



# データのインポートとTidy Data

HDS 中村 知繁

# 本章のゴール

1. さまざまな形式の外部データをPythonにインポートする方法を学ぶ。
2. **Tidy Data (整然データ)** の概念と、その重要性を理解する。
3. データをTidyな形式に整形する方法を習得する。

# これまでの復習と本章の位置づけ

- 本章:
  - **Tidy Data:** データ分析の標準形式
  - 実世界のデータは必ずしも Tidy ではない!
    - → 自分でインポートし、Tidy に整形する必要がある。
- この知識は、今後の統計モデリングや機械学習の基盤となります。

# 1. データインポート

分析はデータ入手から！主なスプレッドシート形式:

## 1. CSV (Comma Separated Values) `.csv`:

- カンマ区切りテキストファイル。
- 1行 = 1観測値、最初の行は列名が多い。
- 広く使われる交換フォーマット。

## 2. Excel `.xlsx`:

- Microsoft Excel形式。
- データ + 書式設定、数式、メタデータ。

Pythonでは `pandas` ライブラリが主役！

## 1.1. 必要なライブラリのインポート

まずは、おなじみのライブラリをインポートしましょう。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# seabornのグラフスタイルを設定
sns.set_theme(style="whitegrid")
```

## 1.2. CSVファイルのインポート (1/2)

seaborn の `iris` データセットを一度CSVとして保存し、それを読み込みます。

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

# irisデータセットをCSVファイルとして保存(Colab環境の一時領域に保存)
csv_file_path = 'iris_dataset.csv'
iris_data.to_csv(csv_file_path, index=False) # index=False: DFのインデックスを保存しない

# 保存したCSVファイルをpandasで読み込む
iris_from_csv = pd.read_csv(csv_file_path)
```

`pd.read_csv()` でファイルパスやURLからデータを読み込めます。

## 1.2. CSVファイルのインポート (2/2)

読み込んだデータを確認します。

```
print("CSVから読み込んだirisデータ:")
print(iris_from_csv.head())
```

出力例:

```
CSVから読み込んだirisデータ:
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1         3.5          1.4         0.2    setosa
1           4.9         3.0          1.4         0.2    setosa
2           4.7         3.2          1.3         0.2    setosa
3           4.6         3.1          1.5         0.2    setosa
4           5.0         3.6          1.4         0.2    setosa
```

## 1.3. Excelファイルのインポート (1/2)

同様に `iris` データセットをExcelファイル (`.xlsx`) として保存し、読み込みます。

```
# irisデータセットをExcelファイルとして保存
excel_file_path = 'iris_dataset.xlsx'
# engine='openpyxl' が必要となることがあります
iris_data.to_excel(excel_file_path, index=False, engine='openpyxl')

# 保存したExcelファイルをpandasで読み込む
iris_from_excel = pd.read_excel(excel_file_path)
```

`pd.read_excel()` を使用します。  
(環境により `openpyxl` ライブラリが必要)

## 1.3. Excelファイルのインポート (2/2)

読み込んだデータを確認します。

```
print("\nExcelから読み込んだirisデータ:")
print(iris_from_excel.head())
```

出力例:

```
Excelから読み込んだirisデータ:
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1         3.5          1.4         0.2    setosa
1           4.9         3.0          1.4         0.2    setosa
2           4.7         3.2          1.3         0.2    setosa
3           4.6         3.1          1.5         0.2    setosa
4           5.0         3.6          1.4         0.2    setosa
```

## 2. Tidy Data (整然データ) ✨

データの「形式」は分析の効率を左右します。

**Hadley Wickham** 提唱の「Tidy Data」が指針となります。

“ Tidy Dataの3原則:

1. 各 **変数 (variable)** が **列 (column)** を形成する。
2. 各 **観測値 (observation)** が **行 (row)** を形成する。
3. 各種類の **観測ユニット (observational unit)** が **テーブル (table)** を形成する。

”

この原則に従うと `pandas`, `seaborn` での処理が格段に楽になります。

## 2.1. Tidy Dataの具体例 (1/3)

- ワイド形式 (Wide Format): 1観測に対し複数の変数が横に広がる。
- ロング形式 (Long Format) / ナロー形式: Tidy Dataの典型。

例: `seaborn` の `mpg` データセット (燃費データ)

```
mpg_raw = sns.load_dataset('mpg')
print("元のmpgデータ（一部）:")
print(mpg_raw.head())
```

出力例 (一部):

```
  mpg cylinders displacement horsepower weight acceleration model_year origin          name
0 18.0           8        307.0      130.0  3504.0         12.0       70     usa  chevrolet chevelle malibu
1 15.0           8        350.0      165.0  3693.0         11.5       70     usa        buick skylark 320
```

## 2.1. Tidy Dataの具体例 (2/3)

デモ用に `mpg` データを加工し、ワイド形式の要約データを作成します。  
生産国 (`origin_name`) ごとに、いくつかの性能指標の平均値を計算。

```
# origin列を文字列にマッピング（例示のため）
origin_map = {1: 'usa', 2: 'europe', 3: 'japan'}
if mpg_raw['origin'].dtype != 'object':
    mpg_raw['origin_name'] = mpg_raw['origin'].map(origin_map)
else:
    mpg_raw['origin_name'] = mpg_raw['origin']

# 各'origin_name'の平均的な性能値を持つワイド形式データ
mpg_summary_wide = mpg_raw.groupby('origin_name')[['horsepower', 'weight', 'acceleration']].mean().reset_index()

print("\n加工後のmpg要約データ（ワイド形式）:")
print(mpg_summary_wide)
```

## 2.1. Tidy Dataの具体例 (3/3)

`mpg_summary_wide` (ワイド形式) の出力例:

	origin_name	horsepower	weight	acceleration
0	europe	80.558824	2423.308824	16.788235
1	japan	79.835443	2221.227848	16.172152
2	usa	119.048980	3361.930612	15.033061

- `horsepower`, `weight`, `acceleration` が独立した列に。
- これを国別・指標別に比較するグラフを描くには？ → Tidy形式へ！

## 2.2. ワイド形式からTidy（ロング）形式へ: `melt()` (1/2)

pandas の `melt()` 関数で、ワイド形式 → ロング形式に変換！

```
mpg_summary_tidy = mpg_summary_wide.melt(  
    id_vars=['origin_name'], # そのまま保持する列（識別子）  
    value_vars=['horsepower', 'weight', 'acceleration'], # ロング形式にする列  
    var_name='metric_type', # ↑の列名が入る新しい列の名前  
    value_name='value' # ↑の値が入る新しい列の名前  
)  
  
print("\nTidy形式に変換されたmpg要約データ:")  
print(mpg_summary_tidy.sort_values(by=['origin_name', 'metric_type']).head(6))
```

## 2.2. ワイド形式からTidy（ロング）形式へ: `melt()` (2/2)

`mpg_summary_tidy` (ロング形式) の出力例 (一部):

	origin_name	metric_type	value
3	europe	acceleration	16.788235
0	europe	horsepower	80.558824
6	europe	weight	2423.308824
4	japan	acceleration	16.172152
1	japan	horsepower	79.835443
7	japan	weight	2221.227848

- 原則1: 各変数 (`origin_name`, `metric_type`, `value`) が列を形成。
- 原則2: 各行 (国・指標・値の組) が観測値を形成。
- これで可視化しやすくなります！

# Tidyデータを用いた可視化



Tidy形式の `mpg_summary_tidy` を使って棒グラフを作成。

```
plt.figure(figsize=(10, 6))
sns.barplot(x='origin_name', y='value', hue='metric_type', data=mpg_summary_tidy)
plt.title('生産国別・性能指標別 平均値')
plt.xlabel('生産国')
plt.ylabel('平均値')
plt.legend(title='性能指標')
plt.show()
```

(実行すると、生産国別・性能指標別の平均値を示す棒グラフが表示されます)

### 3. ケーススタディ: 航空会社の乗客数とTidy Data

`seaborn` の `flights` データセット（年・月・乗客数）を使用。  
元々 Tidy に近いですが、ワイド形式 ⇄ ロング形式の変換練習をします。

```
flights_data = sns.load_dataset('flights')
print("元のflightsデータ (ロング形式):")
print(flights_data.head())
```

出力例:

```
   year month  passengers
0  1949    Jan        112
1  1949    Feb        118
2  1949    Mar        132
3  1949    Apr        129
4  1949    May        121
```

### 3.1. ロング形式からワイド形式へ: `pivot_table()` (1/2)

`pivot_table()` で「年を行、月を列、値に乗客数」のワイド形式に。

```
flights_wide = flights_data.pivot_table(  
    index='year',      # 新しいDFのインデックス  
    columns='month',   # 新しいDFの列  
    values='passengers' # 新しいDFの値  
)  
  
print("\nワイド形式に変換されたflightsデータ (年 x 月):")  
print(flights_wide.head())
```

### 3.1. ロング形式からワイド形式へ: `pivot_table()` (2/2)

`flights_wide` の出力例 (一部):

month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
year												
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
...												

- 各月が独立した列になっており、Tidyではありません。

### 3.2. ワイド形式からTidy（ロング）形式へ戻す: `melt()` (1/2)

このワイド形式の `flights_wide` を `melt()` で再びロング形式へ。  
まず、インデックス `year` を列に戻します。

```
flights_wide_reset = flights_wide.reset_index() # 'year'を列に
print("\nインデックスをリセットしたワイド形式データ (一部):")
print(flights_wide_reset.head(2))

# meltを使ってTidy（ロング）形式に変換
flights_tidy_again = flights_wide_reset.melt(
    id_vars=['year'],           # 識別子
    var_name='month',          # 月の列名が入る新しい列名
    value_name='passengers'   # 乗客数が入る新しい列名
)
```

### 3.2. ワイド形式からTidy（ロング）形式へ戻す: `melt()` (2/2)

変換後のデータを確認(元の `flights_data` と同じ構造に)。

```
# 月の順序を元のデータセットのカテゴリ順に合わせる（オプション）
month_order = flights_data['month'].cat.categories
flights_tidy_again['month'] = pd.Categorical(flights_tidy_again['month'], categories=month_order, ordered=True)

print("\n再びTidy形式に変換されたflightsデータ（一部）:")
print(flights_tidy_again.sort_values(by=['year', 'month']).head())
```

出力例:

```
   year month  passengers
0   1949    Jan        112
12  1949    Feb        118
24  1949    Mar        132
36  1949    Apr        129
48  1949    May        121
```

### 3.3. Tidyデータを用いた時系列プロット

Tidy形式のデータ (`flights_tidy_again` または元の `flights_data`) で時系列プロット。

```
plt.figure(figsize=(12, 7))
sns.lineplot(x='year', y='passengers', hue='month', data=flights_tidy_again) # または flights_data
plt.title('月別・年別 航空会社乗客数の推移')
plt.xlabel('年')
plt.ylabel('乗客数')
plt.legend(title='月', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

(実行すると、月別の乗客数推移を示す線グラフが表示されます)

## 4. まとめ



本章で学んだこと:

### 1. データインポート:

- `pd.read_csv()`, `pd.read_excel()` で外部データを読み込む方法。

### 2. Tidy Dataの概念:

- Hadley Wickhamの3原則 (変数=列, 観測値=行, ユニット=テーブル)。
- Tidy Data (ロング形式) が分析や可視化になぜ重要なか。

### 3. データ整形:

- `melt()`: ワイド形式 → Tidy (ロング) 形式へ変換。
- `pivot_table()`: ロング形式 → ワイド形式へ変換 (補足)。

### 4. 実践的応用:

- Tidy Dataで複雑なグラフも効率的に作成可能。