▾ pandas

▾ Importing pandas

Getting started and checking your pandas setup

Difficulty: *easy*

1. Import pandas under the name `pd` .

```
import pandas as pd
```

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

```
pd.__version__
```

```
'1.3.5'
```

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

```
pd.show_versions()
```

```
INSTALLED VERSIONS
------------------
commit          : 66e3805b8cabe977f40c05259cc3fcf7ead5687d
python          : 3.7.15.final.0
python-bits     : 64
OS              : Linux
OS-release      : 5.10.133+
Version         : #1 SMP Fri Aug 26 08:44:51 UTC 2022
machine         : x86_64
processor       : x86_64
byteorder       : little
LC_ALL          : None
LANG            : en_US.UTF-8
LOCALE          : en_US.UTF-8

pandas          : 1.3.5
numpy           : 1.21.6
pytz            : 2022.5
dateutil        : 2.8.2
pip             : 21.1.3
setuptools      : 57.4.0
Cython          : 0.29.32
```

```
pytest            : 3.6.4
hypothesis        : None
sphinx            : 1.8.6
blosc             : None
feather           : 0.4.1
xlsxwriter        : None
lxml.etree        : 4.9.1
html5lib          : 1.0.1
pymysql           : None
psycopg2          : 2.9.5 (dt dec pq3 ext lo64)
jinja2            : 2.11.3
IPython           : 7.9.0
pandas_datareader : 0.9.0
bs4               : 4.6.3
bottleneck        : None
fsspec            : 2022.10.0
fastparquet       : None
gcsfs             : None
matplotlib        : 3.2.2
numexpr           : 2.8.4
odfpy             : None
openpyxl          : 3.0.10
pandas_gbq        : 0.13.3
pyarrow           : 6.0.1
pyxlsb            : None
s3fs              : None
scipy             : 1.7.3
sqlalchemy        : 1.4.42
tables            : 3.7.0
tabulate          : 0.8.10
xarray            : 0.20.2
xlrd              : 1.1.0
xlwt              : 1.3.0
numba             : 0.56.3
```

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`.

```python
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']

df = pd.DataFrame(data, index=labels)
print(df)
```

```
      animal  age  visits priority
a        cat  2.5       1      yes
b        cat  3.0       3      yes
c      snake  0.5       2       no
d        dog  NaN       3      yes
e        dog  5.0       2       no
f        cat  2.0       3       no
g      snake  4.5       1       no
h        cat  NaN       1      yes
i        dog  7.0       2       no
j        dog  3.0       1       no
```

5. Display a summary of the basic information about this DataFrame and its data.

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   animal    10 non-null     object
 1   age       8 non-null      float64
 2   visits    10 non-null     int64
 3   priority  10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
```

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

```
df.iloc[:3]
```

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no

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

```
df.loc[:, ['animal', 'age']]
```

	animal	age
a	cat	2.5
b	cat	3.0
c	snake	0.5
d	dog	NaN
e	dog	5.0
f	cat	2.0
g	snake	4.5
h	cat	NaN
i	dog	7.0
j	dog	3.0

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

```
df.loc[df.index[[3, 4, 8]], ['animal', 'age']]
```

	animal	age
d	dog	NaN
e	dog	5.0
i	dog	7.0

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

```
df[df['visits'] > 3]
```

	animal	age	visits	priority

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

```
df[df['age'].isnull()]
```

	animal	age	visits	priority
d	dog	NaN	3	yes
h	cat	NaN	1	yes

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

```
df[(df['animal'] == 'cat') & (df['age'] < 3)]
```

	animal	age	visits	priority
a	cat	2.5	1	yes
f	cat	2.0	3	no

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

```
df[df['age'].between(2, 4)]
```

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
f	cat	2.0	3	no
j	dog	3.0	1	no

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

```
df.loc['f', 'age'] = 1.5
```

14. Calculate the sum of all visits (the total number of visits).

```
df['visits'].sum()
```

```
    19
```

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

```
df.groupby('animal')['age'].mean()
```

```
    animal
    cat      2.333333
    dog      5.000000
    snake    2.500000
    Name: age, dtype: float64
```

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.

```
df.loc['k'] = [5.5, 'dog', 'no', 2]

# and then deleting the new row...

df = df.drop('k')
```

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

```
df['animal'].value_counts()
```

```
    cat      4
    dog      4
    snake    2
    Name: animal, dtype: int64
```

18. Sort `df` first by the values in the 'age' in *decending* order, then by the value in the 'visit' column in *ascending* order.

```
df.sort_values(by=['age', 'visits'], ascending=[False, True])
```

	animal	age	visits	priority
i	dog	7.0	2	no
e	dog	5.0	2	no
g	snake	4.5	1	no
j	dog	3.0	1	no
b	cat	3.0	3	yes
a	cat	2.5	1	yes
f	cat	1.5	3	no
c	snake	0.5	2	no

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`.

```
df['priority'] = df['priority'].map({'yes': True, 'no': False})
```

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

```
df['animal'] = df['animal'].replace('snake', 'python')
```

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).

```
df.pivot_table(index='animal', columns='visits', values='age', aggfunc='mean')
```

visits	1	2	3
animal			
cat	2.5	NaN	2.25
dog	3.0	6.0	NaN
python	4.5	0.5	NaN

▾ 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"

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, 5, 6, 7, 7]})
```

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

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

df.loc[df['A'].shift() != df['A']]

# Alternatively, we could use drop_duplicates() here. Note
# that this removes *all* duplicates though, so it won't
# work as desired if A is [1, 1, 2, 2, 1, 1] for example.

df.drop_duplicates(subset='A')
```

	A
0	1
1	2
3	3
4	4
5	5
8	6
9	7

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?

```
df = pd.DataFrame(np.random.random(size=(5, 3)))
```

```
df.sub(df.mean(axis=1), axis=0)
```

	0	1	2
0	-0.232296	0.349936	-0.117639
1	-0.338420	-0.149314	0.487734
2	0.272257	-0.157685	-0.114573
3	0.056414	0.026204	-0.082618
4	-0.301327	0.038646	0.262681

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? (Find that column's label.)

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

df.sum().idxmin()
```

```
    'h'
```

25. How do you count how many unique rows a DataFrame has (i.e. ignore all rows that are duplicates)?

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

len(df) - df.duplicated(keep=False).sum()

# or perhaps more simply...

len(df.drop_duplicates(keep=False))
```

```
    5
```

The next three puzzles are slightly harder...

26. You have a DataFrame that consists of 10 columns of floating--point numbers. Suppose that 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.)

```
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, 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)


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

```
    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 numbers 'vals'. For example:

```
 df = pd.DataFrame({'grps': list('aaabbcaabcccbbc'),
                    '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.

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

df.groupby('grps')['vals'].nlargest(3).sum(level=0)
```

```
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: FutureWarning: Using the
      after removing the cwd from sys.path.
    grps
    a    409
    b    156
    c    345
    Name: vals, dtype: int64
```

28. A DataFrame 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'.

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

```
df.groupby(pd.cut(df['A'], np.arange(0, 101, 10)))['B'].sum()
```

```
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
Name: B, dtype: int64
```

▾ 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'.

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

izero = np.r_[-1, (df == 0).values.nonzero()[0]]  # indices of zeros
idx = np.arange(len(df))
y = df['X'] != 0
df['Y'] = idx - izero[np.searchsorted(izero - 1, idx) - 1]

# http://stackoverflow.com/questions/30730981/how-to-count-distance-to-the-previous-zero-in-p
# credit: Behzad Nouri
```

Here's an alternative approach based on a [cookbook recipe](#):

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

x = (df['X'] != 0).cumsum()
y = x != x.shift()
df['Y'] = y.groupby((y != y.shift()).cumsum()).cumsum()
```

30. Consider a DataFrame containing rows and columns of purely numerical data. Create a list of the row-column index locations of the 3 largest values.

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

df.unstack().sort_values()[-3:].index.tolist()

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

31. Given a DataFrame with a column of group IDs, 'grps', and a column of corresponding integer values, 'vals', replace any negative values in 'vals' with the group mean.

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

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': list('aabbabbbabab'),
                       '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)

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

def replace(group):
    mask = group<0
    group[mask] = group[~mask].mean()
    return group

df.groupby(['grps'])['vals'].transform(replace)
```

```
 0      13.6
 1      28.0
 2      13.6
 3       4.0
 4      13.6
 5      28.0
 6      13.6
 7      13.6
 8       8.0
 9      28.0
10      28.0
11      12.0
12      16.0
13      13.6
14      13.6
Name: vals, dtype: float64
```

▾ 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`.

```
dti = pd.date_range(start='2015-01-01', end='2015-12-31', freq='B')
s = pd.Series(np.random.rand(len(dti)), index=dti)
s
```

```
    2015-01-01    0.609538
    2015-01-02    0.881374
    2015-01-05    0.832511
    2015-01-06    0.028224
    2015-01-07    0.141001
                   ...
    2015-12-25    0.685694
    2015-12-28    0.874250
    2015-12-29    0.422635
    2015-12-30    0.133565
    2015-12-31    0.788243
    Freq: B, Length: 261, dtype: float64
```

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

```
s[s.index.weekday == 2].sum()
```

```
    25.806536514886236
```

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

```
s.resample('M').mean()
```

```
    2015-01-31    0.449417
    2015-02-28    0.456119
    2015-03-31    0.511630
    2015-04-30    0.511415
    2015-05-31    0.487637
    2015-06-30    0.466271
    2015-07-31    0.514411
```

```
      2015-08-31     0.512055
      2015-09-30     0.506418
      2015-10-31     0.511358
      2015-11-30     0.475331
      2015-12-31     0.435428
      Freq: M, dtype: float64
```

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

```
s.groupby(pd.Grouper(freq='4M')).idxmax()
```

```
      2015-01-31     2015-01-15
      2015-05-31     2015-02-03
      2015-09-30     2015-08-24
      2016-01-31     2015-12-15
      Freq: 4M, dtype: datetime64[ns]
```

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

```
pd.date_range('2015-01-01', '2016-12-31', freq='WOM-3THU')
```

```
      DatetimeIndex(['2015-01-15', '2015-02-19', '2015-03-19', '2015-04-16',
                     '2015-05-21', '2015-06-18', '2015-07-16', '2015-08-20',
                     '2015-09-17', '2015-10-15', '2015-11-19', '2015-12-17',
                     '2016-01-21', '2016-02-18', '2016-03-17', '2016-04-21',
                     '2016-05-19', '2016-06-16', '2016-07-21', '2016-08-18',
                     '2016-09-15', '2016-10-20', '2016-11-17', '2016-12-15'],
                    dtype='datetime64[ns]', freq='WOM-3THU')
```

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(!)', '<Air France> (12)', '(British Airways. )',
                    '12. Air France', '"Swiss Air"']})
```

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

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

```
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(!)', ' (12)', '(British Airways. )',
                            '12. Air France', '"Swiss Air"']})

df['FlightNumber'] = df['FlightNumber'].interpolate().astype(int)
df
```

	From_To	FlightNumber	RecentDelays	Airline
0	LoNDon_paris	10045	[23, 47]	KLM(!)
1	MAdrid_miLAN	10055	[]	(12)
2	londON_StockhOlm	10065	[24, 43, 87]	(British Airways.)
3	Budapest_PaRis	10075	[13]	12. Air France
4	Brussels_londOn	10085	[67, 32]	"Swiss Air"

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 with the correct values. Assign the correct column names to this temporary DataFrame.

```
temp = df.From_To.str.split('_', expand=True)
temp.columns = ['From', 'To']
temp
```

	From	To	
0	LoNDon	paris	

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

| **4** | Brussels | londOn |

```
temp['From'] = temp['From'].str.capitalize()
temp['To'] = temp['To'].str.capitalize()
temp
```

	From	To	
0	London	Paris	
1	Madrid	Milan	
2	London	Stockholm	
3	Budapest	Paris	
4	Brussels	London	

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

```
df = df.drop('From_To', axis=1)
df = df.join(temp)
df
```

	FlightNumber	RecentDelays	Airline	From	To	
0	10045	[23, 47]	KLM(!)	London	Paris	
1	10055	[]	(12)	Madrid	Milan	
2	10065	[24, 43, 87]	(British Airways.)	London	Stockholm	
3	10075	[13]	12. Air France	Budapest	Paris	
4	10085	[67, 32]	"Swiss Air"	Brussels	London	

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

```
df['Airline'] = df['Airline'].str.extract('([a-zA-Z\s]+)', expand=False).str.strip()
# note: using .strip() gets rid of any leading/trailing spaces
df
```

	FlightNumber	RecentDelays	Airline	From	To
0	10045	[23, 47]	KLM	London	Paris
1	10055	[]		Madrid	Milan
2	10065	[24, 43, 87]	British Airways	London	Stockholm
3	10075	[13]	Air France	Budapest	Paris
4	10085	[67, 32]	Swiss Air	Brussels	London

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`.

```
delays = df['RecentDelays'].apply(pd.Series)

delays.columns = ['delay_{}'.format(n) for n in range(1, len(delays.columns)+1)]

df = df.drop('RecentDelays', axis=1).join(delays)

df
```

	FlightNumber	Airline	From	To	delay_1	delay_2	delay_3
0	10045	KLM	London	Paris	23.0	47.0	NaN
1	10055		Madrid	Milan	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

The DataFrame should look much better now.

▾ Using MultiIndexes

Go beyond flat DataFrames with additional index levels

Difficulty: *medium*

Previous exercises have seen us analysing data from DataFrames equipped with a single index level. However, pandas also gives you the possibilty of indexing your data using *multiple* levels. This is very much like adding new dimensions to a Series or a DataFrame. For example, a Series is 1D, but by using a MultiIndex with 2 levels we gain of much the same functionality as a 2D DataFrame.

The set of puzzles below explores how you might use multiple index levels to enhance data analysis.

To warm up, we'll look make a Series with two index levels.

44. Given the lists `letters = ['A', 'B', 'C']` and `numbers = list(range(10))`, construct a MultiIndex object from the product of the two lists. Use it to index a Series of random numbers. Call this Series `s`.

```
letters = ['A', 'B', 'C']
numbers = list(range(10))

mi = pd.MultiIndex.from_product([letters, numbers])
s = pd.Series(np.random.rand(30), index=mi)
s
```

```
A  0    0.469096
   1    0.750318
   2    0.729866
   3    0.560741
   4    0.770753
   5    0.805680
   6    0.105796
   7    0.693090
   8    0.334203
   9    0.154230
B  0    0.389993
   1    0.820586
   2    0.682819
   3    0.805714
   4    0.249166
   5    0.085652
   6    0.767687
   7    0.039850
   8    0.234391
   9    0.345153
C  0    0.374892
   1    0.386239
   2    0.824587
   3    0.020665
   4    0.092250
   5    0.346555
```

```
     6     0.117335
     7     0.819907
     8     0.227121
     9     0.027275
dtype: float64
```

45. Check the index of `s` is lexicographically sorted (this is a necessary proprty for indexing to work correctly with a MultiIndex).

```
s.index.is_lexsorted()

# or more verbosely...
s.index.lexsort_depth == s.index.nlevels
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: MultiInde
  """"Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: FutureWarning: MultiInde
  after removing the cwd from sys.path.
True
```

46. Select the labels `1`, `3` and `6` from the second level of the MultiIndexed Series.

```
s.loc[:, [1, 3, 6]]
```

```
A   1     0.750318
    3     0.560741
    6     0.105796
B   1     0.820586
    3     0.805714
    6     0.767687
C   1     0.386239
    3     0.020665
    6     0.117335
dtype: float64
```

47. Slice the Series `s`; slice up to label 'B' for the first level and from label 5 onwards for the second level.

```
s.loc[pd.IndexSlice[:'B', 5:]]

# or equivalently without IndexSlice...
s.loc[slice(None, 'B'), slice(5, None)]
```

```
A   5     0.805680
    6     0.105796
    7     0.693090
    8     0.334203
```

```
      9     0.154230
   B  5     0.085652
      6     0.767687
      7     0.039850
      8     0.234391
      9     0.345153
   dtype: float64
```

48. Sum the values in `s` for each label in the first level (you should have Series giving you a total for labels A, B and C).

```
s.sum(level=0)
```

```
   /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Using the
     """Entry point for launching an IPython kernel.
   A    5.373774
   B    4.421010
   C    3.236826
   dtype: float64
```

49. Suppose that `sum()` (and other methods) did not accept a `level` keyword argument. How else could you perform the equivalent of `s.sum(level=1)` ?

```
s.unstack().sum(axis=0)
```

```
   0    1.233981
   1    1.957143
   2    2.237272
   3    1.387120
   4    1.112169
   5    1.237887
   6    0.990817
   7    1.552847
   8    0.795715
   9    0.526658
   dtype: float64
```

50. Exchange the levels of the MultiIndex so we have an index of the form (letters, numbers). Is this new Series properly lexsorted? If not, sort it.

```
new_s = s.swaplevel(0, 1)

if not new_s.index.is_lexsorted():
    new_s = new_s.sort_index()

new_s
```

```
     /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: MultiInde
       This is separate from the ipykernel package so we can avoid doing imports until
     0   A      0.469096
         B      0.389993
         C      0.374892
     1   A      0.750318
         B      0.820586
         C      0.386239
     2   A      0.729866
         B      0.682819
         C      0.824587
     3   A      0.560741
         B      0.805714
         C      0.020665
     4   A      0.770753
         B      0.249166
         C      0.092250
     5   A      0.805680
         B      0.085652
         C      0.346555
     6   A      0.105796
         B      0.767687
         C      0.117335
     7   A      0.693090
         B      0.039850
         C      0.819907
     8   A      0.334203
         B      0.234391
         C      0.227121
     9   A      0.154230
         B      0.345153
         C      0.027275
     dtype: float64
```

Minesweeper

Generate the numbers for safe squares in a Minesweeper grid

Difficulty: *medium* to *hard*

If you've ever used an older version of Windows, there's a good chance you've played with
Minesweeper. If you're not familiar with the game, imagine a grid of squares: some of these
squares conceal a mine. If you click on a mine, you lose instantly. If you click on a safe square, you
reveal a number telling you how many mines are found in the squares that are immediately
adjacent. The aim of the game is to uncover all squares in the grid that do not contain a mine.

In this section, we'll make a DataFrame that contains the necessary data for a game of
Minesweeper: coordinates of the squares, whether the square contains a mine and the number of
mines found on adjacent squares.

51. Let's suppose we're playing Minesweeper on a 5 by 4 grid, i.e.

```
X = 5
Y = 4
```

To begin, generate a DataFrame `df` with two columns, `'x'` and `'y'` containing every coordinate for this grid. That is, the DataFrame should start:

```
   x  y
0  0  0
1  0  1
2  0  2
```

```
X = 5
Y = 4

p = pd.core.reshape.util.cartesian_product([np.arange(X), np.arange(Y)])
df = pd.DataFrame(np.asarray(p).T, columns=['x', 'y'])
df
```

	x	y
0	0	0
1	0	1
2	0	2
3	0	3
4	1	0

52. For this DataFrame `df`, create a new column of zeros (safe) and ones (mine). The probability of a mine occuring at each location should be 0.4.

| **7** | **1** | **3** |

```
df['mine'] = np.random.binomial(1, 0.4, X*Y)
df
```

```
       x  y  mine
```
🪄

53. Now create a new column for this DataFrame called `'adjacent'`. This column should contain the number of mines found on adjacent squares in the grid.

(E.g. for the first row, which is the entry for the coordinate `(0, 0)`, count how many mines are found on the coordinates `(0, 1)`, `(1, 0)` and `(1, 1)`.)

```
     4   1  0     0
df['adjacent'] = \
    df.merge(df + [ 1,  1, 0], on=['x', 'y'], how='left')\
      .merge(df + [ 1, -1, 0], on=['x', 'y'], how='left')\
      .merge(df + [-1,  1, 0], on=['x', 'y'], how='left')\
      .merge(df + [-1, -1, 0], on=['x', 'y'], how='left')\
      .merge(df + [ 1,  0, 0], on=['x', 'y'], how='left')\
      .merge(df + [-1,  0, 0], on=['x', 'y'], how='left')\
      .merge(df + [ 0,  1, 0], on=['x', 'y'], how='left')\
      .merge(df + [ 0, -1, 0], on=['x', 'y'], how='left')\
      .iloc[:, 3:]\
      .sum(axis=1)


# An alternative solution is to pivot the DataFrame
# to form the "actual" grid of mines and use convolution.
# See https://github.com/jakevdp/matplotlib_pydata2013/blob/master/examples/minesweeper.py

from scipy.signal import convolve2d

mine_grid = df.pivot_table(columns='x', index='y', values='mine')
counts = convolve2d(mine_grid.astype(complex), np.ones((3, 3)), mode='same').real.astype(int)
df['adjacent'] = (counts - mine_grid).ravel('F')
```

```
    /usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:9203: FutureWarning: Passing
      validate=validate,
    --------------------------------------------------------------------------
    AttributeError                          Traceback (most recent call last)
    <ipython-input-56-9221d4b18ba7> in <module>
         19 mine_grid = df.pivot_table(columns='x', index='y', values='mine')
         20 counts = convolve2d(mine_grid.astype(complex), np.ones((3, 3)),
    mode='same').real.astype(int)
    ---> 21 df['adjacent'] = (counts - mine_grid).ravel('F')

    /usr/local/lib/python3.7/dist-packages/pandas/core/generic.py in __getattr__(self,
    name)
       5485         ):
       5486             return self[name]
    -> 5487         return object.__getattribute__(self, name)
       5488
       5489     def __setattr__(self, name: str, value) -> None:

    AttributeError: 'DataFrame' object has no attribute 'ravel'
```

54. For rows of the DataFrame that contain a mine, set the value in the `'adjacent'` column to NaN.

```
df.loc[df['mine'] == 1, 'adjacent'] = np.nan
```

55. Finally, convert the DataFrame to grid of the adjacent mine counts: columns are the x coordinate, rows are the y coordinate.

```
df.drop('mine', axis=1).set_index(['y', 'x']).unstack()
```

	adjacent				
x	0	1	2	3	4
y					
0	1.0	1.0	3.0	NaN	2.0
1	2.0	NaN	3.0	NaN	2.0
2	NaN	3.0	4.0	3.0	3.0
3	2.0	NaN	2.0	NaN	NaN

Plotting

Visualize trends and patterns in data

Difficulty: *medium*

To really get a good understanding of the data contained in your DataFrame, it is often essential to create plots: if you're lucky, trends and anomalies will jump right out at you. This functionality is baked into pandas and the puzzles below explore some of what's possible with the library.

56. Pandas is highly integrated with the plotting library matplotlib, and makes plotting DataFrames very user-friendly! Plotting in a notebook environment usually makes use of the following boilerplate:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

matplotlib is the plotting library which pandas' plotting functionality is built upon, and it is usually aliased to `plt`.

`%matplotlib inline` tells the notebook to show plots inline, instead of creating them in a separate window.

`plt.style.use('ggplot')` is a style theme that most people find agreeable, based upon the styling of R's ggplot package.

For starters, make a scatter plot of this random data, but use black X's instead of the default markers.
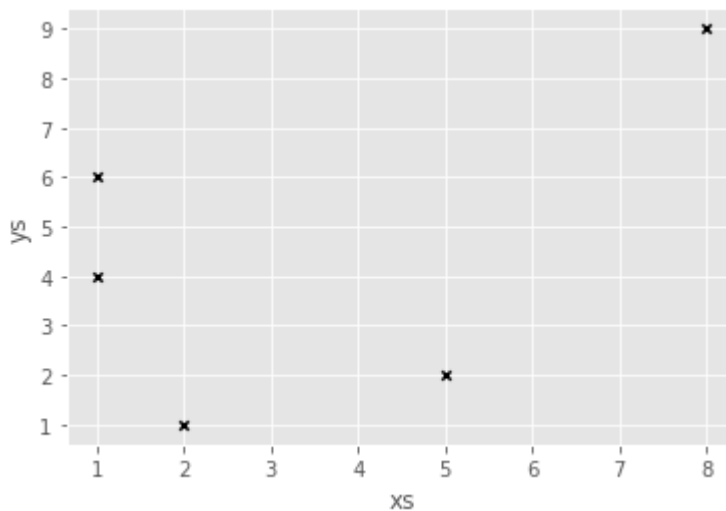
`df = pd.DataFrame({"xs":[1,5,2,8,1], "ys":[4,2,1,9,6]})`

Consult the [documentation](#) if you get stuck!

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

df = pd.DataFrame({"xs":[1,5,2,8,1], "ys":[4,2,1,9,6]})

df.plot.scatter("xs", "ys", color = "black", marker = "x")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbceb198190>
```

57. Columns in your DataFrame can also be used to modify colors and sizes. Bill has been keeping track of his performance at work over time, as well as how good he was feeling that day, and whether he had a cup of coffee in the morning. Make a plot which incorporates all four features of this DataFrame.

(Hint: If you're having trouble seeing the plot, try multiplying the Series which you choose to represent size by 10 or more)

The chart doesn't have to be pretty: this isn't a course in data viz!

```
df = pd.DataFrame({"productivity":[5,2,3,1,4,5,6,7,8,3,4,8,9],
                   "hours_in"    :[1,9,6,5,3,9,2,9,1,7,4,2,2],
                   "happiness"   :[2,1,3,2,3,1,2,3,1,2,2,1,3],
                   "caffienated" :[0,0,1,1,0,0,0,0,1,1,0,1,0]})
```

```
df = pd.DataFrame({"productivity":[5,2,3,1,4,5,6,7,8,3,4,8,9],
                   "hours_in"    :[1,9,6,5,3,9,2,9,1,7,4,2,2],
                   "happiness"   :[2,1,3,2,3,1,2,3,1,2,2,1,3],
                   "caffienated" :[0,0,1,1,0,0,0,0,1,1,0,1,0]})

df.plot.scatter("hours_in", "productivity", s = df.happiness * 30, c = df.caffienated)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbce21d46d0>
```
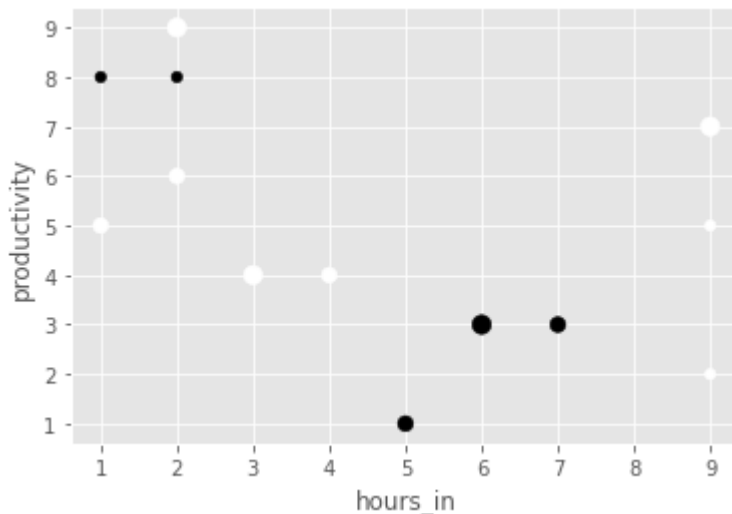

58. What if we want to plot multiple things? Pandas allows you to pass in a matplotlib *Axis* object for plots, and plots will also return an Axis object.

Make a bar plot of monthly revenue with a line plot of monthly advertising spending (numbers in millions)

```
df = pd.DataFrame({"revenue":[57,68,63,71,72,90,80,62,59,51,47,52],
                   "advertising":[2.1,1.9,2.7,3.0,3.6,3.2,2.7,2.4,1.8,1.6,1.3,1.9],
                   "month":range(12)
                  })
```

```
df = pd.DataFrame({"revenue":[57,68,63,71,72,90,80,62,59,51,47,52],
                   "advertising":[2.1,1.9,2.7,3.0,3.6,3.2,2.7,2.4,1.8,1.6,1.3,1.9],
                   "month":range(12)
                  })
```

```
ax = df.plot.bar("month", "revenue", color = "green")
df.plot.line("month", "advertising", secondary_y = True, ax = ax)
ax.set_xlim((-1,12))
```

```
(-1.0, 12.0)
```
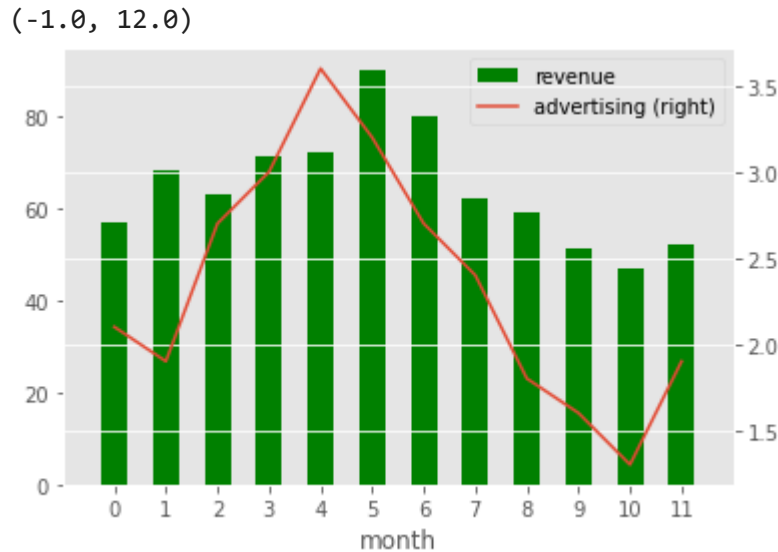

Now we're finally ready to create a candlestick chart, which is a very common tool used to analyze stock price data. A candlestick chart shows the opening, closing, highest, and lowest price for a stock during a time window. The color of the "candle" (the thick part of the bar) is green if the stock closed above its opening price, or red if below.

Candlestick Example

This was initially designed to be a pandas plotting challenge, but it just so happens that this type of plot is just not feasible using pandas' methods. If you are unfamiliar with matplotlib, we have provided a function that will plot the chart for you so long as you can use pandas to get the data into the correct format.

Your first step should be to get the data in the correct format using pandas' time-series grouping function. We would like each candle to represent an hour's worth of data. You can write your own aggregation function which returns the open/high/low/close, but pandas has a built-in which also does this.

The below cell contains helper functions. Call `day_stock_data()` to generate a DataFrame containing the prices a hypothetical stock sold for, and the time the sale occurred. Call `plot_candlestick(df)` on your properly aggregated and formatted stock data to print the candlestick chart.

```
import numpy as np
def float_to_time(x):
    return str(int(x)) + ":" + str(int(x%1 * 60)).zfill(2) + ":" + str(int(x*60 % 1 * 60)).zf
```

```python
def day_stock_data():
    #NYSE is open from 9:30 to 4:00
    time = 9.5
    price = 100
    results = [(float_to_time(time), price)]
    while time < 16:
        elapsed = np.random.exponential(.001)
        time += elapsed
        if time > 16:
            break
        price_diff = np.random.uniform(.999, 1.001)
        price *= price_diff
        results.append((float_to_time(time), price))



    df = pd.DataFrame(results, columns = ['time','price'])
    df.time = pd.to_datetime(df.time)
    return df


#Don't read me unless you get stuck!
def plot_candlestick(agg):
    """
    agg is a DataFrame which has a DatetimeIndex and five columns: ["open","high","low","clos
    """
    fig, ax = plt.subplots()
    for time in agg.index:
        ax.plot([time.hour] * 2, agg.loc[time, ["high","low"]].values, color = "black")
        ax.plot([time.hour] * 2, agg.loc[time, ["open","close"]].values, color = agg.loc[time

    ax.set_xlim((8,16))
    ax.set_ylabel("Price")
    ax.set_xlabel("Hour")
    ax.set_title("OHLC of Stock Value During Trading Day")
    plt.show()
```

59. Generate a day's worth of random stock data, and aggregate / reformat it so that it has hourly summaries of the opening, highest, lowest, and closing prices

```python
df.set_index("time", inplace = True)
agg = df.resample("H").ohlc()
agg.columns = agg.columns.droplevel()
agg["color"] = (agg.close > agg.open).map({True:"green",False:"red"})
agg.head()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-63-ed4141a6e198> in <module>
----> 1 df.set_index("time", inplace = True)
      2 agg = df.resample("H").ohlc()
      3 agg.columns = agg.columns.droplevel()
      4 agg["color"] = (agg.close > agg.open).map({True:"green",False:"red"})
      5 agg.head()
```

$\hat{\vee}$ 1 frames

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py in set_index(self, keys,
drop, append, inplace, verify_integrity)
    5449
    5450            if missing:
-> 5451                raise KeyError(f"None of {missing} are in the columns")
    5452
```

60. Now that you have your properly-formatted data, try to plot it yourself as a candlestick chart.
Use the `plot_candlestick(df)` function above, or matplotlib's <u>plot documentation</u> if you get
stuck.

```
plot_candlestick(agg)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-64-73f4e3092d60> in <module>
----> 1 plot_candlestick(agg)

NameError: name 'plot_candlestick' is not defined
```

SEARCH STACK OVERFLOW

Double-click (or enter) to edit

Colab paid products - Cancel contracts here

0s   completed at 11:21 PM