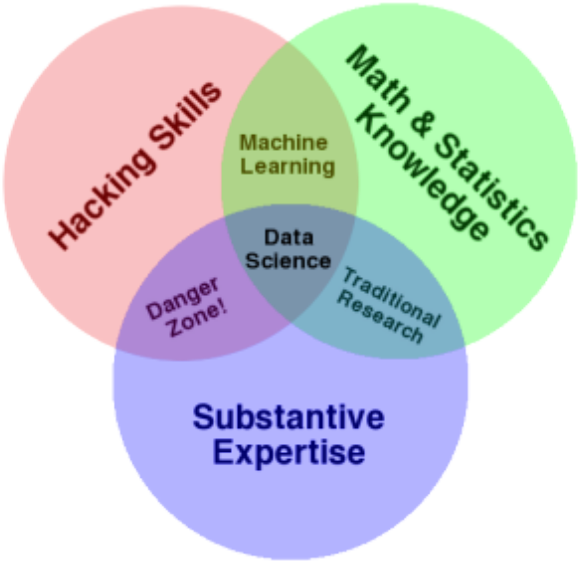
Usando El Tipo Series de Python Pandas

En estas notas se realizan pruebas con la estructura de datos "Series".



Inicialización con el tipo "Series"

Un objeto "Series" es un vector con datos indexados. Lo que sigue son algunas formas de inicialización.

In [1]:

```
#importacion estandar de pandas
import pandas as pd

edad = pd.Series([10, 20, 14, 11])
edad
```

Out[1]:

```
0    10
1    20
```

```
2    14
3    11
dtype: int64
```

Si no se especifican indices, se asigna una secuencia de indices por defecto. Ahora asignando indices.

In [2]:

```
bacteria = pd.Series([10, 20, 14, 11],
                     index=['a', 'b', 'c', 'd'])
bacteria
```

Out[2]:

```
a    10
b    20
c    14
d    11
dtype: int64
```

A partir de un diccionario

In [3]:

```
bacteria_dict = {'a': 10, 'b': 20, 'c': 14, 'd': 11}
pd.Series(bacteria_dict)
```

Out[3]:

```
a    10
b    20
c    14
d    11
dtype: int64
```

Una "Serie" contiene los datos (*values*) en un objeto de tipo "NumPy array"

In [4]:

```
# Datos
bacteria.values
```

Out[4]:

```
array([10, 20, 14, 11], dtype=int64)
```

Y los indices en un objeto "Index" de pandas.

In [5]:

```
# Indices
bacteria.index
```

Out[5]:

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

Seleccionando Datos

Seleccionando datos segun sus posiciones

In [6]:

```
bacteria['b']
```

Out[6]:

```
20
```

In [7]:

```
bacteria[1]
```

Out[7]:

```
20
```

In [8]:

```
bacteria[['a','b']]
```

Out[8]:

```
a    10
b    20
dtype: int64
```

Seleccionando con condiciones

In [9]:

```
bacteria[(bacteria > 10) & \
         (bacteria < 20)]
```

Out[9]:

```
c    14
d    11
dtype: int64
```

In [10]:

```
bacteria
```

Out[10]:

```
a    10
b    20
c    14
d    11
dtype: int64
```

Operaciones Basicas

In [11]:

```
bacteria = bacteria + 5
print bacteria
```

```
a    15
b    25
c    19
d    16
dtype: int64
```

In [12]:

```
bacteria = bacteria + bacteria
print bacteria
```

```
a    30
b    50
c    38
d    32
dtype: int64
```

In [13]:

```
bacteria = bacteria > 30
print bacteria
```

```
a    False
b     True
c     True
```

```
d     True
dtype: bool
```

Luego se puede verificar si alguno o todos son True

In [14]:

```
# retorna True si algun elemento es True
print bacteria.any()
# retorna True si todos los elementos son True
print bacteria.all()
```

```
True
False
```

Aplicando funciones

In [15]:

```
bacteria = pd.Series([10, 20, 14, 11], index=['a', 'b', 'c', 'd'])
def f(x):
    if x % 2 != 0:
        return x + 100
    else:
        return x

bacteria.apply(f)
```

Out[15]:

```
a     10
b     20
c     14
d    111
dtype: int64
```

¿Por que evitar el "for"?

In [16]:

```
%%timeit
# mostrando performance
```

```
ds = pd.Series(range(10000))
for counter in range(len(ds)):
    ds[counter] = f(ds[counter])
```

```
1 loops, best of 3: 181 ms per loop
```

In [17]:

```
%%timeit

ds = pd.Series(range(10000))
ds = ds.apply(f)
```

```
10 loops, best of 3: 25.6 ms per loop
```

En conclusion es mejor usar "apply" que usar "for"

Usando "copy"

Algo basico de python pero que a veces se olvida

In [18]:

```
x = pd.Series([10, 20, 14, 11])
x
```

Out[18]:

```
0    10
1    20
2    14
3    11
dtype: int64
```

Aqui parece que copiamos el contenido de x a y

In [19]:

```
y = x
y
```

Out[19]:

```
0    10
1    20
```

```
2    14
3    11
dtype: int64
```

Cambiamos el valor de 10 a 100

In [20]:

```
y[0]
```

Out[20]:

```
10
```

In [21]:

```
y[0] = 100
# otra manera de cambiar un dato
y.loc[1] = 400
```

In [22]:

```
y
```

Out[22]:

```
0    100
1    400
2     14
3     11
dtype: int64
```

Pero tambien cambió x

In [23]:

```
x
```

Out[23]:

```
0    100
1    400
2     14
3     11
dtype: int64
```

Entonces usar "y = x.copy()" solo cuando sea necesario para no sobrecargar la memoria

Otras funciones

Descriptores estadisticos

In [24]:

```
bacteria.describe(percentiles=[0.25, 0.5, 0.75])
```

Out[24]:

```
count     4.00
mean     13.75
std       4.50
min      10.00
25%      10.75
50%      12.50
75%      15.50
max      20.00
dtype: float64
```

Cambiar el tipo de los datos

In [25]:

```
import numpy as np


bacteria.astype(np.float64)
```

Out[25]:

```
a    10
b    20
c    14
d    11
dtype: float64
```

Trabajando con Datos faltantes (missing data)

Normalmente los datos faltantes se representan como:

- "NA" = Not available

- "NaN" = Not a number
- None = Null/nonetype object

Los siguientes son ejemplos de inicializacion de datos que tienen datos faltantes.

In [26]:

```
pd.Series([632, 569, None], index=['a', 'c', 'e'])
```

Out[26]:

```
a    632
c    569
e    NaN
dtype: float64
```

In [27]:

```
pd.Series([632, 569, np.nan], index=['a', 'c', 'e'])
```

Out[27]:

```
a    632
c    569
e    NaN
dtype: float64
```

In [28]:

```
bacteria_dict = {'a': 632, 'b': 1638, 'c': 569, 'd': 'None'}
pd.Series(bacteria_dict, index=['a','c','e'])
```

Out[28]:

```
a    632
c    569
e    NaN
dtype: float64
```

In [29]:

```
bacteria_dict = {'a': 632, 'b': 1638, 'c': 569, 'd': None}
pd.Series(bacteria_dict, index=['a','c','e'])
```

Out[29]:

```
a    632
c    569
e    NaN
dtype: float64
```

In [30]:

```
bacteria_dict = {'a': 632, 'b': 1638, 'c': 569, 'd': 'NA'}
pd.Series(bacteria_dict, index=['a','c','e'])
```

Out[30]:

```
a    632
c    569
e    NaN
dtype: float64
```

In [31]:

```
bacteria_dict = {'a': 632, 'b': 1638, 'c': 569, 'd': 'NaN'}
bacteria_nueva = pd.Series(bacteria_dict, index=['a','c','e'])

bacteria_nueva
```

Out[31]:

```
a    632
c    569
e    NaN
dtype: float64
```

Verificando si tenemos datos faltantes

In [32]:

```
bacteria_nueva.notnull()
```

Out[32]:

```
a     True
c     True
e    False
dtype: bool
```

Son datos completos?

In [33]:

```
bacteria_nueva.notnull().all()
```

Out[33]:

```
False
```

Entonces presenta datos faltantes, luego algo que podemos hacer es rellenar con promedios

In [34]:

```
# usar "inplace=True" para modificar la "Serie"
bacteria_nueva.fillna(bacteria_nueva.mean())
```

Out[34]:

```
a    632.0
c    569.0
e    600.5
dtype: float64
```

O sino eliminar los datos faltantes

In [35]:

```
bacteria_nueva.dropna()
```

Out[35]:

```
a    632
c    569
dtype: float64
```

← Newer Posts

Older Posts →

Links

Github: https://github.com/rgap

Bitbucket: https://bitbucket.org/relguzman

Researchgate: https://www.researchgate.net/profile/Rel_Guzman

LinkedIn: https://www.linkedin.com/in/relguzman

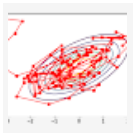
Facebook: https://www.facebook.com/rel.guzman

Twitter: https://twitter.com/rgapa

Some Random Posts

Tipos De Anuncios En Google Adwords

Estos Son Los Principales Formatos De Anuncios En...



Sampling: Metropolis Hastings

Gibbs Sampling Converges Slowly And Generates Samples Too...

Search

SEARCH