Python para Análisis de Datos

Módulo 04

Pandas

Pandas

Pandas es un módulo muy popular para manipulación y análisis de datos. Está hecho sobre Numpy y provee estructuras de datos diseñadas para trabajar con datos con de tablas. A diferencia de un array de numpy, pandas permite trabajar con distintos tipos de datos para cada columna asemejándose más a lo que se usa en bases de datos relacionales.

Tiene dos estructuras de datos: la **serie** y el **dataframe**. El primero es un arreglo unidimensional etiquetado y con un tipo de dato determinado. El dataframe es una estructura bidimensional similar a una tabla que admite etiquetas para filas y columnas. Cada columna de un dataframe es una serie.

Series

La **serie** es una estructura de datos unidimensional en donde cada dato tiene además una etiqueta. Es una estructura mutable a la que no sólo se le puede cambiar sus elementos (como en un array de numpy) si no que además se le pueden agregar y quitar elementos (a diferencia de un array).

Se refiere a las etiquetas colectivamente como **index,** y las etiquetas no tienen por qué ser únicas. Estas etiquetas añaden funcionalidad para manipular los datos. Para crear una serie se usa el constructor `pd.Series()` que es capaz de convertir a serie distintas estructuras como listas, diccionarios, arrays, etc. También tiene un atributo *nombre,* que es el que se utiliza como etiqueta de la columna cuando forma parte de un dataframe.

Series

```
pd.Series([5,7,2])
```

```
0    5
1    7
2    2
dtype: int64
```

```
pd.Series([5,7,2], index=["a","b","c"], dtype="int8", name="numeros")
```

```
a    5
b    7
c    2
Name: numeros, dtype: int8
```

```
pd.Series({"Martín":8, "Juan":9, "Lucía":7},name="notas")
```

```
Martín    8
Juan      9
Lucía     7
Name: notas, dtype: int64
```

Dataframe

El **dataframe** es la estructura más utilizada de pandas. Es una estructura bidimensional tipo tabla con etiquetas para filas y columnas, lo que da más flexibilidad a la hora de manipular los datos. Cada columna puede tener su propio tipo de dato.

Es mutable tanto en los datos como en su tamaño, por lo que se pueden añadir o quitar filas y columnas. Cada columna es una serie en donde todas las series tienen el mismo `index.`

Se puede construir dataframes a partir de muchas estructuras diferentes como listas, arrays y diccionarios.

Dataframe

```
# desde listas anidadas

pd.DataFrame([[0,1,2], [3,4,5], [6,7,8]])
```

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

Dataframe

```
# desde listas anidadas y definiendo index y columns

pd.DataFrame([[0,1,2], [3,4,5],[6,7,8]], index=["a", "b", "c"],
             columns=["lunes","martes","miércoles"])
```

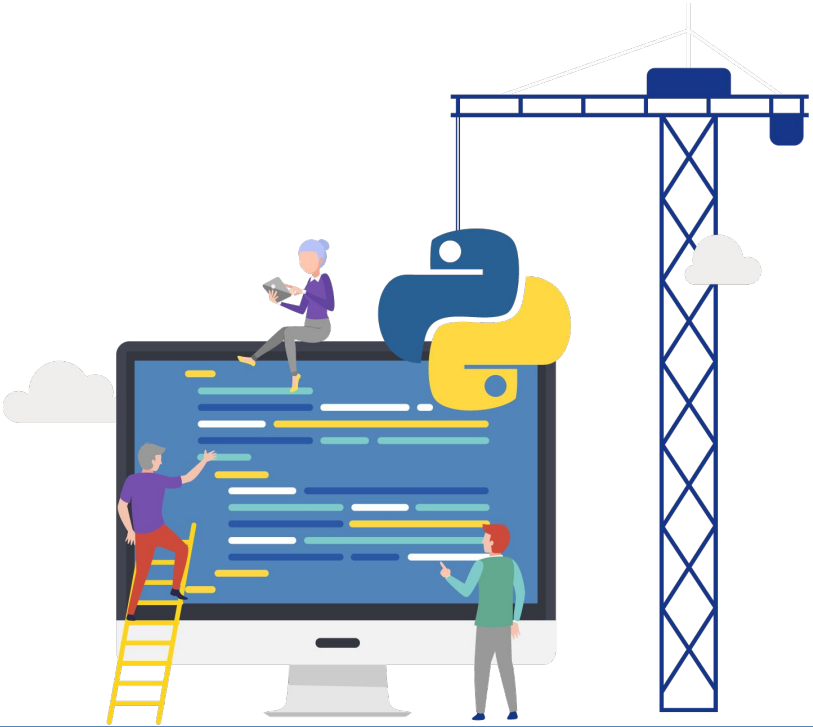
	lunes	martes	miércoles
a	0	1	2
b	3	4	5
c	6	7	8

Dataframe

```
# desde un array de numpy

array = np.random.randint(0,50,(3,5))
pd.DataFrame(array)
```

	0	1	2	3	4
0	31	44	13	1	7
1	45	1	34	27	32
2	7	37	6	22	40



Dataframe

```
# desde un diccionario

pd.DataFrame({"Columna 1": [1,2,3], "Columna 2": [4,5,6]}, index=["A","B","C"])
```

	Columna 1	Columna 2
A	1	4
B	2	5
C	3	6

Lectura de archivos

Pandas tiene numerosas funciones para leer diferentes formatos de archivos y construir dataframes a partir de ellos. Esto facilita enormemente la tarea de importar los datos. Entre los formatos que se pueden leer podemos encontrar csv, excel, json, html, sql, entre otros.

Los nombres de estas funciones empiezan con el prefijo `read_`, por ejemplo, `pd.read_csv`, `pd.read_excel`, etc. Cada una de estas funciones tiene numerosos parámetros para controlar cómo se realiza la lectura si es necesario.

Lectura de archivos

Para practicar las funcionalidades que ofrece pandas vamos a trabajar con un dataset muy conocido, el del Titanic. En el material del Alumni se provee el archivo *"titanic.csv"*, que vamos a abrir con la función `read_csv`. El primer parámetro puede ser un string con la ruta absoluta o relativa al archivo. Si el archivo se encuentra en el mismo directorio que la jupyter notebook, se puede poner simplemente el nombre del archivo. Además, se pueden pasar URLs de archivos csv.

La función va a crear un DataFrame a partir del archivo. En general no hace falta especificar más parámetros pero si es necesario se puede controlar cada detalle de cómo se realiza la lectura del archivo.

```
data = pd.read_csv("titanic.csv")
data
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

Explorando el dataframe

Se pueden visualizar filas del comienzo del dataframe con el método `head`, y del final con el método `tail`. En ambos se puede especificar la cantidad de filas a imprimir. El método info imprime información sobre un DataFrame que incluye el tipo de índice y los tipos de columna, los valores no nulos y el uso de memoria.

También se puede ver los atributos `dtype` (tipo de dato de cada columna), `shape` (dimensiones del dataframe), `size` (cantidad de elementos). Las etiquetas se pueden ver con los atributos `index` y `columns`, que además permiten reasignarlos para modificar las etiquetas. El atributo `values` devuelve una representación del dataframe como un array de numpy.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
data.shape
```

```
(891, 12)
```

```
data.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
data.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

¡Muchas gracias!

¡Sigamos trabajando!