ANÁLISE DE DATOS

CON

PANDAS

Sushmita Mitra

- Graduada en física en 1988-89.
- Científica de datos da India do centro de investigación Indian Statistical Institute.
- Recibiu moitos premios polos seu traballos en computación neuronal.

Pandas

- O módulo ou libraría **`pandas`** proporciona estruturas de datos e operacións para a manipulación de tablas númericas e series temporais, coma un Excel para python. Páxina web da axuda: https://pandas.pydata.org/docs/user_guide/index.html
- Instalar **`pandas`** co comando: **`pip3 install pandas`** no teu terminal.
- Normalmente impórtase con **`import pandas as pd`**, para evitar interferencias con outros módulos.
- Coñecer a versión de **`pandas`** instalada:
  - **import pandas as pd**
  - **print(pd.__version__)**

Pandas: funcionalidades

- Proporciona os tipos de datos **`DataFrame`** e **`Series`**
- Permite ler e escribir datos desde formatos de arquivo variados: Follas de **`Excel (.xlsx)`**, arquivos **`CSV`** (Comma-separated values), arquivos de texto, bases de datos, etc.
- Transforma os datos como ordear filas ou extraer subconxuntos.
- Realiza diversas estatísticas sobre os datos.

Pandas: tipo de datos **Series**

- Unha estrutura unidimensional etiquetada que almacena datos de calquer tipo: enteiros, cadeas de caracteres, obxetos de Python, etc.
- Creación de obxetos tipo Series:
  - Desde unha lista de valores:
    - **s = pd.Series([1, 3, 5, 7, 6, 8])** # saída

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Pandas: tipo de datos **DataFrame**

- Unha estrutura bidimensional que almacena os datos en forma de táboas, con filas e columnas.
- Creación de DataFrame desde un diccionario de obxetos onde as claves son as etiquetas das columnas e os datos son os valores das columnas. Pódese poñer índices as filas con index (por defecto pon números 0,1,2,3,...)
  - import pandas as pd
  - from numpy import *
  - df=pd.DataFrame({'A':1.0, 'B': range(3), 'C':['Maria', 'Pablo', 'Uxia']}, index=([10, 20, 30]))
  - df #contido de df

```
    A  B      C
10  1.0  0  Maria
20  1.0  1  Pablo
30  1.0  2   Uxia
```

Pandas: tipo de datos **DataFrame**

- Tipo de dato de cada columna no DataFrame df:
  - **df.dtypes**
- Visualiza as n primeiras filas do DataFrame df (por defecto visualiza 10):
  - **df.head(n)**
- Visualiza as n últimas filas do DataFrame df:
  - **df.tail(n)**
- Visualiza as etiquetas das columnas e índices das filas do DataFrame df:
  - **df.columns** # saída Index(['A', 'B', 'C'], dtype='object')
  - **df.index** # Index([10, 20, 30], dtype='int64')

Pandas: tipo de datos **DataFrame**

- Convertir no DataFrame **df** nun dato de Numpy (sen etiquetas de columna nen índices de fila):
  - **df.to_numpy()**
- Devolve unha estatística básica do DataFrame **df**:
  - **df.describe()**
- Acceso a unha columna do DataFrame df: df[“etiqueta da columna]
  - **df[“A”]**

```
Out[25]:
10    1.0
20    1.0
30    1.0
Name: A, dtype: float64
```

Pandas: tipo de datos **DataFrame**

- Acceso a un rango de filas do DataFrame df: df[inicio:fin]
  - **Df[0:2]**

```
Out[28]:
      A  B      C
10  1.0  0  Maria
20  1.0  1  Pablo
```

- Acceso a datos concretos do **DataFrame df:**

`DataFrame.at`

Access a single value for a row/column pair by label.

`DataFrame.iat`

Access a single value for a row/column pair by integer position.

`DataFrame.loc`

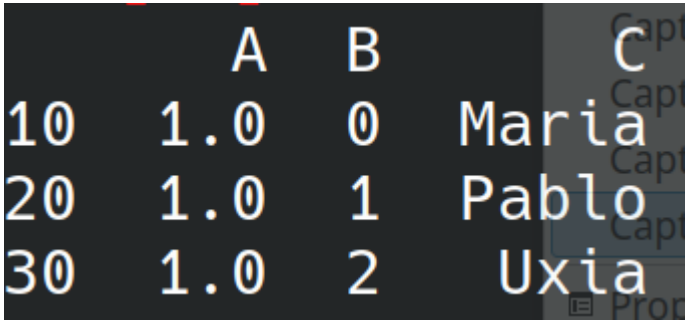
Access a group of rows and columns by label(s).

`DataFrame.iloc`

Access a group of rows and columns by integer position(s).

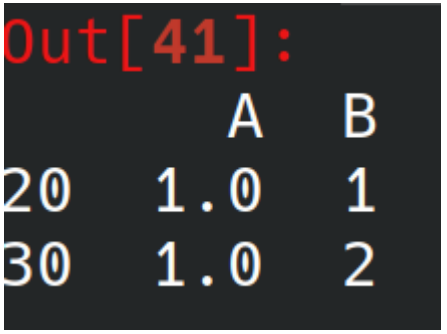
Pandas: tipo de datos **DataFrame**

- DataFrame df:



	A	B	C
10	1.0	0	Maria
20	1.0	1	Pablo
30	1.0	2	Uxia

- Acceso a datos concretos do **DataFrame df:**
  - **df.at[20,"C"]** # saída Pablo
  - **df.iat[1,2]** # saída Pablo
  - **df.loc[20:30,["A", "B"]]** # saída
  - **df.iloc[1:3, 0:2]** #saída



Out[41]:

	A	B
20	1.0	1
30	1.0	2

Pandas: tipo de datos **DataFrame**

- Seleccionar filas do **DataFrame df** que cumpren algunha condición:
  - **df[df["B"] > 1]** # saída

```
Out[44]:
     A  B     C
30  1.0  2  Uxia
```

- Pero tamén se pode realizar convertindo a columna B a un array de Numpy e facer as operacións con Numpy.

Pandas: lectura de arquivos

- Importa pandas cun alias:
  - import pandas as pd
- Ler arquivo CSV “exemplo.csv” e almacénao no obxeto de tipo DataFrame de nome df (tamén se pode usar para ler un arquivo de texto especificando o separador)
  - df = pd.read_csv("exemplo.csv")
- Ler arquivo Excel:
  - df = pd.read_excel('exemplo.xlsx')
- Ler arquivo JSON:
  - df = pd.read_json('exemplo.json')

Pandas: escritura de arquivos

- Escribir un o obxeto de tipo DataFrame e nome **df** no arquivo CSV “exemplo.csv”
  - df.to_csv("exemplo.csv")
- Escribir nun arquivo de tipo Excel:
  - df.to_excel('exemplo.xlsx')
- Escribir nun arquivo de tipo JSON (JavaScript Object Notation):
  - df.to_json('exemplo.json')

Pandas: exemplo

- Arquivo “exemplo.xlsx”:

	A	B	C	D	E	F
1	MUESTRA	f1	f2	f3	f4	saida
2	1	7,1	4,2	3,5	2,4	alto
3	2	6,4	4,4	6,2	0,7	baixo
4	3	4,1	2,6	6,2	0,7	baixo
5	4	7,71	4,46	6,26	0,68	alto
6	5	6,05	3,57	5,11	1,79	alto
7	6	3,47	2,95	7,04	0,22	alto
8	7	2,87	3,01	5,71	0,37	baixo
9	8	7,79	6,61	7,99	0,09	baixo
10	9	8,19	6,97	9,11	0,13	baixo
11	10	6,27	4,79	7,84	0,11	baixo
12						

- Ler arquivo:
  - datos=pd.read_excel('exemplo.xlsx')

Pandas: exemplo

- Estatistica básica: **datos.describe()**

```
Out[56]:
       MUESTRA         f1         f2         f3         f4
count  10.00000  10.000000  10.000000  10.000000  10.000000
mean    5.50000   5.990547   4.351100   6.496390   0.715602
std     3.02765   1.896235   1.476520   1.603620   0.772836
min     1.00000   2.868667   2.608333   3.458333   0.085882
25%     3.25000   4.549167   3.150333   5.837167   0.150294
50%     5.50000   6.352549   4.275000   6.245000   0.519667
75%     7.75000   7.548250   4.705510   7.637578   0.720167
max    10.00000   8.194118   6.971765   9.110588   2.375000
```

- Convertir columna f1 a array de numpy no **DataFrame datos**:
  - **f1=datos["f1"].to_numpy()** # f1 array([7.075 , 6.43333333, 4.05 , 7.706 , 6.04666667, 3.46933333, 2.86866667, 7.79058824, 8.19411765, 6.27176471])
- Acceso a un subconxunto de datos no **DataFrame datos:**
  - **datos.iloc[2:5,3:6]**

```
Out[60]:
          f3        f4  saida
2  6.216667  0.666667  baixo
3  6.256667  0.680667   alto
4  5.107333  1.785333   alto
```

Pandas: exemplo

- Gardar o **DataFrame** **datos** no arquivo con nome “proba.xlsx”:
  - datos.to_excel('proba.xlsx')