Be a more efficient data scientist, master pandas with this guide

Félix Revert
Aug 15, 2018 · 4 min read

Python is open source. It's great, but has the inherent problem of open source: many packages do (or try to do) the same thing. If you're new to Python, it's hard to know the best package for a specific task. You need someone who has experience to tell you. And I tell you today: there's one package you absolutely need to learn for data science, and it's called **pandas**.

And what's really interesting with pandas is that many other packages are hidden in it. Pandas is a core package with additional features from a variety of other packages. And that's great because you can work only using pandas.

pandas is like Excel in Python: it uses tables (namely **DataFrame**) and operates transformations on the data. But it can do a lot more.

If you're already familiar with Python, you can go straight to the 3rd paragraph

Let's start:

```
import pandas as pd
```

load_pandas.py hosted with ♡ by GitHub view raw

Don't ask me why "pd" and not "p" or any other, it's just like that. Deal with it :)

The most elementary functions of pandas

Reading data

```
data = pd.read_csv('my_file.csv')
```

read_csv.py hosted with ♡ by GitHub view raw

```
data = pd.read_csv('my_file.csv', sep=';', encoding='latin-1', nrows=1000, skiprows=[2,5])
```

read_csv.py hosted with ♡ by GitHub view raw

sep means separator. If you're working with French data, csv separator in Excel is ";" so you need to explicit it. Encoding is set to "latin-1" to read French characters. nrows=1000 means reading the first 1000 rows. skiprows=[2,5] means you will remove the 2nd and 5th row when reading the file

The most usual functions: read_csv, read_excel
Some other great functions: read_clipboard (which I use way too often, copying data from Excel or from the web), read_sql

Writing data

```
data.to_csv('my_new_file.csv', index=None)
```

write_data.py hosted with ♡ by GitHub view raw

index=None will simply write the data as it is. If you don't write index=None, you'll get an additional first column of 1,2,3, ... until the last row.

I usually don't go for the other functions, like .to_excel, .to_json, .to_pickle since .to_csv does very well the job. And because csv is the most common way to save tables. There's also the .to_clipboard if you're like me an Excel maniac who wants to paste your results from Python to Excel.

Checking the data

```
data.shape
```

shape.py hosted with ♡ by GitHub view raw

Gives (#rows, #columns)

```
data.describe()
```

describe.py hosted with ♡ by GitHub view raw

Computes basic statistics

Seeing the data

```
data.head(3)
```

head.py hosted with ♡ by GitHub view raw

Print the first 3 rows of the data. Similarly to .head(), .tail() will look at the last rows of the data.

```
data.loc[8]
```

loc.py hosted with ♡ by GitHub view raw

Print the 8th row

```
data.loc[8, 'column_1']
```

loc_single_point.py hosted with ♡ by GitHub view raw

Print the value of the 8th row on "column_1"

```
data.loc[range(4,6)]
```

loc_subset.py hosted with ♡ by GitHub view raw

Subset from row 4 to 6 (excluded)

The basic functions of pandas

Logical operations

```
data[data['column_1']=='french']
```

```
data[(data['column_1']=='french') & (data['year_born']==1990) & ~(data['city']=='London')]
```

and.py hosted with ♡ by GitHub view raw

Subset the data thanks to logical operations. To use & (AND), ~ (NOT) and | (OR), you have to add "(" and ")" before and after the logical operation.

```
data[data['column_1'].isin(['french', 'english'])]
```

subset.py hosted with ♡ by GitHub view raw

Instead of writing multiple ORs for the same column, use the .isin() function

Basic plotting

This feature is made possible thanks to the matplotlib package. As we said in the intro, it's usable directly in pandas.

```
data['column_numerical'].plot()
```

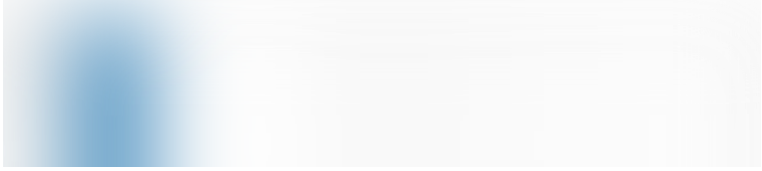
plot.py hosted with ♡ by GitHub view raw

Example of .plot() output

```
data['column_numerical'].hist()
```

hist.py hosted with ♡ by GitHub view raw

Plots the distribution (histogram)



Example of .hist() output

```
%matplotlib inline
```

plot_in_Jupyter.py hosted with ♡ by GitHub view raw

If you're working with Jupyter, don't forget to write this line (only once in the notebook), before plotting

Updating the data

```
data.loc[8, 'column_1'] = 'english'
```

update_data.py hosted with ♡ by GitHub view raw

Replace the value in the 8th row at the 'column_1' by 'english'

```
data.loc[data['column_1']=='french', 'column_1'] = 'French'
```

complex_loc.py hosted with ♡ by GitHub view raw

Change values of multiple rows in one line

Alright, now you can do things that were easily accessible in Excel. Let's dig in some amazing things that are not doable in Excel.

Medium level functions

Counting occurrences

```
data['column_1'].value_counts()
```

value_counts.py hosted with ♡ by GitHub view raw

Example of .value_counts() output

Operations on full rows, columns, or all data

```
1  data['column_1'].map(len)
```

map.py hosted with ♡ by GitHub view raw

The len() function is applied to each element of the 'column_1'

The .map() operation applies a function to each element of a column.

```
1  data['column_1'].map(len).map(lambda x: x/100).plot()
```

map_combination.py hosted with ♡ by GitHub view raw

A great pandas feature is the chaining method. It helps you do multiple operations (.map() and .plot() here) in one line, for more simplicity and efficiency

```
1  data.apply(sum)
```

apply.py hosted with ♡ by GitHub view raw

.apply() applies a function to columns. Use .apply(, axis=1) to do it on the rows.

.applymap() applies a function to all cells in the table (DataFrame).

tqdm, the one and only

When working with large datasets, pandas can take some time running .map(), .apply(), .applymap() operations. tqdm is a very useful package that helps predict when theses operations will finish executing (yes I lied, I said we would use only pandas).

```
from tqdm import tqdm_notebook
tqdm_notebook().pandas()
```

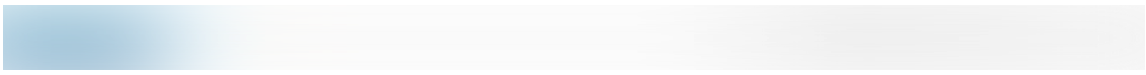
tqdm_pandas.py hosted with ♡ by GitHub view raw

setup of tqdm with pandas

```
data['column_1'].progress_map(lambda x: x.count('e'))
```

tqdm_map.py hosted with ♡ by GitHub view raw

Replace .map() by .progress_map(), same for .apply() and .applymap()



This is the progress bar you get in Jupyter with tqdm and pandas

Correlation and scatter matrices

```
data.corr()
data.corr().applymap(lambda x: int(x*100)/100)
```

corr.py hosted with ♡ by GitHub view raw

.corr() will give you the correlation matrix

```
pd.plotting.scatter_matrix(data, figsize=(12,8))
```

scatter_matrix.py hosted with ♡ by GitHub view raw

Example of scatter matrix. It plots all combinations of two columns in the same chart.

Advanced operations in pandas

The SQL join

Joining in pandas is overly simple.

```
1  data.merge(other_data, on=['column_1', 'column_2', 'column_3'])
```

merge.py hosted with ♡ by GitHub view raw

Joining on 3 columns takes just one line

Grouping

Not quite simple at the beginning, you need to master the syntax first and you'll see yourself using this feature all the time.

```
1  data.groupby('column_1')['column_2'].apply(sum).reset_index()
```

groupby.py hosted with ♡ by GitHub view raw

Group by a column, the select another column on which to operate a function. The .reset_index() reshapes your data as a DataFrame (table)

As explained previously, chain your functions in one line for optimal code

Iterating over rows

```
dictionary = {}

for i,row in data.iterrows():
  dictionary[row['column_1']] = row['column_2']
```

iterrows.py hosted with ♡ by GitHub view raw

The .iterrows() loops through 2 variables together: the index of the row and the row (**i** and **row** in the code above).

Overall pandas is one of the reason why Python is such a great language

There are many other interesting pandas features I could have shown, but it's already enough to understand why a data scientist cannot do without pandas.

To sum up, pandas is

- simple to use, hiding all the complex and abstract computations behind
- (generally) intuitive
- fast, if not the fastest data analysis package (it highly optimized in C)

It is THE tool that helps a data scientist to quickly read and understand data and be more efficient at his role.

I hope you found this article useful, and if you did, consider giving at least 50 claps :)

Data Science Python Data Analysis Machine Learning Coding

About Help Legal