Data structuring, part 1

The Pandas way

Andreas Bjerre-Nielsen

Recap

Which Python containers have learned about so far?

- ...
- ...

Which containers can we turn into a `numpy array`?

- ...

Agenda

1. motivation
2. the numpy module
3. overview of the pandas module:
4. the pandas series
   - working with series and numeric procedures
   - boolean series
5. more tools:
   - inspecting and selecting observations
   - modifying DataFrames
   - dataframe IO

Why we structure data

Motivation

Why do we want to learn data structuring?

- Data never comes in the form of our model. We need to 'wrangle' our data.

Can our machine learning models not do this for us?

- Not yet :). The current version needs **tidy** data. What is tidy?

One row per observation.

Loading the software

In [33]:

```
import numpy as np
import pandas as pd
```

Numpy module

Numpy overview

What is the numpy (http://www.numpy.org/) module?

`numpy` is a Python framework similar to matlab

- fast and versatile for manipulating arrays
- linear algebra tools available
- many machine learning and statistics libraries are built with `numpy`

Numpy arrays

What is a numpy array?

An n-dimensional array:

- 1-d: vector;
- 2-d: matrix;
- 3-d: tensor.

We see use cases later.

Pandas overview

Pandas motivation

Why use Pandas?

1. simplicity - Pandas is built with Python's simplicity
2. flexible and powerful tools for manipulating data
3. speed - build on years of research about numeric computation
4. development - breathtaking speed of new tools coming

Pandas data types

How do we work with data in Pandas?

- We use two fundamental data stuctures: `Series` and `DataFrame`.

Pandas data frames (1)

What is a `DataFrame`?

- A matrix with labelled columns and rows (which are called indices). Example:

```
In [34]:
df = pd.DataFrame([[1,2], [3, 4]],
                  columns=['A', 'B'],
                  index=['i', 'ii'])

print(df)
```

```
    A  B
i   1  2
ii  3  4
```

- An object with powerful data tools.

Pandas data frames (2)

How are pandas dataframes built?

Pandas dataframes can be though of as numpy arrays with some additional stuff.

Most functions from `numpy` can be applied directly to Pandas. We can convert a DataFrame to a `numpy` matrix with `values` attribute.

```
In [35]: df.values
```

```
Out[35]: array([[1, 2],
                [3, 4]], dtype=int64)
```

To note: In Python we can describe it as a list of lists of a dict of dicts.

Pandas series

What is a `Series`?

- A vector/list with labels for each entry. Example:

```
In [36]:  my_series = pd.Series([1,True,1/3,'a'])
          my_series
```

```
Out[36]:  0           1
          1        True
          2    0.333333
          3           a
          dtype: object
```

What data structure does this remind us of?

- A mix of Python list and dictionary (more info follows)

Series vs DataFrames

How are Series related to DataFrames?

Every column is a series. Example: access as object method:

```
In [37]: print(df.A)
```

```
i     1
ii    3
Name: A, dtype: int64
```

Another option is access as key:

```
In [38]: print(df['B'])
```

```
i     2
ii    4
Name: B, dtype: int64
```

To note: The latter option more robust as variables named same as methods, e.g. `count`, cannot be accesed.

Indices and column names

Why don't we just use matrices?

- inspection of data is quicker
- keep track of rows after deletion
- indices may contain fundamentally different data structures
    - e.g. time series, hierarchical groups
- facilitates complex operation:
    - merging datasets
    - split-apply-combine

Working with pandas Series

Generate Series (1)

Let's revisit our series

```
In [39]: my_series
```

```
Out[39]: 0           1
         1        True
         2    0.333333
         3           a
         dtype: object
```

Components in series

- `index`: label for each observation
- `values`: observation data
- `dtype`: the format of the series - `object` means any data type is allowed
    - note: the object dtype is SLOW!

Generate Series (2)

How do we set custom index?

Example:

```
In [40]: num_data = range(0,3)
         indices = ['B','C','A']
         my_series2 = pd.Series(num_data, index=indices)
         my_series2
```

```
Out[40]: B    0
         C    1
         A    2
         dtype: int64
```

Generation Series (3)

Can a dictionary be converted to a series?

Yes, we just put into the Series class constructor. Example:

```
In [41]: d = {'yesterday':0, 'today':1, 'tomorrow':3}

         my_series3 = pd.Series(d)
         my_series3
```

```
Out[41]: today        1
         tomorrow     3
         yesterday    0
         dtype: int64
```

Note: Same is true for DataFrames which requires that each value in the dictionary is also a dictionary.

Generation Series (4)

Can we convert series to dictionaries?

- Yes, in most cases.

- ***WARNING!#@***: Series indices are NOT unique

```
In [42]: pd.Series(range(3),index=['A', 'A','A']).to_dict()
```

```
Out[42]: {'A': 2}
```

The power of pandas

How is the series different from a dict?

- We will see that pandas Series have powerful methods and operations.
- It is both key and index based.

Converting data types

The data type of a series can be converted with the **astype** method:

```
In [43]: my_series3.astype(np.float64) # np.str
```

```
Out[43]: today        1.0
         tomorrow     3.0
         yesterday    0.0
         dtype: float64
```

Numeric procedures

Numeric operations (1)

How can we basic arithmetic operations with arrays, series and dataframes?

Like Python data! An example:

```
In [44]: my_arr1 = np.array([2, 3, 2])
         my_arr2 = my_arr1 ** 2
         my_arr2
```

```
Out[44]: array([4, 9, 4], dtype=int32)
```

Numeric operations (2)

Are other numeric python operators the same??

Yes /, //, -, *, ** etc. behave as expected.

Why is this useful?

- vectorized operations are VERY fast;
- requires very little code.

Numeric operations (3)

Can we do the same with two vectors?

- Yes, we can also do elementwise addition, multiplication, subtractions etc. of series. Example:

```
In [45]: my_arr1 + my_arr2
```

```
Out[45]: array([ 6, 12,  6])
```

Numeric methods (1)

Pandas series has powerful numeric methods built-in, example:

In [46]:

```
my_series2.median()
```

Out[46]:

```
1.0
```

Other useful methods include: **mean**, **quantile**, **min**, **max**, **var**, **describe**, **quantile** and many more.

In [47]:

```
my_series2.describe()
```

Out[47]:

```
count    3.0
mean     1.0
std      1.0
min      0.0
25%      0.5
50%      1.0
75%      1.5
max      2.0
dtype: float64
```

Numeric methods (2)

An important method is `value_counts`. This counts number for each observation.

Example:

```
In [48]: my_series4 = pd.Series(my_arr2)
         my_series4.unique()
```

```
Out[48]: array([4, 9], dtype=int64)
```

```
In [49]: my_series4.value_counts()
```

```
Out[49]: 4    2
         9    1
         dtype: int64
```

What is observation in the value_counts output - index or data?

Numeric methods (3)

Are there other powerful numeric methods?

Yes: examples include

- **unique**, **nunique**: the unique elements and the count of unique elements
- **cut**, **qcut**: partition series into bins
- **diff**: difference every two consecutive observations
- **cumsum**: cumulative sum
- **nlargest**, **nsmallest**: the n largest elements
- **idxmin**, **idxmax**: index which is minimal/maximal
- **corr**: correlation matrix

Check series documentation (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html) for more information.

Boolean Series

Logical expression for Series

Can we test an expression for all elements?

Yes: **==**, **!=** work for a single object or Series with same indices. Example:

```
In [50]: my_series3 > 0
```

```
Out[50]: today          True
         tomorrow       True
         yesterday     False
         dtype: bool
```

What datatype is returned?

Logical expression in Series (2)

Can we check if elements in a series equal some element in a container?

Yes, the `isin` method. Example:

```
In [52]: my_rng = range(2)
         print(list(my_rng))
         # my_series3.isin(my_rng)
```

```
[0, 1]
```

Power of boolean series (1)

Can we combine boolean Series?

Yes, we can use:

- the and operator, also known as &
- the `or` operator, also known as |

In [53]:

```
# (my_series3 > 0) & (my_series3 == 1)
```

What datatype is returned?

Power of boolean series (2)

Why do we care for boolean series (and arrays)?

Because we can use the to select rows based on their content.

In [54]:

```
# my_series6 = pd.Series(data=[17, 18, 18],
#                          index=['April 1', 'April 2', 'April 3'],
#                          name='age')
# print(my_series6)
# my_series6[my_series6>17]
```

NOTE: Boolean selection is extremely useful for dataframes!!

Inspecting and selecting observations

Viewing series and dataframes

How can we view the contents in our dataset?

- We can use `print` our dataset
- We can visualize patterns by plotting (from tomorrow)

The head and tail

We select the *first* rows in a DataFrame or Series with the head method.

```
In [55]: n = 3 # number of observations
         my_series7 = pd.Series(np.random.normal(size=[100]))
         my_series7.head(n)
```

```
Out[55]: 0    0.128634
         1    1.129369
         2    1.153792
         dtype: float64
```

The `tail` method selects the last observations in a DataFrame.

Row selection (1)

*How can we select certain rows in a Series when for given index **keys**?*

WIth the `loc` attribute. Example:

In [56]:

```
my_loc = 'tomorrow'
# my_loc = ['today', 'tomorrow']
# my_series3.loc[my_loc]
```

Row selection (2)

How can we select certain rows in a Series when for given index ***integers****?*

The `iloc` method selects rows for provided index integers.

```
In [57]:  my_series3.iloc[1]
          # my_series7.iloc[10:13]

Out[57]:  3
```

Row selection (3)

Do our tools for vieving specific rows, i.e. `loc`, `iloc` work for DataFrames?

- Yes, we can use both `loc` and `iloc`. As default they work the same.

In [60]:

```
my_idx = ['i', 'ii', 'iii']
my_cols = ['a','b']
my_data = [[1, 2], [3, 4], [5, 6]]
my_df = pd.DataFrame(my_data, columns=my_cols, index=my_idx)
my_df.loc[['i']]
```

Out[60]:

	a	b
i	1	2

Row selection (4)

How are `loc`, `iloc` different for DataFrames?

- For DataFrames we can also specify columns.

In [59]:

```
idx_keep = ['i','ii']
cols_keep = ['a']
print(my_df.loc[idx_keep, cols_keep])
```

```
     a  b
i    1  2
ii   3  4
iii  5  6
    a
i   1
ii  3
```

Columns selection

How can we select columns in a DataFrame?

- Option 1: using the `[]` and providing a list of columns.
- Option 2: using `loc` and setting row selection as `:`.

In [79]:

```
# my_df[cols_keep]
```

Selection quiz

What does `:` *do in* `iloc` *or* `loc`*?*

Select all rows (columns).

Modifying DataFrames

Chaging the index (1)

How can we change the index of a DataFrame?

We change set a DataFrame's index index using its method `set_index`. Example:

In [87]:

```
# my_df.set_index('a')
```

Chaging the index (2)

Is our DataFrame changed? I.e. does it have a new index?

We can use the keyword `inplace` which will replace the DataFrame:

```
In [65]: my_df_a = my_df.set_index('a')
         my_df_copy = my_df.copy()
         my_df_copy.set_index('a', inplace=True)
         my_df_copy.head(2)
```

Out[65]:

	b
a	
1	2
3	4

Chaging the index (3)

Sometimes we wish to remove the index. This is done with the `reset_index` method:

In [66]:

```
my_df.reset_index()
my_df
```

Out[66]:

	a	b
0	1	2
1	3	4
2	5	6

By specifying the keyword `drop`=True we delete the index. Note `inplace` also works.

To note: Indices can have multiple levels, in this case `level` can be specified to delete a specific level.

Sorting data

A DataFrame can be sorted with `sort_values`; this method takes one or more columns to sort by.

```
In [85]: my_df.sort_values(by='a', ascending=False)
```

Out[85]:

	a	b
iii	5	6
ii	3	4
i	1	2

To note: Many key word arguments are possible for sort_values, including ascending if for one or more valuable we want descending values. Sorting by index is possible with `sort_index`.

DataFrame IO: loading and storing

Reading DataFrames (1)

Download the file from url:

```
In [76]: url = 'https://api.statbank.dk/v1/data/FOLK1A/CSV?lang=en&Tid=*'
```

Reading DataFrames (2)

Now let's try opening it:

- As local file:

In [84]:

```
abs_path = 'C:/Users/bvq720/Downloads/FOLK1A.csv' # absolute path
# rel_path = 'FOLK1A.csv' # relative path

df = pd.read_csv(abs_path) # open the file as dataframe
```

What are absolute and relative paths?

Reading DataFrames (3)

Now let's try opening it online file:

In [81]:

```
df = pd.read_csv(url,sep=';')
df.head(3)
```

Out[81]:

	TID	INDHOLD
0	2008Q1	5475791
1	2008Q2	5482266
2	2008Q3	5489022

Reading other data types

Other pandas readers include: excel, sql, sas, stata and many more.

To note: an incredibly fast and useful module for reading and writing data is feather (https://github.com/wesm/feather).

Storing data

Data can be stored in a particular format with to_(FORMAT) where (FORMAT) is the file type such as csv. Let's try with to_csv:

```
In [14]: df.to_csv('DST_people_count.csv', index=False)
```

Should we always set `index=False`. Yes, unless time series!!! Otherwise the index will be exported too!

The end

Return to agenda