Organizing data: pandas

Aksel Hiorth

University of Stavanger

Aug 19, 2022

Contents

1 What is Pandas?

Pandas is a Python package that among many things are used to handle data, and perform operations on groups of data. It is built on top of Numpy, which makes it easy to perform vectorized operations. Pandas is written by Wes McKinney, and one of it objectives is according to the official website " providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python"[1]. Pandas

[1] `https://pandas.pydata.org/`

also has excellent functions for reading and writing excel and csv files. An excel file is read directly into memory in what is called a `DataFrame` in Pandas. A DataFrame is a two dimensional object where data are typically stored in column or row format. Pandas has a lot of functions that can be used to calculate statistical properties of the data frame as a whole. In this chapter we will focus on basic data manipulation, stuff you might do in excel, but can be done much faster in Python and Pandas.

2 Creating a data frame

In the following we will assume that you have imported pandas, like this

```
import pandas as pd
```

2.1 From empty DataFrame

This is perhaps the most basic way of creating a DataFrame, first we create an empty DataFrame

```
df = pd.DataFrame()
```

Variable name.

Note that we often use `df` as a variable name for a DataFrame, this is a choice, but it is a usually a good choice as someone else reading the code could infer from a name that `df` is a DataFrame. If you need more than one DataFrame variable you could use `df1`, `df2`, etc. or even better to use a descriptive name, `df_sales_data`.

Next, we can add columns to the DataFrame

```
df=pd.DataFrame()
df['a']=[0,1,2,3]
df['b']=[4,5,6,7]
df['c']=['hammer','saw','rock','nail']
print(df) # to view data frame
```

Note that all columns needs to have the same size.

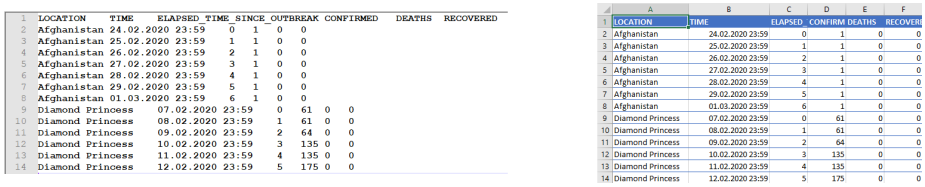
`pd.Series()`.

Even if we initialize the DataFrame column with a list, the command `type(df['a'])` will tell you that the column in the DataFrame are of type

`pd.Series()`. Thus the fundamental objects in Pandas are of type `Series`. Series are more flexible, and it is possible to calculate `df['a']/df['b']`, whereas `[0,1,2,3]/[4,5,6,7]` is not possible.

2.2 From file

Assume you have some data organized in excel or in a csv file. The csv file could just be a file with column data, they could be separated by a comma or tab



LOCATION	TIME	ELAPSED_TIME_SINCE_OUTBREAK	CONFIRMED	DEATHS	RECOVERED
Afghanistan	24.02.2020 23:59	0	1	0	0
Afghanistan	25.02.2020 23:59	1	1	0	0
Afghanistan	26.02.2020 23:59	2	1	0	0
Afghanistan	27.02.2020 23:59	3	1	0	0
Afghanistan	28.02.2020 23:59	4	1	0	0
Afghanistan	29.02.2020 23:59	5	1	0	0
Afghanistan	01.03.2020 23:59	6	1	0	0
Diamond Princess	07.02.2020 23:59	0	61	0	0
Diamond Princess	08.02.2020 23:59	1	61	0	0
Diamond Princess	09.02.2020 23:59	2	64	0	0
Diamond Princess	10.02.2020 23:59	3	135	0	0
Diamond Princess	11.02.2020 23:59	4	135	0	0
Diamond Princess	12.02.2020 23:59	5	175	0	0

	A	B	C	D	E	F
1	LOCATION	TIME	ELAPSED	CONFIRM	DEATHS	RECOVER
2	Afghanistan	24.02.2020 23:59	0	1	0	0
3	Afghanistan	25.02.2020 23:59	1	1	0	0
4	Afghanistan	26.02.2020 23:59	2	1	0	0
5	Afghanistan	27.02.2020 23:59	3	1	0	0
6	Afghanistan	28.02.2020 23:59	4	1	0	0
7	Afghanistan	29.02.2020 23:59	5	1	0	0
8	Afghanistan	01.03.2020 23:59	6	1	0	0
9	Diamond Princess	07.02.2020 23:59	0	61	0	0
10	Diamond Princess	08.02.2020 23:59	1	61	0	0
11	Diamond Princess	09.02.2020 23:59	2	64	0	0
12	Diamond Princess	10.02.2020 23:59	3	135	0	0
13	Diamond Princess	11.02.2020 23:59	4	135	0	0
14	Diamond Princess	12.02.2020 23:59	5	175	0	0

Figure 1: Official Covid-19 data, and example of files (left) tab separated (right) excel file.

```
df=pd.read_excel('file.xlsx') # excel file
df=pd.read_csv('file.csv',sep=',') # csv comma separated file
df=pd.read_csv('file.csv',sep='\t') # csv tab separated file
```

If the excel file has several sheets, you can give the sheet name directly, e.g. `df=pd.read_excel('file.xlsx',sheet_name="Sheet1")`, for more information see the documentation[2].

Accessing files.

Accessing files from python can be painful. If excel files are open in excel, Windows will not allow a different program to access it - always remember to close the file before opening it. Sometimes we are not in the right directory, to check which directory you are in, you can always do the following

```
import os
print(os.getcwd()) # prints current working directory
```

2.3 Create DataFrame from dictionary

A DataFrame can be quite easily be generated from a dictionary. A dictionary is a special data structure, where an unique key is associated with a data type (key:value pair). In this case, the key would be the title of the column, and the

[2] `https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html`

value would be the data in the columns. To generate the excel file in figure 1, we can do (see figure 2 for the final result)

```
import datetime as dt
a=dt.datetime(2020,2,24,23,59) # 24/2-2020 23:59
b=dt.datetime(2020,2,7,23,59)
my_dict={'LOCATION':7*['Afghanistan'] + 6*['Diamond Princess'],
'TIME':[a+dt.timedelta(days=i) for i in range(7)] +
[b+dt.timedelta(days=i) for i in range(6)],
'ELAPSED_TIME_SINCE_OUTBREAK':[0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5],
'CONFIRMED':7*[1]+[61, 61, 64, 135, 135, 175],
'DEATHS':13*[0],
'RECOVERED': 13*[0]}
df=pd.DataFrame(my_dict)
print(df) # to view
```

	LOCATION	TIME	ELAPSED_TIME_SINCE_OUTBREAK	CONFIRMED	DEATHS	RECOVERED
0	Afghanistan	2020-02-24 23:59:00	0	1	0	0
1	Afghanistan	2020-02-25 23:59:00	1	1	0	0
2	Afghanistan	2020-02-26 23:59:00	2	1	0	0
3	Afghanistan	2020-02-27 23:59:00	3	1	0	0
4	Afghanistan	2020-02-28 23:59:00	4	1	0	0
5	Afghanistan	2020-02-29 23:59:00	5	1	0	0
6	Afghanistan	2020-03-01 23:59:00	6	1	0	0
7	Diamond Princess	2020-02-07 23:59:00	0	61	0	0
8	Diamond Princess	2020-02-08 23:59:00	1	61	0	0
9	Diamond Princess	2020-02-09 23:59:00	2	64	0	0
10	Diamond Princess	2020-02-10 23:59:00	3	135	0	0
11	Diamond Princess	2020-02-11 23:59:00	4	135	0	0
12	Diamond Princess	2020-02-12 23:59:00	5	175	0	0

Figure 2: The resulting DataFrame of Covid-19 data.

Note that all columns must have the same length to create the DataFrame. We can easily save the data frame to excel format and open it in excel

```
df.to_excel('covid19.xlsx', index=False) # what happens if you put index=True?
```

Index column.

Whenever you create a DataFrame Pandas by default create an index column, it contains an integer for each row starting at zero. It can be accessed by `df.index`, and it is also possible to define another column as index column.

2.4 Manipulating DataFrames

2.5 Selecting columns

If we want to pick out a specific column we can access it in the following ways

```
time=df['TIME'] # by the name, alternatively
time=df[df.columns[1]]
time=df.loc[:,['TIME']] # by loc[] if we use name
time=df.iloc[:,1] # by iloc, pick column number 1
```

The `loc[]` and `iloc[]` functions allows for list slicing, one can then pick e.g. every second element in the column by `time=df.iloc[::2,1]` etc. The difference is that `loc[]` uses the name, and `iloc[]` the index (usually an integer).

Special characters.

Sometimes when reading files from excel, headers may contains invisible characters like newline `\n` or tab `\t` or maybe Norwegian special letters that have not been read in properly. If you have problem accessing a column by name do `print(df.columns)` and check if the name matches what you would expect.

2.6 Selecting rows

Typically you would select rows based on a criterion, the syntax in Pandas is that you enter a series containing `True` and `False` for the rows you want to pick out, e.g. to pick out all entries with Afghanistan we can do

```
df[df['LOCATION'] == 'Afghanistan']
```

The innermost statement `df['LOCATION'] == 'Afghanistan'` gives a logical vector with the value `True` for the five last elements and `False` for the rest. Then we pass this to the DataFrame, and in one go the unwanted elements are removed. It is also possible to use several criteria, e.g. only extracting data after a specific time

```
df[(df['LOCATION'] == 'Afghanistan') & (df['ELAPSED_TIME_SINCE_OUTBREAK'] > 2)]
```

Note that the parenthesis are necessary, otherwise the logical operation would fail.

2.7 Performing mathematical operations on DataFrames

When performing mathematical operations on DataFrames there are at least two strategies

- Extract columns from the DataFrame and perform mathematical operations on the columns using Numpy, leaving the original DataFrame intact
- To operate directly on the data in the DataFrame using the Pandas library

Speed and performance.

Using Pandas or Numpy should in principle be equally fast. The advice is to not worry about performance before it is necessary. Use the methods you are confident with, and try to be consistent. By consistent, we mean that if you have found one way of doing a certain operation stick to that one and try not to implement many different ways of doing the same thing.

We can always access the individual columns in a DataFrame by the syntax `df['column_name']`.

Example: mathematical operations on DataFrames.

1. Create a DataFrame with one column (`a`) containing ten thousand random uniformly distributed numbers between 0 and 1 (checkout `np.random.uniform`[3])
2. Add two new columns: one which all elements of `a` is squared and one where the sine function is applied to column `a`
3. Calculate the inverse of all the numbers in the DataFrame
4. Make a plot of the results (i.e. `a` vs `a*a`, and `a` vs `sin(a)`)

Solution.

1. First we make the DataFrame

```
import numpy as np
import pandas as pd
N=10000
a=np.random.uniform(0,1,size=N)
df=pd.DataFrame() # empty DataFrame
df['a']=a
```

If you like you could also try to use a dictionary. Next, we add the new columns

```
df['b']=df['a']*df['a'] # alternatively np.square(df['a'])
df['c']=np.sin(df['a'])
```

[3] https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html

1. The inverse of all the numbers in the DataFrame can be calculated by simply doing

```
1/df
```

Note: you can also do `df+df` and many other operations on the whole DataFrame.

1. To make plots there are several possibilities. Personally, I tend most of the time to use the `matplotlib`[4] library, simply because I know it quite well, but Pandas has a great deal of very simple methods you can use to generate nice plots with very few commands.

Matplotlib:

```
import matplotlib.pyplot as plt
plt.plot(df['a'],df['b'], '*', label='$a^2$')
plt.plot(df['a'],df['c'], '^', label='$\sin(a)$')
plt.legend()
plt.grid() # make small grid lines
plt.show()
```

Pandas plotting: First, let us try the built in plot command in Pandas

```
df.plot()
```

If you compare this plot with the previous plot, you will see that Pandas plots all columns versus the index columns, which is not what we want. But, we can set `a` to be the index column

```
df=df.set_index('a')
df.plot()
```

We can also make separate plots

```
df.plot(subplots=True)
```

or scatter plots

```
df=df.reset_index()
df.plot.scatter(x='a',y='b')
df.plot.scatter(x='a',y='c')
```

Note that we have to reset the index, otherwise there are no column named `a`.

[4] `https://matplotlib.org/`

2.8 Joining two DataFrames

Appending DataFrames. The DataFrame with the Covid-19 data in the previous section could have been created from two separate DataFrames, using `concat()`[5]. First, create two DataFrames

```
my_dict1={'LOCATION':7*['Afghanistan'],
'TIME':[a+dt.timedelta(days=i) for i in range(7)],
'ELAPSED_TIME_SINCE_OUTBREAK':[0, 1, 2, 3, 4, 5, 6],
'CONFIRMED':7*[1],
'DEATHS':7*[0],
'RECOVERED': 7*[0]}
my_dict2={'LOCATION':6*['Diamond Princess'],
'TIME':[b+dt.timedelta(days=i) for i in range(6)],
'ELAPSED_TIME_SINCE_OUTBREAK':[0, 1, 2, 3, 4, 5],
'CONFIRMED':[61, 61, 64, 135, 135, 175],
'DEATHS':6*[0],
'RECOVERED': 6*[0]}
df1=pd.DataFrame(my_dict1)
df2=pd.DataFrame(my_dict2)
```

Next, add them row wise (see figure 3)

```
df=pd.concat([df1,df2])
print(df) # to view
```

	LOCATION	TIME	ELAPSED_TIME_SINCE_OUTBREAK	CONFIRMED	DEATHS	RECOVERED
0	Afghanistan	2020-02-24 23:59:00	0	1	0	0
1	Afghanistan	2020-02-25 23:59:00	1	1	0	0
2	Afghanistan	2020-02-26 23:59:00	2	1	0	0
3	Afghanistan	2020-02-27 23:59:00	3	1	0	0
4	Afghanistan	2020-02-28 23:59:00	4	1	0	0
5	Afghanistan	2020-02-29 23:59:00	5	1	0	0
6	Afghanistan	2020-03-01 23:59:00	6	1	0	0
0	Diamond Princess	2020-02-07 23:59:00	0	61	0	0
1	Diamond Princess	2020-02-08 23:59:00	1	61	0	0
2	Diamond Princess	2020-02-09 23:59:00	2	64	0	0
3	Diamond Princess	2020-02-10 23:59:00	3	135	0	0
4	Diamond Princess	2020-02-11 23:59:00	4	135	0	0
5	Diamond Princess	2020-02-12 23:59:00	5	175	0	0

Figure 3: The result of `concat()`.

If you compare this DataFrame with the previous one, you will see that the index column is different. This is because when joining two DataFrames Pandas does not reset the index by default, doing `df=pd.concat([df1,df2],ignore_index=True)` resets the index. It is also possible to join DataFrames column vise

[5] `https://pandas.pydata.org/docs/reference/api/pandas.concat.html`

```
pd.concat([df1,df2],axis=1)
```

Merging DataFrames. In the previous example we had two non overlapping DataFrames (separate countries and times). It could also be the case that some of the data was overlapping e.g. continuing with the Covid-19 data, one could assume that there was one data set from one region and one from another region in the same country

```
my_dict1={'LOCATION':7*['Diamond Princess'],
'TIME':[b+dt.timedelta(days=i) for i in range(7)],
'ELAPSED_TIME_SINCE_OUTBREAK':[0, 1, 2, 3, 4, 5, 6],
'CONFIRMED':7*[1],
'DEATHS':7*[0],
'RECOVERED': 7*[0]}
my_dict2={'LOCATION':2*['Diamond Princess'],
'TIME':[b+dt.timedelta(days=i) for i in range(2)],
'ELAPSED_TIME_SINCE_OUTBREAK':[0, 1],
'CONFIRMED':[60, 60],
'DEATHS':2*[0],
'RECOVERED': 2*[0]}
df1=pd.DataFrame(my_dict1)
df2=pd.DataFrame(my_dict2)
```

If we do `pd.concat([df1,df2])` we will simply add all values after each other. What we want to do is to sum the number of confirmed, recovered and deaths for the same date. This can be done in several ways, but one way is to use `pd.DataFrame.merge()`[6].You can specify the columns to merge on, and choose `outer` which is union (all data from both frames) or `inner` which means the intersect (only data which you merge on that exists in both frames). After performing the commands

```
df1.merge(df2,on=['LOCATION','TIME'],how='outer')
df1.merge(df2,on=['LOCATION','TIME'],how='inner')
```

we get the results in figure 4

Clearly in this case we need to choose `outer`. In the merge process pandas adds an extra subscript `_x` and `_y` on columns that contains the same header name. We also need to sum those, which can be done as follows (see figure 5 for the final result)

```
df=df1.merge(df2,on=['LOCATION','TIME'],how='outer')
cols=['CONFIRMED','DEATHS', 'RECOVERED']
for col in cols:
    df[col]=df[[col+'_x',col+'_y']].sum(axis=1) # sum row elements
    df=df.drop(columns=[col+'_x',col+'_y']) # remove obsolete columns
```

[6] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html

	LOCATION	TIME	ELAPSED_TIME_SINCE_OUTBREAK_x	CONFIRMED_x	DEATHS_x	RECOVERED_x	ELAPSED_TIME_SINCE_OUTBREAK_y	CONFIRMED_y	DEATHS_y	RECOVERED_y
0	Diamond Princess	2020-02-07 23:59:00	0	1	0	0	0.0	60.0	0.0	0.0
1	Diamond Princess	2020-02-08 23:59:00	1	1	0	0	1.0	60.0	0.0	0.0
2	Diamond Princess	2020-02-09 23:59:00	2	1	0	0	NaN	NaN	NaN	NaN
3	Diamond Princess	2020-02-10 23:59:00	3	1	0	0	NaN	NaN	NaN	NaN
4	Diamond Princess	2020-02-11 23:59:00	4	1	0	0	NaN	NaN	NaN	NaN
5	Diamond Princess	2020-02-12 23:59:00	5	1	0	0	NaN	NaN	NaN	NaN
6	Diamond Princess	2020-02-13 23:59:00	6	1	0	0	NaN	NaN	NaN	NaN

	LOCATION	TIME	ELAPSED_TIME_SINCE_OUTBREAK_x	CONFIRMED_x	DEATHS_x	RECOVERED_x	ELAPSED_TIME_SINCE_OUTBREAK_y	CONFIRMED_y	DEATHS_y	RECOVERED_y
0	Diamond Princess	2020-02-07 23:59:00	0	1	0	0	0	60	0	0
1	Diamond Princess	2020-02-08 23:59:00	1	1	0	0	1	60	0	0

Figure 4: Merging to dataframes using `outer` (top) and `inner` (bottom).

```
# final clean up
df['ELAPSED_TIME_SINCE_OUTBREAK']=df['ELAPSED_TIME_SINCE_OUTBREAK_x']
df=df.drop(columns=['ELAPSED_TIME_SINCE_OUTBREAK_y','ELAPSED_TIME_SINCE_OUTBREAK_x'])
```

	LOCATION	TIME	CONFIRMED	DEATHS	RECOVERED	ELAPSED_TIME_SINCE_OUTBREAK
0	Diamond Princess	2020-02-07 23:59:00	61.0	0.0	0.0	0
1	Diamond Princess	2020-02-08 23:59:00	61.0	0.0	0.0	1
2	Diamond Princess	2020-02-09 23:59:00	1.0	0.0	0.0	2
3	Diamond Princess	2020-02-10 23:59:00	1.0	0.0	0.0	3
4	Diamond Princess	2020-02-11 23:59:00	1.0	0.0	0.0	4
5	Diamond Princess	2020-02-12 23:59:00	1.0	0.0	0.0	5
6	Diamond Princess	2020-02-13 23:59:00	1.0	0.0	0.0	6

Figure 5: Result of outer merging and summing.

2.9 Grouping and summing

In many cases you want to perform a mathematical operation on some columns. Lets say we wanted to find the total number of confirmed, deaths and recovered cases in the full database. This can be done by `df[['CONFIRMED','DEATHS','RECOVERED']].sum()` or accessing each column individually and sum each of them e.g. `np.sum(df['CONFIRMED'])`. But, in some cases we have a big database, maybe of all the countries in the world, and we want to get the total number of cases in each country. This can be done very elegantly with the command `pd.DataFrame.groupby()`[7] (see figure 6 for final result)

```
df.groupby('LOCATION').sum()
```

What this command do is to sum all columns with the same location, and drop columns that cannot be summed.

[7] `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html`

LOCATION	ELAPSED_TIME_SINCE_OUTBREAK	CONFIRMED	DEATHS	RECOVERED
Afghanistan	21	7	0	0
Diamond Princess	15	631	0	0

Figure 6: The results of `df.groupby('LOCATION').sum()`.

2.10 Simple statistics in Pandas

At the end it is worth mentioning the built in methods `pd.DataFrame.mean`, `pd.DataFrame.median`, `pd.DataFrame.std` which calculates the mean, median and standard deviation on the columns in the DataFrame where it make sense (i.e. avoid strings and dates). To get all these values in one go (and a few more) on can also use `pd.DataFrame.describe()`

```
df.describe()
```

The output is shown in figure 7

```
df.describe()
✓ 0.1s
```

	CONFIRMED	DEATHS	RECOVERED	ELAPSED_TIME_SINCE_OUTBREAK
count	7.000000	7.0	7.0	7.000000
mean	18.142857	0.0	0.0	3.000000
std	29.277002	0.0	0.0	2.160247
min	1.000000	0.0	0.0	0.000000
25%	1.000000	0.0	0.0	1.500000
50%	1.000000	0.0	0.0	3.000000
75%	31.000000	0.0	0.0	4.500000
max	61.000000	0.0	0.0	6.000000

Figure 7: Output from the describe command.

References