

Importing pandas

Getting started and checking your pandas setup

Difficulty: *easy*

1. Import pandas under the alias `pd`.

```
In [ ]:
```

2. Print the version of pandas that has been imported.

```
In [ ]:
```

3. Print out all the version information of the libraries that are required by the pandas library.

```
In [ ]:
```

DataFrame basics

A few of the fundamental routines for selecting, sorting, adding and aggregating data in DataFrames

Difficulty: *easy*

Note: remember to import numpy using:

```
import numpy as np
```

Consider the following Python dictionary `data` and Python list `labels`:

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

(This is just some meaningless data I made up with the theme of animals and trips to a vet)

4. Create a DataFrame `df` from this dictionary `data` which has the index `labels`.

```
In [ ]:
```

```
import numpy as np
```

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}
```

```
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

5. Display a summary of the basic information about this DataFrame and its data (*hint: there is a single method that can be called on the DataFrame*).

```
In [ ]:
```

6. Return the first 3 rows of the DataFrame `df`.

```
In [ ]:
```

7. Select just the 'animal' and 'age' columns from the DataFrame `df`.

```
In [ ]:
```

8. Select the data in rows `[3, 4, 8]` and in columns `['animal', 'age']`.

```
In [ ]:
```

9. Select only the rows where the number of visits is greater than 3.

```
In [ ]:
```

10. Select the rows where the age is missing, i.e. it is `NaN`.

```
In [ ]:
```

11. Select the rows where the animal is a cat *and* the age is less than 3.

```
In [ ]:
```

12. Select the rows the age is between 2 and 4 (inclusive).

```
In [ ]:
```

13. Change the age in row 'f' to 1.5.

```
In [ ]:
```

14. Calculate the sum of all visits in `df` (i.e. find the total number of visits).

```
In [ ]:
```

15. Calculate the mean age for each different animal in `df`.

```
In [ ]:
```

16. Append a new row 'k' to `df` with your choice of values for each column. Then delete that row to return the original DataFrame.

```
In [ ]:
```

17. Count the number of each type of animal in `df`.

```
In [ ]:
```

18. Sort `df` first by the values in the 'age' in *descending* order, then by the value in the 'visits' column in *ascending* order (so row `i` should be first, and row `d` should be last).

```
In [ ]:
```

19. The 'priority' column contains the values 'yes' and 'no'. Replace this column with a column of boolean values: 'yes' should be `True` and 'no' should be `False`.

```
In [ ]:
```

20. In the 'animal' column, change the 'snake' entries to 'python'.

```
In [ ]:
```

21. For each animal type and each number of visits, find the mean age. In other words, each row is an animal, each column is a number of visits and the values are the mean ages (*hint: use a pivot table*).

```
In [ ]:
```

DataFrames: beyond the basics

Slightly trickier: you may need to combine two or more methods to get the right answer

Difficulty: *medium*

The previous section was tour through some basic but essential DataFrame operations. Below are some ways that you might need to cut your data, but for which there is no single "out of the box" method.

22. You have a DataFrame `df` with a column 'A' of integers. For example:

```
df = pd.DataFrame({'A': [1, 2, 2, 3, 4, 5, 5, 6, 7, 7]})
```

How do you filter out rows which contain the same integer as the row immediately above?

You should be left with a column containing the following values:

```
1, 2, 3, 4, 5, 6, 7
```

```
In [ ]:
```

23. Given a DataFrame of numeric values, say

```
df = pd.DataFrame(np.random.random(size=(5, 3))) # a 5x3 frame of float values
```

how do you subtract the row mean from each element in the row?

```
In [ ]:
```

24. Suppose you have DataFrame with 10 columns of real numbers, for example:

```
df = pd.DataFrame(np.random.random(size=(5, 10)), columns=list('abcdefghij'))
```

Which column of numbers has the smallest sum? Return that column's label.

```
In [ ]:
```

25. How do you count how many unique rows a DataFrame has (i.e. ignore all rows that are duplicates)? As input, use a DataFrame of zeros and ones with 10 rows and 3 columns.

```
df = pd.DataFrame(np.random.randint(0, 2, size=(10, 3)))
```

```
In [ ]:
```

The next three puzzles are slightly harder.

26. In the cell below, you have a DataFrame `df` that consists of 10 columns of floating-point numbers. Exactly 5 entries in each row are NaN values.

For each row of the DataFrame, find the *column* which contains the *third* NaN value.

You should return a Series of column labels: `e, c, d, h, d`

```
In [8]:
```

```
import pandas as pd
import numpy as np
nan = np.nan

data = [[0.04, nan, nan, 0.25, nan, 0.43, 0.71, 0.51, nan, nan],
        [ nan, nan, nan, 0.04, 0.76, nan, nan, nan, 0.67, 0.76, 0.16],
        [ nan, nan, 0.5, nan, 0.31, 0.4, nan, nan, 0.24, 0.01],
        [0.49, nan, nan, 0.62, 0.73, 0.26, 0.85, nan, nan, nan],
        [ nan, nan, 0.41, nan, 0.05, nan, 0.61, nan, 0.48, 0.68]]

columns = list('abcdefghij')

df = pd.DataFrame(data, columns=columns)
```

```
# write a solution to the question here
```

```
Out[8]:
```

```
   a    b    c    d    e    f    g    h    i    j
0  0.04 NaN NaN  0.25 NaN  0.43  0.71  0.51 NaN NaN
1  NaN NaN NaN  0.04 0.76 NaN NaN  0.67 0.76 0.16
2  NaN NaN 0.50 NaN 0.31 0.40 NaN NaN 0.24 0.01
3  0.49 NaN NaN 0.62 0.73 0.26 0.85 NaN NaN NaN
4  NaN NaN 0.41 NaN 0.05 NaN 0.61 NaN 0.48 0.68
```

```
In [12]:
```

```
(df.isnull().cumsum(axis=1)==3).idxmax(axis=1)
```

```
Out[12]:
```

```
0    e
1    c
2    d
3    h
4    d
dtype: object
```

27. A DataFrame has a column of groups 'grps' and and column of integer values 'vals':

```
df = pd.DataFrame({'grps': list('aabbcaabcccbcb'),
                   'vals': [12,345,3,1,45,14,4,52,54,23,235,21,57,3,87]})
```

For each group, find the sum of the three greatest values. You should end up with the answer as follows:

```
grps
a    409
b    156
c    345
```

```
In [ ]:
```

```
df = pd.DataFrame({'grps': list('aabbcaabcccbcb'),
                   'vals': [12,345,3,1,45,14,4,52,54,23,235,21,57,3,87]})

# write a solution to the question here
```

28. The DataFrame `df` constructed below has two integer columns 'A' and 'B'. The values in 'A' are between 1 and 100 (inclusive).

For each group of 10 consecutive integers in 'A' (i.e. `(0, 10]`, `(10, 20]`, ...), calculate the sum of the corresponding values in column 'B'.

The answer should be a Series as follows:

```
A
(0, 10]    635
(10, 20]   360
(20, 30]   315
(30, 40]   306
(40, 50]   750
(50, 60]   284
(60, 70]   424
(70, 80]   526
(80, 90]   835
(90, 100]  852
```

```
In [ ]:
```

```
df = pd.DataFrame(np.random.RandomState(8765).randint(1, 101, size=(100, 2)), columns = ["A", "B"])

# write a solution to the question here
```

DataFrames: harder problems

These might require a bit of thinking outside the box...

...but all are solvable using just the usual pandas/NumPy methods (and so avoid using explicit `for` loops).

Difficulty: *hard*

29. Consider a DataFrame `df` where there is an integer column 'X':

```
df = pd.DataFrame({'X': [7, 2, 0, 3, 4, 2, 5, 0, 3, 4]})
```

For each value, count the difference back to the previous zero (or the start of the Series, whichever is closer). These values should therefore be

```
[1, 2, 0, 1, 2, 3, 4, 0, 1, 2]
```

Make this a new column 'Y':

```
In [ ]:
```

```
[29-32]
```

```
In [2]:
```

```
import pandas as pd
df = pd.DataFrame({'X': [7, 2, 0, 3, 4, 2, 5, 0, 3, 4]})
```

```
In [3]:
```

```
df
```

```
Out[3]:
```

```
   X
0  7
1  2
2  0
3  3
4  4
5  2
6  5
7  0
8  3
9  4
```

```
In [ ]:
```

30. Consider the DataFrame constructed below which contains rows and columns of numerical data.

Create a list of the column-row index locations of the 3 largest values in this DataFrame. In this case, the answer should be:

```
[(5, 7), (6, 4), (2, 5)]
```

```
In [ ]:
```

```
df = pd.DataFrame(np.random.RandomState(30).randint(1, 101, size=(8, 8)))
```

31. You are given the DataFrame below with a column of group IDs, 'grps', and a column of corresponding integer values, 'vals':

```
df = pd.DataFrame({'vals': np.random.RandomState(31).randint(-30, 30, size=15),
                   'grps': np.random.RandomState(31).choice(["A", "B"], 15)})
```

Create a new column 'patched_vals' which contains the same values as the 'vals' any negative values in 'vals' with the group mean:

```
vals  grps  patched_vals
0    -12    A         13.6
1     -7    B         28.0
2    -14    A         13.6
3     -4    A          4.0
4     -7    A         13.6
5     -2    B         28.0
6     -8    A         13.6
7     -1    A         13.6
8     -8    A          8.0
9     -2    B         28.0
10    28    A         28.0
11    12    A         12.0
12    16    A         16.0
13   -24    A         13.6
14   -12    A         13.6
```

```
In [ ]:
```

32. Implement a rolling mean over groups with window size 3, which ignores NaN value. For example consider the following DataFrame:

```
>>> df = pd.DataFrame({'group': ['aabbbaabcccbabab'],
                       'value': [1, 2, 3, np.nan, 2, 3, np.nan, 1, 7, 3, np.nan, 8]})

>>> df
  group  value
0     a     1.0
1     a     2.0
2     b     3.0
3     b    NaN
4     a     2.0
5     b     3.0
6     b    NaN
7     b     1.0
8     a     7.0
9     b     3.0
10    a    NaN
11    b     8.0
```

The goal is to compute the Series:

```
0    1.000000
1    1.500000
2    3.000000
3    3.000000
4    1.666667
5    3.000000
6    3.000000
7    2.000000
8    3.666667
9    2.000000
10   4.500000
11   4.000000
```

E.g. the first window of size three for group 'b' has values 3.0, NaN and 3.0 and occurs at row index 5. Instead of being NaN the value in the new column at this row index should be 3.0 (just the two non-NaN values are used to compute the mean (3+3)/2)

```
In [ ]:
```

Series and DatetimeIndex

Exercises for creating and manipulating Series with datetime data

Difficulty: *easy/medium*

pandas is fantastic for working with dates and times. These puzzles explore some of this functionality.

33. Create a DatetimeIndex that contains each business day of 2015 and use it to index a Series of random numbers. Let's call this Series `s`.

```
In [ ]:
```

34. Find the sum of the values in `s` for every Wednesday.

```
In [ ]:
```

35. For each calendar month in `s`, find the mean of values.

```
In [ ]:
```

36. For each group of four consecutive calendar months in `s`, find the date on which the highest value occurred.

```
In [ ]:
```

37. Create a DatetimeIndex consisting of the third Thursday in each month for the years 2015 and 2016.

```
In [ ]:
```

Cleaning Data

Making a DataFrame easier to work with

Difficulty: *easy/medium*

It happens all the time: someone gives you data containing malformed strings, Python, lists and missing data. How do you tidy it up so you can get on with the analysis?

Take this monstrosity as the DataFrame to use in the following puzzles:

```
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_StockhOlM',
                              'Budapest_PaRis', 'Brussels_lonDon'],
                  'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
                  'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
                  'Airline': ['KLM(1)', '<Air France> (12)', '(British Airways. )',
                              '12. Air France', '"Swiss Air"']})
```

Formatted, it looks like this:

```
   From_To  FlightNumber  RecentDelays  Airline
0  LoNDon_paris    10045.0      [23, 47]    KLM(1)
1  MAdrid_miLAN         NaN           []  <Air France> (12)
2  londON_StockhOlM    10065.0  [24, 43, 87]  (British Airways. )
3  Budapest_PaRis         NaN          [13]      12. Air France
4  Brussels_lonDon    10085.0      [67, 32]    "Swiss Air"
```

(It's some flight data I made up; it's not meant to be accurate in any way.)

```
In [20]:
```

```
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_StockhOlM',
                              'Budapest_PaRis', 'Brussels_lonDon'],
                  'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
                  'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
                  'Airline': ['KLM(1)', '<Air France> (12)', '(British Airways. )',
                              '12. Air France', '"Swiss Air"']})
```

38. Some values in the the `FlightNumber` column are missing (they are `NaN`). These numbers are meant to increase by 10 with each row so 10055 and 10075 need to be put in place. Modify `df` to fill in these missing numbers and make the column an integer column (instead of a float column).

```
In [ ]:
```

39. The `From_To` column would be better as two separate columns! Split each string on the underscore delimiter `_` to give a new temporary DataFrame called 'temp' with the correct values. Assign the correct column names 'From' and 'To' to this temporary DataFrame.

```
In [ ]:
```

40. Notice how the capitalisation of the city names is all mixed up in this temporary DataFrame 'temp'. Standardise the strings so that only the first letter is uppercase (e.g. 'londON' should become 'London')

```
In [ ]:
```

41. Delete the `From_To` column from `df` and attach the temporary DataFrame 'temp' from the previous questions.

```
In [ ]:
```

42. In the `Airline` column, you can see some extra punctuation and symbols have appeared around the airline names. Pull out just the airline name. E.g. '(British Airways.)' should become 'British Airways'.

```
In [21]:
```

```
df
```

```
Out[21]:
```

```
   From_To  FlightNumber  RecentDelays  Airline
0  LoNDon_paris    10045.0      [23, 47]    KLM(1)
1  MAdrid_miLAN         NaN           []    Air France
2  londON_StockhOlM    10065.0  [24, 43, 87]  British Airways
3  Budapest_PaRis         NaN          [13]      12. Air France
4  Brussels_lonDon    10085.0      [67, 32]    Swiss Air
```

```
In [22]:
```

```
df['Airline'] = df['Airline'].str.extract('([a-zA-ZAa]+)')
```

```
Out[22]:
```

```
   From_To  FlightNumber  RecentDelays  Airline
0  LoNDon_paris    10045.0      [23, 47]    KLM
1  MAdrid_miLAN         NaN           []    Air France
2  londON_StockhOlM    10065.0  [24, 43, 87]  British Airways
3  Budapest_PaRis         NaN          [13]      12. Air France
4  Brussels_lonDon    10085.0      [67, 32]    Swiss Air
```

43. In the `RecentDelays` column, the values have been entered into the DataFrame as a list. We would like each first value in its own column, each second value in its own column, and so on. If there isn't an Nth value, the value should be `NaN`.

Expand the Series of lists into a DataFrame named `delays`, rename the columns `delay_1`, `delay_2`, etc. and replace the unwanted `RecentDelays` column in `df` with `delays`.

```
In [ ]:
```

The DataFrame should look much better now.

```
   FlightNumber  Airline  From  To  delay_1  delay_2  delay_3
0    10045      KLM      London  Paris    23.0    47.0      NaN
1    10055      Air France  Madrid  Paris    NaN      NaN      NaN
2    10065      British Airways  London  Stockholm    24.0    43.0    87.0
3    10075      Air France  Budapest  Paris    13.0      NaN      NaN
4    10085      Swiss Air  Brussels  London    67.0    32.0      NaN
```

```
In [ ]:
```