

Introduzione a Pandas

Obiettivi

- Caricare dati da file
- Rinominare e modificare le colonne
- Cercare dati in base a criteri
- Modificare i dati



Prima di cominciare

VISTI LA SCORSA VOLTA:

- Caricamento di file .csv in un dataframe
- Visualizzazione del dataframe come tabella
- Rimozione di duplicati e correzione di dati mancanti/sbagliati
- · Ricerca di dati tramite chiave

ESEMPI DI FILE CHE POSSONO DIVENTARE UN DATAFRAME:

- Excel: pandas.read_excel('pandasExcel.xlsx', 'Sheet1'), con il modulo 'xlrd'
- Csv: pandas.read_csv('Book1.csv'), visto la scorsa volta. Lo stesso comando si può usare su un file di testo (.txt), previa definizione del carattere separatore: pandas.read_csv('myFile.txt', delimiter='\t')
- NetCDF, formato particolare che vedremo più avanti
- Database SQL, con il modulo sqlite3:

```
connection = sqlite3.connect('mydatabase.db')
pandas.read_sql('select * from Employee', connection)
```

Column type

Le colonne di un dataframe sono serie, che contengono dati tutti dello stesso tipo (int, string...). È possibile cambiare il tipo dei dati contenuti, modificando I dati stessi di conseguenza.

```
df.varname = df.varname.astype(newtype)
```

Esempio:

```
df['timestamp'] = df.Datetime.values.astype(np.int64)
```

Un metodo specifico per le date:

```
df['Datetime'] = pd.to_datetime(np.array(datetime_array),
format='%Y-%m-%d %H:%M:%S')
```

Modificare i dati

CANCELLARE RIGHE E COLONNE:

```
df.drop([0]) cancella la riga di indice 0 (cioè la prima)
df.drop(['Carolina']) cancella la riga in cui compare il nome Carolina
df.drop(df.index[-1]) cancella l'ultima riga
```

df.drop(['A'], axis=1) cancella la colonna A. axis indica che stiamo indicando una colonna (axis=0 sono le righe)

CANCELLARE DATI RIPETUTI:

df.drop_duplicates() cancella le righe ripetute (df.duplicates() per controllare se sono presenti righe ripetute)

df.drop_duplicates(['nome']) cancella tutte le righe che hanno lo stesso valore nella
colonna 'nome'

nota: df['A'].nunique() conta invece il numero di entry non ripetute

Modificare i dati

FUNZIONI

È possibile applicare una funzione ai dati di un dataframe:

df.apply(np.sqrt) radice quadrata di ogni elemento del dataframe df['A'].apply(np.sqrt) radice quadrata degli elementi della colonna 'A'

df['A'].sum() somma di tutti gli elementi della colonna 'A' (qualora possibile)

Ordinare i dati

ORDINARE IN BASE AI VALORI DI UNA COLONNA:

Si possono ordinare i dati (le righe) in base al valore di una certa colonna:

```
df.sort_values(by=['nome'], inplace=True) ordine crescente (default)
    df.sort_values(by=['nome'], inplace=True, ascending=False) ordine
decrescente (va specificato)
```

Nota: inplace=True vuol dire che modifica il dato esistente, invece che crearlo nuovo

ORDINARE IN BASE AI VALORI DI PIÙ COLONNE:

Si possono ordinare i dati anche tenendo conto di più colonne. In questo caso la prima colonna specificata avrà la precedenza, poi seguirà la seconda eccetera:

```
df.sort_values(by=['nome','cognome'], inplace=True) ordine crescente
(default)
```

df.sort_values(by=['nome','cognome'], inplace=True, ascending=False)
ordine decrescente (va specificato)

Selezionare i dati

• SELEZIONE IN BASE ALLA POSIZIONE:

Se so quali righe voglio, è facile:

df.[0:3] mostra le prime 3 righe del dataframe

df.[-1] questa è l'ultima riga

SELEZIONARE IN BASE AL VALORE:

A volte non so dove sia il dato, ma so il suo valore:

df.loc[df['nome'] == 'Carolina'] righe in cui nella colonna 'nome' compare il valore 'Carolina

s_africa = dataset.sel(lat=slice(-35, -52), lon=slice(25, 44)) questa è una maschera, ossia un criterio di selezione

s_africa_temp = np.array(s_africa.variables['air']) ho selezionato dal dataframe i valori relativi a un'area geografica: il Sud Africa

Alcuni esempi realistici – 1

· Creiamo un dataframe da file .csv e modifichiamo il formato della data:

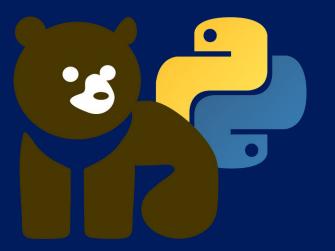
```
def open_ASIM_housekeeping(filename):
    df = pd.read_csv(filename)
    df['t'] = pd.to_datetime(df['t'], format='%d-%b-%Y %H:%M:%S')
    df['timestamp'] = df.t.values.astype(np.int64) // 10 ** 9
    return df
```



Alcuni esempi realistici – 2

 Creiamo un dataframe da file .csv e modifichiamo il formato della data, versione più complessa:

```
def open_ISS_position(filename):
  df = pd.read csv(filename, sep=" ", names=['Date', 'time', 'delete', 'longitude', 'latitude'])
  date array = np.array(df['Date'])
  time_array = np.array(df['time'])
  datetime_array = []
  for d, t in zip(date_array, time_array):
     datetime_string = "%s %s" % (d, t)
     datetime_array.append(datetime_string)
  df['Datetime'] = pd.to_datetime(np.array(datetime_array), format='%Y-%m-%d %H:%M:%S')
  df['timestamp'] = df.Datetime.values.astype(np.int64) // 10 ** 9
  df.drop(['delete', 'Date', 'time'], axis=1, inplace=True)
  return df
```



...and KISS!

Keep It Simply Stupid

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

JOIN US!