Python Biella Group

Pandas 05

Indexing e Multiindex

Ricette sui DataFrames

Presenta

Mario

Python Biella Group

Obiettivi della serata

- Migliorare la conoscenza di Pandas
  - In particolare, Dataframes (2d)

Cos’ e’ pandas?

pandas is a high-performance open source library for data analysis in Python developed by Wes McKinney in 2008
Available for free and is distributed with a 3-Clause BSD License under the open source initiative

pandas stands for **panel data**, a reference to the tabular format in which it processes the data

Over the years, it has become the **de-facto standard library for data analysis** using Python
There's a large community behind it, rapid iteration, features, and enhancements are continuously made

Some key features of pandas include the following:

- It **facilitates loading/importing data** from varied sources, such as CSV and databases such as SQL.

- It **can process a variety of datasets in different formats**: time series, tabular heterogeneous, and matrix data.

- It **can handle myriad operations on datasets**: subsetting, slicing, filtering, merging, groupBy, re-ordering, and re-shaping.

- It **can deal with missing data according to rules defined by the user/developer**, such as ignore, convert to 0, and so on.

- It **can be used for parsing and munging (conversion) of data** as well as **modeling and statistical analysis**.

- **It integrates well with other Python libraries** such as statsmodels, SciPy, and scikit-learn.

- It **delivers fast performance** and can be sped up even more by making use of **Cython** (C extensions to Python).

NumPy, pandas e R

The pandas package was created by Wes McKinney in 2008 as a result of frustrations he encountered while working on time series data in R.

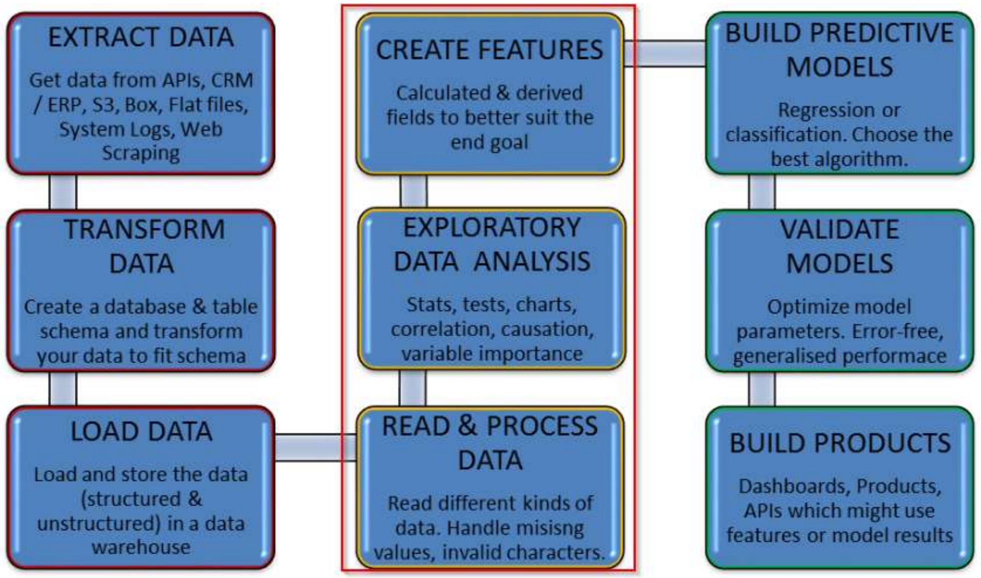
Built on top of NumPy and provides features not available in it.

It provides fast, easy-to-understand data structures and helps fill the gap between Python and a language like R.

NumPy deals with **homogeneous** blocks of data.

Using **pandas** helps to deal with data in a tabular structure composed of **different data types**.

Utilizzo di pandas in Data Analytics / 1


EXTRACT DATA
Get data from APIs, CRM / ERP, S3, Box, Flat files, System Logs, Web Scraping
TRANSFORM DATA
Create a database & table schema and transform your data to fit schema
LOAD DATA
Load and store the data (structured & unstructured) in a data warehouse
CREATE FEATURES
Calculated & derived fields to better suit the end goal
EXPLORATORY DATA ANALYSIS
Stats, tests, charts, correlation, causation, variable importance
READ & PROCESS DATA
Read different kinds of data. Handle misisng values, invalid characters.
BUILD PREDICTIVE MODELS
Regression or classification. Choose the best algorithm.
VALIDATE MODELS
Optimize model parameters. Error-free, generalised performace
BUILD PRODUCTS
Dashboards, Products, APIs which might use features or model results

Indispensabile per questi steps di “Data Science”:
- Read & Process Data
- Exploratory Data Analysis
- Create Features

pandas

Utilizzo di pandas in Data Analytics / 2

pandas is suited well for the following types of dataset:
•Tabular with heterogeneous type columns
•Ordered and unordered time series
•Matrix/array data with labeled or unlabeled rows and columns

pandas can perform the following operations on data with finesse:
•Easy handling of missing and NaN data
•Addition and deletion of columns
•Automatic and explicit data alignment with labels
•GroupBy for aggregating and transforming data using split-apply-combine
•Converting differently indexed Python or NumPy data to DataFrame
•Slicing, indexing, hierarchical indexing, and subsetting of data
•Merging, joining, and concatenating data
•I/O methods for flat files, HDF5, feather, and parquet formats
•Time series functionality

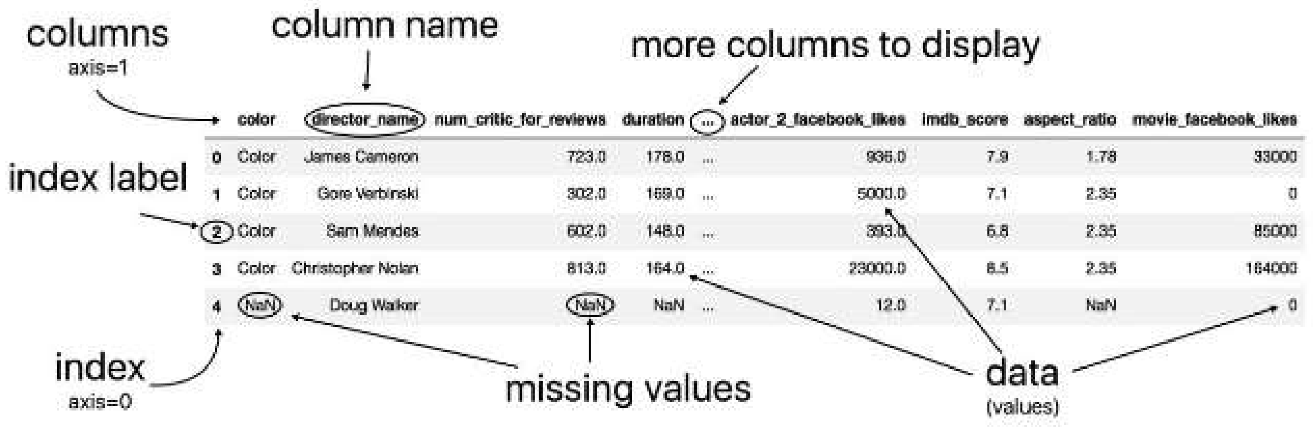
Strutture dati

pandas.Series one-dimensional (1D) pandas data structure: 1D data and the index
A Series is really a 1D NumPy array (composed of any data type) coupled with an array of labels (together called the **index** of the series).

pandas.DataFrame two-dimensional (2D) pandas tabular data structure: 2D data structure composed of rows and columns.
Each column of a DataFrame is a pandas Series.
These columns should be of the same length, but they can be of different data types—float, int, bool, and so on.
DataFrames are both value-mutable and size-mutable.
This lets us perform operations that would alter values held within the DataFrame or add/delete columns to/from the DataFrame.
Similar to a Series, which has a name and index as attributes, a DataFrame has column names and a row index.
The row index can be made of either numerical values or strings such as month names.
Indexes are needed for fast lookups as well as proper aligning and joining of data in pandas multilevel indexing is also possible in DataFrames.

~~**pandas.Panel** three-dimensional (3D) pandas data structure~~ DEPRECATED! -> multi-indexing in DataFrames

Anatomia di un dataframe



	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

Columns and the **index** is in **bold** font, which makes them easy to identify.
By convention, the terms **index label** and **column name** refer to the individual members of the index and columns, respectively.
The term index refers to all the index labels as a whole just as the term columns refers to all the column names as a whole

The columns and the index provide **labels** for the columns and rows of the DataFrame and allow for direct and easy access to different subsets of data.
When multiple Series or DataFrames are combined, the indexes align first before any calculation occurs.
Collectively, the columns and the index are known as the **axes**: a DataFrame has two axes--a vertical axis (the index) and a horizontal axis(the columns).
Pandas borrows convention from NumPy and uses the integers 0/1 as another way of referring to the vertical/horizontal axis.

DataFrame data (values) is always in regular font and is an entirely separate component from the columns or index.
Pandas uses **NaN** (not a number) to represent **missing values**.
Notice that even though the color column has only string values, it uses NaN to represent a missing value.

Indexing

(select & where)

Indicizzazione numerica -> Accesso/Selezione/Filtro per posizione:

.iloc[row_indexer, col_indexer]

col_indexer opzionale, default = :

Parametri:

- An integer, e.g. 5
- A list or array of integers, e.g. [4, 3, 0]
- A slice object with ints, e.g. 1:7
- A boolean array

Indicizzazione per etichetta -> Accesso/Selezione/Filtro per etichetta:

.loc[row_indexer, col_indexer]

col_indexer opzionale, default = :

Parametri:

- A single label, e.g. 5 or 'a'
- A list or array of labels, e.g. ['a', 'b', 'c']
- A slice object with labels, e.g. 'a':'f'. (start and stop included)
- A boolean array

Indicizzazione per etichetta -> Accesso/Selezione/Filtro per etichetta:

[indexer] (magic method: __getitem__)

Simile a .loc ma seleziona solo per colonna – filtra solo per riga

Parametri:

- A single label, e.g. 5 or 'a'
- A list or array of labels, e.g. ['a', 'b', 'c']
- A boolean array (len = num. of rows)
- A slice object with labels, e.g. 'a':'f'. (start and stop included)

Selezione di una colonna / riga -> Serie
Selezione di più colonne / righe -> DataFrame

Forte utilizzo dello slicing

Indexing and Selecting in pandas
Basic Indexing

[ ] usata per indicizzare e fare subset di lista / per slicing di arrays NumPy ed è <u>operatore di base per indicizzare in pandas</u>
Series: slicing specificando la label o l'indice posizionale
DataFrame: permette di specificare la label per colonna ma non di trattare le righe (nè con indice, nè con label)
Per specificare la riga: sequenza di [] con indice o label
# Accessing a single element in a DataFrame -> df["colB"]["R3"], df["colB"][1]
Per fare <u>subset di valori multipli, una lista di labels</u> deve essere passata tra []. -> df[["colA", "colB"]]

To access a single entity (a column, value, or item), **the square bracket operator can be replaced by the dot operator**.
Let's subset colA in the DataFrame using the dot (.) operator -> df.colA
By using **two dot operators in a chain**, an individual element can be accessed -> df.colA.R3

Range slicing
Slicing by supplying the **start** and **end** position to subset a range of values can be done in pandas objects, just as in NumPy arrays.
The [:] operator helps in range slicing. As always with a range in Python, the value after the colon is excluded when slicing.
Range slicing can be done by providing either the start or end index. If the end index is not provided, values are sliced from the given starting index to the end of the data structure. Likewise, when only the end index is given, the first row is considered as the starting position for slicing

Range slicing can be made even more interesting through a **property to select rows at evenly spaced intervals**.
For instance, you can select only the odd-numbered rows or even-numbered rows this way:
# Select odd rows -> df[::2]   # Select even rows -> df[1::2]   # Reverse the rows -> df[::-1]

Python Biella Group

Indexing and Selecting in pandas
Labels, integer, and mixed indexing

Operatore .loc : indicizzazione per etichetta (label-oriented indexing)

- a single label such as ["colC"], [2], or ["R1"]—*nei casi in cui la label è un intero, non si riferisce alla posizione dell'indice, ma è una label!*
- a list or array of labels, for example, ["colA", "colB"];
- a slice object with labels, for example, "colB":"colD“ (unlike the ranges in Python, both the exteriors are included in the selected data);
- a Boolean array.

pandas objects can also be filtered based on logical conditions applied to values within the objects

Ex -> (serie) ser_loc1.loc[ser_loc1 > 13]; (df) df_loc2.loc[df_loc2["Asia"] > 11, :]

Operatore .iloc : indicizzazione numerica (integer-based indexing)

Same four cases as label-oriented indexing: single labels, a list of labels, range slicing, and Boolean arrays.

For Boolean array-based indexing with the iloc operator, the array must be extracted using logical conditions around array values.

# Indexing with Boolean array -> df_loc2.iloc[(df_loc2["Asia"] > 11).values, :]

Operatori .iat e**.at** – equivalenti (rispettivamente) a .iloc e .loc

While .loc and .iloc support the selection of multiple values, .at and .iat can only extract a single scalar value.

Hence they require row and column indices for slicing

~~.ix operator: indicizzazione mista (mixed label and integer-based indexing)~~ -> Deprecato

Condizioni di filtro

pandas

(where conditions)

<Pandas Serie> <operatore> <elemento di confronto>

OUTPUT = Pandas Serie (booleana) che può essere usata come filtro

Le condizioni possono essere combinate con operatori booleani ma non sono lazy (*)

DataFrame è un oggetto 2D – Sta allo sviluppatore decidere se accedere per riga o per colonna

(*) "Normalmente" Python valuta le espressioni in modo "**lazy**".
Nell'espressione (x and y), prima valuta x; se x è false, il suo valore è ritornato, altrimenti valuta anche y e ritorna il suo valore.
Nell'espressione (x or y), prima valuta x; se x è true, ritorna il suo valore altrimenti valuta y e il valore risultante è restituito.

Multiindex

pandas

MultiIndex o **indice gerarchico**
E' un indice che usa valori multipli nella sua chiave

Quando si usa?
- Performance? (non ci sono benefici evidenti)
- «Autodocumentare» i dati, dare una struttura ai dati
- In combinazione con stack() e unstack()

...and JOIN US!

- Sito: https://pythonbiella.herokuapp.com/
- GitHub: https://github.com/PythonGroupBiella/MaterialeLezioni
- YouTube: https://www.youtube.com/channel/UCkvQcNjmC_duLhvDxeUPJAg
- Telegram: https://t.me/joinchat/AAAAAFGSWcxhSln_SRhseQ