In [4]:
```
import numpy as np # imports a fast numerical programming library
import scipy as sp #imports stats functions, amongst other things
import matplotlib as mpl # this actually imports matplotlib
import matplotlib.cm as cm #allows us easy access to colormaps
import matplotlib.pyplot as plt #sets up plotting under plt
#sets up pandas table display
import seaborn as sns #sets up styles and gives us more plotting options
```

Annoucement:

There are totally 24 questions, 23 of them are followed by a blank cell. Please fill in the code in the blank cell, and remember to display the target data.

Information you may need to complete this project: Google, Pandas Official documents, Pandas cheatsheets.

Attention

Because Jupyter Notebook will save the result of all codes that were executed before, if you are not sure whether your code is right and it may jeopardize the dataset (for example: deleting some records), there are several ways to reset your data:

- Rerun the cell that creates the data
- Delete your current cell or comment it, and select "restart & run all" in the Kernel menu.
- Create a temporary variable that exactly copies the original dataframe, and test your codes on the temporary dataframe first.

1. Import pandas under the name `pd` .

In [91]:
```
import pandas as pd
```

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

In [92]:
```
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']
```

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

In [93]:
```
df = pd.DataFrame(data, index=labels)
df
```

Out[93]:

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

3. Print a summary of the basic information of the dataframe

In [94]:
```
print df.info()
print '\n'
print df.columns
print '\n'
print df.index
print '\n'
print df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
age         8 non-null float64
animal      10 non-null object
priority    10 non-null object
visits      10 non-null int64
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
None


Index([u'age', u'animal', u'priority', u'visits'], dtype='object')


Index([u'a', u'b', u'c', u'd', u'e', u'f', u'g', u'h', u'i', u'j'], dtype='ob
ject')


             age     visits
count   8.000000  10.000000
mean    3.437500   1.900000
std     2.007797   0.875595
min     0.500000   1.000000
25%     2.375000   1.000000
50%     3.000000   2.000000
75%     4.625000   2.750000
max     7.000000   3.000000
```

4. Return the first 3 rows of the DataFrame df.

In [95]:
```
print df.iloc[:3]
```

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

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

In [97]:
```
print df.iloc[[3,4,8], [1,0]];
# this also selects the appropriate data, but gives
# DeprecationWarming: .ix is deprecated
# df.ix[[3,4,8], ['animal', 'age']]
```

```
  animal  age
d    dog  NaN
e    dog  5.0
i    dog  7.0
```

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

In [98]:
```
df[df['visits'] > 3]
```

Out[98]:

	age	animal	priority	visits

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

In [99]:
```
df[df['age'].isnull()]
```

Out[99]:

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

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

In [100]:
```
df[(df['animal'] == 'cat') & (df['age'] < 3)]
```

Out[100]:

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

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

In [101]:
```
df.loc['f', 'age'] = 1.5
df
```

Out[101]:

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

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

In [102]:
```
df['visits'].sum()
```

Out[102]:
```
19
```

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

In [103]:
```
df.groupby('animal')['age'].mean()
```

Out[103]:
```
animal
cat      2.333333
dog      5.000000
snake    2.500000
Name: age, dtype: float64
```

12. 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 [106]:
```
df.loc['k'] = [1.5, 'dog', 'yes', 1]
print df
print '\n'
df = df.drop('k')
print df
```

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


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

13. Count the number of each type of animal in `df`. http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.value_counts.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.value_counts.html)

In [107]:
```
df['animal'].value_counts()
```

Out[107]:
```
cat      4
dog      4
snake    2
Name: animal, dtype: int64
```

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

In [108]:
```
df.sort_values(by=['age', 'visits'], ascending=[False, True])
```

Out[108]:

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

15. 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`. And print the new dataframe. refer http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.map.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.map.html) (hint: use a dict or lambda function as argument)

In [109]:
```
df['priority'] = df['priority'].map({'yes': True, 'no': False})
print df
```

```
   age animal  priority  visits
a  2.5    cat      True       1
b  3.0    cat      True       3
c  0.5  snake     False       2
d  NaN    dog      True       3
e  5.0    dog     False       2
f  1.5    cat     False       3
g  4.5  snake     False       1
h  NaN    cat      True       1
i  7.0    dog     False       2
j  3.0    dog     False       1
```

16. In the 'animal' column, change the 'snake' entries to 'python'. Try googling it.

In [110]:
```
df['animal'] = df['animal'].replace('snake', 'python')
df
```

Out[110]:

	age	animal	priority	visits
a	2.5	cat	True	1
b	3.0	cat	True	3
c	0.5	python	False	2
d	NaN	dog	True	3
e	5.0	dog	False	2
f	1.5	cat	False	3
g	4.5	python	False	1
h	NaN	cat	True	1
i	7.0	dog	False	2
j	3.0	dog	False	1

17. 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 `pd.pivot_table`).

In [111]:
```
df.pivot_table(index = 'animal', columns = 'visits', values = 'age', aggfunc = 'mean')
```

Out[111]:

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

18. 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` . hint: use `pd.date_range` (change the freq argument to "B" to only choose business days), `np.random.rand`

In [113]:
```
dateTimeIndex = pd.date_range(start = '2015-01-01', end = '2015-12-31', freq =
'B')
print dateTimeIndex
print '\n'
s = pd.Series(np.random.rand(len(dateTimeIndex)), index = dateTimeIndex)
print s
```

```
DatetimeIndex(['2015-01-01', '2015-01-02', '2015-01-05', '2015-01-06',
               '2015-01-07', '2015-01-08', '2015-01-09', '2015-01-12',
               '2015-01-13', '2015-01-14',
               ...
               '2015-12-18', '2015-12-21', '2015-12-22', '2015-12-23',
               '2015-12-24', '2015-12-25', '2015-12-28', '2015-12-29',
               '2015-12-30', '2015-12-31'],
              dtype='datetime64[ns]', length=261, freq='B')
```

2015-01-01	0.910826
2015-01-02	0.481425
2015-01-05	0.060279
2015-01-06	0.725052
2015-01-07	0.750149
2015-01-08	0.291876
2015-01-09	0.956948
2015-01-12	0.934083
2015-01-13	0.146549
2015-01-14	0.517683
2015-01-15	0.997568
2015-01-16	0.336646
2015-01-19	0.652175
2015-01-20	0.090073
2015-01-21	0.447626
2015-01-22	0.652946
2015-01-23	0.477521
2015-01-26	0.064512
2015-01-27	0.657046
2015-01-28	0.042803
2015-01-29	0.417128
2015-01-30	0.185573
2015-02-02	0.446516
2015-02-03	0.171058
2015-02-04	0.097808
2015-02-05	0.982990
2015-02-06	0.288648
2015-02-09	0.776665
2015-02-10	0.125568
2015-02-11	0.803884
...	...
2015-11-20	0.237285
2015-11-23	0.909968
2015-11-24	0.085465
2015-11-25	0.534953
2015-11-26	0.765949
2015-11-27	0.178583
2015-11-30	0.410723
2015-12-01	0.969433
2015-12-02	0.293473
2015-12-03	0.178641
2015-12-04	0.634025
2015-12-07	0.453489
2015-12-08	0.663244
2015-12-09	0.507017
2015-12-10	0.822721
2015-12-11	0.796618

```
2015-12-14    0.560058
2015-12-15    0.966747
2015-12-16    0.023136
2015-12-17    0.354777
2015-12-18    0.631930
2015-12-21    0.166570
2015-12-22    0.492938
2015-12-23    0.838012
2015-12-24    0.191111
2015-12-25    0.642807
2015-12-28    0.432437
2015-12-29    0.148145
2015-12-30    0.923227
2015-12-31    0.392378
Freq: B, Length: 261, dtype: float64
```

19. Given the following data:

In [114]:
```
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"']})
```

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), print the dataframe. Hint:(http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.interpolate.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.interpolate.html))

In [115]:
```
df['FlightNumber'] = df['FlightNumber'].interpolate().astype(int)
print df
```

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

20. The From_To column would be better as two separate columns! Choose this column and give it to a temporary dataframe "temp". Then split temp by the underscore delimiter _ so that it has two columns ("From" and "To"), with the correct values. Assign the correct column names "From" and "To" to this temporary DataFrame. and then print the dataframe "temp".

In [116]:
```
temp = df['From_To'].str.split('_', n = 1, expand = True)
temp.columns = ['From', 'To']
print temp
```

```
       From         To
0    LoNDon      paris
1    MAdrid      miLAN
2    londON  StockhOlm
3  Budapest      PaRis
4  Brussels     londOn
```

21. 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".)

In [117]:
```
temp['From'] = temp.From.str.title()
temp['To'] = temp.To.str.title()
temp
```

Out[117]:

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

22. Delete the From_To column from `df` and attach the temporary DataFrame from the previous questions, print df.

In [118]:
```
df.drop(columns = ['From_To'], inplace = True)
df['From'] = temp['From']
df['To'] = temp['To']
print df
```

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

23. 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'`. (this sentence is a little bit difficult, so I write it for you, try to understand it, it's useful)

In [119]:
```
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
```

Out[119]:

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

24. For the following dataset:

In [120]:
```
users = pd.read_table('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user',
                      sep='|', index_col='user_id')
```

Find the mean age per occupation (Hint: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html))

In [121]:
```
users.groupby('occupation').age.mean()
```

Out[121]:
```
occupation
administrator    38.746835
artist           31.392857
doctor           43.571429
educator         42.010526
engineer         36.388060
entertainment    29.222222
executive        38.718750
healthcare       41.562500
homemaker        32.571429
lawyer           36.750000
librarian        40.000000
marketing        37.615385
none             26.555556
other            34.523810
programmer       33.121212
retired          63.071429
salesman         35.666667
scientist        35.548387
student          22.081633
technician       33.148148
writer           36.311111
Name: age, dtype: float64
```