【Python】 Pandas

目次

はじめに

■はじめに

- ▶ Pandas ラ
- ▶ ※ ほとんどの変換用メソッドは仮引数 `inplace` をもち、 `True` にするとその `obj` じたいを更新する（破壊的変更を行う）。
- ▶ ※ 削減のために、 `idx` は `['r0', …]` などを、 `col` は `['c0', …]` などを指すこととする。

生成

■スカラーの生成

- ▶ 欠損値
- ▶ 今日の日付や
  現在の日時
- ▶ 文字列から日時 (日付) に

■シリーズの生成

- ▶ リストからシリに
- ▶ 辞書からシリに
- ▶ ※ 仮引数 `dtype` で型を指定できる。文字列なら `pd.StringDtype()` 。
- ▶ デフの列からシリに
- ▶ デフの行からシリに

【Python】 Pandas

目次

- はじめに
- 生成
- 書き出し
- シリーズに対する処理
- データフレームに対する処理

はじめに

■はじめに

- ▶ Pandas ラ　　$ pip install pandas　　import pandas as pd
- ▶ ※ ほとんどの変換用メソッドは仮引数 `inplace` をもち、 `True` にするとその `obj` じたいを更新する（破壊的変更を行う）。
- ▶ ※ 削減のために、 `idx` は `['r0', …]` などを、 `col` は `['c0', …]` などを指すこととする。

生成

■スカラーの生成

- ▶ 欠損値　　pd.NA　　※ `None` は避けるべき。
- ▶ 今日の日付や
  現在の日時　　from datetime import date　pd.Timestamp(date.today())
  ‥ datetime　pd.Timestamp(datetime.now())
- ▶ 文字列から日時 (日付) に　pd.Timestamp(*str*)　※様々なフォーマットに対応

■シリーズの生成

- ▶ リストからシリに　　ser = pd.Series(l, index=idx)　　※ `name=` で名前を設定可能
- ▶ 辞書からシリに　　(d = {'r0': *v0*, …} 等で)　　ser = pd.Series(l)　　※〃
- ▶ ※ 仮引数 `dtype` で型を指定できる。文字列なら `pd.StringDtype()` 。
- ▶ デフの列からシリに　　df['r2']　か　df.loc[:, 'r2']　か　df.iloc[:, 2]
- ▶ デフの行からシリに　　df.loc['r2']　　か　　df.iloc[2]

■データフレームの生成

- ※ 1列のデータフレームをつくることも可能。ただ、複数列あるデフを1列に変換するという行為をすれば往々にして、シリーズ型に還元されるので注意。
- 空のデータフレーム
- リストの辞書からデフに
- 2次元辞書からデフに
- 2次元ﾘｽﾄ,行列からデフに
- シリの辞書からデフに
- 辞,シリのﾘｽﾄからデフに
- CSVからデフに
- Excelファからデフに
- ※ 以上は仮引数 `dtype` で型を指定できる。文字列なら `pd.StringDtype()` 。
- HTML中の表たちから
  デフのリストに

書き出し

■書き出し

- CSVとして書き出し

シリーズに対する処理

■メタ情報の参照・変更

シリーズの名前について

- シリの名前を変更

行について

- ある行名が存在するか
- 行数

■データフレームの生成

- ※ 1列のデータフレームをつくることも可能。ただ、複数列あるデフを1列に変換するという行為をすれば往々にして、シリーズ型に還元されるので注意。
- 空のデータフレーム　df = pd.DataFrame()
- リストの辞書からデフに　(d = {'c0': [*v00*, ...], ...} 等で)　df = pd.DataFrame(d, index=idx)
- 2次元辞書からデフに　(d = {'c0': {'r0': *v00*, ...}, ...} 等で)　df = pd.DataFrame(d)
- 2次元ﾘｽﾄ,行列からデフに　df = pd.DataFrame(l, index=idx, columns=col)
- シリの辞書からデフに　df = pd.DataFrame(d)
- 辞,シリのﾘｽﾄからデフに　(l = [{'c0': *v00*, ...}, ...] 等で)　df = pd.DataFrame(l, index=idx)
- CSVからデフに
  (列名, 行名のないCSVから)　pd.read_csv(パス, header=None)
  (列名のみあるCSVから)　pd.read_csv(パス)
  (列名, 行名があるCSVから)　pd.read_csv(パス, index_col=0)
- Excelファからデフに　pd.read_excel(パス, sheet_name=シート名か番号※, header=‥, index_col=‥)　※番号なら `0` 始まりの整数
- ※ 以上は仮引数 `dtype` で型を指定できる。文字列なら `pd.StringDtype()` 。
- HTML中の表たちから
  デフのリストに　pd.read_html(パス※)
  ※Path型, HTML様文字列, URL, バッファも可。

書き出し

■書き出し

- CSVとして書き出し　obj.to_csv(パス, header=列名含めるか, index=行名含めるか)

シリーズに対する処理

■メタ情報の参照・変更

シリーズの名前について

- シリの名前を変更　ser.name = '名前'

行について

- ある行名が存在するか　'r2' in ser　か　'r2' in ser.index
- 行数　len(obj)　か　obj.shape[0]

- ▶ 行名を一括で変更
- ▶ 行名オブをつくる
- ▶ ※ 行名オブは基本的に「行名のないシリーズ」として考えてよく、シリーズと同じような扱い方ができる。

■要素の参照・変更

- ▶ ある値の要素があるか
- ▶ 要素（１つ）が欠損値か
- ▶ 要素 (1 つ) を参照
- ▶ 要素（複数）を参照
- ▶ 一部の要素 (1 複) を編集
- ▶ 要素を１つずつ取り出す

■一般処理

- ▶ ビューからコピーへ
- ▶ 欠損値を置換
- ▶ ある値をもつ要素を置換
- ▶ 値が他と重複しているか

■単射処理

- ▶ 文字列型に変換
- ▶ 各値が欠測値か・否か
- ▶ ある複数の値のどれかか
- ▶ 関数を適用して変換

数値処理

- ▶ 比較演算して真偽値に
- ▶ 初等算術
- ▶ 指数関数

文字列処理

- ▶ 正規表現を用いて置換
- ▶ *regexp*で始まるか

- ▶ 行名を一括で変更 obj.index = ['r0', 'r1', 'r2', ...]
- ▶ 行名オブをつくる pd.Index(イテオブ)
- ▶ ※ 行名オブは基本的に「行名のないシリーズ」として考えてよく、シリーズと同じような扱い方ができる。

■要素の参照・変更

- ▶ ある値の要素があるか x in obj.values ※: boolスカラー
- ▶ 要素（１つ）が欠損値か 要素 is pd.NA ※: boolスカラー
- ▶ 要素 (1 つ) を参照 ser['r2'] ※: スカラー (dtypeの型)
- ▶ 要素（複数）を参照 ser[['r2', ...]] ※: シリーズ
- ▶ 一部の要素 (1 複) を編集 ser[['r2':'r5']] = ser[['r2':'r5']].str.upper() ser[['r2':'r5']] = 10 等
- ▶ 要素を１つずつ取り出す for x in ser:↲ ‥

■一般処理

- ▶ ビューからコピーへ obj.copy()
- ▶ 欠損値を置換 obj.fillna(値) ※デフなら列ごとに違う値にする方法もある
- ▶ ある値をもつ要素を置換 obj.replace(*what*, *repl*) や obj.replace([*what1*, ...], *repl*)
  ※ `what` や `repl` は文字列型に限定されない！！
- ▶ 値が他と重複しているか ser.duplicated(keep=False)

■単射処理

- ▶ 文字列型に変換 obj.astype(str)
- ▶ 各値が欠測値か・否か obj.isnull() ・ obj.notnull()
- ▶ ある複数の値のどれかか obj.isin([x1, x2, ...])
- ▶ 関数を適用して変換 ser.map(関数名) ※実は辞書を渡すのもOK

数値処理

- ▶ 比較演算して真偽値に obj < 7 obj1 < obj2 (obj != 0) & ~((obj < -1) | (obj > 1))
  ※ 否定の演算子 `~` は *boolObj* 相手にしか使えない。
- ▶ 初等算術 obj - 7 や 3 * obj や obj1 / obj2 や obj += 1
- ▶ 指数関数 import numpy as np np.exp(obj)

文字列処理

- ▶ 正規表現を用いて置換 ser.str.replace(*regexp*, *replacement*, regex=True)
- ▶ *regexp*で始まるか ser.str.match(*regexp*)

- ▶ *regexp*を含むか
- ▶ (大小文字を区別せず) 〃
- ▶ *str*で始まるか
- ▶ *str*で終わるか
- ▶ 前後の空白などを削除

■日時処理

- ▶ 日時様文字列を日時型に
- ▶ ☆ 誕生日から年齢を計算

■並び替え処理

- ▶ 行名で並び替え

■行が増減する変換

行が減る変換

- ▶ 最初の５行だけ抜き出し
- ▶ 行（１複）を削除
- ▶ 行 (１・複) 無作為抽出
- ▶ いくつかの行に絞る
- ▶ 比較演算して 〃
- ▶ 行名オブを用いて 〃
- ▶ 欠損値の行を削除
- ▶ 値の重複をなくす

行が増える変換

- ▶ 上下で**連結**
- ▶ １行新規追加

■列が増える変換（データフレームに）

- ▶ 左右で**連結**
- ▶ 行名を連番に戻す

■行や列が増えるかもしれない変換（データフレームに）

- ▶ 2つのオブを連結**結合**

- ▶ *regexp*を含むか　ser.str.contains(*regexp*)　※ `ser` が欠損値含むなら `na=` も。
- ▶ (大小文字を区別せず) 〃　ser.str.contains(*regexp*, case=Flase)
- ▶ *str*で始まるか　ser.str.startswith(*str*)　※大文字小文字は必ず区別される
- ▶ *str*で終わるか　ser.str.endswith(*str*)　※ 〃
- ▶ 前後の空白などを削除　ser.str.strip()　※ `.lstrip()` `.rstrip()` もある

■日時処理

- ▶ 日時様文字列を日時型に　pd.to_datetime(ser)
- ▶ ☆ 誕生日から年齢を計算

■並び替え処理

- ▶ 行名で並び替え　obj.sort_index()　※降順なら `ascending=False` に。

■行が増減する変換

行が減る変換

- ▶ 最初の５行だけ抜き出し　obj.head()
- ▶ 行（１複）を削除　obj = obj.drop(['r2', ...]※)　※1行なら `[]` 不要
- ▶ 行 (１・複) 無作為抽出　obj = obj.sample()　・　obj = obj.sample(n=*num*)
- ▶ いくつかの行に絞る　obj[*boolSeries*※]　※行名は `obj` と同じである必要
- ▶ 比較演算して 〃　ser[ser < 7]
- ▶ 行名オブを用いて 〃　ser[行名オブ]
- ▶ 欠損値の行を削除　ser.dropna()
- ▶ 値の重複をなくす　ser.unique()

行が増える変換

- ▶ 上下で**連結**　obj = pd.concat([obj0, ...])
- ▶ １行新規追加　ser['r8'] = *v8*

■列が増える変換（データフレームに）

- ▶ 左右で**連結**　df = pd.concat([obj0, ...], axis=1)　※: **デフ**
  ※ 共通した行名のみにしたいなら `join='inner'` に。
- ▶ 行名を連番に戻す　df = obj.reset_index()　※行名だった列が列として加わる

■行や列が増えるかもしれない変換（データフレームに）

- ▶ 2つのオブを連結**結合**　pd.merge(obj0, obj1, on='キー列の名前')　※: デフ

データフレームに対する処理

■メタ情報の参照・変更

行名について

- ▶ ある行名が存在するか
- ▶ 行数
- ▶ 行名オブをつくる
- ▶ ※ 行名オブは基本的に「行名のないシリーズ」として考えてよく、シリーズと同じような扱い方ができる。

列名について

- ▶ ある列名が存在するか
- ▶ 列数
- ▶ ある列の名前を変更
- ▶ 列名を一括で変更

■要素の参照・変更

- ▶ ある値の要素があるか
- ▶ 要素（１つ）が欠損値か
- ▶ 要素 (1 つ) を参照
- ▶ 一部の要素 (1 複) を編集
- ▶ 列（１つ）を参照
- ▶ 列（複数）を参照
- ▶ 行（１つ）を参照
- ▶ 行（複数）を参照
- ▶ １列ずつ処理
- ▶ １行ずつ処理

■一般処理

- ▶ ビューからコピーへ

データフレームに対する処理

■メタ情報の参照・変更

行名について

- ▶ ある行名が存在するか　'r2' in df.index
- ▶ 行数　len(obj)　か　obj.shape[0]
- ▶ 行名オブをつくる　pd.Index(イテオブ)
- ▶ ※ 行名オブは基本的に「行名のないシリーズ」として考えてよく、シリーズと同じような扱い方ができる。

列名について

- ▶ ある列名が存在するか　'c2' in df　か　'c2' in df.columns
- ▶ 列数　len(df.columns)　か　df.shape[1]
- ▶ ある列の名前を変更　df = df.rename(columns={'c2': *new_name_c2*, ...})
- ▶ 列名を一括で変更　df.columns = col　や　df.rename(columns=str.upper)

■要素の参照・変更

- ▶ ある値の要素があるか　x in obj.values　※: boolスカラー
- ▶ 要素（１つ）が欠損値か　要素 is pd.NA　※: boolスカラー
- ▶ 要素 (1 つ) を参照　df.at['r2', 'c3']　か　df.iat[2, 3]　※: 〃
- ▶ 一部の要素 (1 複) を編集　df['c2'] = ser['c2'].str.upper()　df['c2'] =　= 10　等
- ▶ 列（１つ）を参照　df['c2']　や　df.loc[:, 'c2']　や　df.iloc[:, 2]　※: シリ
- ▶ 列（複数）を参照　df[['c2', ...]]　や　df.loc[:, ['c2', 'c5']]　や　df.iloc[:, [2, 5]]
  ※スライスも可能。ただ `2:7` は `7` も含むことに注意
- ▶ 行（１つ）を参照　df.loc['r2']　か　df.iloc[2]　※: シリ
- ▶ 行（複数）を参照　df.loc[['r2', ...]]　や　df.iloc[[2, ...]]　※スライスも可能
- ▶ １列ずつ処理　for column_name, column_ser in df.items():↲　‥
- ▶ １行ずつ処理　for index, row_ser in df.iterrows():↲　‥　※**コピーっぽい**！

■一般処理

- ▶ ビューからコピーへ　obj.copy()

■単写処理

- ▶ 文字列型に変換
- ▶ 各値が欠測値か・否か
- ▶ ある複数の値のどれかか

数値処理

- ▶ 比較演算して真偽値に
- ▶ 初等算術
- ▶ 指数関数

■並び替え処理

- ▶ 列 (1 複) の値で並び替え
- ▶ 列名で並び替え

■列が増減する変換

列が減る変換

- ▶ 列 (1 ・複) 無作為抽出
- ▶ 列 （ 1 複）を削除
- ▶ 欠損値を含む列を削除
- ▶ 列 （ 1 複）を行名に抜擢

列が減る変換（シリーズに）

- ▶ 行中の全ての値が他の行と重複しているか
- ▶ ☆ 行中のいずれかの値が他の行と重複しているか
- ▶ 行中の全ての値が真に評価できるか
- ▶ 行中のいずれかの値が真に評価できるか

列が増える変換

- ▶ 左右で**連結**
- ▶ 空の 1 列を新規追加
- ▶ 1 **列**新規追加

■行が増減する変換

■単写処理

- ▶ 文字列型に変換　obj.astype(str)
- ▶ 各値が欠測値か・否か　obj.isnull()　・　obj.notnull()
- ▶ ある複数の値のどれかか　obj.isin([x1, x2, ...])

数値処理

- ▶ 比較演算して真偽値に　obj < 7　obj1 < obj2　(obj != 0) & ~((obj < -1) | (obj > 1))
  ※ 否定の演算子 `~` は *boolObj* 相手にしか使えない。
- ▶ 初等算術　obj - 7　や　3 * obj　や　obj1 / obj2　や　obj += 1
- ▶ 指数関数　import numpy as np　　np.exp(obj)

■並び替え処理

- ▶ 列 (1 複) の値で並び替え　df.sort_values(by=['c2', ...]※)　　※1列なら `[]` 不要
- ▶ 列名で並び替え　df.sort_index(axis=1)　　※降順なら `ascending=False` に。

■列が増減する変換

列が減る変換

- ▶ 列 (1 ・複) 無作為抽出　df = df.sample(axis=1)　・　df = df.sample(axis=1, n=*num*)
- ▶ 列 （ 1 複）を削除　df = df.drop(['c2', ...]※, axis=1)　※1列なら `[]` 不要
- ▶ 欠損値を含む列を削除　df = df.dropna(axis=1)
- ▶ 列 （ 1 複）を行名に抜擢　df = df.set_index(['c2', ...]※)　　※1列なら `[]` 不要

列が減る変換（シリーズに）

- ▶ 行中の全ての値が他の行と重複しているか　df.duplicated(keep=False)
- ▶ ☆ 行中のいずれかの値が他の行と重複しているか
- ▶ 行中の全ての値が真に評価できるか　df.all(axis=1)
- ▶ 行中のいずれかの値が真に評価できるか　df.any(axis=1)

列が増える変換

- ▶ 左右で**連結**　df = pd.concat([obj0, ...], axis=1)　　※: **デフ**
  ※ 共通した行名のみにしたいなら `join='inner'` に。
- ▶ 空の 1 列を新規追加　df['c5'] = pd.NA
- ▶ 1 **列**新規追加　df['c5'] = [*v05*, *v15*, ...]　　や　　df['c5'] = ser

■行が増減する変換

行が減る変換

- ▶ 行（１複）を削除
- ▶ いくつかの行に絞る
- ▶ 列 (１複) で比較演算し〃
- ▶ 行名オブを用いて〃
- ▶ クエリ文を用いて〃
- ▶ 欠損値を含む行を削除
- ▶ 全てが欠損値の行を削除
- ▶ 欠損値を含む行のみに
- ▶ 列 (１複) で重複をなくす
- ▶ 全値が他と重複している行のみに

行が増える変換

- ▶ 上下で**連結**
- ▶ 空の１行を新規追加
- ▶ １**行**新規追加

■行や列が増えるかもしれない変換

- ▶ 転置

行が減る変換

- ▶ 行（１複）を削除　obj = obj.drop(['r2', ...]※)　※1行なら `[]` 不要
- ▶ いくつかの行に絞る　obj[*boolSeries*※]　※行名は `obj` と同じである必要
- ▶ 列 (１複) で比較演算し〃　df[df['c2'] < 7]　※: **デフ**（ `.iloc[0]` でシリにできる）
- ▶ 行名オブを用いて〃　df.loc[行名オブ]
- ▶ クエリ文を用いて〃　df.query('*query*')
- ▶ 欠損値を含む行を削除　df.dropna()　※ `n` 個以上の時に限るなら `thresh=n` を渡す
- ▶ 全てが欠損値の行を削除　df.dropna(how='all')
- ▶ 欠損値を含む行のみに　df[df.isnull().any(axis=1)]
- ▶ 列 (１複) で重複をなくす　df.drop_duplicates(['c2', ...])
- ▶ 全値が他と重複している行のみに　df[df.duplicated(keep=False)]

行が増える変換

- ▶ 上下で**連結**　obj = pd.concat([obj0, ...])
- ▶ 空の１行を新規追加　df.loc['r5'] = pd.NA
- ▶ １**行**新規追加　df.loc['r5'] = [*v50*, *v51*, ...]　や　df.loc['r5'] = ser

■行や列が増えるかもしれない変換

- ▶ 転置　df.T　※~~ビュー~~コピー（NumPyとは違う）