li class="list_item">休館日</li>
             </ul>
         </div>
     </body>

</html>
```

出力

```
X県施設情報
```

解答の雛形

```
from bs4 import BeautifulSoup

html = """
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
            <p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
            <ul id="list_place">
                <li class="list_item">図書館</li>
                <li class="list_item">美術館</li>
                <li class="list_item">体育館</li>
            </ul>
            <p id="explanation_item">施設については以下の項目について説明します。</p>
            <ul id="list_info">
                <li class="list_item">利用料</li>
                <li class="list_item">休館日</li>
            </ul>
        </div>
    </body>

</html>
"""

## ここに解答を入力
```

問題6

問題文

施設案内に関する以下のようなWebページのHTML形式のデータが格納されている変数 `html` がある。出力例のように、このHTML形式のデータに含まれる、開始タグと終了タグの間の文字列が `休館日` であるタグとその中身を出力するプログラムを記述せよ。

```
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
             <p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
             <ul id="list_place">
                 <li class="list_item">図書館</li>
                 <li class="list_item">美術館</li>
                 <li class="list_item">体育館</li>
             </ul>
             <p id="explanation_item">施設については以下の項目について説明します。</p>
             <ul id="list_info">
                 <li class="list_item">利用料</li>
                 <li class="list_item">休館日</li>
             </ul>
         </div>
     </body>

```

```
24 </html>
```

出力

```
1 <li class="list_item">休館日</li>
```

解答の雛形

```
from bs4 import BeautifulSoup

html = """
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
            <p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
            <ul id="list_place">
                <li class="list_item">図書館</li>
                <li class="list_item">美術館</li>
                <li class="list_item">体育館</li>
            </ul>
            <p id="explanation_item">施設については以下の項目について説明します。</p>
            <ul id="list_info">
                <li class="list_item">利用料</li>
                <li class="list_item">休館日</li>
            </ul>
        </div>
    </body>

</html>
"""

## ここに解答を入力
```

問題7

問題文

施設案内に関する以下のようなWebページのHTML形式のデータが格納されている変数 `html` がある。出力例のように、このHTML形式のデータに含まれる、各施設の名前と利用料と休館日のタプルのリストを出力するプログラムを記述せよ。

施設名・利用料・休館日をタプルにまとめる際には、施設名、利用料、休館日の順でまとめること。また、各施設の情報（施設名・利用料・休館日）をリストにまとめる際には、 `図書館` 、 `美術館` 、 `体育館` の順でまとめること。

```
1 <html>
2
3     <head>
4         <title>X県施設情報</title>
5     </head>
6
7     <body>
8         <div>
9             <h1 id="top">施設案内</h1>
```

```
            <p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
            <ul id="list_place">
                <li class="list_item">図書館</li>
                <li class="list_item">美術館</li>
                <li class="list_item">体育館</li>
            </ul>
            <p id="explanation_item">施設については以下の項目について説明します。</p>
            <ul id="list_info">
                <li class="list_item">利用料</li>
                <li class="list_item">休館日</li>
            </ul>
            <table border="1">
                <thead>
                    <tr>
                        <th>施設名</th>
                        <th>利用料</th>
                        <th>休館日</th>
                    </tr>
                </thead>
                <tbody>
                    <tr id="first">
                        <td class="place">図書館</td>
                        <td class="price">0円</td>
                        <td class="day">水曜日</td>
                    </tr>
                    <tr id="second">
                        <td class="place">美術館</td>
                        <td class="price">1000円</td>
                        <td class="day">月曜日</td>
                    </tr>
                    <tr id="third">
                        <td class="place">体育館</td>
                        <td class="price">500円</td>
                        <td class="day">水曜日</td>
                    </tr>
                </tbody>
            </table>
        </div>
    </body>

</html>
```

出力

```
[('図書館', '0円', '水曜日'), ('美術館', '1000円', '月曜日'), ('体育館', '500円', '水曜日')]
```

解答の雛形

```
from bs4 import BeautifulSoup

html = """
<html>

    <head>
        <title>X県施設情報</title>
    </head>

    <body>
        <div>
            <h1 id="top">施設案内</h1>
            <p id="explanation_place">このサイトでは、X県の施設に以下の施設について説明します。</p>
            <ul id="list_place">
                <li class="list_item">図書館</li>
                <li class="list_item">美術館</li>
                <li class="list_item">体育館</li>
            </ul>
```

```
            <p id="explanation_item">施設については以下の項目について説明します。</p>
            <ul id="list_info">
                <li class="list_item">利用料</li>
                <li class="list_item">休館日</li>
            </ul>
            <table border="1">
                <thead>
                    <tr>
                        <th>施設名</th>
                        <th>利用料</th>
                        <th>休館日</th>
                    </tr>
                </thead>
                <tbody>
                    <tr id="first">
                        <td class="place">図書館</td>
                        <td class="price">0円</td>
                        <td class="day">水曜日</td>
                    </tr>
                    <tr id="second">
                        <td class="place">美術館</td>
                        <td class="price">1000円</td>
                        <td class="day">月曜日</td>
                    </tr>
                    <tr id="third">
                        <td class="place">体育館</td>
                        <td class="price">500円</td>
                        <td class="day">水曜日</td>
                    </tr>
                </tbody>
            </table>
        </div>
    </body>

</html>
"""

## ここに解答を入力
```

5章: Webスクレイピングとデータ分析

本章の流れ

本章では、2章から4章で学んだpandasとWebスクレイピングを使い、Webページからデータを抽出して、さらにデータの加工・分析を行う。具体的には、4章で学んだWebスクレイピングを使用し、Webページから分析に必要なデータを抽出する。2章と3章で学んだpandasを使用し、抽出したデータからデータフレームを作成後、データフレーム内のデータの加工や統計量の算出を行い、データ分析を行う。

Webページのデータ抽出からデータ分析を一貫して行うことで、実践的なWebスクレイピングとpandasを使用したデータの加工・分析について学ぶ。

本章で行うWebスクレイピングでは、実際にWebサイトにアクセスし、HTMLドキュメントを取得することはせず、取得したHTMLドキュメントをパースするところから行う。

本章でのデータ分析の流れ

今回、以下の流れでデータ分析を行う。データ分析は目的を持って行うため、まず、データ分析の目的を理解するところから始める。

1. データ分析の目的の理解
2. WebスクレイピングするWebページについての理解
3. 目的に沿ったWebスクレイピングの実施
4. データの確認
5. データの加工
6. データ分析

データ分析の目的の理解

今回、飲料メーカーXのマーケティング担当者として、競合他社であるメーカーA、メーカーB、メーカーCについて、メーカーごと、さらにメーカー内でも商品の種別ごとに、特徴や傾向がないか調査することが目的である。

調査結果を元に、メーカーXでは、競合他社に対抗するための商品開発を行う予定である。

WebスクレイピングするWebページについての理解

調査を進める中で、以下のような各メーカーごとに情報がまとまっている飲料水に関するレビューサイトを見つけたため、このレビューサイトのデータを抽出・分析することにした。

メーカー名	種別	商品名	レビュー件数	評価	値段
A	お茶	緑茶	20	50	70
A	炭酸飲料	メロンソーダ	30	30	90
A	果汁入り飲料	レモンジュース	40	20	100
A	お茶	ウーロン茶	50	40	80
A	炭酸飲料	エナジーソーダ	60	1000	50
A	果汁入り飲料	ピーチジュース	70	30	70
A	お茶	麦茶	25	40	80
A	炭酸飲料	ストロングソーダ	0	0	100
A	果汁入り飲料	ミックスジュース	77	0	70
B	お茶	緑茶	80	60	150
B	炭酸飲料	ライチソーダ	90	60	180
B	果汁入り飲料	アップルジュース	100	70	700
B	お茶	麦茶	55	60	150
B	炭酸飲料	メロンソーダ	30	60	200
B	果汁入り飲料	オレンジジュース	86	90	900
B	お茶	ジャスミン茶	15	50	150
B	炭酸飲料	ブドウ風味ソーダ	0	0	200
B	果汁入り飲料	ミックスジュース	122	30	100
C	お茶	柚子茶	50	80	500
C	炭酸飲料	クラフトソーダ	20	90	800
C	果汁入り飲料	マンゴージュース	10	90	1000
C	お茶	アールグレイ	42	80	700
C	炭酸飲料	クラフトコーラ	20	90	800
C	果汁入り飲料	ストロベリージュース	20	100	1000
C	お茶	プーアル茶	32	80	700
C	炭酸飲料	プレミアムメロンソーダ	12	90	800
C	果汁入り飲料	梅ジュース	20	80	1000

Webスクレイピングを行う前に、どのようなWebページであるか理解するところから始める。

ページ全体を見ると、各メーカーの商品ごとにレビュー件数と評価、値段が表形式で記載されていることがわかる。

表の各列について詳しく見ていこう。

- `メーカ名` 列には、該当する商品を製造しているメーカーが記載されている。値は、 `A` 、 `B` 、 `C` の三種類である。
- `種別` 列には、該当する商品の種別が記載されている。値は、 `お茶` 、 `炭酸飲料` 、 `果汁入り飲料` の三種類である。
- `商品名` 列には、評価対象となっている商品名が記載されている。
- `レビュー件数` 列には、該当の商品に対しての寄せられたレビューの数が記載されている。レビューが一件もない場合には、 `0` と記載されている。
- `評価` 列には、レビューを元に点数付けした該当の商品に対しての味に関する評価が、 `0` から `100` までの数字で記載されている。値が大きいほど、味に対して良い評価がされていることを示す。Webページのバグによって、一部 `100` を越える値が記載されている。また、 `レビュー件数` が `0` の場合は、 `0` が記載されている。
- `値段` 列には、該当する商品の値段が記載されている。

このレビューサイトのHTMLドキュメントを得るため、HTTPリクエストを送信した結果、HTTPレスポンスから以下のようなHTMLドキュメントを取得できた（取得したHTMLドキュメントの全体については、次節のソースコード中に記載する）。以下のHTMLドキュメントから分析に必要なデータを抽出する。

```
<html>

<head>
    <title>飲料水レビューサイト</title>
</head>

<body>
    <div dir="ltr">
        <table cellspacing="0" cellpadding="0" border="1">
            <thead>
                <tr>
                    <th style="width:100px;" class="column-headers-background">メーカー名</th>
                    <th style="width:100px;" class="column-headers-background">種別</th>
                    <th style="width:198px;" class="column-headers-background">商品名</th>
                    <th style="width:100px;" class="column-headers-background">レビュー件数</th>
                    <th style="width:100px;" class="column-headers-background">評価</th>
                    <th style="width:100px;" class="column-headers-background">値段</th>
                </tr>
            </thead>
            <tbody>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">A</td>
                    <td class="kind" dir="ltr">お茶</td>
                    <td class="item" dir="ltr">緑茶</td>
                    <td class="review-number" dir="ltr">20</td>
                    <td class="review-point" dir="ltr">50</td>
                    <td class="price" dir="ltr">70</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">A</td>
                    <td class="kind" dir="ltr">炭酸飲料</td>
                    <td class="item" dir="ltr">メロンソーダ</td>
                    <td class="review-number" dir="ltr">30</td>
                    <td class="review-point" dir="ltr">30</td>
                    <td class="price" dir="ltr">90</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">A</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">レモンジュース</td>
                    <td class="review-number" dir="ltr">40</td>
                    <td class="review-point" dir="ltr">20</td>
                    <td class="price" dir="ltr">100</td>
                </tr>

                <!-- 中略 -->

                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">梅ジュース</td>
                    <td class="review-number" dir="ltr">20</td>
                    <td class="review-point" dir="ltr">80</td>
                    <td class="price" dir="ltr">1000</td>
                </tr>
          </tbody>
         </table>
     </div>
</body>

</html>
```

目的に沿ったWebスクレイピングの実施

各メーカーの商品の分析を行う上で、Webページに記載されていた表形式がそのまま分析で使用しやすい形であるため、前節のHTMLドキュメ

ントから、Webページの表形式をそのまま維持する以下のようなデータフレームの作成を目指す。

```
    メーカー名       種別           商品名  レビュー件数    評価    値段
0       A       お茶           緑茶      20    50    70
1       A     炭酸飲料        メロンソーダ      30    30    90
2       A   果汁入り飲料        レモンジュース      40    20   100
3       A       お茶         ウーロン茶      50    40    80
4       A     炭酸飲料        エナジーソーダ      60  1000    50
5       A   果汁入り飲料        ピーチジュース      70    30    70
6       A       お茶           麦茶      25    40    80
7       A     炭酸飲料       ストロングソーダ       0     0   100
8       A   果汁入り飲料        ミックスジュース      77     0    70
9       B       お茶           緑茶      80    60   150
10      B     炭酸飲料        ライチソーダ      90    60   180
11      B   果汁入り飲料        アップルジュース     100    70   700
12      B       お茶           麦茶      55    60   150
13      B     炭酸飲料        メロンソーダ      30    60   200
14      B   果汁入り飲料        オレンジジュース      86    90   900
15      B       お茶         ジャスミン茶      15    50   150
16      B     炭酸飲料       ブドウ風味ソーダ       0     0   200
17      B   果汁入り飲料        ミックスジュース     122    30   100
18      C       お茶           柚子茶      50    80   500
19      C     炭酸飲料        クラフトソーダ      20    90   800
20      C   果汁入り飲料        マンゴージュース      10    90  1000
21      C       お茶         アールグレイ      42    80   700
22      C     炭酸飲料        クラフトコーラ      20    90   800
23      C   果汁入り飲料     ストロベリージュース      20   100  1000
24      C       お茶          プーアル茶      32    80   700
25      C     炭酸飲料    プレミアムメロンソーダ      12    90   800
26      C   果汁入り飲料          梅ジュース      20    80  1000
```

どのように上記のようなデータフレームを作成するか、プログラムを記述していく前に、データフレームを作成するために必要な情報がHTMLでどのように構造化されているか理解する必要がある。

まず、データフレームの列名（ `メーカー名` や `種別` など）がHTMLドキュメント内でどのように構造化されているか確認する。

HTMLドキュメントを観察すると、各 `<th>` が列名を持っていることがわかる。そのため、各 `<th>` の要素を取得することで、列名を取得することができると考えられる。

続いて、各行の情報（ `A` や `お茶` など）がHTMLドキュメント内でどのように構造化されているか確認する。

HTMLドキュメントを観察すると、 `<tbody>` の各 `<tr>` に行の情報を持っていることがわかる。 `<tr>` の要素の中では、各 `<td>` に、 `メーカー名` や `種別` などの値を持っている。 `メーカー名` の値はclass属性が `maker` の `<td>` が持ち、 `種別` の値はclass属性が `kind` の `<td>` が持ち、 `商品名` の値はclass属性が `item` の `<td>` が持ち、 `レビュー件数` の値はclass属性が `review-number` の `<td>` が持ち、 `評価` の値はclass属性が `review-point` の `<td>` が持ち、 `値段` の値はclass属性が `price` の `<td>` が持っている。そのため、 `<tbody>` から各 `<tr>` の要素を取得し、さらに各 `<tr>` に対して、それぞれの列に紐づくclass属性を指定して `<td>` の要素を取得することで、行の各値を取得できると考えられる。

以下は、HTMLドキュメントの構造を観察した結果から、データフレームの作成に必要な要素をHTMLドキュメントから取得し、データフレームを作成するプログラムである。

```python
import pandas as pd
from bs4 import BeautifulSoup

html_text = """
<html>

<head>
    <title>飲料水レビューサイト</title>
</head>

<body>
    <div dir="ltr">
        <table cellspacing="0" cellpadding="0" border="1">
```

```
14              <thead>
15                  <tr>
16                      <th style="width:100px;" class="column-headers-background">メーカー名</th>
17                      <th style="width:100px;" class="column-headers-background">種別</th>
18                      <th style="width:198px;" class="column-headers-background">商品名</th>
19                      <th style="width:100px;" class="column-headers-background">レビュー件数</th>
20                      <th style="width:100px;" class="column-headers-background">評価</th>
21                      <th style="width:100px;" class="column-headers-background">値段</th>
22                  </tr>
23              </thead>
24              <tbody>
25                  <tr style="height: 20px">
26                      <td class="maker" dir="ltr">A</td>
27                      <td class="kind" dir="ltr">お茶</td>
28                      <td class="item" dir="ltr">緑茶</td>
29                      <td class="review-number" dir="ltr">20</td>
30                      <td class="review-point" dir="ltr">50</td>
31                      <td class="price" dir="ltr">70</td>
32                  </tr>
33                  <tr style="height: 20px">
34                      <td class="maker" dir="ltr">A</td>
35                      <td class="kind" dir="ltr">炭酸飲料</td>
36                      <td class="item" dir="ltr">メロンソーダ</td>
37                      <td class="review-number" dir="ltr">30</td>
38                      <td class="review-point" dir="ltr">30</td>
39                      <td class="price" dir="ltr">90</td>
40                  </tr>
41                  <tr style="height: 20px">
42                      <td class="maker" dir="ltr">A</td>
43                      <td class="kind" dir="ltr">果汁入り飲料</td>
44                      <td class="item" dir="ltr">レモンジュース</td>
45                      <td class="review-number" dir="ltr">40</td>
46                      <td class="review-point" dir="ltr">20</td>
47                      <td class="price" dir="ltr">100</td>
48                  </tr>
49                  <tr style="height: 20px">
50                      <td class="maker" dir="ltr">A</td>
51                      <td class="kind" dir="ltr">お茶</td>
52                      <td class="item" dir="ltr">ウーロン茶</td>
53                      <td class="review-number" dir="ltr">50</td>
54                      <td class="review-point" dir="ltr">40</td>
55                      <td class="price" dir="ltr">80</td>
56                  </tr>
57                  <tr style="height: 20px">
58                      <td class="maker" dir="ltr">A</td>
59                      <td class="kind" dir="ltr">炭酸飲料</td>
60                      <td class="item" dir="ltr">エナジーソーダ</td>
61                      <td class="review-number" dir="ltr">60</td>
62                      <td class="review-point" dir="ltr">1000</td>
63                      <td class="price" dir="ltr">50</td>
64                  </tr>
65                  <tr style="height: 20px">
66                      <td class="maker" dir="ltr">A</td>
67                      <td class="kind" dir="ltr">果汁入り飲料</td>
68                      <td class="item" dir="ltr">ピーチジュース</td>
69                      <td class="review-number" dir="ltr">70</td>
70                      <td class="review-point" dir="ltr">30</td>
71                      <td class="price" dir="ltr">70</td>
72                  </tr>
73                  <tr style="height: 20px">
74                      <td class="maker" dir="ltr">A</td>
75                      <td class="kind" dir="ltr">お茶</td>
76                      <td class="item" dir="ltr">麦茶</td>
77                      <td class="review-number" dir="ltr">25</td>
78                      <td class="review-point" dir="ltr">40</td>
79                      <td class="price" dir="ltr">80</td>
80                  </tr>
81                  <tr style="height: 20px">
82                      <td class="maker" dir="ltr">A</td>
83                      <td class="kind" dir="ltr">炭酸飲料</td>
84                      <td class="item" dir="ltr">ストロングソーダ</td>
85                      <td class="review-number" dir="ltr">0</td>
```

```
                    <td class="review-point" dir="ltr">0</td>
                    <td class="price" dir="ltr">100</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">A</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">ミックスジュース</td>
                    <td class="review-number" dir="ltr">77</td>
                    <td class="review-point" dir="ltr">0</td>
                    <td class="price" dir="ltr">70</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">お茶</td>
                    <td class="item" dir="ltr">緑茶</td>
                    <td class="review-number" dir="ltr">80</td>
                    <td class="review-point" dir="ltr">60</td>
                    <td class="price" dir="ltr">150</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">炭酸飲料</td>
                    <td class="item" dir="ltr">ライチソーダ</td>
                    <td class="review-number" dir="ltr">90</td>
                    <td class="review-point" dir="ltr">60</td>
                    <td class="price" dir="ltr">180</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">アップルジュース</td>
                    <td class="review-number" dir="ltr">100</td>
                    <td class="review-point" dir="ltr">70</td>
                    <td class="price" dir="ltr">700</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">お茶</td>
                    <td class="item" dir="ltr">麦茶</td>
                    <td class="review-number" dir="ltr">55</td>
                    <td class="review-point" dir="ltr">60</td>
                    <td class="price" dir="ltr">150</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">炭酸飲料</td>
                    <td class="item" dir="ltr">メロンソーダ</td>
                    <td class="review-number" dir="ltr">30</td>
                    <td class="review-point" dir="ltr">60</td>
                    <td class="price" dir="ltr">200</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">オレンジジュース</td>
                    <td class="review-number" dir="ltr">86</td>
                    <td class="review-point" dir="ltr">90</td>
                    <td class="price" dir="ltr">900</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">お茶</td>
                    <td class="item" dir="ltr">ジャスミン茶</td>
                    <td class="review-number" dir="ltr">15</td>
                    <td class="review-point" dir="ltr">50</td>
                    <td class="price" dir="ltr">150</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">炭酸飲料</td>
                    <td class="item" dir="ltr">ブドウ風味ソーダ</td>
                    <td class="review-number" dir="ltr">0</td>
```

```
                    <td class="review-point" dir="ltr">0</td>
                    <td class="price" dir="ltr">200</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">B</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">ミックスジュース</td>
                    <td class="review-number" dir="ltr">122</td>
                    <td class="review-point" dir="ltr">30</td>
                    <td class="price" dir="ltr">100</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">お茶</td>
                    <td class="item" dir="ltr">柚子茶</td>
                    <td class="review-number" dir="ltr">50</td>
                    <td class="review-point" dir="ltr">80</td>
                    <td class="price" dir="ltr">500</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">炭酸飲料</td>
                    <td class="item" dir="ltr">クラフトソーダ</td>
                    <td class="review-number" dir="ltr">20</td>
                    <td class="review-point" dir="ltr">90</td>
                    <td class="price" dir="ltr">800</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">マンゴージュース</td>
                    <td class="review-number" dir="ltr">10</td>
                    <td class="review-point" dir="ltr">90</td>
                    <td class="price" dir="ltr">1000</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">お茶</td>
                    <td class="item" dir="ltr">アールグレイ</td>
                    <td class="review-number" dir="ltr">42</td>
                    <td class="review-point" dir="ltr">80</td>
                    <td class="price" dir="ltr">700</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">炭酸飲料</td>
                    <td class="item" dir="ltr">クラフトコーラ</td>
                    <td class="review-number" dir="ltr">20</td>
                    <td class="review-point" dir="ltr">90</td>
                    <td class="price" dir="ltr">800</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">ストロベリージュース</td>
                    <td class="review-number" dir="ltr">20</td>
                    <td class="review-point" dir="ltr">100</td>
                    <td class="price" dir="ltr">1000</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">お茶</td>
                    <td class="item" dir="ltr">プーアル茶</td>
                    <td class="review-number" dir="ltr">32</td>
                    <td class="review-point" dir="ltr">80</td>
                    <td class="price" dir="ltr">700</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">炭酸飲料</td>
                    <td class="item" dir="ltr">プレミアムメロンソーダ</td>
                    <td class="review-number" dir="ltr">12</td>
```

```
                    <td class="review-point" dir="ltr">90</td>
                    <td class="price" dir="ltr">800</td>
                </tr>
                <tr style="height: 20px">
                    <td class="maker" dir="ltr">C</td>
                    <td class="kind" dir="ltr">果汁入り飲料</td>
                    <td class="item" dir="ltr">梅ジュース</td>
                    <td class="review-number" dir="ltr">20</td>
                    <td class="review-point" dir="ltr">80</td>
                    <td class="price" dir="ltr">1000</td>
                </tr>
            </tbody>
        </table>
    </div>
</body>

</html>
"""

# BeautifulSoupオブジェクトを作成
soup = BeautifulSoup(html_text)

# 列名を取得
ths = soup.find_all('th')

# 列名を格納するためのリスト
columns = [th.get_text() for th in ths]

# 表の中身を取得
trs = soup.tbody.find_all('tr')

# データフレームに格納するためのリスト
data = []

# trsからデータを取り出す
for tr in trs:
    maker = tr.find(class_='maker').get_text()
    kind = tr.find(class_='kind').get_text()
    item = tr.find(class_='item').get_text()
    review_number = int(tr.find(class_='review-number').get_text())
    review_point = int(tr.find(class_='review-point').get_text())
    price = int(tr.find(class_='price').get_text())

    # 取り出したデータをリストに追加
    data.append([maker, kind, item, review_number, review_point, price])

# データフレームに変換
df = pd.DataFrame(data, columns=columns)

print(df)
```

```
   メーカー名       種別           商品名  レビュー件数    評価    値段
0      A      お茶          緑茶      20    50    70
1      A    炭酸飲料       メロンソーダ      30    30    90
2      A  果汁入り飲料       レモンジュース      40    20   100
3      A      お茶         ウーロン茶      50    40    80
4      A    炭酸飲料       エナジーソーダ      60  1000    50
5      A  果汁入り飲料       ピーチジュース      70    30    70
6      A      お茶          麦茶      25    40    80
7      A    炭酸飲料      ストロングソーダ       0     0   100
8      A  果汁入り飲料      ミックスジュース      77     0    70
9      B      お茶          緑茶      80    60   150
10     B    炭酸飲料       ライチソーダ      90    60   180
11     B  果汁入り飲料      アップルジュース     100    70   700
12     B      お茶          麦茶      55    60   150
13     B    炭酸飲料       メロンソーダ      30    60   200
14     B  果汁入り飲料      オレンジジュース      86    90   900
15     B      お茶        ジャスミン茶      15    50   150
16     B    炭酸飲料      ブドウ風味ソーダ       0     0   200
17     B  果汁入り飲料      ミックスジュース     122    30   100
```

```
20  18     C       お茶          柚子茶       50    80   500
21  19     C     炭酸飲料       クラフトソーダ       20    90   800
22  20     C  果汁入り飲料       マンゴージュース       10    90  1000
23  21     C       お茶         アールグレイ       42    80   700
24  22     C     炭酸飲料       クラフトコーラ       20    90   800
25  23     C  果汁入り飲料    ストロベリージュース       20   100  1000
26  24     C       お茶          プーアル茶       32    80   700
27  25     C     炭酸飲料   プレミアムメロンソーダ       12    90   800
28  26     C  果汁入り飲料          梅ジュース       20    80  1000
```

上記のプログラムについて詳しく解説する。

初めに、HTMLドキュメントから情報を取得するためにBeautifulSoupが必要になり、データフレームを作成するためにpandasが必要になるため、それぞれインポートしている。

```
1  import pandas as pd
2  from bs4 import BeautifulSoup
```

続く処理で、変数 `html_text` にHTMLドキュメントを格納している。

以下の処理で、 `html_text` に格納している文字列から、 `BeautifulSoup` クラスのオブジェクトとして、 `soup` を生成している。

```
1  soup = BeautifulSoup(html_text)
```

ここまでの処理で、HTMLドキュメントから必要な情報を取得する準備が整ったので、続く処理からは、実際にHTMLドキュメントから必要な情報を取得していく処理を行う。

まずは、列名の取得を行う。

列名は、各 `<th>` に含まれているため、以下の処理でHTMLドキュメントに含まれる `<th>` を全て取得する。

```
1  ths = soup.find_all('th')
```

続く以下の処理で、取得した `<th>` の開始タグと終了タグの間の文字列を取得し、列名のリストを作成し、 `columns` に格納する。実行後の `columns` の中には `['メーカー名', '種別', '商品名', 'レビュー件数', '評価', '値段']` というリストが格納されている。

```
1  columns = [th.get_text() for th in ths]
```

列名の取得を行う処理は以上である。

続いて、行の情報の取得を行う。

行の情報は、 `<tbody>` の各 `<tr>` が持っているため、以下の処理で `<tbody>` の `<tr>` を全て取得する。

```
1  trs = soup.tbody.find_all('tr')
```

続いて、取り出した行の情報を格納する `data` を定義する。 `data` には、空のリストが初期値として与えられている。

```
1  data = []
```

データフレームを作成するために、 `data` に行の情報を二次元リストの形で格納する。以下の処理で、 `for` 文を使い `trs` から `<tr>` の要

素を一つずつ取り出し、取り出した `<tr>` に対して取り出す `<td>` のclass属性を指定することで、各列の値を取り出している。各列の値を取り出し終えたら、取り出した値をリストにして、 `data` に追加している。

```
for tr in trs:
    maker = tr.find(class_='maker').get_text()
    kind = tr.find(class_='kind').get_text()
    item = tr.find(class_='item').get_text()
    review_number = int(tr.find(class_='review-number').get_text())
    review_point = int(tr.find(class_='review-point').get_text())
    price = int(tr.find(class_='price').get_text())

    data.append([maker, kind, item, review_number, review_point, price])
```

上記の処理を実行した後の `data` には以下の二次元リストが格納されている。

```
[['A', 'お茶', '緑茶', 20, 50, 70], ['A', '炭酸飲料', 'メロンソーダ', 30, 30, 90], ['A', '果汁入り飲料', 'レモンジュース', 40,
```

最後に以下の処理で、取得した列名が格納されたリスト `columns` と、行の情報が格納された二次元リスト `data` を使い、データフレーム `df` を作成する。 `DataFrame()` メソッドでは、二次元リストからもデータフレームを作成できる。また、列名については、 `columns=` で列名を持つリストなどを指定することで、データフレームの列名を指定できる。

```
df = pd.DataFrame(data, columns=columns)
```

最終的に作成したデータフレームが以下である。

```
   メーカー名       種別          商品名  レビュー件数    評価    値段
0      A      お茶          緑茶      20    50    70
1      A    炭酸飲料       メロンソーダ      30    30    90
2      A  果汁入り飲料       レモンジュース      40    20   100
3      A      お茶        ウーロン茶      50    40    80
4      A    炭酸飲料       エナジーソーダ      60  1000    50
5      A  果汁入り飲料       ピーチジュース      70    30    70
6      A      お茶          麦茶      25    40    80
7      A    炭酸飲料      ストロングソーダ       0     0   100
8      A  果汁入り飲料      ミックスジュース      77     0    70
9      B      お茶          緑茶      80    60   150
10     B    炭酸飲料       ライチソーダ      90    60   180
11     B  果汁入り飲料      アップルジュース     100    70   700
12     B      お茶          麦茶      55    60   150
13     B    炭酸飲料       メロンソーダ      30    60   200
14     B  果汁入り飲料      オレンジジュース      86    90   900
15     B      お茶        ジャスミン茶      15    50   150
16     B    炭酸飲料      ブドウ風味ソーダ       0     0   200
17     B  果汁入り飲料      ミックスジュース     122    30   100
18     C      お茶          柚子茶      50    80   500
19     C    炭酸飲料       クラフトソーダ      20    90   800
20     C  果汁入り飲料      マンゴージュース      10    90  1000
21     C      お茶        アールグレイ      42    80   700
22     C    炭酸飲料       クラフトコーラ      20    90   800
23     C  果汁入り飲料    ストロベリージュース      20   100  1000
24     C      お茶         プーアル茶      32    80   700
25     C    炭酸飲料   プレミアムメロンソーダ      12    90   800
26     C  果汁入り飲料         梅ジュース      20    80  1000
```

データの確認

前節で、HTMLドキュメントから、必要な情報を抽出し、データフレームを作成した。

データ分析を行う前に、欠損値や外れ値、異常値がデータ内に存在しないか確認する。もし、欠損値や異常値がデータ内に存在する場合は、目

的を達成するためにはどのように対応するのがよいか検討し、必要に応じてデータを加工する。

現在のデータフレームの並びでは、メーカー内での商品の並びが種別ごとにまとまっていないため、実際のデータを確認しづらい。そのため、欠損値や異常値がデータ内に存在するか確認する前に、メーカーごと、かつ同じメーカー内は種別ごとに並ぶようにソートを行う。

以下のように、`sort_values()` メソッドに複数列名を指定することで、複数の列に対してソートを行うことができる。

```
sort_df = df.sort_values(by=["メーカー名","種別"])
print(sort_df)
```

```
   メーカー名       種別           商品名  レビュー件数    評価    値段
0      A      お茶          緑茶      20    50    70
3      A      お茶        ウーロン茶      50    40    80
6      A      お茶          麦茶      25    40    80
2      A  果汁入り飲料       レモンジュース      40    20   100
5      A  果汁入り飲料       ピーチジュース      70    30    70
8      A  果汁入り飲料      ミックスジュース      77     0    70
1      A    炭酸飲料        メロンソーダ      30    30    90
4      A    炭酸飲料       エナジーソーダ      60  1000    50
7      A    炭酸飲料      ストロングソーダ       0     0   100
9      B      お茶          緑茶      80    60   150
12     B      お茶          麦茶      55    60   150
15     B      お茶        ジャスミン茶      15    50   150
11     B  果汁入り飲料       アップルジュース     100    70   700
14     B  果汁入り飲料       オレンジジュース      86    90   900
17     B  果汁入り飲料      ミックスジュース     122    30   100
10     B    炭酸飲料        ライチソーダ      90    60   180
13     B    炭酸飲料        メロンソーダ      30    60   200
16     B    炭酸飲料      ブドウ風味ソーダ       0     0   200
18     C      お茶          柚子茶      50    80   500
21     C      お茶        アールグレイ      42    80   700
24     C      お茶         プーアル茶      32    80   700
20     C  果汁入り飲料       マンゴージュース      10    90  1000
23     C  果汁入り飲料    ストロベリージュース      20   100  1000
26     C  果汁入り飲料         梅ジュース      20    80  1000
19     C    炭酸飲料        クラフトソーダ      20    90   800
22     C    炭酸飲料        クラフトコーラ      20    90   800
25     C    炭酸飲料   プレミアムメロンソーダ      12    90   800
```

上記のプログラムでは、まず `メーカー名` 列でソートを行い、続いて、同一の `メーカー名` 列の値の中で `種別` 列でソートを行っている。`メーカー名` 列と `種別` 列でソートすることによって、メーカーごとかつ種別ごとに商品が並び、データが見やすくなった。

ソートを終えたところで、欠損値がデータ内に存在しないか確認する。`isnull()` メソッドを使い、欠損値があるか確認する。

```
print(sort_df.isnull())
```

```
    メーカー名    種別   商品名  レビュー件数    評価    値段
0   False  False  False   False  False  False
3   False  False  False   False  False  False
6   False  False  False   False  False  False
2   False  False  False   False  False  False
5   False  False  False   False  False  False
8   False  False  False   False  False  False
1   False  False  False   False  False  False
4   False  False  False   False  False  False
7   False  False  False   False  False  False
9   False  False  False   False  False  False
12  False  False  False   False  False  False
15  False  False  False   False  False  False
11  False  False  False   False  False  False
14  False  False  False   False  False  False
17  False  False  False   False  False  False
```

```
17  10  False  False  False   False  False  False
18  13  False  False  False   False  False  False
19  16  False  False  False   False  False  False
20  18  False  False  False   False  False  False
21  21  False  False  False   False  False  False
22  24  False  False  False   False  False  False
23  20  False  False  False   False  False  False
24  23  False  False  False   False  False  False
25  26  False  False  False   False  False  False
26  19  False  False  False   False  False  False
27  22  False  False  False   False  False  False
28  25  False  False  False   False  False  False
```

上記の実行結果から、今回扱うデータには欠損値がないことがわかった。

欠損値の確認を終えた後、外れ値や異常値がデータ内に存在しないか確認する。`describe()`メソッドを使い、`レビュー件数`列、`評価`列、`値段`列の統計量を算出し、外れ値や異常値が存在しないか確認する。

```
1  print(sort_df[["レビュー件数","評価","値段"]].describe())
```

```
1            レビュー件数          評価           値段
2  count   27.000000    27.000000    27.000000
3  mean    43.555556    91.481481   397.777778
4  std     32.400538   184.050656   362.993254
5  min      0.000000     0.000000    50.000000
6  25%     20.000000    35.000000    95.000000
7  50%     32.000000    60.000000   180.000000
8  75%     65.000000    85.000000   750.000000
9  max    122.000000  1000.000000  1000.000000
```

上記のプログラムの実行結果が、`レビュー件数`列、`評価`列、`値段`列の統計量を算出した結果である。

`レビュー件数`列は、平均値が約44件で、最小値0件、最大値122件となっているため、レビューの件数としては現実的な値に収まっており、外れ値や異常値は存在しないように見える。

`評価`列は、上限値が100である列であるにもかかわらず、平均値が約91と非常に高い。最大値が1000となっており、上限の100を超えている。そのため、異常値が存在することがわかる。また、最小値も0となっており、評価の数値の範囲内ではあるが、評価の値としては全ての人が0と評価しているのは現実的には考えづらく、別途調査が必要になる。

`値段`列は、平均値が約398円で、最小値50円、最大値1000円となっているため、飲料の値段としては現実的な値に収まっており、外れ値や異常値は存在しないように見える。

以上から、`評価`列に異常値が存在することがわかる。具体的にどの列が異常値となっているか、`評価`列が100を超える行でフィルタリングして確認する。

```
1  print(sort_df[sort_df["評価"] > 100])
```

```
1    メーカー名    種別       商品名  レビュー件数    評価  値段
2  4     A  炭酸飲料  エナジーソーダ       60  1000  50
```

フィルタリングの結果から、メーカーAのエナジーソーダという商品の評価が異常であることがわかった。

スクレイピング元のレビューサイトを詳しく調べると、レビューサイトの不具合でメーカーAのエナジーソーダの評価が100倍されていたことがわかったため、本来の評価は10となる。分析前のデータ加工で、メーカーAのエナジーソーダの評価について修正を行う。

続いて、評価が0など評価が低い商品にどのようなものがあるか、`評価`列が10未満の行でフィルタリングして確認する。

```
print(sort_df[sort_df["評価"] < 10])
```

```
    メーカー名      種別         商品名  レビュー件数  評価   値段
8       A  果汁入り飲料  ミックスジュース       77   0   70
7       A    炭酸飲料  ストロングソーダ        0   0  100
16      B    炭酸飲料  ブドウ風味ソーダ        0   0  200
```

フィルタリングの結果から、メーカーAのミックスジュースとストロングソーダ、メーカーBのブドウ風味ソーダが評価が0であることがわかった。

スクレイピング元のレビューサイトを詳しく調べると、メーカーAのミックスジュースは本当に評価が0になっていることがわかった。また、メーカーAのストロングソーダとメーカーBのブドウ風味ソーダについては、レビュー件数が0になっており、評価の元となる情報が一切ないため評価が0となっていた。

メーカーAのミックスジュースについては、レビュー件数が77件あるなかで本当に評価が0であったため、そのままデータ分析に用いることにした。

メーカーAのストロングソーダとメーカーBのブドウ風味ソーダのようにレビュー件数が0の商品については、今回、評価や値段を元にメーカーごとの商品の特徴や傾向を調べようとしているため、そもそもレビュー件数が0件で評価が行われていない商品を分析に含めてしまうと目的を達成するための正確な分析結果を得ることができないと判断した。そのため、メーカーAのストロングソーダとメーカーBのブドウ風味ソーダのようにレビュー件数が0の商品については、分析前のデータ加工で、分析データから外す処理を行う。

`レビュー件数` 列の値が0の商品が他にもないか、フィルタリングを行い確認する。

```
print(sort_df[sort_df["レビュー件数"] == 0])
```

```
    メーカー名    種別         商品名  レビュー件数  評価   値段
7       A  炭酸飲料  ストロングソーダ        0   0  100
16      B  炭酸飲料  ブドウ風味ソーダ        0   0  200
```

```

```

フィルタリングの結果から、 `レビュー件数` が0の商品は、メーカーAのストロングソーダとメーカーBのブドウ風味ソーダのみということがわかった。

他にも外れ値や異常値がないか、平均値と標準偏差を用いて調査を行った。平均値をμ、標準偏差をσで表し、$\mu \pm 2\sigma$の範囲外の値を外れ値とした場合、以下のように、メーカーBのミックスジュースの `レビュー件数` に外れ値を発見した。

```
# 平均値を算出
mean = sort_df["レビュー件数"].mean()
# 標準偏差を算出
std = sort_df["レビュー件数"].std()
# μ - 2σ を計算
lower_bound = mean - 2 * std
# μ + 2σ を計算
upper_bound = mean + 2 * std
# 外れ値を含む行をフィルタリング
outlier = sort_df[(sort_df["レビュー件数"] < lower_bound) | (sort_df["レビュー件数"] > upper_bound)]
print(outlier)
```

```
    メーカー名       種別        商品名  レビュー件数  評価   値段
17      B  果汁入り飲料  ミックスジュース      122  30  100
```

スクレイピング元のレビューサイトを詳しく調べると、メーカーBのミックスジュースは本当にレビュー件数が122になっていることがわかった。そのため、メーカーBのミックスジュースについては、そのままデータ分析に用いることにした。

データの加工

前節で行ったデータの確認で、異常値や分析から外すべきデータを見つけることができた。本節では、前節で見つけた異常値の修正と、分析から外すべきデータの削除を行い、データ分析ができるようにデータを加工する。

前節から、分析の目的を達成するようなデータを作成するために、現在のデータに対して、以下の修正が必要である。

- メーカーAのエナジーソーダという商品の評価を10に変更する。
- `レビュー件数` 列が0の行の削除する。

まずは、メーカーAのエナジーソーダという商品の評価を10に変更する。分析用のデータを作成していくため、 `analysis_df` にデータフレーム `sort_df` をコピーして、 `analysis_df` に対してデータの加工を行う。

```python
# sort_dfをコピーしてanalysis_dfを定義
analysis_df = sort_df.copy()

# 行番号4の評価列の値を変更
analysis_df.loc[4, "評価"] = 10

print(analysis_df)
```

```
    メーカー名       種別          商品名  レビュー件数   評価    値段
0       A       お茶          緑茶      20   50    70
3       A       お茶       ウーロン茶      50   40    80
6       A       お茶          麦茶      25   40    80
2       A  果汁入り飲料      レモンジュース      40   20   100
5       A  果汁入り飲料      ピーチジュース      70   30    70
8       A  果汁入り飲料     ミックスジュース      77    0    70
1       A    炭酸飲料      メロンソーダ      30   30    90
4       A    炭酸飲料     エナジーソーダ      60   10    50
7       A    炭酸飲料     ストロングソーダ       0    0   100
9       B       お茶          緑茶      80   60   150
12      B       お茶          麦茶      55   60   150
15      B       お茶       ジャスミン茶      15   50   150
11      B  果汁入り飲料      アップルジュース     100   70   700
14      B  果汁入り飲料      オレンジジュース      86   90   900
17      B  果汁入り飲料      ミックスジュース     122   30   100
10      B    炭酸飲料      ライチソーダ      90   60   180
13      B    炭酸飲料      メロンソーダ      30   60   200
16      B    炭酸飲料     ブドウ風味ソーダ       0    0   200
18      C       お茶          柚子茶      50   80   500
21      C       お茶       アールグレイ      42   80   700
24      C       お茶        プーアル茶      32   80   700
20      C  果汁入り飲料      マンゴージュース      10   90  1000
23      C  果汁入り飲料    ストロベリージュース      20  100  1000
26      C  果汁入り飲料         梅ジュース      20   80  1000
19      C    炭酸飲料      クラフトソーダ      20   90   800
22      C    炭酸飲料      クラフトコーラ      20   90   800
25      C    炭酸飲料   プレミアムメロンソーダ      12   90   800
```

メーカーAのエナジーソーダはデータフレーム内では、行番号が4の行に存在するため、行番号4と、 `評価` 列を指定して値の変更を行った。値の変更後の `analysis_df` の出力結果から、メーカーAのエナジーソーダが正しく修正されていることがわかる。

続いて、分析用のデータフレームから、 `レビュー件数` が0の行を削除する。 `レビュー件数` が0の行を削除するには、 `レビュー件数` が0より

大きい行でフィルタリングすればよい。

```
analysis_df = analysis_df[analysis_df["レビュー件数"] > 0]

print(analysis_df)
```

```
    メーカー名       種別           商品名  レビュー件数   評価    値段
0       A       お茶          緑茶       20   50    70
3       A       お茶       ウーロン茶       50   40    80
6       A       お茶          麦茶       25   40    80
2       A   果汁入り飲料       レモンジュース       40   20   100
5       A   果汁入り飲料       ピーチジュース       70   30    70
8       A   果汁入り飲料      ミックスジュース       77    0    70
1       A     炭酸飲料       メロンソーダ       30   30    90
4       A     炭酸飲料      エナジーソーダ       60   10    50
9       B       お茶          緑茶       80   60   150
12      B       お茶          麦茶       55   60   150
15      B       お茶       ジャスミン茶       15   50   150
11      B   果汁入り飲料      アップルジュース      100   70   700
14      B   果汁入り飲料      オレンジジュース       86   90   900
17      B   果汁入り飲料      ミックスジュース      122   30   100
10      B     炭酸飲料       ライチソーダ       90   60   180
13      B     炭酸飲料       メロンソーダ       30   60   200
18      C       お茶          柚子茶       50   80   500
21      C       お茶       アールグレイ       42   80   700
24      C       お茶        プーアル茶       32   80   700
20      C   果汁入り飲料      マンゴージュース       10   90  1000
23      C   果汁入り飲料    ストロベリージュース       20  100  1000
26      C   果汁入り飲料         梅ジュース       20   80  1000
19      C     炭酸飲料       クラフトソーダ       20   90   800
22      C     炭酸飲料       クラフトコーラ       20   90   800
25      C     炭酸飲料   プレミアムメロンソーダ       12   90   800
```

フィルタリング後の `analysis_df` の出力結果から、 `レビュー件数` が0の行が `analysis_df` から削除されたことがわかる。

データ分析

前節までで、データ分析に必要なデータを用意できた。本節では、前節で加工したデータを使って目的に沿って分析を行っていく。

今回のデータ分析の目的は、競合他社であるメーカーA、メーカーB、メーカーCについて、メーカーごと、さらにメーカー内でも商品の種別ごとに、特徴や傾向がないか調査することであった。

飲料の特徴や傾向を掴むには、今回使用するデータの中では、飲料のおいしさと値段について分析するのが適していると考えられるため、 `評価` 列と `値段` 列に対して以下の集計を行っていく。

- 各メーカーごとに、商品がどのくらいおいしいか傾向を掴むため、メーカーごとに `評価` 列の値の平均値を算出する。
- 各メーカーの商品種別ごとに、商品がどのくらいおいしいか傾向を掴むため、各メーカーの商品種別ごとに `評価` 列の値の平均値を算出する。
- 各メーカーごとに、商品の価格の傾向を掴むため、メーカーごとに `値段` 列の値の平均値を算出する。
- 各メーカーの商品種別ごとに、商品の価格の傾向を掴むため、各メーカーの商品種別ごとに `値段` 列の値の平均値を算出する。
- 各メーカーごとに、商品のおいしさにどのくらいバラつきがあるか傾向を掴むため、メーカーごとに `評価` 列の値の標準偏差を算出する。
- 各メーカーの商品種別ごとに、商品のおいしさにどのくらいバラつきがあるか傾向を掴むため、各メーカーの商品種別ごとに `評価` 列の値の標準偏差を算出する。
- 各メーカーごとに、商品の価格にどのくらいバラつきがあるか傾向を掴むため、メーカーごとに `値段` 列の値の標準偏差を算出する。
- 各メーカーの商品種別ごとに、商品の価格にどのくらいバラつきがあるか傾向を掴むため、各メーカーの商品種別ごとに `値段` 列の値の

標準偏差を算出する。

まず、以下の集計を行う。

- 各メーカーごとに、商品がどのくらいおいしいか傾向を掴むため、メーカーごとに `評価` 列の値の平均値を算出する。

`groupby()` メソッドを使い、 `メーカ名` をグループ分けするキーに指定することで、メーカーごとにグループ化できる。さらに、 `評価` について集計したいため、 `評価` 列を指定し、 `mean()` メソッドを使用することで、メーカーごとの評価の平均値を集計できる。

```
print(analysis_df.groupby("メーカー名")["評価"].mean())
```

```
メーカー名
A    27.500000
B    60.000000
C    86.666667
Name: 評価, dtype: float64
```

上記のグループ集計の結果から、評価の上限が100である中で、メーカーAの評価の平均値は約28であるため、メーカーAの商品は、味についてはおいしくないと評価されていることがわかる。メーカーBの評価の平均値は60であるため、メーカーBの商品は、味については中程度にはおいしいと評価されていることがわかる。メーカーCの評価の平均値は約87であるため、メーカーCの商品は、味についてはとてもおいしいと評価されていることがわかる。

上記の結果から、メーカーごとの商品の味の傾向は以下であることがわかる。

- メーカーA：おいしくない。
- メーカーB：中程度にはおいしい。
- メーカーC：とてもおいしい。

続いて、以下の集計を行う。

- 各メーカーの商品種別ごとに、商品がどのくらいおいしいか傾向を掴むため、各メーカーの商品種別ごとに `評価` 列の値の平均値を算出する。

`groupby()` メソッドを使い、 `メーカ名` 、 `種別` の順にグループ分けするキーに指定することで、各メーカーの商品種別ごとにグループ化できる。さらに、 `評価` について集計したいため、 `評価` 列を指定し、 `mean()` メソッドを使用することで、各メーカーの商品種別ごとの評価の平均値を集計できる。

```
print(analysis_df.groupby(["メーカー名", "種別"])["評価"].mean())
```

```
メーカー名  種別
A       お茶          43.333333
        果汁入り飲料     16.666667
        炭酸飲料        20.000000
B       お茶          56.666667
        果汁入り飲料     63.333333
        炭酸飲料        60.000000
C       お茶          80.000000
        果汁入り飲料     90.000000
        炭酸飲料        90.000000
Name: 評価, dtype: float64
```

上記のグループ集計の結果から、メーカーAでは、果汁入り飲料と炭酸飲料は特においしくないが、お茶はそれほどマズくないということがわかる。メーカーBは、どれも中程度にはおいしい。細かい違いではあるが、メーカーBでは、果汁入り飲料、炭酸飲料、お茶の順でおいしいということがわかる。メーカーCはどれもとてもおいしく、果汁入り飲料と炭酸飲料は特においしいことがわかる。

上記の結果から、各メーカーの商品種別ごとの商品の味の傾向は以下であることがわかる。

- メーカーA：果汁入り飲料と炭酸飲料は特においしくない。お茶についても、おいしくはないが、果汁入り飲料と炭酸飲料ほどではない。
- メーカーB：どれも中程度にはおいしい。
- メーカーC：どれもとてもおいしい。果汁入り飲料と炭酸飲料は特においしい。

続いて、以下の集計を行う。

- 各メーカーごとに、商品の価格の傾向を掴むため、メーカーごとに `値段` 列の値の平均値を算出する。

`groupby()` メソッドを使い、 `メーカ名` をグループ分けするキーに指定することで、メーカーごとにグループ化できる。さらに、 `値段` について集計したいため、 `値段` 列を指定し、 `mean()` メソッドを使用することで、メーカーごとの値段の平均値を集計できる。

```
print(analysis_df.groupby("メーカー名")["値段"].mean())
```

```
メーカー名
A     76.250000
B    316.250000
C    811.111111
Name: 値段, dtype: float64
```

上記のグループ集計の結果から、メーカーAの値段の平均値は約76円であるため、メーカーAの商品は、飲料としては安いことがわかる。メーカーBの値段の平均値は約316円であるため、メーカーBの商品は、飲料としては高いことがわかる。メーカーCの値段の平均値は約811円であるため、メーカーCの商品は、飲料としてはとても高いことがわかる。

上記の結果から、メーカーごとの商品の価格の傾向は以下であることがわかる。

- メーカーA：安い。
- メーカーB：高い。
- メーカーC：とても高い。

続いて、以下の集計を行う。

- 各メーカーの商品種別ごとに、商品の価格の傾向を掴むため、各メーカーの商品種別ごとに `値段` 列の値の平均値を算出する。

`groupby()` メソッドを使い、 `メーカ名` 、 `種別` の順にグループ分けするキーに指定することで、各メーカーの商品種別ごとにグループ化できる。さらに、 `値段` について集計したいため、 `値段` 列を指定し、 `mean()` メソッドを使用することで、各メーカーの商品種別ごとの値段の平均値を集計できる。

```
print(analysis_df.groupby(["メーカー名", "種別"])["値段"].mean())
```

```
メーカー名  種別
A       お茶          76.666667
        果汁入り飲料      80.000000
        炭酸飲料         70.000000
B       お茶         150.000000
        果汁入り飲料     566.666667
        炭酸飲料        190.000000
C       お茶         633.333333
        果汁入り飲料    1000.000000
        炭酸飲料        800.000000
Name: 値段, dtype: float64
```

上記のグループ集計の結果から、メーカーAでは、どの商品も同様に安いことがわかる。細かい違いではあるが、メーカーAでは炭酸飲料、お茶、果汁入り飲料の順に安いことがわかる。メーカーBは、お茶と炭酸飲料が標準的な飲料の価格で、果汁入り飲料が高価格帯であることがわかる。メーカーCはどれも高価格帯であり、特に果汁入り飲料と炭酸飲料が高いことがわかる。

上記の結果から、各メーカーの商品種別ごとの商品の価格の傾向は以下であることがわかる。

- メーカーA：どれも安い。
- メーカーB：お茶と炭酸飲料は標準的な価格。果汁入り飲料は高価格帯。
- メーカーC：どれも高価格帯であり、特に果汁入り飲料と炭酸飲料の価格帯が高い。

続いて、以下の集計を行う。

- 各メーカーごとに、商品のおいしさにどのくらいバラつきがあるか傾向を掴むため、メーカーごとに `評価` 列の値の標準偏差を算出する。

グループ集計の指定方法は平均値を求める際と同様である。標準偏差を求めるため `std()` メソッドを使用する。

```
print(analysis_df.groupby("メーカー名")["評価"].std())
```

```
メーカー名
A    16.690459
B    16.903085
C     7.071068
Name: 評価, dtype: float64
```

上記のグループ集計の結果から、メーカーCの標準偏差がメーカーAとメーカーBと比べて倍以上低いため、メーカーCの商品は他の2メーカーと比べて、商品ごとの味のバラつきが少ないことがわかる。

続いて、以下の集計を行う。

- 各メーカーの商品種別ごとに、商品のおいしさにどのくらいバラつきがあるか傾向を掴むため、各メーカーの商品種別ごとに `評価` 列の値の標準偏差を算出する。

グループ集計の指定方法は平均値を求める際と同様である。標準偏差を求めるため `std()` メソッドを使用する。

```
print(analysis_df.groupby(["メーカー名", "種別"])["評価"].std())
```

```
メーカー名  種別
A       お茶          5.773503
        果汁入り飲料    15.275252
        炭酸飲料       14.142136
B       お茶          5.773503
        果汁入り飲料    30.550505
        炭酸飲料        0.000000
C       お茶          0.000000
        果汁入り飲料    10.000000
        炭酸飲料        0.000000
Name: 評価, dtype: float64
```

上記のグループ集計の結果から、メーカーBの炭酸飲料、メーカーCのお茶と炭酸飲料については各メーカーの種別ごとであればどの商品を選んでも同様のおいしさであることがわかる。メーカーAの商品とメーカーBのお茶についても、各メーカーの種別ごとであれば、そこまで味にバラつきがないことがわかる。全体を見て、メーカーBの果汁入り飲料の味の評価にバラつきがあった。

実際に、メーカーBの果汁入り飲料についてどのように評価がバラついているか実際のデータからみてみよう。

```
print(analysis_df[(analysis_df["メーカー名"] == "B") & (analysis_df["種別"] == "果汁入り飲料")])
```

```
    メーカー名       種別         商品名  レビュー件数  評価   値段
11      B  果汁入り飲料  アップルジュース      100  70  700
14      B  果汁入り飲料  オレンジジュース       86  90  900
17      B  果汁入り飲料  ミックスジュース      122  30  100
```

メーカーBの果汁入り飲料の実際のデータを確認すると、メーカーBの果汁入り飲料には、100円から900円までの商品が展開されており、安い商品は低い評価を受けて、値段が高くなるほど高い評価を受けていることがわかる。

データを分析する際には、集計するだけでなく、必要に応じて実際のデータを確認することも重要である。

上記の結果から、各メーカーの商品種別ごとの商品の評価の傾向は以下であることがわかる。

- メーカーA：多少バラつきはあるが、どの商品も安定しておいしくない。
- メーカーB：お茶と炭酸飲料は安定して中程度においしい。果汁入り飲料は、安い商品はおいしくなく、高くなるほどおいしい。
- メーカーC：果汁入り飲料については多少バラつきはあるが、どの商品も安定してとてもおいしい。

続いて、以下の集計を行う。

- 各メーカーごとに、商品の価格にどのくらいバラつきがあるか傾向を掴むため、メーカーごとに `値段` 列の値の標準偏差を算出する。

グループ集計の指定方法は平均値を求める際と同様である。標準偏差を求めるため `std()` メソッドを使用する。

```
print(analysis_df.groupby("メーカー名")["値段"].std())
```

```
メーカー名
A     15.059406
B    304.674885
C    169.148193
Name: 値段, dtype: float64
```

上記のグループ集計の結果から、メーカーAは安定して安く、メーカーBとメーカーCは高い中でも様々な価格の商品が展開されていることがわかる。

最後に、以下の集計を行う。

- 各メーカーの商品種別ごとに、商品の価格にどのくらいバラつきがあるか傾向を掴むため、各メーカーの商品種別ごとに `値段` 列の値の標準偏差を算出する。

グループ集計の指定方法は平均値を求める際と同様である。標準偏差を求めるため `std()` メソッドを使用する。

```
print(analysis_df.groupby(["メーカー名", "種別"])["値段"].std())
```

```
メーカー名  種別
A       お茶           5.773503
        果汁入り飲料      17.320508
        炭酸飲料         28.284271
B       お茶           0.000000
        果汁入り飲料     416.333200
        炭酸飲料         14.142136
C       お茶         115.470054
```

```
9         果汁入り飲料      0.000000
10         炭酸飲料        0.000000
11 Name: 値段, dtype: float64
```

上記のグループ集計の結果から、メーカーAは全体的に多少価格がバラついている。細かいところだが、炭酸飲料、果汁入り飲料、お茶の順で価格がバラついていることがわかる。メーカーBはお茶と炭酸飲料の価格のバラつきは少なく、果汁入り飲料の価格が大きくバラついていることがわかる。メーカーCはお茶の価格にバラつきがあり、果汁入り飲料と炭酸飲料には価格のバラつきがないことがわかる。

上記の結果から、各メーカーの商品種別ごとの商品の価格の傾向は以下であることがわかる。

- メーカーA：多少バラつきはあるが総じて安い。
- メーカーB：お茶と炭酸飲料に関しては、どれも標準的な価格である。果汁入り飲料については、100円から900円まで幅広い価格帯の商品が展開されている。
- メーカーC：お茶については値段帯のバラつきはあるものの、どれも総じてとても高い。

今回行う集計は以上となる。

ここまでの集計結果から得られた考察をまとめる。

まずは各メーカーごとの味と値段の全体的な傾向は以下である。

- メーカーA：ほとんどの商品が標準的な価格より安いが、味はおいしくない。格安指向のメーカー。
- メーカーB：標準的な価格の商品がメインであるが、一部値段が高い商品もある。味については一部を除き全体的に並みのおいしさ。一般大衆向け指向のメーカー。
- メーカーC：高い価格帯の商品を主に扱っている。味については、高いだけあり全体的に評価が高く、とてもおいしい。高級志向のメーカー。

さらに各メーカーで、商品種別ごとの味と値段の傾向は以下である。

- メーカーA：お茶、果汁入り飲料、炭酸飲料のどれも安いが、どれもおいしくはない。特に果汁入り飲料、炭酸飲料については味の評価がとても低い。
- メーカーB：お茶と炭酸飲料については、標準的な価格で並みのおいしさである。果汁入り飲料については、標準的な価格の商品から高い商品まで広く展開されており、値段が高くなればなれるだけ評価が高くなる。
- メーカーC：お茶、果汁入り飲料、炭酸飲料の高い分おいしい。お茶と比べて、果汁入り飲料、炭酸飲料は価格帯が高いが、その分味の評価はお茶よりも高い。

本章で実施するデータ分析は以上である。

問題1

問題文

飲料に関するレビューが記載されている以下のようなWebページのHTML形式のデータが格納されている変数 `html_text` がある。出力例のように、このHTML形式のデータからpandasのデータフレームを作成し、作成したデータフレームを出力するプログラムを記述せよ。

作成するデータフレームは、行番号に `0` から `7` までの数字を持ち、列名は左から `メーカー名` 、 `種別` 、 `商品名` 、 `レビュー件数` 、 `評価` 、 `値段` とする。

各行について、各列に以下の要素の文字列が格納される。

- `メーカー名` の列にはclass名が `maker` の要素の文字列が値として格納される。
- `種別` の列にはclass名が `kind` の要素の文字列が値として格納される。

- `商品名` の列にはclass名が `item` の要素の文字列が値として格納される。
- `レビュー件数` の列にはclass名が `review-number` の要素の文字列が値として格納される。
- `評価` の列にはclass名が `review-point` の要素の文字列が値として格納される。
- `値段` の列にはclass名が `price` の要素の文字列が値として格納される。

また、各行の順番については、インデックス `0` の行にはメーカー名が `A` の `緑茶` 、インデックス `1` の行にはメーカー名が `A` の `メロンソーダ` といったように、出力例と同様の順番に出力すること。

```
<html>

<head>
    <title>飲料水レビューサイト</title>
</head>

<body>
    <div id="column-header">
        <span style="width:100px;" class="column-header-name">メーカー名</span>
         <span style="width:100px;" class="column-header-name">種別</span>
         <span style="width:198px;" class="column-header-name">商品名</span>
         <span style="width:100px;" class="column-header-name">レビュー件数</span>
         <span style="width:100px;" class="column-header-name">評価</span>
         <span style="width:100px;" class="column-header-name">値段</span>
     </div>
     <div id="rows">
         <p style="height: 20px">
             <span class="maker" dir="ltr">A</span>
             <span class="kind" dir="ltr">お茶</span>
             <span class="item" dir="ltr">緑茶</span>
             <span class="review-number" dir="ltr">20</span>
             <span class="review-point" dir="ltr">50</span>
             <span class="price" dir="ltr">70</span>
         </p>
         <p style="height: 20px">
             <span class="maker" dir="ltr">A</span>
             <span class="kind" dir="ltr">炭酸飲料</span>
             <span class="item" dir="ltr">メロンソーダ</span>
             <span class="review-number" dir="ltr">30</span>
             <span class="review-point" dir="ltr">30</span>
             <span class="price" dir="ltr">90</span>
         </p>
         <p style="height: 20px">
             <span class="maker" dir="ltr">A</span>
             <span class="kind" dir="ltr">お茶</span>
             <span class="item" dir="ltr">ウーロン茶</span>
             <span class="review-number" dir="ltr">50</span>
             <span class="review-point" dir="ltr">40</span>
             <span class="price" dir="ltr">80000</span>
         </p>
         <p style="height: 20px">
             <span class="maker" dir="ltr">A</span>
             <span class="kind" dir="ltr">炭酸飲料</span>
             <span class="item" dir="ltr">エナジーソーダ</span>
             <span class="review-number" dir="ltr">12022</span>
             <span class="review-point" dir="ltr">98</span>
             <span class="price" dir="ltr">50</span>
         </p>
         <p style="height: 20px">
             <span class="maker" dir="ltr">B</span>
             <span class="kind" dir="ltr">お茶</span>
             <span class="item" dir="ltr">緑茶</span>
             <span class="review-number" dir="ltr">80</span>
             <span class="review-point" dir="ltr">60</span>
             <span class="price" dir="ltr">150</span>
         </p>
         <p style="height: 20px">
             <span class="maker" dir="ltr">B</span>
             <span class="kind" dir="ltr">炭酸飲料</span>
             <span class="item" dir="ltr">ライチソーダ</span>
             <span class="review-number" dir="ltr">90</span>
```

```
            <span class="review-point" dir="ltr">60</span>
            <span class="price" dir="ltr">180</span>
        </p>
        <p style="height: 20px">
            <span class="maker" dir="ltr">B</span>
            <span class="kind" dir="ltr">お茶</span>
            <span class="item" dir="ltr">麦茶</span>
            <span class="review-number" dir="ltr">55</span>
            <span class="review-point" dir="ltr">60</span>
            <span class="price" dir="ltr">150</span>
        </p>
        <p style="height: 20px">
            <span class="maker" dir="ltr">B</span>
            <span class="kind" dir="ltr">炭酸飲料</span>
            <span class="item" dir="ltr">メロンソーダ</span>
            <span class="review-number" dir="ltr">30</span>
            <span class="review-point" dir="ltr">60</span>
            <span class="price" dir="ltr">200</span>
        </p>
    </div>
</body>

</html>
```

出力

```
   メーカー名    種別       商品名  レビュー件数  評価    値段
0     A    お茶       緑茶      20  50    70
1     A  炭酸飲料    メロンソーダ      30  30    90
2     A    お茶     ウーロン茶      50  40  80000
3     A  炭酸飲料   エナジーソーダ    12022  98    50
4     B    お茶       緑茶      80  60    150
5     B  炭酸飲料    ライチソーダ      90  60    180
6     B    お茶       麦茶      55  60    150
7     B  炭酸飲料    メロンソーダ      30  60    200
```

解答の雛形

```
import pandas as pd
from bs4 import BeautifulSoup

html_text = """
<html>

<head>
    <title>飲料水レビューサイト</title>
</head>

<body>
    <div id="column-header">
        <span style="width:100px;" class="column-header-name">メーカー名</span>
        <span style="width:100px;" class="column-header-name">種別</span>
        <span style="width:198px;" class="column-header-name">商品名</span>
        <span style="width:100px;" class="column-header-name">レビュー件数</span>
        <span style="width:100px;" class="column-header-name">評価</span>
        <span style="width:100px;" class="column-header-name">値段</span>
    </div>
    <div id="rows">
        <p style="height: 20px">
            <span class="maker" dir="ltr">A</span>
            <span class="kind" dir="ltr">お茶</span>
            <span class="item" dir="ltr">緑茶</span>
            <span class="review-number" dir="ltr">20</span>
            <span class="review-point" dir="ltr">50</span>
            <span class="price" dir="ltr">70</span>
        </p>
```

```
        <p style="height: 20px">
            <span class="maker" dir="ltr">A</span>
            <span class="kind" dir="ltr">炭酸飲料</span>
            <span class="item" dir="ltr">メロンソーダ</span>
            <span class="review-number" dir="ltr">30</span>
            <span class="review-point" dir="ltr">30</span>
            <span class="price" dir="ltr">90</span>
        </p>
        <p style="height: 20px">
            <span class="maker" dir="ltr">A</span>
            <span class="kind" dir="ltr">お茶</span>
            <span class="item" dir="ltr">ウーロン茶</span>
            <span class="review-number" dir="ltr">50</span>
            <span class="review-point" dir="ltr">40</span>
            <span class="price" dir="ltr">80000</span>
        </p>
        <p style="height: 20px">
            <span class="maker" dir="ltr">A</span>
            <span class="kind" dir="ltr">炭酸飲料</span>
            <span class="item" dir="ltr">エナジーソーダ</span>
            <span class="review-number" dir="ltr">12022</span>
            <span class="review-point" dir="ltr">98</span>
            <span class="price" dir="ltr">50</span>
        </p>
        <p style="height: 20px">
            <span class="maker" dir="ltr">B</span>
            <span class="kind" dir="ltr">お茶</span>
            <span class="item" dir="ltr">緑茶</span>
            <span class="review-number" dir="ltr">80</span>
            <span class="review-point" dir="ltr">60</span>
            <span class="price" dir="ltr">150</span>
        </p>
        <p style="height: 20px">
            <span class="maker" dir="ltr">B</span>
            <span class="kind" dir="ltr">炭酸飲料</span>
            <span class="item" dir="ltr">ライチソーダ</span>
            <span class="review-number" dir="ltr">90</span>
            <span class="review-point" dir="ltr">60</span>
            <span class="price" dir="ltr">180</span>
        </p>
        <p style="height: 20px">
            <span class="maker" dir="ltr">B</span>
            <span class="kind" dir="ltr">お茶</span>
            <span class="item" dir="ltr">麦茶</span>
            <span class="review-number" dir="ltr">55</span>
            <span class="review-point" dir="ltr">60</span>
            <span class="price" dir="ltr">150</span>
        </p>
        <p style="height: 20px">
            <span class="maker" dir="ltr">B</span>
            <span class="kind" dir="ltr">炭酸飲料</span>
            <span class="item" dir="ltr">メロンソーダ</span>
            <span class="review-number" dir="ltr">30</span>
            <span class="review-point" dir="ltr">60</span>
            <span class="price" dir="ltr">200</span>
        </p>
    </div>
</body>

</html>
"""

# ここに解答を入力
```

問題2

問題文

以下のように飲料のレビューに関するデータが格納されたデータフレーム `df` がある。分析を行うためのデータフレームを作成するため、データフレーム `df` に対して以下の加工を行う。

- レビュー件数が外れ値のデータを分析対象のデータに含めるのは不適切と判断した。平均値をμ、標準偏差をσで表し、$\mu \pm 2\sigma$の範囲外の値を外れ値とした場合、 `レビュー件数` 列に外れ値を含む行をデータフレーム `df` から取り除く。
- インデックス `2` の行の `値段` 列の値が、レビューサイトのバグで、本来の値から1000倍されていた。正しい値に修正するため、インデックス `2` の行の `値段` 列の値を `80` に変更する。

上記の修正を行ったpandasのデータフレームを作成し、出力例のように、作成したデータフレームを出力するプログラムを記述せよ。

```
   メーカー名    種別       商品名  レビュー件数  評価     値段
0     A    お茶        緑茶       20  50     70
1     A  炭酸飲料    メロンソーダ       30  30     90
2     A    お茶     ウーロン茶       50  40  80000
3     A  炭酸飲料   エナジーソーダ    12022  98     50
4     B    お茶        緑茶       80  60    150
5     B  炭酸飲料    ライチソーダ       90  60    180
6     B    お茶        麦茶       55  60    150
7     B  炭酸飲料    メロンソーダ       30  60    200
```

出力

```
   メーカー名    種別       商品名  レビュー件数  評価    値段
0     A    お茶        緑茶       20  50    70
1     A  炭酸飲料   メロンソーダ       30  30    90
2     A    お茶     ウーロン茶       50  40    80
4     B    お茶        緑茶       80  60   150
5     B  炭酸飲料   ライチソーダ       90  60   180
6     B    お茶        麦茶       55  60   150
7     B  炭酸飲料   メロンソーダ       30  60   200
```

解答の雛形

```
import pandas as pd

data = {
    "メーカー名": ["A", "A", "A", "A", "B", "B", "B", "B"],
    "種別": ["お茶", "炭酸飲料", "お茶", "炭酸飲料", "お茶", "炭酸飲料", "お茶", "炭酸飲料"],
    "商品名": ["緑茶", "メロンソーダ", "ウーロン茶", "エナジーソーダ", "緑茶", "ライチソーダ", "麦茶", "メロンソーダ"],
    "レビュー件数": [20, 30, 50, 12022, 80, 90, 55, 30],
    "評価": [50, 30, 40, 98, 60, 60, 60, 60],
    "値段": [70, 90, 80000, 50, 150, 180, 150, 200],
}

df = pd.DataFrame(data)

# ここに解答を入力
```

問題3

問題文

以下のように飲料のレビューに関するデータを分析用に加工したデータフレーム `df` がある。出力例のように、 `メーカー名` の `種別` ごとに、 `レビュー件数` の平均値を出力するプログラムを記述せよ。

期待する出力結果には、`Name: レビュー件数, dtype: float64` という文字列も含まれる。これは、どの列に対して平均値を算出したかという情報と、平均値のデータ型に関する情報である。平均値を出力する処理を正しく実装すればこの情報が出力されるため、作成するプログラムの出力結果にも `Name: レビュー件数, dtype: float64` を含めたままにすること。

```
   メーカー名    種別      商品名  レビュー件数  評価   値段
0     A    お茶      緑茶      20  50   70
1     A  炭酸飲料  メロンソーダ      30  30   90
2     A    お茶   ウーロン茶      50  40   80
4     B    お茶      緑茶      80  60  150
5     B  炭酸飲料  ライチソーダ      90  60  180
6     B    お茶      麦茶      55  60  150
7     B  炭酸飲料  メロンソーダ      30  60  200
```

出力

```
メーカー名  種別
A       お茶      35.0
        炭酸飲料    30.0
B       お茶      67.5
        炭酸飲料    60.0
Name: レビュー件数, dtype: float64
```

解答の雛形

```
import pandas as pd

data = {
    "メーカー名": ["A", "A", "A", "B", "B", "B", "B"],
    "種別": ["お茶", "炭酸飲料", "お茶", "お茶", "炭酸飲料", "お茶", "炭酸飲料"],
    "商品名": ["緑茶", "メロンソーダ", "ウーロン茶", "緑茶", "ライチソーダ", "麦茶", "メロンソーダ"],
    "レビュー件数": [20, 30, 50, 80, 90, 55, 30],
    "評価": [50, 30, 40, 60, 60, 60, 60],
    "値段": [70, 90, 80, 150, 180, 150, 200],
}

df = pd.DataFrame(data)

# ここに解答を入力
```