Manipulação de Dados com Pandas

Linguagem de Programação Aplicada

Prof. Alex Kutzke

17 de abril de 2021

Pandas

O que é Pandas

- Pacote que provê ferramentas de alta performance e de fácil aplicação para trabalho com estruturas de dados e análise de dados em Python:

```
import pandas as pd
pd?
```

Importando

- Convenção: importar com alias `pd`:

```
import pandas as pd
```

- Os elementos `Series` e `DataFrame` são comumente importados ao espaço de nomes do próprio programa (por serem de uso comum):

```
from pandas import Series, DataFrame
```

Introdução às Estruturas de Dados do Pandas

Series

- É uma estrutura composta por um array unidimensional de *valores* e outro, de mesmo tamanho, de *índices*;

```
obj = pd.Series([4, 7, -5, 3])
obj
```

```
0    4
1    7
2   -5
3    3
dtype: int64
```

Valores e índices

```
obj.values
obj.index  # like range(4)
```

- É possível estipular os valores dos índices:

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
obj2.index
```

```
Index(['d', 'b', 'a', 'c'], dtype='object')
```

Valores e índices

```
obj2['a']
obj2['d'] = 6
obj2[['c', 'a', 'd']]
```

```
c    3
a   -5
d    6
dtype: int64
```

Valores e índices

- Muitas das operações dos ndarrays da NumPy são permitidas:

```
import numpy as np

obj2[obj2 > 0]
obj2 * 2
np.exp(obj2)
```

```
d     403.428793
b    1096.633158
a       0.006738
c      20.085537
dtype: float64
```

Valores e índices

- Pertinência por índice:

```
'b' in obj2
'e' in obj2
```

Inicialização por dicionário

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3 = pd.Series(sdata)
```

```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

Alteração de Índices

```
states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states)
obj4
```

```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

- Novos valores são adicionados como `NaN`.

Null ou Não Null

```
pd.isnull(obj4)
pd.notnull(obj4)
```

```
California    False
Ohio           True
Oregon         True
Texas          True
dtype: bool
```

```
obj4.isnull()
```

```
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

Alinhamento de índices

- As estruturas do Pandas conseguem perceber (e alinhar) índices iguais em objetos diferentes:
  - Isso é levado em conta, automaticamente, nas mais diversas operações:

```
obj3 + obj4
```

```
California         NaN
Ohio           70000.0
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

Atribuição de nomes

- Para facilitar a leitura, é possível adicionar nomes para o objeto e para os índices:

```
obj4.name = 'population'
obj4.index.name = 'state'
```

```
state
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
Name: population, dtype: float64
```

Alteração de índices

```
obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
obj
```

```
Bob      4
Steve    7
Jeff    -5
Ryan     3
dtype: int64
```

DataFrame

- Pense em uma lista de `Series`;

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
```

```
    state  year  pop
0    Ohio  2000  1.5
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
5  Nevada  2003  3.2
```

```
frame.head()
```

Mais sobre o DataFrame

- Especificando a ordenação das colunas:

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

```
frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
                      index=['one', 'two', 'three', 'four',
                             'five', 'six'])
frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

Acessando colunas

```
frame2['state']
frame2.year
```

```
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year, dtype: int64
```

Acessando linhas

```
frame2.loc['three']
```

```
year     2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

Alterando valores

- Similar ao NumPy:

```
frame2['debt'] = 16.5
frame2['debt'] = np.arange(6.)
```

```
val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
frame2['debt'] = val
```

```
frame2['eastern'] = frame2.state == 'Ohio'
```

Nomeando dados

```
pop = {'Nevada': {2001: 2.4, 2002: 2.9},
       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
frame3 = pd.DataFrame(pop)
```

```
frame3.index.name = 'year'; frame3.columns.name = 'state'
```

Reindexing

```
obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
```

```
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64
```

```
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
```

```
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

Reindexing

```
frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
                     index=['a', 'c', 'd'],
                     columns=['Ohio', 'Texas', 'California'])
frame2 = frame.reindex(['a', 'b', 'c', 'd'])
```

Apagando entradas

```
obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
new_obj = obj.drop('c')
obj.drop(['d', 'c'])
```

```
a    0.0
b    1.0
e    4.0
dtype: float64
```

```
obj.drop('c', inplace=True)
obj
```

```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

Indexação, Seleção e Filtros

Indexação

- Bastante similar ao NumPy

```
obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
obj['b']
obj[1]
obj[2:4]
obj[['b', 'a', 'd']]
obj[[1, 3]]
obj[obj < 2]
```

Exemplos de indexação com DataFrame

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
data[data['three'] > 5]
```

```
data < 5
data[data < 5] = 0
```

Exemplos loc e iloc

```
data.iloc[2, [3, 0, 1]]
data.iloc[2]
data.iloc[[1, 2], [3, 0, 1]]
```

```
data.loc[:'Utah', 'two']
data.iloc[:, :3][data.three > 5]
```

Aritmética e Alinhamento de Dados

Aritmética

```
s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
               index=['a', 'c', 'e', 'f', 'g'])
s1 + s2
```

```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

Aritmética com DataFrame

- Considere:

```
df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),
                   index=['Ohio', 'Texas', 'Colorado'])
df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),
                   index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

Aritmética com DataFrame

```
df1 + df2
```

```
                b   c     d   e
        Colorado  NaN NaN   NaN NaN
```

```
        Ohio      3.0 NaN   6.0 NaN
        Oregon    NaN NaN   NaN NaN
        Texas     9.0 NaN  12.0 NaN
        Utah      NaN NaN   NaN NaN
```

Valores de preenchimento

```
df1.add(df2, fill_value=0)
```

Funções aritméticas com operadores invertidos

- Funções aritméticas podem começar com `r`, indicando que a ordem dos operadores deve ser invertida;
- Exemplo, as operações abaixo são são idênticas:

```
1 / df1
df1.rdiv(1)
```

Operações entre Series e DataFrames

```
frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                     columns=list('bde'),
                     index=['Utah', 'Ohio', 'Texas', 'Oregon'])
series = frame.iloc[0]
```

```
frame - series
```

Aplicação de Funções e Mapeamento

```
frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),
                     index=['Utah', 'Ohio', 'Texas', 'Oregon'])
frame
np.abs(frame)
```

- Entre outras tantas funções.

Aplicação de Funções e Mapeamento

- Aplicar uma função sobre as colunas;

```
f = lambda x: x.max() - x.min()
frame.apply(f)
```

- Ou por linhas:

```
frame.apply(f, axis='columns')
```

- Ou, ainda, por elemento:

```
format = lambda x: '%.2f' % x
frame.applymap(format)
```

Sorting

- Ordenação “por linhas”:

```
frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
                     index=['three', 'one'],
                     columns=['d', 'a', 'b', 'c'])
frame.sort_index()
frame.sort_index(axis=1)
frame.sort_index(axis=1, ascending=False)
```

- Ordenação “por colunas”:

```
frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})
frame.sort_values(by='b')
frame.sort_values(by=['a', 'b'])
```

Exemplo: MovieLens 1M Dataset

Importando dados

- Pandas possui vários métodos para leitura e importação de dados nos mais diferentes formatos:

```
import pandas as pd
# Make display smaller
pd.options.display.max_rows = 10

unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('datasets/movielens/users.dat', sep='::',
                      header=None, names=unames)

rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('datasets/movielens/ratings.dat', sep='::',
                        header=None, names=rnames)

mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
                       header=None, names=mnames)
```

Unindo dados

- Algo como um join;
- Aqui, o alinhamento automático dos dados faz todo o trabalho “sujo”:

```
data = pd.merge(pd.merge(ratings, users), movies)
```

Extraindo novas conclusões

- Criação de uma tabela “pivô”, contendo a média de avaliação para todos os filmes, separado por sexo do avaliador:

```
mean_ratings = data.pivot_table('rating', index='title',
                                columns='gender', aggfunc='mean')
```

Filmes com mais de 250 classificações

- Agrupa avaliações por filme (*title*);
- Seleciona aqueles que receberam mais de 250 avaliações:

```
ratings_by_title = data.groupby('title').size()
ratings_by_title[:10]
active_titles = ratings_by_title.index[ratings_by_title >= 250]
active_titles
```

Filmes com mais de 250 classificações

```
# Select rows on the index
mean_ratings = mean_ratings.loc[active_titles]
```

- E os filmes melhor avaliados pela mulheres:

```
top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)
top_female_ratings[:10]
```

Avaliando discrepâncias nas avaliações

- Insere nova coluna (diff) com a diferença entre as avaliações de homens e mulheres:

```
mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
```

- Ordena pela nova coluna.

```
sorted_by_diff = mean_ratings.sort_values(by='diff')
sorted_by_diff[:10]
```

Avaliando discrepâncias nas avaliações

```
sorted_by_diff[::-1][:10]
```

- Qual a interpretação do comando acima?

Outras métricas:

```
# Standard deviation of rating grouped by title
rating_std_by_title = data.groupby('title')['rating'].std()
# Filter down to active_titles
rating_std_by_title = rating_std_by_title.loc[active_titles]
# Order Series by value in descending order
rating_std_by_title.sort_values(ascending=False)[:10]
```

Exercícios

Trabalhe com o Pandas sobre a base de dados MovieLens e retire novas informações. Por exemplo: 1. Os 10 filmes melhor classificados (com mais de 300 avaliações); 2. Os 10 filmes pior classificados (com mais de 300 avaliações); 3. Quantidade de filmes por gênero (considerar apenas o primeiro gênero de cada filme); 4. Os 10 filmes melhor classificados no gênero "comedy".

Referências

- McKinney, Wes - Python para Análise de Dados: Tratamento de Dados com Pandas, Numpy e IPython, Editora Novatec, 1a Edição, 2019;