データラングリング入門

Python pandas を使ったデータ加工

HDS 中村 知繁

T. Nakamura | Juntendo Univ.

2025/02/08

これまでの学習と本日のテーマ

これまでの学習:

- データフレームの基本的な確認方法
- `matplotlib` や `seaborn` を使ったデータの可視化
  - 散布図、折れ線グラフ、箱ひげ図、ヒストグラム、棒グラフなど

この章で学ぶこと：データラングリング
`pandas` ライブラリを使った主要なデータ加工・整形操作を学びます。

データラングリングとは？

データ分析の前処理として非常に重要な工程です。主な操作には以下のようなものがあります。

1. **行のフィルタリング**: 条件に合う行だけを選ぶ。
2. **列の集計**: 列データから平均や合計などの要約統計量を計算する。
3. **グループ化**: 特定の列の値に基づいて行をグループにまとめ、グループごとの集計を行う。
4. **列の作成・変換**: 既存の列から新しい列を作ったり、列の値を変換したりする。
5. **並べ替え**: 特定の列の値に基づいて行を並び替える。
6. **結合**: 複数のデータフレームを特定のキーを使ってつなぎ合わせる。

なぜ `pandas` を学ぶのか？

- Pythonでデータ分析を行う際の中心的ライブラリ。
- **SQLとの類似性**:
  - `pandas`の操作は、データベース操作言語であるSQL（Structured Query Language）の考え方と共通する部分が多くあります。
  - SQLは大量データを効率的に扱うために広く利用されており、`pandas`を学ぶことはSQLの理解にも繋がります。

準備：ライブラリとデータセット

まず、必要なライブラリをインポートし、サンプルデータセットを読み込みます。
今回は `seaborn` パッケージに含まれる `iris` データセット（アヤメの花のデータ）を使用します。

```
import pandas as pd
import seaborn as sns
import numpy as np # 数値計算で利用

# seabornからirisデータセットをロード
iris_df = sns.load_dataset('iris')

# データの最初の5行を確認
print("--- データセットの最初の5行 ---")
print(iris_df.head())
# print("\n--- データセットの基本情報 ---")
# iris_df.info() # スライドでは出力を省略する場合あり
# print("\n--- データセットの要約統計量 ---")
# print(iris_df.describe()) # スライドでは出力を省略する場合あり
```

メソッドチェーン：処理を繋げる書き方

`pandas` では、複数のデータ操作を `.`（ドット）で繋げて連続的に記述する**メソッドチェーン**がよく使われます。

例： データフレーム `df` に操作 `f()`、`g()`、`h()` を順番に適用

```
# result = df.f().g().h()
```

読み方：

1. `df` を用意。
2. `df` に `f()` を適用。
3. その結果に `g()` を適用。
4. さらにその結果に `h()` を適用。

この章で学ぶ操作の多くは、メソッドチェーンで簡潔に記述できます。

1. 行のフィルタリング：条件に合うデータを選ぶ

データセットの中から、特定の条件を満たす行だけを抽出します。

例1：品種 (`species`) が `setosa` のデータを抽出

```
# 通常のブールインデックス
setosa_df = iris_df[iris_df['species'] == 'setosa']
print(setosa_df.head())
```

`query` メソッドを使った場合 (よりSQLライク)

```
setosa_df_query = iris_df.query("species == 'setosa'")
print(setosa_df_query.head())
```

行のフィルタリング：複数の条件

複数の条件を組み合わせることも可能です。

- `&`: AND (かつ)
- `|`: OR (または)
- `~`: NOT (否定)
- 比較演算子: `>`, `<`, `>=`, `<=`, `!=`

例2：「品種が `versicolor` で、かつ花びらの長さ (`petal_length`) が4.5cmより大きい」データ

```
versicolor_long_petal = iris_df[
    (iris_df['species'] == 'versicolor') & (iris_df['petal_length'] > 4.5)
]
print(versicolor_long_petal)
```

```
# queryメソッドの場合
# versicolor_long_petal_query = iris_df.query(
#     "species == 'versicolor' and petal_length > 4.5"
# )
```

行のフィルタリング：`isin()` メソッド

リストに含まれるいずれかの値でフィルタリングする場合、`isin()` メソッドが便利です。

例3：品種が `setosa` または `virginica` のデータ

```
setosa_virginica_df = iris_df[iris_df['species'].isin(['setosa', 'virginica'])]
print(setosa_virginica_df.sample(5)) # ランダムに5行表示
```

2. 列の集計：データの特徴を掴む

データフレームの列に対し、平均、合計、最大値、最小値などの**要約統計量**を計算します。

例： `sepal_length` （がくの長さ）の平均と標準偏差

```
mean_sl = iris_df['sepal_length'].mean()
std_sl = iris_df['sepal_length'].std()
print(f"がくの長さ 平均: {mean_sl:.2f} cm, 標準偏差: {std_sl:.2f} cm")
```

複数の統計量を一度に計算 (`agg` メソッド)

```
summary_stats = iris_df['sepal_length'].agg(['mean', 'std', 'min', 'max'])
print(summary_stats)
```

欠損値(`NaN`)は、多くの集計関数でデフォルトで無視されます。

主な集計関数

- `mean()`: 平均値
- `std()`: 標準偏差
- `median()`: 中央値
- `min()`, `max()`: 最小値、最大値
- `sum()`: 合計
- `count()`: 件数 (非欠損値の数)
- `quantile(q=0.25)`: 四分位数 (例: 0.25で第一四分位数)
- `nunique()`: ユニークな値の数
- `var()`: 分散

3. グループ化：グループごとの特徴を見る

特定の列の値に基づいてデータをグループ分けし、各グループに対して集計処理を行います。
`groupby()` メソッドを使用します。

例1：品種 (`species`) ごとに各特徴量の平均値を計算

```
species_mean_df = iris_df.groupby('species').mean()
print(species_mean_df)
```

`groupby()` 実行時点ではグループ化オブジェクトが生成され、その後に集計関数 (`mean()`, `sum()`, `agg()` など) を適用すると計算が実行されます。

グループ化：発展的な集計

例2：品種ごとの花びらの長さ (`petal_length`) の平均

```
petal_len_mean_by_species = iris_df.groupby('species')['petal_length'].mean()
print(petal_len_mean_by_species)
```

例3：品種ごとの花びらの長さの集計 (平均, 標準偏差, 個数)

```
petal_summary = iris_df.groupby('species')['petal_length'].agg(['mean', 'std', 'count'])
print(petal_summary)
```

4. 列の作成・変換：新しい情報を加える

既存の列の値を使って新しい列を作成したり、列の値を変換したりします。

例：花びらの面積 (`petal_area`) を `petal_length * petal_width` で計算

方法1：直接代入

```
iris_df_mutated = iris_df.copy() # 元のデータを変更しないためにコピー
iris_df_mutated['petal_area'] = \
    iris_df_mutated['petal_length'] * iris_df_mutated['petal_width']
print(iris_df_mutated[['petal_length', 'petal_width', 'petal_area']].head())
```

列の作成・変換：`assign()` メソッド

`assign()` メソッドは、新しい列を追加したデータフレームを返します (元のデータフレームは変更しません)。メソッドチェーンに適しています。

方法2：`assign` メソッド

```
iris_df_assigned = iris_df.assign(
    petal_area = iris_df['petal_length'] * iris_df['petal_width'],
    sepal_area = iris_df['sepal_length'] * iris_df['sepal_width']
)
print(iris_df_assigned[['species', 'petal_area', 'sepal_area']].head())
```

5. 並べ替え：データを順序付ける

特定の列の値に基づいて、データフレームの行を昇順または降順に並べ替えます。
`sort_values()` メソッドを使用します。

例1：花びらの長さ (`petal_length`) が大きい順（降順）に並べ替え

```
sorted_iris_df = iris_df.sort_values(by='petal_length', ascending=False)
print(sorted_iris_df[['species', 'petal_length']].head())
```

- `ascending=True`: 昇順 (デフォルト)
- `ascending=False`: 降順

並べ替え：複数キーでのソート

複数のキー（列）を指定して並べ替えることも可能です。

例2：品種 (`species`) で昇順、その中で花びらの長さ (`petal_length`) で降順

```
sorted_multi_key = iris_df.sort_values(
    by=['species', 'petal_length'],
    ascending=[True, False] # speciesは昇順, petal_lengthは降順
)
print(sorted_multi_key[['species', 'petal_length']].head(10))
```

6. 結合：複数のデータを繋ぎ合わせる

2つ以上のデータフレームを、共通のキー（列）をもとに横に繋ぎ合わせます。
`merge()` メソッドを使用します。

例： `iris` データに日本語の品種名情報を結合
まず、日本語品種名を持つデータフレームを作成します。

```
species_jpn_df = pd.DataFrame({
    'species': ['setosa', 'versicolor', 'virginica'],
    'species_jpn': ['ヒオウギアヤメ', 'ブルーフラッグ', 'バージニカ']
})
print("--- 日本語品種名データフレーム ---")
print(species_jpn_df)
```

結合：`merge()` の実行

`on` 引数でキーとなる列を指定します。`how` 引数で結合方法を指定します（`left`, `right`, `inner`, `outer`）。

```
merged_iris_df = pd.merge(iris_df, species_jpn_df, on='species', how='left')
# how='left': 左側(iris_df)の全行を残し、右側(species_jpn_df)から情報を付加

print("\n--- 日本語品種名と結合したデータ（サンプル） ---")
print(merged_iris_df.sample(5))
```

キー列名が左右で異なる場合は `left_on`, `right_on` を使用します。

その他の便利な操作

上記以外にも、データラングリングで役立つ基本的な操作があります。

- **列の選択**: 特定の列だけを選び出す。
- **列名の変更**: 列の名前を変更する。
- **上位/下位N件の取得**: 特定の条件でトップNのデータを抽出。

その他の操作：列の選択

特定の列だけを選び出すには、列名をリストで指定します。

```
# 'species', 'petal_length', 'petal_width' の3列を選択
selected_columns_df = iris_df[['species', 'petal_length', 'petal_width']]
print(selected_columns_df.head())
```

特定の列を削除するには `drop()` メソッドを使用します。

```
# 'sepal_width' 列を削除 (axis=1で列を指定)
# dropped_df = iris_df.drop('sepal_width', axis=1)
# print(dropped_df.head())
```

その他の操作：列名の変更

`rename()` メソッドで列名を変更します。

```
renamed_df = iris_df.rename(columns={
    'sepal_length': 'gaku_nagasa_cm', # がくの長さ(cm)
    'sepal_width': 'gaku_haba_cm',    # がくの幅(cm)
    'petal_length': 'kaben_nagasa_cm',# 花弁の長さ(cm)
    'petal_width': 'kaben_haba_cm'    # 花弁の幅(cm)
})
print(renamed_df.head())
```

その他の操作：上位/下位N件の取得

`nlargest()` や `nsmallest()` メソッドで、特定の列の値が大きい/小さい順にN件のデータを取得します。

```
# petal_length が大きい上位3件
top_3_petal_length = iris_df.nlargest(3, 'petal_length')
print("--- 花弁の長さが大きい上位3件 ---")
print(top_3_petal_length)

# petal_length が小さい上位3件
# smallest_3_petal_length = iris_df.nsmallest(3, 'petal_length')
```

まとめ：データラングリング操作一覧

操作	`pandas` での主な実現方法	説明
行のフィルタリング	`[]` (ブールインデックス), `query()`	条件に合う行を選択する
列の集計	`mean()`, `sum()`, `agg()` など	列の値を要約統計量でまとめる
グループ化	`groupby()` と集計関数	グループごとに集計する
列の作成・変換	直接代入, `assign()`	既存の列から新しい列を作成したり、値を変換したりする
並べ替え	`sort_values()`	特定の列の値に基づいて行を並び替える
結合	`merge()`	共通のキーを使って複数のデータフレームを繋ぎ合わせる
列の選択	`[]`, `loc[]`, `iloc[]`, `drop()`	特定の列を選び出す、または削除する
列名の変更	`rename()`	列の名前を変更する
上位/下位N件取得	`head()`, `tail()`, `nlargest()`, `nsmallest()`	データの一部や極端な値を持つデータを抽出する

終わりに

データラングリングは、データ分析プロジェクトにおいて非常に重要なスキルです。
分析に適した形にデータを整形することで、より洞察に満ちた結果を得ることができます。

様々なデータセットでこれらの操作を練習し、習熟を目指しましょう！ 🌸