Programação II
+
Estruturas de Dados para Bioinformática

Análise de dados tabulares (pandas)
Parte 1

Hugo Pacheco

DCC/FCUP
22/23

NumPy

- Desenhado para programação numérica sobre arrays
- Otimizado para álgebra de arrays, operações elemento-a-elemento
- Tem algumas limitações para dados menos estruturados:
  - Não permite atribuir nomes a linhas/colunas (e.g., cabeçalhos CSV)
  - Não suporta elementos de tipos diferentes (inteiros, datas, strings, etc)
  - Pouco suporte para transformar a estrutura dos arrays
  - Pouco suporte para lidar com dados mal formatados ou em falta

Arrays ⇝ Tabelas

- Tabelas ≃ folhas de cálculo ≃ bases de dados

Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78

- Colunas potencialmente de diferentes tipos

```
name : str, age : int, gender : str, rating : float
```

Pandas

- Biblioteca construída em cima do *numpy*
- Generalização de arrays para tabelas:
  - Permite atribuir nomes a linhas/colunas
  - Permite diferentes tipos por coluna
  - Oferece operações de manipulação e transformação de tabelas
  - Vasto suporte para lidar com dados em falta

Pandas

- Dois tipos principais:

  - *Series*: um vetor de elementos do mesmo tipo, ao estilo de um array 1D *numpy*, com índices ([0,1,…[se não definidos)

A	B	C	D	A
10	50	23	70	34

  - *DataFrame*: uma tabela com colunas de tipos diferentes; primeira coluna define índices; ([0,1,…[se não definidos)

  - nomes de colunas únicos; nomes de índices podem ser repetidos

Índice	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Steve	45	Male	3.9
Katie	38	Female	2.78

Pandas (criação)

- Uma *Series* pode ser criada a partir de vários tipos de sequências (listas, dicionários, array *numpy*)

```
>>> s = pd.Series((1,2,3))
>>> s
0    1
1    2
2    3
dtype: int64
>>> type(s.values)
<class 'numpy.ndarray'>
>>> s.index
RangeIndex(start=0, stop=3, step=1)

>>> s =
pd.Series([1,2,3],index=[1,2,5])
>>> type(s.values)
<class 'numpy.ndarray'>
>>> s.index
Int64Index([1, 2, 5],
dtype='int64')
```

```
>>> s = pd.Series({'a':1,'b':2})
>>> s
a    1
b    2
dtype: int64
>>> s.index
Index(['a', 'b'],
dtype='object')
```

Pandas (criação)

- Um *DataFrame* pode ser criado a partir de vários tipos de sequências (listas, dicionários, array *numpy*, *Series*, *DataFrame*)

```
>>> import pandas as pd
>>> pd.DataFrame([1,2,3,4])
   0
0  1
1  2
2  3
3  4
>>> pd.DataFrame([[1,2],[3,4],
[5,6]],index=list("abc"),colum
ns=list("ab"))
   a  b
a  1  2
b  3  4
c  5  6
```

```
>>> pd.DataFrame({'name':
['steve','lia'],'age':
[25,38],'sex':
['Male','Female']})
    name  age     sex
0  steve   25    Male
1    lia   38  Female
>>>
pd.DataFrame([{'a':1,'b':
2},
{'a':5,'b':10,'c':20}])
   a   b     c
0  1   2   NaN
1  5  10  20.0
```

NumPy $\leq$ *Pandas*

- Projeção e slices de *Series* funcionam tal como para arrays 1D *numpy*

```
>>> s =
pd.Series(range(5),index
=list("abcde"))
>>> s
a    0
b    1
c    2
d    3
e    4
dtype: int64

>>> s['a']
0
```

```
>>> s['b':'d'] = [10,20,30]
>>> s
a     0
b    10
c    20
d    30
e     4

>>> s[s > 2]
d    3
e    4
dtype: int64
```

NumPy $\leq$ *Pandas*

- Projeção e slices de *DataFrame* funcionam quase como para arrays 2D *numpy*; 1ª dimensão para colunas, 2ª dimensão ou *.loc* para linhas

```
>>> df = pd.DataFrame([[1,2],
[3,4]],index=list("ab"),columns
=list("cd"))
>>> df
   c  d
a  1  2
b  3  4

>>> df['c']
a    1
b    3
>>> df['c']['a'] = 10
>>> df
    c  d
a  10  2
b   3  4
```

```
>>> df.loc['b']
c    3
d    4
Name: b, dtype: int64

>>> df.loc['a'] = [5,6]
>>> df
   c  d
a  5  6
b  3  4

>>> df[:].loc['b'] = [0,0]
>>> df
   c  d
a  5  6
b  0  0
```

NumPy $\leq$ *Pandas*

- Operações matemáticas elemento-a-elemento sobre *Series* funcionam tal como arrays 1D *numpy*

```
>>> s = pd.Series({'a':1,'b':2,'c':3})
>>> s
a    1
b    2
c    3
dtype: int64
>>> s + 2
a    3
b    4
c    5
dtype: int64
>>> s + s
a    2
b    4
c    6
dtype: int64
>>> np.sum(s)
6
```

NumPy $\leq$ *Pandas*

- Operações matemáticas elemento-a-elemento sobre *DataFrame* funcionam quase como arrays 2D *numpy*

```
>>> df = pd.DataFrame([(1,2),
(3,4)],index=list("ab"),columns=list("c
d"))
>>> df
   c  d
a  1  2
b  3  4
>>> (df + df) / 3
          c         d
a  0.666667  1.333333
b  2.000000  2.666667
>>> df.max()
c    3
d    4
dtype: int64
>>> df.max().max()
4
```

NumPy $\leq$ *Pandas*

- Funções universais *numpy* podem ser aplicadas a todos os elementos de uma *Series* ou um *DataFrame*

```
>>> s =
pd.Series(range(1,3))
>>> s
0    1
1    2
dtype: int64
>>> np.power(s,2)
0    1
1    4
dtype: int64
```

```
>>> df = pd.DataFrame([[1,2],
[4,5]], columns=['A','B'])
>>> df
   A  B
0  1  2
1  4  5
>>> np.sin(df * np.pi / 4)
                A             B
0  7.071068e-01  1.000000
1  1.224647e-16 -0.707107
```

Pandas (mapeamento)

- Podemos também aplicar uma função:
  - por colunhas
  - por linhas

```
>>> df = pd.DataFrame([[1,2],
[4,5]], columns=['A','B'])
>>> df
   A  B
0  1  2
1  4  5
f = lambda x: x.max() -
x.min()

>>> df.apply(f,axis=0)
A    3
B    3
dtype: int64

>>> df.apply(f,axis=1)
0    1
1    1
dtype: int64
```

Pandas (colunas)

- Podem-se renomear as colunas de um *DataFrame*

```
>>> df = pd.DataFrame([[1,2],[4,5]],
columns=['A','B'])
>>> df
   A  B
0  1  2
1  4  5
>>> df.columns=[True,False]
>>> df
   True   False
0      1      2
1      4      5
```

Pandas (colunas)

- Podem-se renomear as colunas de um *DataFrame* aplicando uma função ou um mapeamento (dicionário) às colunas existentes, criando um novo objeto

```
>>> df = pd.DataFrame([[1,2],[4,5]],
columns=['A','B'])

>>> df = df.rename(columns={'A':'a'})
   a  B
0  1  2
1  4  5
>>> df = df.rename(columns=lambda s:s.lower())
   a  b
0  1  2
1  4  5
```

Pandas (índices)

- Podem-se reordenar os índices de uma *Series* ou de um *DataFrame*, criando um novo objeto

```
>>> s = pd.Series((1,2,3))
>>> s
0    1
1    2
2    3
dtype: int64
>>> s = s.reindex((2,2,1,3))
>>> s
2    3
2    3
1    2
3    NaN
dtype: int64
```

```
>>> df =
pd.DataFrame([{'a':1,'b'
:2},{'a':3,'b':4}])
>>> df
   a  b
0  1  2
1  3  4
>>> df =
df.reindex([1,0])
>>> df
   a  b
1  3  4
0  1  2
```

Pandas (índices)

- Podem-se alterar os índices de uma *Series* ou de um *DataFrame* aplicando uma função ou um mapeamento (dicionário) aos índices existentes, criando um novo objeto

```
>>> s = pd.Series((1,2,3))
>>> s
0    1
1    2
2    3
dtype: int64
>>> s = s.rename(lambda
i : i+1)
>>> s
1    1
2    2
3    3
dtype: int64
```

```
>>> df =
pd.DataFrame([{'a':1,'b'
:2},{'a':3,'b':4}])
>>> df
   a  b
0  1  2
1  3  4
>>> df =
df.rename({0:'a',1:'b'})
>>> df
   a  b
a  1  2
b  3  4
```

Pandas (índices)

- Podem-se converter todos os índices em colunas

```
>>> s =
pd.Series((1,2,3),index=
list("abc"))
>>> s
a    1
b    2
c    3
dtype: int64
>>> s.reset_index()
  index  0
0     a  1
1     b  2
2     c  3
```

```
>>> df =
pd.DataFrame([{'a':1,'b':2}
,
{'a':3,'b':4}],index=list("
cd"))
>>> df
   a  b
c  1  2
d  3  4
>>> df.reset_index()
  index  a  b
0     c  1  2
1     d  3  4
```

Pandas (índices)

- Podem-se converter colunas de um *DataFrame* em índices

```
>>> s =
pd.Series((1,2,3))
>>> s
0    1
1    2
2    3
>>> s.index =
list("abc")
>>> s
a    1
b    2
c    3
dtype: int64
```

```
>>> df =
pd.DataFrame([{'a':1,'b':2}
,{'a':3,'b':4}])
>>> df
   a  b
0  1  2
1  3  4
>>>
df.set_index(keys='a',inpla
ce=True)
>>> df
   b
a
1  2
3  4
```

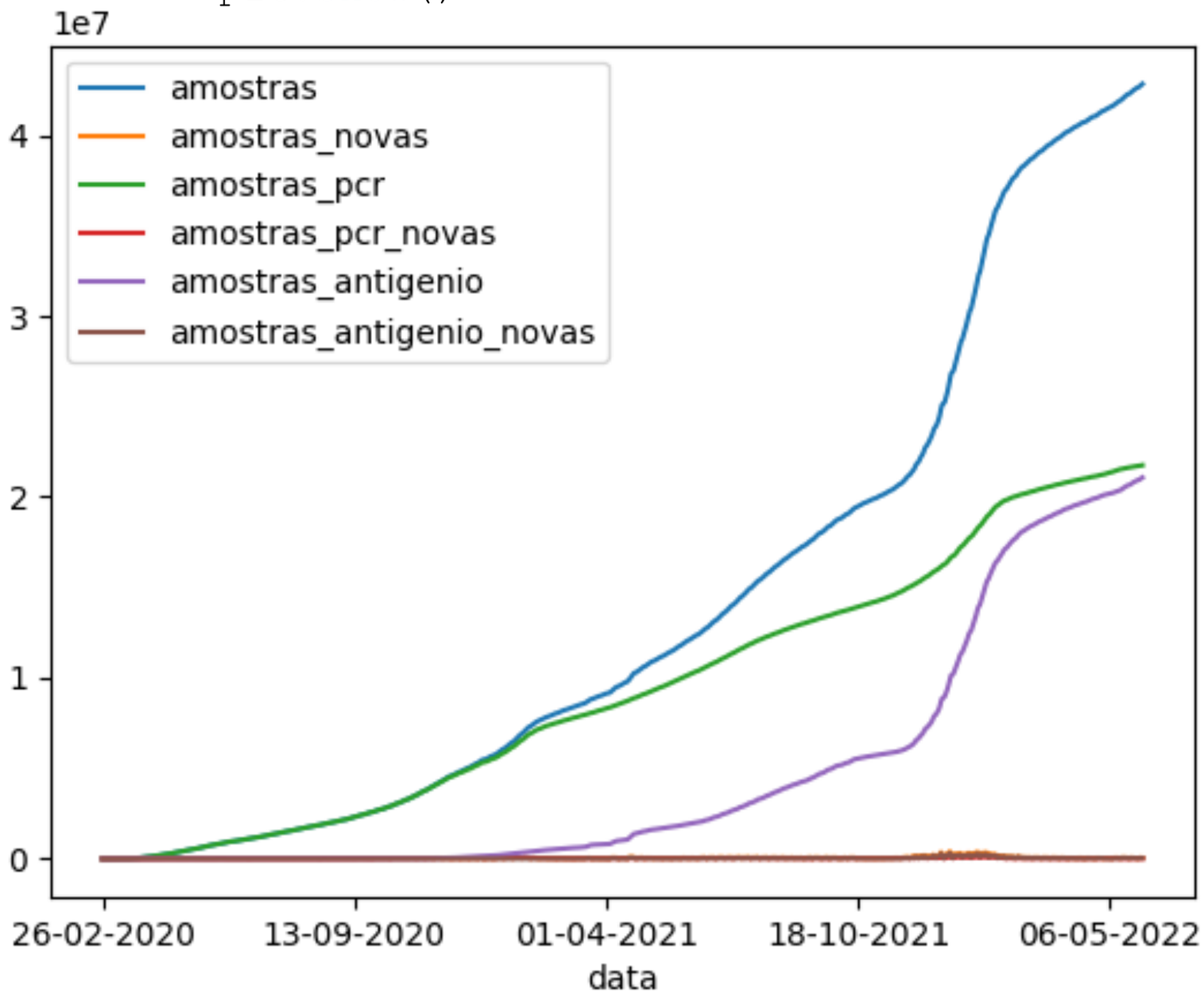
Exemplo *Pandas*

- Ler ficheiro CSV com o número de testes COVID-19 diários, disponível aqui

```
>>> import pandas as pd
>>> amostras = pd.read_csv('amostras.csv',index_col='data')
>>> amostras.fillna(0,inplace=True)
>>> amostras
            amostras  ...  amostras_antigenio_novas
data                  ...
26-02-2020       0.0  ...                       0.0
27-02-2020       0.0  ...                       0.0
28-02-2020       0.0  ...                       0.0
29-02-2020       0.0  ...                       0.0
01-03-2020      25.0  ...                       0.0
...              ...  ...                       ...
```

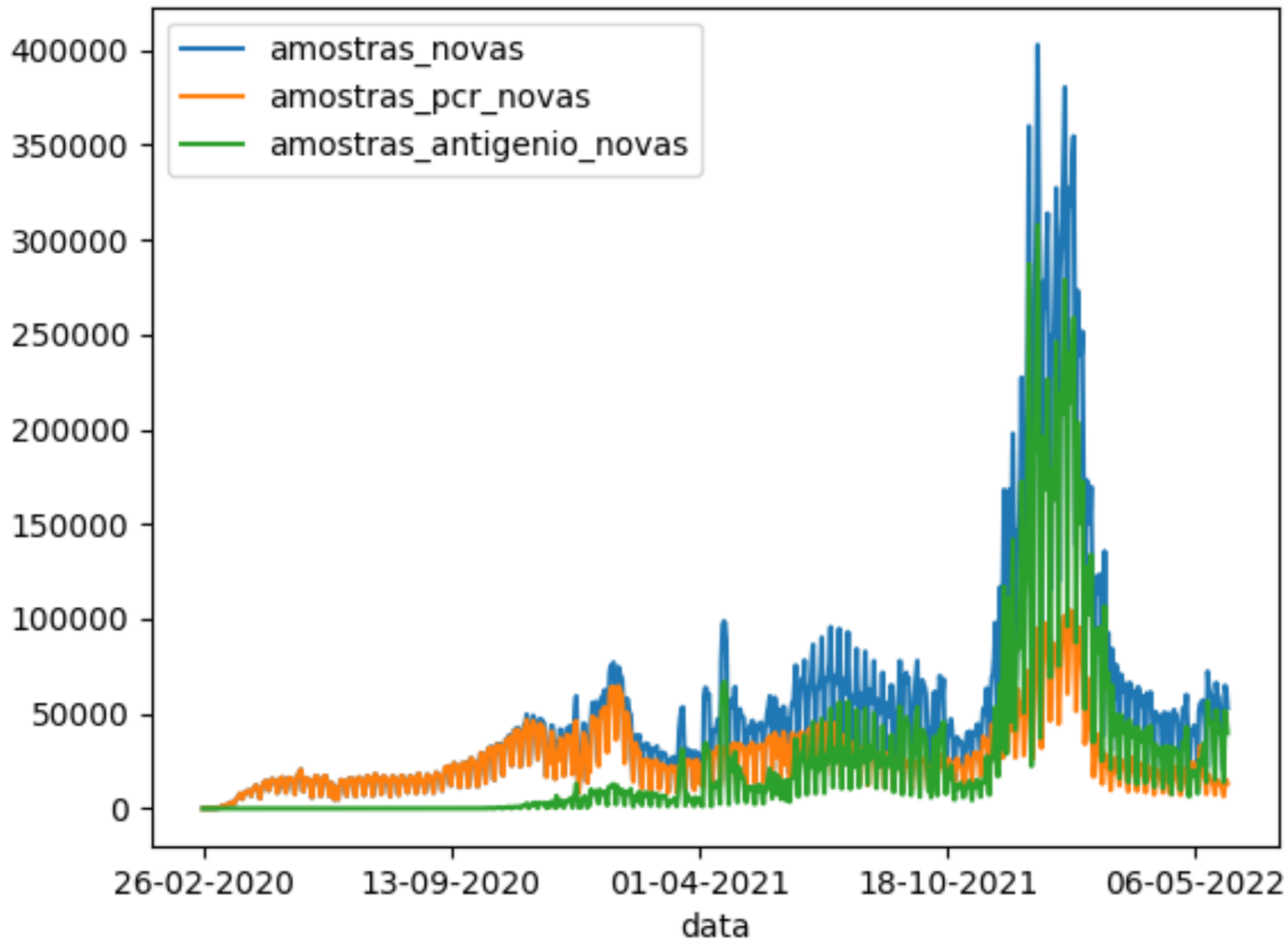
Exemplo *Pandas*

```
import matplotlib.pyplot as plt
amostras.plot()
plt.show()
```



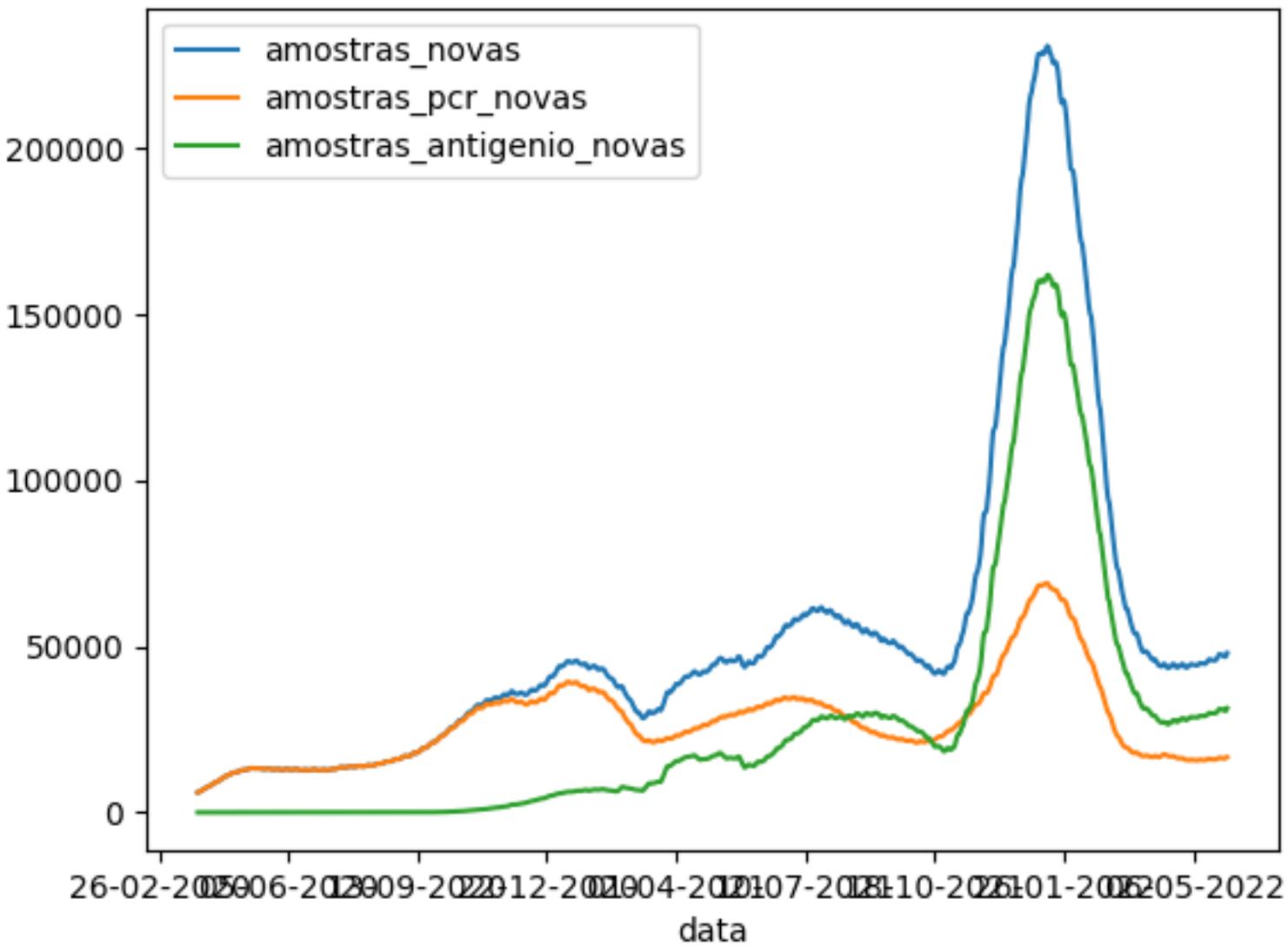
Exemplo *Pandas*

```
novas_colunas = [c for c in amostras.columns if c.endswith('_novas')]
novas_amostras = amostras[novas_colunas].copy()
novas_amostras.plot()
plt.show()
```



Exemplo *Pandas*

```
smooth = novas_amostras.copy()
for i in range(len(smooth)) :
    smooth.iloc[i] = novas_amostras.iloc[i-30:i+30].mean()
smooth.plot()
plt.show()
```



Exemplo *Pandas*

- Algumas estatísticas

```
#1º dia antigénio
>>> amostras[amostras['amostras_antigenio'] > 0].index[0]

# dias e valores dos máximos de novas amostras por categoria
>>> max_datas = novas_amostras.idxmax()
>>> { k : (data,novas_amostras[k][data]) for k,data in
dict(max_datas).items() }

# dias de Dezembro de 2021 acima da média de novas amostras de 2021
>>> novas = novas_amostras.rename(lambda str :
pd.to_datetime(str,infer_datetime_format=True))
>>> novas2021 = novas [ novas.index.year == 2021 ]
>>> means2021 = novas2021.mean()

>>> above_mean2021 = (novas2021['amostras_novas'] >=
means2021['amostras_novas'])
>>> dec_above_mean2021 = novas2021[above_mean2021 &
(novas2021.index.month == 12)]
```