pandas入門

みんなのPython勉強会#106
7/18, 2024
辻真吾（@tsjshg）

自己紹介

- Pythonを使ったデータサイエンスが得意です
  - 大学の研究所に所属しています supported by SoftBank
  - アークエルテクノロジーズ株式会社
  - 株式会社RATH
- Stapyスタッフ
  - いつもありがとうございます
- www.tsjshg.info

お知らせ

ソフトバンクのAxross事業部が人材を募集しています。

バックエンドエンジニア【Axross事業部】

システムエンジニア（システム開発・社内SE・システム運用）

AI／DX人材育成プラットフォームの機能強化を行うバックエンドエンジニア

勤務地：竹芝本社（東京都港区海岸）／リモートワークも可（在宅勤務・サテライトオフィスなど）

#女性積極採用 #エンジニア #新規事業 #バックエンド #AI戦略室

賃金	月給：336,475円～949,000円 想定理論年収：5,690,900円～19,939,000円 （月給：基本給＋勤務実績に応じた時間外手当＋自己成長支援金[10,000円]＋WorkStyle支援金[4,000円]、想定理論年収：月給＋賞与＋特別加算賞与＋各種支援金） ※時間外手当は一般職のみとなり、かつ、実際の時間外手当は、勤務実績に応じて変動します（上記は20時間相当で計算） ※賞与、特別加算賞与は会社業績、個人別評価に応じて変動します ※自己成長支援金は対象外の等級もあります

あらすじ

データサイエンスの実践に欠かせない表形式のデータ（DataFrame）を扱うライブラリであるpandasの話をします

本日のバージョン

- Python==3.12.2
- pandas==2.2.2
- numpy==2.0.0
  - 6月に2006年以来のメージャーバージョンアップ

まずはここから

```
import pandas as pd
```

ほとんどの場合pdと略される

DataFrameと呼ばれる表形式のデータが扱える

辞書から作る

```
df = pd.DataFrame({"col1": [1, 2, -3],
                   "col2": [0.1, 0.2, -0.3]},
                   index=["idx1", "idx2", "idx3"])
```

	col1	col2
idx1	1	0.1
idx2	2	0.2
idx3	-3	-0.3

辞書のキーが列名、値が列方向に並ぶ。

リストから作る

```
df = pd.DataFrame([[1, 0.1],
                   [2, 0.2],
                   [-3, 0.3]],
                   columns=["col1", "col2"],
                   index=["idx1", "idx2", "idx3"])
```

	col1	col2
idx1	1	0.1
idx2	2	0.2
idx3	-3	-0.3

リストの各要素がそれぞれの行になる。

各種ファイルからの読み込み

外部ファイルを読み込んでDataFrameを作る

- CSV
- Excel
- SQLを発行してデータベースから
- Parquet
- などなど

列ごとにデータ型が違う

	col1	col2
idx1	1	0.1
idx2	2	0.2
idx3	-3	-0.3

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3 entries, idx1 to idx3
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   col1    3 non-null      int64
 1   col2    3 non-null      float64
dtypes: float64(1), int64(1)
memory usage: 72.0+ bytes
```

列を取り出す

```
df["col1"]
```

```
# Output
idx1    1
idx2    2
idx3   -3
Name: col1, dtype: int64
```

データ型は1次元配列のSeries

```
type(df["col1"])
```

```
# Output
pandas.core.series.Series
```

要素へのアクセス方法

```
# 列へのアクセス
df.loc[:, "col1"]
```

```
# Output
idx1    1
idx2    2
idx3   -3
Name: col1, dtype: int64
```

名前ではなく位置を指定

```
# 同じ結果
df.iloc[:, 0]
```

行へのアクセス

```
df.loc["idx1", :]
```

```
# Output
col1    1.0
col2    0.1
Name: idx1, dtype: float64
```

```
# 同じ結果
df.iloc[0, :]
```

- locまたはilocを使う書き方が基本
- 実はいろいろな書き方ができて、それが問題となる場合もある（あとで詳しく）

NumPyが使われている

```
df['col1'].values
```

```
# Output
array([ 1,  2, -3])
```

データの操作

```
df
```

	col1	col2
idx1	1	0.1
idx2	2	0.2
idx3	-3	-0.3

```
df > 0
```

	col1	col2
idx1	True	True
idx2	True	True
idx3	False	False

値の変更

負の値をすべて0にする

```
df[df < 0] = 0
```

	col1	col2
idx1	1	0.1
idx2	2	0.2
idx3	0	0.0

NumPyの配列でもできることではあるが、行と列に名前が付けられるDataFrameでこれができるところが便利

フィルタリング

```
# 1列目が正の値かどうか
df["col1"] > 0
```

真偽値のSeriesが帰ってくる

```
# Output
idx1     True
idx2     True
idx3    False
Name: col1, dtype: bool
```

```
df[df["col1"] > 0]
```

	col1	col2
idx1	1	0.1
idx2	2	0.2

データの保存

pickle形式が便利

```
# 書き込み
df.to_pickle('my_df.pkl')
# 読み込み
df = pd.read_pickle('my_df.pkl')
```

pickleには0〜5までプロトコル（バージョン）があるので注意！手元のPCでpickle化したものをGoogle Colabで読み込もうとしたとき失敗したりする。

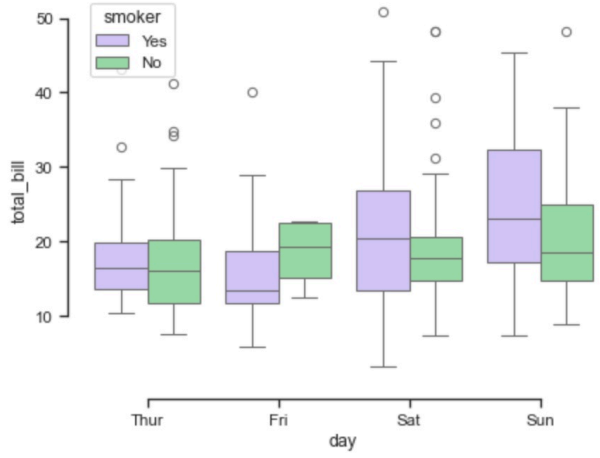
```
df.to_pickle('my_df.pkl', protocol=4)
```

各種ライブラリとの関係

- 可視化ライブラリseabornやplotlyがpandasのDataFrameを前提としている
- 機械学習ライブラリscikit-learnもNumPyのarrayからDataFrameへシフト

seabornの例

Grouped boxplots



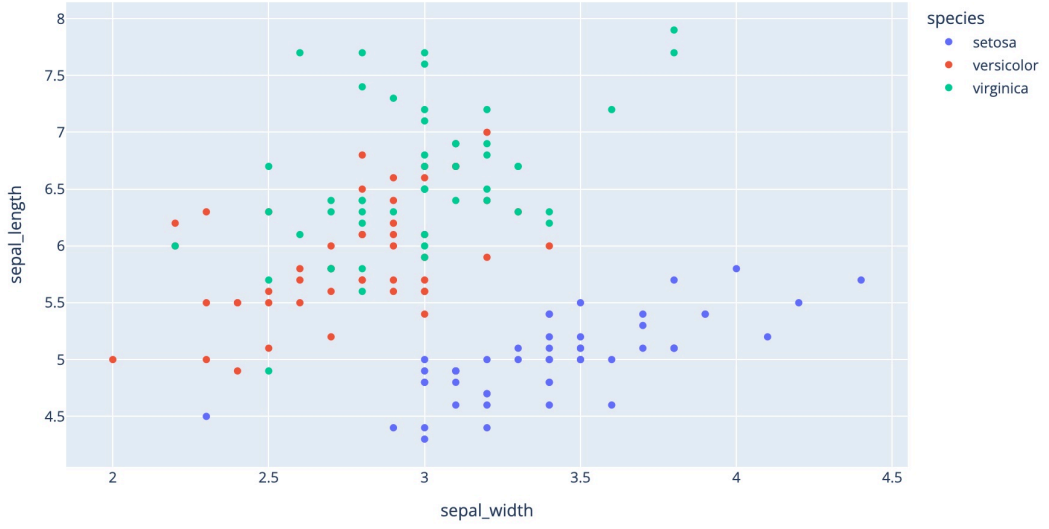
seaborn components used: `set_theme()`, `load_dataset()`, `boxplot()`, `despine()`

```
import seaborn as sns
sns.set_theme(style="ticks", palette="pastel")

# Load the example tips dataset
tips = sns.load_dataset("tips")

# Draw a nested boxplot to show bills by day and time
sns.boxplot(x="day", y="total_bill",
            hue="smoker", palette=["m", "g"],
            data=tips)
sns.despine(offset=10, trim=True)
```

seaborn Grouped boxplots

plotlyの例

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species")
fig.show()
```



plotly express

アヤメのサンプルデータを利用

```
import plotly.express as px
df = px.data.iris()
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5	3.6	1.4	0.2	setosa	1

データの概要

```
df.describe(include='all')
```

	sepal_length	sepal_width	petal_length	petal_width	species	species_id
count	150	150	150	150	150	150
unique	nan	nan	nan	nan	3	nan
top	nan	nan	nan	nan	setosa	nan
freq	nan	nan	nan	nan	50	nan
mean	5.84333	3.054	3.75867	1.19867	nan	2
std	0.828066	0.433594	1.76442	0.763161	nan	0.819232
min	4.3	2	1	0.1	nan	1
25%	5.1	2.8	1.6	0.3	nan	1
50%	5.8	3	4.35	1.3	nan	2
75%	6.4	3.3	5.1	1.8	nan	3
max	7.9	4.4	6.9	2.5	nan	3

品種ごとの数

```
df.value_counts('species')
```

```
# Output
species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

品種ごとの平均

```
df.groupby('species').mean()
```

species	sepal_length	sepal_width	petal_length	petal_width	species_id
setosa	5.006	3.418	1.464	0.244	1
versicolor	5.936	2.77	4.26	1.326	2
virginica	6.588	2.974	5.552	2.026	3

こうした計算が簡単にできるところはかなり便利

groupbyの役割

グループ分けされたDataFrameが取得できる

```
df.groupby('species').get_group('setosa').head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5	3.6	1.4	0.2	setosa	1

行ごとのループ

0.1 × sepal_length + 0.2 × petal_width

```
result = []
for i in df.index:
    result.append(0.1 * df.loc[i,'sepal_length'] + 0.2 * df.loc[i
df['result'] = result
```

applyメソッドを使おう

```
df['result'] = df.apply(lambda x: 0.1*x['sepal_length']
                                                + 0.2*x['pet
```

- lambda記法で無名関数を作る
- axisで計算する方向を指定
  - axis=0（デフォルト）関数の引数xに列が順に入る
  - axis=1 xに行が順に入る

axisはややこしいので、いろいろ試して慣れるしかない気がします

Copy-on-Write

DataFrameの一部に値を代入する場合には注意が必要

ちょっとややこしいので、順を追って説明します

詳しくは公式ドキュメントへ

いろいろな書き方がある

1行目と2行目、1列目からなる部分にアクセスする書き方（この他にもいろいろできる）

	col1	col2
idx1	1	0.1
idx2	2	0.2
idx3	-3	-0.3

```
df.loc[['idx1','idx2'], 'col1'] # 基本はこれ（行と列を同時に選択）
df.loc[['idx1', 'idx2']]['col1'] # 行を選んだ後に列を選択
df['col1'][['idx1','idx2']] # 列を選んだ後に行を選択
```

	col1
idx1	1
idx2	2

この2箇所を100に変更

行と列を同時に選択する書き方で指定する（これが基本）

```
df.loc[['idx1','idx2'], 'col1'] = 100
```

	col1	col2
idx1	100	0.1
idx2	100	0.2
idx3	-3	-0.3

ダメな例1

代入しても無視される

```
# 行を選んだあとに列を指定
df.loc[['idx1', 'idx2']]['col1'] = 100
```

	col1	col2
idx1	1	0.1
idx2	2	0.2
idx3	-3	-0.3

ダメな例2

将来はエラーになりますという警告がでるが、代入はできる

```
# 列を選んだあとに行を指定
df['col1'][['idx1','idx2']] = 100
```

```
FutureWarning: ChainedAssignmentError: behaviour will change in p
You are setting values through chained assignment. Currently this
まだまだ続く...
```

	col1	col2
idx1	100	0.1
idx2	100	0.2
idx3	-3	-0.3

どういうこと？

要素のアクセス方法にいろいろな書き方があり、そこに代入操作が加わると書き方によって挙動が違う！

なんてこった・・・

さすがにそれはどうなんだ？ということで、次期バージョン（Ver.3）から仕様が変わって、行と列に同時にアクセスする書き方以外はエラーになって代入できなくなる予定

この機能を今すぐONにできる

```
pd.options.mode.copy_on_write = True
```

公式ドキュメントでは、この機能を今すぐONにすることが推奨されています。

まとめ

- DataFrameを中心にpandasの基本を紹介
  - 基本的な使い方
  - groupby, applyメソッド
  - Copy-on-Writeに関することなど
- pandasはデータが大きくなると処理速度の遅さが気になってくる
  - これを解決するすごい方法が次のお話し