10-PandasSeries

August 11, 2020

1 Pandas Series

- Started by Wes MacKinney with a first release in 2011.
- Based on NumPy, it is the most used library for all things data.
- Motivated by the toolbox in R for manipulating data easily.
- A lot of names in Pandas come from R world.
- It is Open source (BSD)

https://pandas.pydata.org/

```
import pandas as pd
```

"*Pandas provides high-performance, easy-to-use data structures and data analysis tools in Python*"

- Self-describing data structures
- Data loaders to/from common file formats
- Plotting functions
- Basic statistical tools.

[1]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

```
pd.set_option("display.max_rows", 8)
plt.rcParams['figure.figsize'] = (9, 6)
```

1.1 Series

- A Series contains a one-dimensional array of data, *and* an associated sequence of labels called the *index*.
- The index can contain numeric, string, or date/time values.
- When the index is a time value, the series is a time series.
- The index must be the same length as the data.
- If no index is supplied it is automatically generated as `range(len(data))`.

[2]:
```
pd.Series([1,3,5,np.nan,6,8])
```

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

[3]:
```
pd.Series(index=pd.period_range('09/11/2017', '09/18/2017', freq="D"))
```

```
<ipython-input-3-579d6b723cc5>:1: DeprecationWarning: The default dtype for
empty Series will be 'object' instead of 'float64' in a future version. Specify
a dtype explicitly to silence this warning.
  pd.Series(index=pd.period_range('09/11/2017', '09/18/2017', freq="D"))
```

[3]:
```
2017-09-11   NaN
2017-09-12   NaN
2017-09-13   NaN
2017-09-14   NaN
2017-09-15   NaN
2017-09-16   NaN
2017-09-17   NaN
2017-09-18   NaN
Freq: D, dtype: float64
```

1.1.1 Exercise

- Create a text with `lorem` and count word occurences with a `collection.Counter`. Put the result in a `dict`.

[4]:
```
from lorem import text
from collections import Counter
```

```
import operator

c = Counter(filter(None,text().strip().replace('.','').replace('\n',' ').
 ↪lower().split(' ')))
result = dict(sorted(c.most_common(),key=operator.itemgetter(1),reverse=True))
result
```

[4]:
```
{'porro': 10,
 'tempora': 10,
 'velit': 9,
 'sit': 9,
 'ut': 8,
 'ipsum': 7,
 'voluptatem': 7,
 'est': 7,
 'adipisci': 7,
 'magnam': 7,
 'labore': 7,
 'neque': 7,
 'dolorem': 7,
 'quiquia': 6,
 'numquam': 6,
 'modi': 6,
 'aliquam': 6,
 'dolor': 5,
 'eius': 5,
 'amet': 5,
 'etincidunt': 5,
 'quisquam': 5,
 'non': 5,
 'consectetur': 5,
 'quaerat': 4,
 'sed': 3,
 'dolore': 3}
```

1.1.2 Exercise

- From the results create a Pandas series name latin_series with words in alphabetical order as index.

[5]:
```
df = pd.Series(result)
df
```

[5]:
```
porro          10
tempora        10
velit           9
```

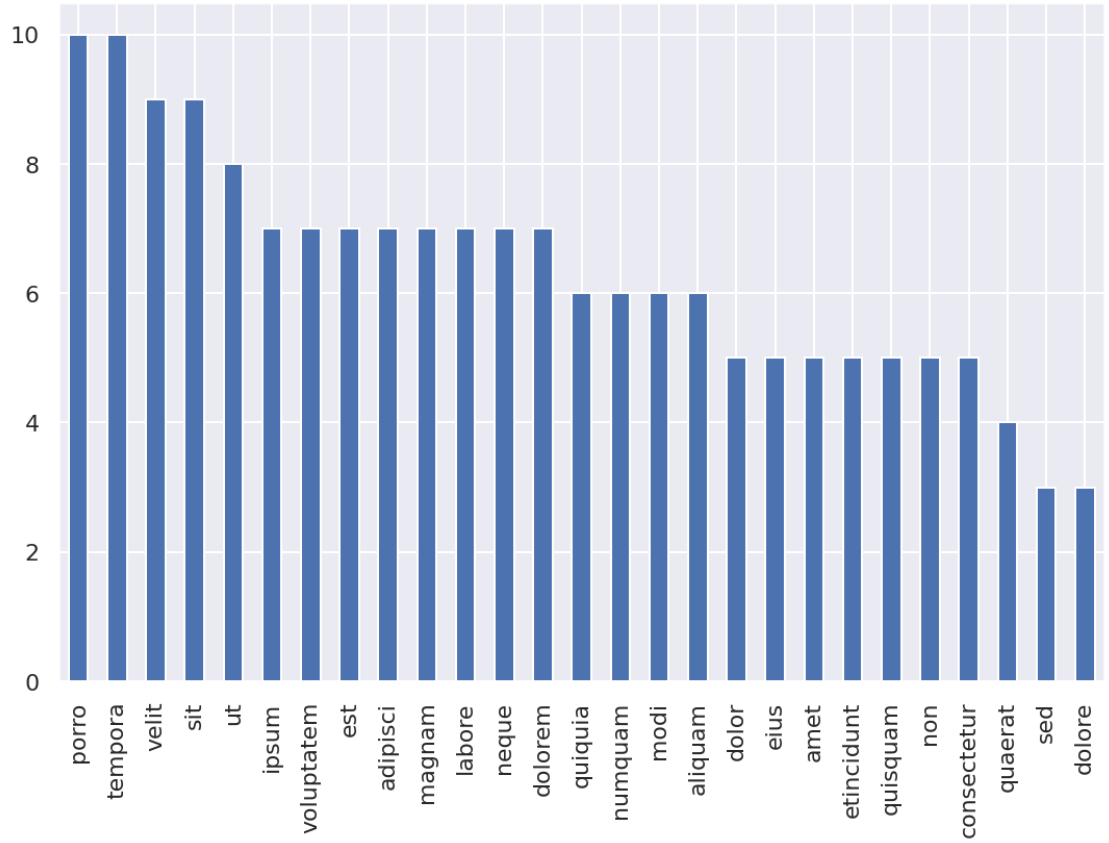
```
sit             9
                ..
consectetur     5
quaerat         4
sed             3
dolore          3
Length: 27, dtype: int64
```

1.1.3 Exercise

- Plot the series using 'bar' kind.

[6]:
```
df.plot(kind='bar')
```

[6]:
```
<AxesSubplot:>
```



1.1.4 Exercise

- Pandas provides explicit functions for indexing `loc` and `iloc`.

- Use `loc` to display the number of occurrences of 'dolore'.
- Use `iloc` to diplay the number of occurrences of the last word in index.

[7]:
```
df.loc['dolore']
```

[7]: 3

[8]:
```
df.iloc[-1]
```

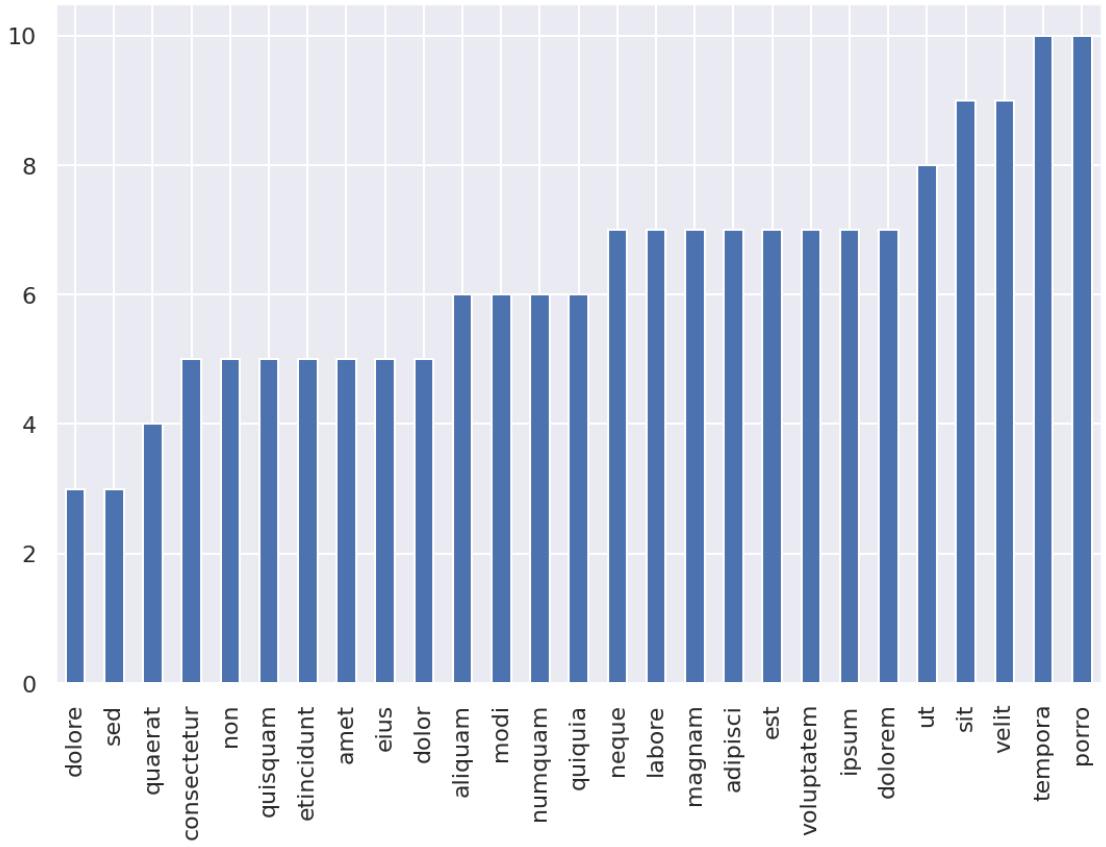
[8]: 3

1.1.5 Exercise

- Sort words by number of occurrences.
- Plot the Series.

[9]:
```
df = df.sort_values()
df.plot(kind='bar')
```

[9]: <AxesSubplot:>



1.1.6 Full globe temperature between 1901 and 2000.

We read the text file and load the results in a pandas dataframe. In cells below you need to clean the data and convert the dataframe to a time series.

[10]:
```
import os
here = os.getcwd()

filename = os.path.join(here,"data","monthly.land.90S.90N.df_1901-2000mean.dat.
 ↪txt")

df = pd.read_table(filename, sep="\s+",
                   names=["year", "month", "mean temp"])
df
```

[10]:

	year	month	mean temp
0	1880	1	-0.0235
1	1880	2	-0.4936
2	1880	3	-0.6785
3	1880	4	-0.2829
…	…	…	…
1580	2011	9	-999.0000
1581	2011	10	-999.0000
1582	2011	11	-999.0000
1583	2011	12	-999.0000

[1584 rows x 3 columns]

1.1.7 Exercise

- Insert a third column with value one named ”day” with `.insert`.
- convert df index to datetime with `pd.to_datetime` function.
- convert df to Series containing only ”mean temp” column.

[11]:
```
df.insert(loc=2,column='day',value=np.ones(len(df)))
df
```

[11]:

	year	month	day	mean temp
0	1880	1	1.0	-0.0235
1	1880	2	1.0	-0.4936
2	1880	3	1.0	-0.6785
3	1880	4	1.0	-0.2829
…	…	…	…	…
1580	2011	9	1.0	-999.0000
1581	2011	10	1.0	-999.0000
1582	2011	11	1.0	-999.0000
1583	2011	12	1.0	-999.0000

```
[1584 rows x 4 columns]
```

[12]:
```
df.index = pd.to_datetime(df[['year','month','day']])
df
```

[12]:
```
            year  month  day  mean temp
1880-01-01  1880      1  1.0    -0.0235
1880-02-01  1880      2  1.0    -0.4936
1880-03-01  1880      3  1.0    -0.6785
1880-04-01  1880      4  1.0    -0.2829
…            …    …  …         …
2011-09-01  2011      9  1.0  -999.0000
2011-10-01  2011     10  1.0  -999.0000
2011-11-01  2011     11  1.0  -999.0000
2011-12-01  2011     12  1.0  -999.0000

[1584 rows x 4 columns]
```

[13]:
```
df = df['mean temp']
df
```

[13]:
```
1880-01-01     -0.0235
1880-02-01     -0.4936
1880-03-01     -0.6785
1880-04-01     -0.2829
                …
2011-09-01   -999.0000
2011-10-01   -999.0000
2011-11-01   -999.0000
2011-12-01   -999.0000
Name: mean temp, Length: 1584, dtype: float64
```

[14]:
```
type(df)
```

[14]:
```
pandas.core.series.Series
```

1.1.8 Exercise

- Display the beginning of the file with `.head`.

[15]:
```
df.head()
```

[15]:
```
1880-01-01   -0.0235
1880-02-01   -0.4936
1880-03-01   -0.6785
```

```
1880-04-01   -0.2829
1880-05-01   -0.1261
Name: mean temp, dtype: float64
```

1.1.9 Exercise

- Display the end of the file with `.tail`.

[16]:
```
df.tail()
```

[16]:
```
2011-08-01   -999.0
2011-09-01   -999.0
2011-10-01   -999.0
2011-11-01   -999.0
2011-12-01   -999.0
Name: mean temp, dtype: float64
```

In the dataset, -999.00 was used to indicate that there was no value for that year.

1.1.10 Exercise

- Display values equal to -999 with `.values`.
- Replace the missing value (-999.000) by `np.nan`

[17]:
```
df[df.values == -999]
```

[17]:
```
2011-07-01   -999.0
2011-08-01   -999.0
2011-09-01   -999.0
2011-10-01   -999.0
2011-11-01   -999.0
2011-12-01   -999.0
Name: mean temp, dtype: float64
```

[18]:
```
df2 = df.copy()
df2[df == -999.0] = np.nan  # For this indexing we need a copy
df2.tail()
```

[18]:
```
2011-08-01   NaN
2011-09-01   NaN
2011-10-01   NaN
2011-11-01   NaN
2011-12-01   NaN
Name: mean temp, dtype: float64
```

Once they have been converted to np.nan, missing values can be removed (dropped).

1.1.11 Exercise

- Remove missing values with `.dropna`.

[19]:
```
df = df2.dropna()
df.tail()
```

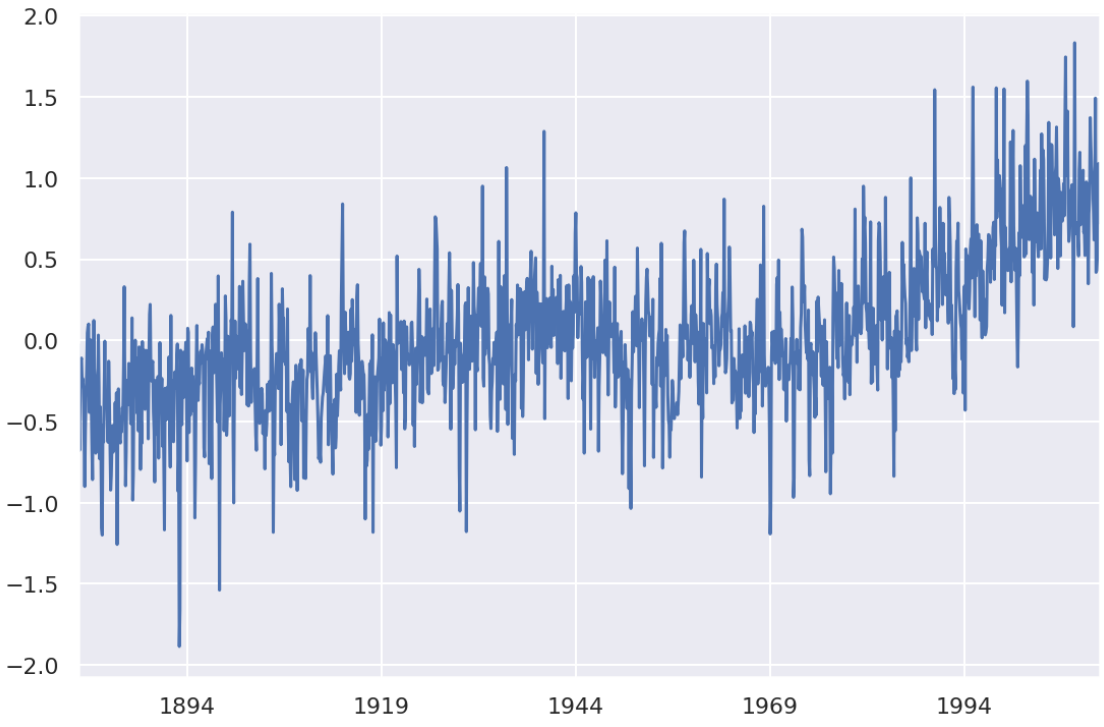
[19]:
```
2011-02-01    0.5113
2011-03-01    0.8618
2011-04-01    1.0897
2011-05-01    0.7247
2011-06-01    0.8550
Name: mean temp, dtype: float64
```

1.1.12 Exercise

- Generate a basic visualization using `.plot`.

[20]:
```
df.plot()
```

[20]:
```
<AxesSubplot:>
```



1.1.13 Exercise

Convert df index from timestamp to period is more meaningfull since it was measured and averaged over the month. Use `to_period` method.

[21]:
```
df = df.to_period('M')
df
```

[21]:
```
1880-01   -0.0235
1880-02   -0.4936
1880-03   -0.6785
1880-04   -0.2829
             …
2011-03    0.8618
2011-04    1.0897
2011-05    0.7247
2011-06    0.8550
Freq: M, Name: mean temp, Length: 1578, dtype: float64
```

1.2 Resampling

Series can be resample, downsample or upsample. - Frequencies can be specified as strings: ”us”, ”ms”, ”S”, ”T”, ”H”, ”D”, ”B”, ”W”, ”M”, ”A”, ”3min”, ”2h20”, ... - More aliases at http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases

1.2.1 Exercise

- With `resample` method, convert df Series to 10 year blocks:

[22]:
```
df.resample('10A').mean()
```

[22]:
```
1880   -0.386485
1890   -0.316798
1900   -0.256431
1910   -0.247673
          …
1980    0.188519
1990    0.463572
2000    0.785452
2010    0.884700
Freq: 10A-DEC, Name: mean temp, Length: 14, dtype: float64
```

1.2.2 Saving Work

HDF5 is widely used and one of the most powerful file format to store binary data. It allows to

store both Series and DataFrames.

[23]:
```
with pd.HDFStore("data/pandas_series.h5") as writer:
    df.to_hdf(writer, "/temperatures/full_globe")
```

1.2.3 Reloading data

[24]:
```
with pd.HDFStore("data/pandas_series.h5") as store:
    df = store["/temperatures/full_globe"]
```