Data Analytics

–

Python

–

Pandas

gamma tech school

TEMA 3 – Pandas

Índice

1. Introducción

Pandas es una librería open source que facilita el trabajo con estructuras de datos y su análisis.

Puede trabajar con muchos tipos de datos distintos: series de tiempo, matriciales, tabulares, entre otros.

Nos permite trabajar con DataFrame, el objeto principal de pandas, así como estructuras tabulares bidimensionales y las Series, ambas basadas en una array de Numpy.

Gracias a pandas tendremos mucha facilidad para limpiar, filtrar, rellenar datos faltantes, fusionar conjuntos de datos, agruparlos, transformarlos, estudiarlos…

Todo pudiendo cargar nuestros datos desde una gran variedad de archivos (CSV, Excel, SQL, JSON…) así como exportarlos de nuevo en el formato deseado.

2. Conociendo Numpy

Numpy

Es una librería fundamental para el análisis de datos.

Nos proporciona una poderosa estructura de datos: **ndarray**.

- **ndarray** tiene un comportamiento similar a una lista de python, pero con la diferencia que en este caso tienen homogeneidad (todos los datos que contienen son del mismo tipo) y pueden ser manipulados de manera más eficiente en términos de memoria y rendimiento.
- Añade funciones para realizar operaciones matemáticas y avanzadas, así como operaciones vectorizadas.
- Facilita manipular los datos y es fundamental para gestionar grandes conjuntos de datos.
- Es la piedra angular para otras librerías fundamentales en el análisis de datos, como son: pandas, matplotlib, SciPy, scikit-learn…

2.1 Qué es un array

- Similar a una **lista** de Python.
- Contiene **datos homogéneos**: todos los datos del array son del mismo tipo.
- Permite **operaciones entre arrays**:
  - No puedo sumar dos listas, las va a concatenar.
  - Puedo sumar dos arrays, sin necesidad de usar bucles sumará los valores.
- Tienen **número de dimensiones** (1D, 2D, 3D…)
- Tienen **tamaño** (*shape*) que indica la cantidad de elementos que tiene el array en cada dimensión.
- También indica el **tipo de dato** que tienen los elementos que hay dentro del array.

2.1 Qué es un array

```
1 a = np.array([[1, 2, 3], [4, 5, 6]])
2 a
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
1 print(f'Tamaño: {a.shape} --> 2 dimensiones de tres elementos.')          # tamaño
2 print(f'Cantidad de elementos en el total de las dimensiones {a.size}.')  # cantidad de elementos
3 print(f'Tipo de dato de los elementos: {a.dtype}.')                       # tipo de dato de los elementos
4 print(f'Cantidad de dimensiones: {a.ndim}.')                             # dimensiones
```

```
Tamaño: (2, 3) --> 2 dimensiones de tres elementos.
Cantidad de elementos en el total de las dimensiones 6.
Tipo de dato de los elementos: int64.
Cantidad de dimensiones: 2.
```

2.2 Tipos de dato en un array

Data type	Description
`bool_`	Boolean (True or False) stored as a byte
`int_`	Default integer type (same as C `long`; normally either `int64` or `int32`)
intc	Identical to C `int` (normally `int32` or `int64`)
intp	Integer used for indexing (same as C `ssize_t`; normally either `int32` or `int64`)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
`float_`	Shorthand for `float64`.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
`complex_`	Shorthand for `complex128`.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Additionally to `intc` the platform dependent C integer types `short`, `long`, `longlong` and their unsigned versions are defined.

3. Series de pandas

- Es una de las dos estructuras de datos principales en pandas.
- Puede entenderse como una **columna** de una tabla.
- Puede contener cualquier tipo de dato (enteros, flotantes, strings, distintos objetos de Python…)
- Cada elemento, cada fila, se asocia a un **índice**. El índice es **único** para cada fila.

```
1 serie
```

```
0             hola
1                1
2               23
3                4
4       [1, 2, 3]
5     otro string
dtype: object
```

- **Homogeneidad**: a pesar que una Series puede contener cualquier tipo de dato, todos los elementos deben ser del mismo tipo. Esto favorecerá que, en combinación con numpy, se puedan hacer operaciones de manera eficiente.
- **Inmutabilidad del tamaño**: los valores de una Series se pueden cambiar, pero su tamaño no. Podemos añadir más elementos (de otra lista, de otra Series…), pero hacer eso nos creará una Series nueva.

```
1 seriedata
```

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

3. Series de pandas - métodos

- **.head() / .tail()**: Retorna las primeras/n últimas n filas de la Serie.
- **.describe()**: Proporciona estadísticas resumidas para Series numéricas o categóricas.
- **.max() / .min()**: Encuentra el valor máximo/mínimo.
- **.mean() / .median()**: Calcula la media/mediana.
- **.sum()**: Suma los valores.
- **.unique()**: Retorna los valores únicos.
- **.value_counts()**: Cuenta la ocurrencia de cada valor único.
- **.apply()**: Aplica una función a cada elemento.
- **.map()**: Mapea valores según una correspondencia dada o función.
- **.sort_values()**: Ordena la Serie.
- **.sort_index()**: Ordena la Serie por su índice.
- **.isna() / .notna() / .isnull() / .notnull()**: Detecta valores NaN o no NaN.
- **.fillna()**: Rellena los valores NaN con un valor especificado.
- **.dropna()**: Elimina los elementos NaN.
- **.astype()**: Cambia el tipo de dato de la Serie.
- **.copy()**: Crea una copia de la Serie.
- **.append()**: Añade elementos de otra Serie o lista.
- **.reset_index()**: Restablece el índice de la Serie a un índice entero incremental.

4. DataFrame

- Estructura de datos principal de pandas.
- Es una estructura tabular bidimensional.
  - Tiene filas y columnas.
  - Similar a una hoja de cálculo, una entidad de SQL…
- Cada columna puede ser de un tipo de dato distinto.

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLar Ba
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	
...	...	...	...	...	...	...	...	...	...	...	
18244	7	2018-02-04	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	
18245	8	2018-01-28	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	
18246	9	2018-01-21	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	
18247	10	2018-01-14	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	
18248	11	2018-01-07	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	

18249 rows × 14 columns

- **Heterogeneidad**: así como una Serie, a pesar de poder albergar todo tipo de dato, debe contener un único tipo de dato; el DataFrame puede contener diferentes tipos de dato, cada columna puede ser un dato distinto (fecha, cadena de texto, números, objetos…)
- **Mutable**: se puede alterar tanto el contenido como el tamaño.
- **Alineación automática**: permite alinear los datos automáticamente basándose en el índice.
- **Índices para filas y columnas**: cada fila y cada columna tiene su propia etiqueta. Permite acceder muy rápidamente a información concreta, o subconjuntos de datos.

4. DataFrame - métodos

Manipulación de Datos

- **.head() / .tail()**: Muestra las primeras/n últimas n filas.
- **.describe()**: Proporciona un resumen estadístico de las columnas numéricas.
- **.drop()**: Elimina filas o columnas especificadas.
- **.dropna()**: Elimina filas o columnas que contengan NaN.
- **.fillna()**: Rellena valores NaN con un valor dado.
- **.sort_values()**: Ordena el DataFrame por una o más columnas.
- **.sort_index()**: Ordena el DataFrame por el índice de las filas o columnas.
- **.reset_index()**: Restablece el índice del DataFrame.
- **.set_index()**: Establece una columna del DataFrame como el índice.

Entrada/Salida (I/O)

- **.read_csv() / .to_csv()**: Lee/escribe en archivos CSV.
- **.read_excel() / .to_excel()**: Lee/escribe en archivos Excel.
- **.read_sql() / .to_sql()**: Lee/escribe en bases de datos SQL.

Selección y Filtrado

- **.loc[] / .iloc[]**: Seleccionan datos por etiquetas o posiciones.
- **.query()**: Filtra filas que cumplen una condición.
- **.filter()**: Filtra elementos según el eje especificado.

Agregación y Agrupación

- **.groupby()**: Agrupa datos basándose en valores de una o más columnas.
- **.pivot_table()**: Crea una tabla dinámica.
- **.agg()**: Realiza una agregación utilizando una o más operaciones.

Combinación y Unión

- **.merge()**: Combina DataFrames basándose en columnas comunes (similar a JOIN en SQL).
- **.concat()**: Concatena o apila objetos pandas a lo largo de un eje.
- **.join()**: Une columnas con otra DataFrame.

Transformación

- **.apply()**: Aplica una función a lo largo de un eje del DataFrame.
- **.applymap()**: Aplica una función elemento por elemento.

5. NaN – Ausencia de valor

- Significa “Not A Number”.
- Se representa en el estándar IEEE de aritmética en punto flotante para representar valores indefinidos o no reales.
- En numpy y pandas representan valores faltantes o incalculables.

- Usando **numpy**:
  - np.nan para representarlo.
  - Una operación matemática con NaN dará como resultado NaN, que no es ningún valor, incluido sí mismo. Es decir que np.nan == np.nan dará False.
  - Permite detectarlos, con métodos como np.isnan()
- Usando **pandas**:
  - Trata el NaN como un NA, ausente. Representa datos faltantes tanto en Series como en DataFrame.
  - Generalmente una operación con NaN va a dar como resultado NaN. No obstante, pandas facilita el manejo de operaciones con datos faltantes.
  - Permite detectarlos (.isna() / .isnull()), rellenarlos (.fillna()) incluso eliminarlos (.dropna())

5. NaN – Ausencia de valor

Consideraciones:

- **Cálculos**: la existencia de NaN puede influir muchísimo en el resultado de nuestros cálculos. Hay que hacer una buena exploración primero de la información.
- **Comparaciones**: no son efectivas, por ello se tienen que emplear los diferentes métodos para poder detectarlos correctamente.
- **Análisis**: debemos analizar bien el comportamiento de nuestros datos para tomar una decisión que será fundamental para la limpieza de los NaN.
- **Limpieza**: es crucial tomar la decisión durante el proceso de análisis que menos influye al resultado.

6. Cargando datos...

Cargar archivos

Pandas facilita la carga de archivos para explorar nuestros datos y analizarlos. Permite, de una manera muy sencilla, parsear la información para hacerla accesible y legible en forma de DataFrame.

Los formatos más habituales:

	Lectura	Escritura
CSV	pd.read_csv('ruta/archivo')	DataFrame.to_csv('ruta/archivo')
Excel	pd.read_excel('ruta/archivo')	DataFrame.to_excel('ruta/archivo')
JSON	pd.read_json('ruta/archivo')	DataFrame.to_json('ruta/archivo')

Parámetros importantes para la carga de archivos

- **sep =** por defecto va a ser **‘,’.** Puede suceder que el archivo.csv tenga otro tipo de separador, le pasaremos el separador en formato *string* como argumento.
  - **pd.read_table()**: va a tener por defecto el separador **sep= ‘\t’.** Es el tabulador.
- **header =** por defecto va a tomar la primera línea del archivo. En caso de ser otra se le puede indicar.
- **usecols =** se le puede pasar una lista con las columnas que queremos utilizar.
- **index_col =** se le indica la columna, con una *string* que tiene que ocupar el lugar del índice al cargar el archivo.
- **dtype =** se le puede pasar un diccionario indicando el tipo de dato que debe tener cada columna.

7. Conociendo los datos

Introducción a EDA (Exploratory Data Analysis)

- Objetivos:
  - Plantear hipótesis sobre las causas de los fenómenos observados.
- Pasos previos:
  - Conocer los datos:
    - ¿Qué forma tienen, columnas, filas…?
    - ¿Qué ocurre con los NaN?
    - ¿Necesaria limpieza?
- EDA:
  - Análisis con gráficos.
  - Estudio de comportamiento.
  - Estudio de correlaciones.

7. Conociendo los datos

Análisis exploratorio

- **.head() / .tail()**: Muestra las primeras/n últimas n filas.
- **.describe()**: Proporciona un resumen estadístico de las columnas numéricas.
- **.shape()**: Devuelve una tupla (filas, columnas).
- **.info()**: Aporta información rápida sobre el rango de índices, nombre de las columnas, tipos de dato.
- **.columns**: Devuelve una lista con el nombre de las columnas.
- **.sample()**: Una muestra aleatoria de tantas filas como se le indique en el argumento.
- **.copy()**: Hace una copia, mejor trabajar sobre ella para no estropear los datos originales, hasta estar seguros.
- **.dtype() / .astype()**: Muestra / asigna el tipo de dato.
- **.isna() / .isnull()**: Cuántos nulos hay.
- **.fillna()**: Rellena los nulos con el criterio tomado.
- **.rename()**: Uno de los modos para cambiar el nombre a cierta información.
- **.drop()**: Permite eliminar los datos que cumplan con la característica que se indique.
- **.value_counts():** Ideal para variables categóricas, cuenta las veces que aparece cada valor.
- **.unique()**: Muestra los valores únicos.
- **.nunique()**: Muestra la cantidad de valores únicos.
- **.hist()**: Muestra gráficamente la distribución de los datos.
- **.corr()**: Hace una matriz de correlación, que sirve para ver la relación entre dos variables.

Consideraciones

- **Llamar a una columna:**
  - df.columna
  - df[‘columna’]
  - df[[‘columna’]]
- **Llamar a varias columnas:**
  - no se puede con la primera sintaxis
  - df[‘columna1’, ‘columna2’, ‘columna3’]
  - df[[‘columna1’, ‘columna2’, ‘columna3’]]
- **Inplace**:
  - todo aquello que afecte a nuestros datos se tiene que grabar dándole un argumento True al parámetro inplace.
  - Renombrar, reindexar, borrar, son acciones que están alterando directamente el DataFrame.
- **Axis**:
  - Muchas funciones van a necesitar saber sobre qué deben actuar, columnas o filas.
    - axis = 1 —> **columna**
    - axis = 0 —> **fila** (es el valor por defecto)

8. .iloc[] / .loc[]

Característica	.iloc[]	.loc[]
Tipo de índice	de posición (enteros)	etiqueta
Selección	selecciona filas y columnas por su posición	selecciona filas y columnas por su etiqueta
Uso de índices	df.iloc[0, 0] primera fila, primera columna	df.loc['fila1', 'columna1'] fila1, columna1
Rangos	puede usarse el slicing	admite rangos de etiquetas
Boolean indexing	puede usar arrays booleanos basados en posición	puede usar arrays booleanos basados en etiquetas
Modificaciones	hace modificaciones concretas en la selección especificada según su posición	hace modificaciones concretas en la selección especificada según su etiqueta