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. numpy and pandas overview
3. the pandas series
    - working with series and numeric procedures
    - boolean series
4. 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 [ ]:
```
import numpy as np
import pandas as pd
```

Numpy and Pandas

Numpy overview

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

`numpy` is a Python module similar to matlab

- fast and versatile for manipulating arrays
- linear algebra tools available
- used in some machine learning and statistics packages

Example from yesterday

In [ ]:
```
table = [[1,2],[3,4]]
arr = np.array(table)
```

Pandas motivation

Why use Pandas?

1. simplicity - Pandas is built with Python's simplicity
2. powerful and fast tools for manipulating data from numpy
3. flexibility and new data tools compared with numpy (more info follows)
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 [ ]:
```
df = pd.DataFrame(arr,
                  index=['i', 'ii'],
                  columns=['A', 'B'])

print(df)
```

- An object with powerful data tools.

Pandas data frames (2)

How are pandas dataframes built?

Pandas dataframes can be thought 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` array with `values` attribute.

In [ ]:
```
df.values # .tolist()
```

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 [ ]:
```
L = [1, 1.2, 'abc', True]

my_series = pd.Series(L)
my_series.to_dict()
```

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 key:

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

Another option is access as object method:

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

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 numpy arrays and 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

Generating a Series (1)

Let's revisit our series

In [ ]:
```
my_series
```

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!

Generating a Series (2)

How do we set custom index?

Example:

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

Generating a Series (3)

Can a dictionary be converted to a series?

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

In [ ]:
```
d = {'yesterday': 0, 'today': 1, 'tomorrow':3}
my_series3 = pd.Series(d)
my_series3
```

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

Generating a Series (4)

Can we convert series to dictionaries?

- Yes, in most cases.

In [ ]:
```
my_series3.to_dict()
```

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

In [ ]:
```
s = pd.Series(range(3),index=['A', 'A', 'A'])
s.index.duplicated().sum()
```

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 (i.e. sequential).

Converting data types

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

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

Numeric procedures

Numeric operations (1)

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

Like Python data! An example:

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

Numeric operations (2)

Are other numeric python operators the same??

Numeric operators work `/`, `//`, `-`, `*`, `**` 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 [ ]:
```
my_arr1 + my_arr2
```

Numeric methods (1)

Pandas series has powerful numeric methods built-in. Example of 10 mil. obs:

In [ ]:
```
arr = np.random.normal(size=10**7)
s2 = pd.Series(arr)
s2.median()
```

Other useful methods include: `mean`, `quantile`, `min`, `max`, `std`, `describe`, `quantile` and many more.

In [ ]:
```
my_series2.describe()
```

Numeric methods (2)

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

Example:

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

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 [ ]:
```
my_series3 == 0
```

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

Power of boolean series (1)

Can we combine boolean Series?

Yes, we can use:

- the & operator (*and*)
- the | operator (*or*)

In [ ]:
```
print((my_series3 > 0) & (my_series3 == 1))
```

In [ ]:
```
(df.A > 2) & (df.B<=3) # selection by multiple columns
```

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 [ ]:
```
my_series3[my_series3<3]
```

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 [ ]:
```
arr = np.random.normal(size=[100])
my_series7 = pd.Series(arr)
my_series7.head(3)
```

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 [ ]:
```
# 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 [ ]:
```
# print(my_series3.iloc[1])
# print(my_series3)
# arr = np.random.normal(size=10**6)
pd.Series(arr).iloc[500010:500013]
```

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 [ ]:
```
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)
# print(my_df)
print(my_df.loc['i'])
```

Row selection (4)

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

- For DataFrames we can also specify columns.

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

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 [ ]:
```
print(my_df.loc[:,['b']])
```

Selection quiz

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

Select all rows (columns).

Modifying DataFrames

Modyfying DataFrames

Why do we want to modify DataFrames?

- Because data rarely comes in the form we want it.

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 [ ]:
```
print(my_df.set_index('a'))
```

Chaging the index (2)

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

No, we must overwrite it or make it into a new object:

In [ ]:
```
print(my_df)
my_df_a = my_df.set_index('a').copy()
print(my_df_a)
```

Chaging the index (3)

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

In [ ]:
```
print(my_df_a.reset_index(drop=True))
# print(my_df)
```

By specifying the keyword `drop` =True we delete the index.

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

Chaging the column names

Column names can be changed with

In [ ]:
```
print(my_df)
my_df.columns = ['A', 'B']
print(my_df)
```

DataFrame's also have the function called `rename` .

Chaging all column values

How can we can update values in a DataFrame?

In [ ]:

```
print(my_df)

# set uniform value
my_df['B'] = 3
print(my_df)


# set different values
my_df['B'] = [2,17,0]
print(my_df)
```

Chaging specific column values

How can we can update values in a DataFrame?

In [ ]:
```
print(my_df)

# loc, iloc
my_loc2 = ['i', 'iii']
my_df.loc[my_loc2, 'A'] = 10
print(my_df)
```

Sorting data

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

In [ ]:
```
print(my_df.sort_values(by='A', ascending=True))
```

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 (https://api.statbank.dk/v1/data/FOLK1A/CSV?lang=en&Tid=*). Open directly in Pandas.

In [ ]:
```
url = 'https://api.statbank.dk/v1/data/FOLK1A/CSV?lang=en&Tid=*'
df = pd.read_csv(url, sep=';') # open the file as dataframe
print(df.head(2))
```

Reading DataFrames (2)

Now let's try opening the file from the URL (https://api.statbank.dk/v1/data/FOLK1A/CSV?lang=en&Tid=*) as a local file:

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

df = pd.read_csv(abs_path, sep=';') # open the file as dataframe
print(df.head(2))
```

- absolute path: entire path starting from which disk etc.
- relative paths: from where your program, i.e. Jupyter is

Reading other data types

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

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 [ ]:
```
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