Python Programming

by Narendra Allam

Chapter 10.2

Pandas & Matplotlib

Topics Covering

- Pandas
  - series
    - Constructing from dictionaries
    - Custom Index
    - Data filtering
  - Data Frames
    - Constructing from a dictionary with values as lists
    - Custom indexing
    - Rearraning the coloumns
    - Setting values
    - Sum
    - Cumulative sum
    - Assigning a column to the dataframe
    - Adding a new column
    - Deleting a column
    - Slicing
    - Indexing and Advanced indexing
    - Sorting
    - Transposing
    - Sort by
    - Concatenate
    - Merge
    - Join
    - Group By
    - Data Munging
      - Working Missing data
    - Reading Data from CSV, Excel, JSON
    - Writing Data to CSV, Excel, JSON
- Matplotlib
  - Basic Plotting
  - mutiple plots
  - labels
  - legends
  - styles
  - Bar charts
  - Histograms
  - Scatter Plots

- box Plots
- pie plots

Series

A Series is a one-dimensional array-like object containing an array of data, which can be any NumPy data type, and an associated array of data labels, functioning as its index.

In [1]:

```
import pandas as pd
```

In [2]:

```
S = pd.Series([36, 32, 45, 30, 25, 40, 42], dtype=float)
print(S)
```

```
0    36.0
1    32.0
2    45.0
3    30.0
4    25.0
5    40.0
6    42.0
dtype: float64
```

In [3]:

```
S.index
```

Out[3]:

```
RangeIndex(start=0, stop=7, step=1)
```

In [4]:

```
S.values
```

Out[4]:

```
array([36., 32., 45., 30., 25., 40., 42.])
```

In [5]:

```
S[4]
```

Out[5]:

```
25.0
```

In [6]:

```
# custom indices
S = pd.Series([36, 32, 45, 30, 25, 40, 42], index=['Sunday', 'Monday',
                                                   'Tuesday', 'Wednessday',
                                                   'Thursday', 'Friday',
                                                   'Saturday'])
S
```

Out[6]:

```
Sunday         36
Monday         32
Tuesday        45
Wednessday     30
Thursday       25
Friday         40
Saturday       42
dtype: int64
```

In [7]:

```
S['Friday']
```

Out[7]:

```
40
```

In [8]:

```
# custom indices
S = pd.Series([36, 32, 45, 30, 25, 40, 42], index=range(1, 8))
S
```

Out[8]:

```
1    36
2    32
3    45
4    30
5    25
6    40
7    42
dtype: int64
```

In [9]:

```
# checking data availability for indices
expected_dates = [1, 3, 4, 6, 8, 9, 10]

s1 = pd.Series(S, index=expected_dates)
s1
```

Out[9]:

```
1     36.0
3     45.0
4     30.0
6     40.0
8      NaN
9      NaN
10     NaN
dtype: float64
```

In [10]:

```
None == None
```

Out[10]:

```
True
```

In [11]:

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

Out[11]:

```
False
```

In [12]:

```
print ('MAX=', s1.max())
print ('MIN=', s1.min())
print ('AVG=', s1.mean())
print ('STD=', s1.std())
```

```
MAX= 45.0
MIN= 30.0
AVG= 37.75
STD= 6.34428877022476
```

In [13]:

```
s1.describe()
```

Out[13]:

```
count     4.000000
mean     37.750000
std       6.344289
min      30.000000
25%      34.500000
50%      38.000000
75%      41.250000
max      45.000000
dtype: float64
```

In [14]:

```
s1.isnull()
```

Out[14]:

```
1     False
3     False
4     False
6     False
8      True
9      True
10     True
dtype: bool
```

In [15]:

```
s1
```

Out[15]:

```
1     36.0
3     45.0
4     30.0
6     40.0
8      NaN
9      NaN
10     NaN
dtype: float64
```

In [16]:

```
# considering 0s inplace of NaN
s1[s1.isnull()] = 0
```

In [17]:

```
s1.describe()
```

Out[17]:

```
count     7.000000
mean     21.571429
std      20.670891
min       0.000000
25%       0.000000
50%      30.000000
75%      38.000000
max      45.000000
dtype: float64
```

In [18]:

```
# Custom Index
fruits = ['apples', 'oranges', 'cherries', 'pears', 'Mango']
quantities = [20, 33, 52, 10, 40]
S = pd.Series(quantities, index=fruits)
S
```

Out[18]:

```
apples      20
oranges     33
cherries    52
pears       10
Mango       40
dtype: int64
```

In [19]:

```
S['Mango']
```

Out[19]:

```
40
```

In [20]:

```
# Scalar arithmatic
import numpy as np
print((S + 3) * 4)
print("======================")
```

```
apples       92
oranges     144
cherries    220
pears        52
Mango       172
dtype: int64
======================
```

In [21]:

```
np.sin(S)
```

Out[21]:

```
apples      0.912945
oranges     0.999912
cherries    0.986628
pears      -0.544021
Mango       0.745113
dtype: float64
```

In [22]:

```
# fruits with quantity morethan 30
S[S > 30] # filtering
```

Out[22]:

```
oranges     33
cherries    52
Mango       40
dtype: int64
```

In [23]:

```
# Replacing the quantities greater than 30 with custom list of values
S[S > 30] = [30, 40, 50]
S
```

Out[23]:

```
apples      20
oranges     40
cherries    50
pears       10
Mango       40
dtype: int64
```

In [24]:

```
# dictionary
cities = {"London":   8615246,
          "Berlin":   3562166,
          "Madrid":   3165235,
          "Rome":     2874038,
          "Paris":    2273305,
          "Vienna":   1805681,
          "Bucharest":1803425,
          "Hamburg":  1760433,
          "Budapest": 1754000,
          "Warsaw":   1740119,
          "Barcelona":1602386,
          "Munich":   1493900,
          "Milan":    1350680}

city_series = pd.Series(cities, dtype='uint32')
city_series
```

Out[24]:

```
Barcelona    1602386
Berlin       3562166
Bucharest    1803425
Budapest     1754000
Hamburg      1760433
London       8615246
Madrid       3165235
Milan        1350680
Munich       1493900
Paris        2273305
Rome         2874038
Vienna       1805681
Warsaw       1740119
dtype: uint32
```

All the cities with population greater than the average ?

In [25]:

```
city_series[city_series > np.average(city_series)]
```

Out[25]:

```
Berlin    3562166
London    8615246
Madrid    3165235
Rome      2874038
dtype: uint32
```

All the cities with population < 1700000 and > 1300000?

In [26]:

```
city_series[(city_series < 1700000) & (city_series > 1300000)]
```

Out[26]:

```
Barcelona    1602386
Milan        1350680
Munich       1493900
dtype: uint32
```

In [27]:

```
# List can be passed to index, if less elements in the list NaN will be assigned
my_cities = ["London", "Paris", "Zurich", "Berlin",
             "Stuttgart", "Hamburg"]

my_city_series = pd.Series(city_series, index=my_cities)
my_city_series
```

Out[27]:

```
London       8615246.0
Paris        2273305.0
Zurich             NaN
Berlin       3562166.0
Stuttgart          NaN
Hamburg      1760433.0
dtype: float64
```

In [28]:

```
my_city_series.isnull()
```

Out[28]:

```
London       False
Paris        False
Zurich        True
Berlin       False
Stuttgart     True
Hamburg      False
dtype: bool
```

In [29]:

```
my_city_series[my_city_series.isnull()] = 1000000
```

In [30]:

```
my_city_series
```

Out[30]:

```
London       8615246.0
Paris        2273305.0
Zurich       1000000.0
Berlin       3562166.0
Stuttgart    1000000.0
Hamburg      1760433.0
dtype: float64
```

In [31]:

```
city_series = pd.Series(my_city_series, dtype='uint64')
print(city_series)
```

```
London       8615246
Paris        2273305
Zurich       1000000
Berlin       3562166
Stuttgart    1000000
Hamburg      1760433
dtype: uint64
```

In [32]:

```
city_series[city_series.index.str.startswith('B')] = 999999
```

In [33]:

```
city_series
```

Out[33]:

```
London       8615246
Paris        2273305
Zurich       1000000
Berlin        999999
Stuttgart    1000000
Hamburg      1760433
dtype: uint64
```

In [34]:

```
# converting dtype of exisiting series
s = pd.Series(my_city_series, dtype='uint32')
s
```

Out[34]:

```
London       8615246
Paris        2273305
Zurich       1000000
Berlin       3562166
Stuttgart    1000000
Hamburg      1760433
dtype: uint32
```

np.nan is not zero

In [35]:

```
s1 = pd.Series([1, np.nan, 2])
s2 = pd.Series([1, 0, 2])
```

In [36]:

```
s1.describe()
```

Out[36]:

```
count    2.000000
mean     1.500000
std      0.707107
min      1.000000
25%      1.250000
50%      1.500000
75%      1.750000
max      2.000000
dtype: float64
```

In [37]:

```
s2.describe()
```

Out[37]:

```
count    3.0
mean     1.0
std      1.0
min      0.0
25%      0.5
50%      1.0
75%      1.5
max      2.0
dtype: float64
```

Dataframe

The underlying idea of a DataFrame is based on spreadsheets. We can see the data structure of a DataFrame as tabular and spreadsheet-like. It contains an ordered collection of columns. Each column consists of a unique data type, but different columns can have different types, e.g. the first column may consist of integers, while the second one consists of boolean values and so on.

A DataFrame has a row and column index; it's like a dict of Series with a common index.

In [38]:

```
import pandas as pd
cities = {"city_name": ["London", "Berlin", "Madrid", "Rome",
                    "Paris", "Vienna", "Bucharest", "Hamburg",
                    "Budapest", "Warsaw", "Barcelona",
                    "Munich", "Milan"],
          "population": [8615246, 3562166, 3165235, 2874038,
                         2273305, 1805681, 1803425, 1760433,
                         1754000, 1805681, 1602386, 1805681,
                         1350680],
          "country": ["England", "Germany", "Spain", "Italy",
                      "France", "Austria", "Romania",
                      "Germany", "Hungary", "Poland", "Spain",
                      "Germany", "Italy"]}

city_frame = pd.DataFrame(cities)

city_frame
```

Out[38]:

	city_name	country	population
0	London	England	8615246
1	Berlin	Germany	3562166
2	Madrid	Spain	3165235
3	Rome	Italy	2874038
4	Paris	France	2273305
5	Vienna	Austria	1805681
6	Bucharest	Romania	1803425
7	Hamburg	Germany	1760433
8	Budapest	Hungary	1754000
9	Warsaw	Poland	1805681
10	Barcelona	Spain	1602386
11	Munich	Germany	1805681
12	Milan	Italy	1350680

In [39]:

```
### Custom indexing
ordinals = ["first", "second", "third", "fourth",
            "fifth", "sixth", "seventh", "eigth",
            "ninth", "tenth", "eleventh", "twelvth",
            "thirteenth"]

city_frame = pd.DataFrame(cities, index=ordinals)
city_frame
```

Out[39]:

	city_name	country	population
first	London	England	8615246
second	Berlin	Germany	3562166
third	Madrid	Spain	3165235
fourth	Rome	Italy	2874038
fifth	Paris	France	2273305
sixth	Vienna	Austria	1805681
seventh	Bucharest	Romania	1803425
eigth	Hamburg	Germany	1760433
ninth	Budapest	Hungary	1754000
tenth	Warsaw	Poland	1805681
eleventh	Barcelona	Spain	1602386
twelvth	Munich	Germany	1805681
thirteenth	Milan	Italy	1350680

In [40]:

```
# Rearranging the Columns
city_frame = pd.DataFrame(city_frame,
                          columns=[ "country",
                                    "city_name",
                                    "population"])
city_frame
```

Out[40]:

	country	city_name	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
seventh	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [41]:

```
city_frame = city_frame.rename(columns = {'city_name':'cityname'})
```

In [42]:

```
city_frame
```

Out[42]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
seventh	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [43]:

```
city_frame.rename(index = {'eigth':'seventh'}, inplace=True)
```

In [44]:

```
city_frame
```

Out[44]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
seventh	Romania	Bucharest	1803425
seventh	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [45]:

```
city_frame.groupby(city_frame.index).get_group('seventh')
```

Out[45]:

	country	cityname	population
seventh	Romania	Bucharest	1803425
seventh	Germany	Hamburg	1760433

In [46]:

```
city_frame
```

Out[46]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
seventh	Romania	Bucharest	1803425
seventh	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [47]:

```
city_frame.rename(index = {'seventh': 'eigth'}, inplace=True)
city_frame
```

Out[47]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
eigth	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [48]:

```
city_frame
```

Out[48]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
eigth	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [49]:

```
city_frame = city_frame.rename(columns = {'city_name':'cityname'})
```

In [50]:

```
# accessing a column
city_frame['cityname']
```

Out[50]:

```
first            London
second           Berlin
third            Madrid
fourth             Rome
fifth             Paris
sixth            Vienna
eigth         Bucharest
eigth           Hamburg
ninth          Budapest
tenth            Warsaw
eleventh      Barcelona
twelvth          Munich
thirteenth        Milan
Name: cityname, dtype: object
```

In [51]:

```
# accessing a cell
city_frame['cityname']['sixth']
```

Out[51]:

```
'Vienna'
```

In [52]:

```
# Alternate syntax
city_frame.cityname['eigth']
```

Out[52]:

```
eigth    Bucharest
eigth      Hamburg
Name: cityname, dtype: object
```

In [53]:

```
# setting values
# city_frame['cityname']['fourth'] = 'ROME_MODIFIED'
city_frame.set_value('fourth', 'cityname', 'ROME_MODIFIED')
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:3: Future
Warning: set_value is deprecated and will be removed in a future relea
se. Please use .at[] or .iat[] accessors instead
  This is separate from the ipykernel package so we can avoid doing im
ports until
```

Out[53]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	ROME_MODIFIED	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
eigth	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [54]:

```
city_frame.set_value('fourth', 'cityname', 'Rome')
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:1: Future
Warning: set_value is deprecated and will be removed in a future relea
se. Please use .at[] or .iat[] accessors instead
  """Entry point for launching an IPython kernel.
```

Out[54]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
eigth	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

Slicing and views

loc(), iloc()

In [55]:

```
city_frame
```

Out[55]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
eigth	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [56]:

```
city_frame.loc['third': 'tenth', 'country':'cityname']
```

Out[56]:

	country	cityname
third	Spain	Madrid
fourth	Italy	Rome
fifth	France	Paris
sixth	Austria	Vienna
eigth	Romania	Bucharest
eigth	Germany	Hamburg
ninth	Hungary	Budapest
tenth	Poland	Warsaw

In [57]:

```
city_frame.loc['third': 'tenth', 'cityname':'country':-1]
```

Out[57]:

	cityname	country
third	Madrid	Spain
fourth	Rome	Italy
fifth	Paris	France
sixth	Vienna	Austria
eigth	Bucharest	Romania
eigth	Hamburg	Germany
ninth	Budapest	Hungary
tenth	Warsaw	Poland

In [58]:

```
city_frame.loc['seventh']
```

```
---------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
 last)
/usr/local/lib/python3.6/site-packages/pandas/core/indexing.py in _has
_valid_type(self, key, axis)
   1505                 if not ax.contains(key):
-> 1506                     error()
   1507             except TypeError as e:

/usr/local/lib/python3.6/site-packages/pandas/core/indexing.py in erro
r()
   1500                                  .format(key=key,
-> 1501                                          axis=self.obj._get_axis
_name(axis)))
   1502

KeyError: 'the label [seventh] is not in the [index]'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call
 last)
<ipython-input-58-9942e41d2be2> in <module>()
----> 1 city_frame.loc['seventh']

/usr/local/lib/python3.6/site-packages/pandas/core/indexing.py in __ge
titem__(self, key)
   1371
   1372             maybe_callable = com._apply_if_callable(key, self.
obj)
-> 1373             return self._getitem_axis(maybe_callable, axis=axi
s)
   1374
   1375     def _is_scalar_access(self, key):

/usr/local/lib/python3.6/site-packages/pandas/core/indexing.py in _get
item_axis(self, key, axis)
   1624
   1625         # fall thru to straight lookup
-> 1626         self._has_valid_type(key, axis)
   1627         return self._get_label(key, axis=axis)
   1628

/usr/local/lib/python3.6/site-packages/pandas/core/indexing.py in _has
_valid_type(self, key, axis)
   1512                 raise
   1513             except:
-> 1514                 error()
   1515
   1516         return True

/usr/local/lib/python3.6/site-packages/pandas/core/indexing.py in erro
r()
   1499                 raise KeyError(u"the label [{key}] is not in t
he [{axis}]"
   1500                                  .format(key=key,
```

```
-> 1501                                          axis=self.obj._get_axis
_name(axis)))
   1502
   1503             try:

KeyError: 'the label [seventh] is not in the [index]'
```

In [ ]:

```
city_frame.loc['third': 'tenth' : 2, 'cityname':'country':-1]
```

Accessing Specific columns and rows

In [ ]:

```
city_frame.loc[['first', 'sixth', 'tenth'], ['country', 'population']]
```

In [ ]:

```
city_frame.loc['first':'fifth', ['country', 'population']]
```

In [ ]:

```
city_frame.loc['fifth':'first':-1, ['country', 'population']]
```

In [59]:

```
city_frame.iloc[4::-1, [0, 2]]
```

Out[59]:

	country	population
fifth	France	2273305
fourth	Italy	2874038
third	Spain	3165235
second	Germany	3562166
first	England	8615246

In [60]:

```
city_frame.iloc[2:9, :]
```

Out[60]:

	country	cityname	population
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
eigth	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000

In [61]:

```
city_frame
```

Out[61]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
eigth	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680

In [62]:

```
city_frame.sum()
```

Out[62]:

```
country       EnglandGermanySpainItalyFranceAustriaRomaniaGe...
cityname      LondonBerlinMadridRomeParisViennaBucharestHamb...
population                                             34177957
dtype: object
```

In [63]:

```
city_frame['population'].sum()
```

Out[63]:

```
34177957
```

In [64]:

```
city_frame.all()
```

Out[64]:

```
country       True
cityname      True
population    True
dtype: bool
```

In [65]:

```
# Cumulative sum
x = city_frame["population"].cumsum()
print(x)
```

```
first          8615246
second        12177412
third         15342647
fourth        18216685
fifth         20489990
sixth         22295671
eigth         24099096
eigth         25859529
ninth         27613529
tenth         29419210
eleventh      31021596
twelvth       32827277
thirteenth    34177957
Name: population, dtype: int64
```

Adding a new column

In [66]:

```
import numpy as np
# Adding a new column
city_frame['area'] = np.nan
```

In [67]:

```
city_frame
```

Out[67]:

	country	cityname	population	area
first	England	London	8615246	NaN
second	Germany	Berlin	3562166	NaN
third	Spain	Madrid	3165235	NaN
fourth	Italy	Rome	2874038	NaN
fifth	France	Paris	2273305	NaN
sixth	Austria	Vienna	1805681	NaN
eigth	Romania	Bucharest	1803425	NaN
eigth	Germany	Hamburg	1760433	NaN
ninth	Hungary	Budapest	1754000	NaN
tenth	Poland	Warsaw	1805681	NaN
eleventh	Spain	Barcelona	1602386	NaN
twelvth	Germany	Munich	1805681	NaN
thirteenth	Italy	Milan	1350680	NaN

In [68]:

```
area = [1572, 891.85, 605.77, 1285,
        105.4, 414.6, 228, 755,
        525.2, 517, 101.9, 310.4,
        181.8]

city_frame["area"] = area
```

In [69]:

```
city_frame
```

Out[69]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40
thirteenth	Italy	Milan	1350680	181.80

adding a row

In [70]:

```
df = pd.DataFrame([['India', 'Hyderabad', 15000000, 700]],
                  columns = ['country', 'cityname', 'population', 'area'],
                  index = ['fourteenth'])
```

In [71]:

```
df
```

Out[71]:

	country	cityname	population	area
fourteenth	India	Hyderabad	15000000	700

In [72]:

```
city_frame = city_frame.append(df)
```

In [73]:

```
city_frame
```

Out[73]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40
thirteenth	Italy	Milan	1350680	181.80
fourteenth	India	Hyderabad	15000000	700.00

In [74]:

```
city_frame['extra'] = np.nan
```

In [75]:

```
city_frame
```

Out[75]:

	country	cityname	population	area	extra
first	England	London	8615246	1572.00	NaN
second	Germany	Berlin	3562166	891.85	NaN
third	Spain	Madrid	3165235	605.77	NaN
fourth	Italy	Rome	2874038	1285.00	NaN
fifth	France	Paris	2273305	105.40	NaN
sixth	Austria	Vienna	1805681	414.60	NaN
eigth	Romania	Bucharest	1803425	228.00	NaN
eigth	Germany	Hamburg	1760433	755.00	NaN
ninth	Hungary	Budapest	1754000	525.20	NaN
tenth	Poland	Warsaw	1805681	517.00	NaN

Deleting a column

In [76]:

```
city_frame.pop('extra')
```

Out[76]:

```
first        NaN
second       NaN
third        NaN
fourth       NaN
fifth        NaN
sixth        NaN
eigth        NaN
eigth        NaN
ninth        NaN
tenth        NaN
eleventh     NaN
twelvth      NaN
thirteenth   NaN
fourteenth   NaN
Name: extra, dtype: float64
```

In [77]:

```
city_frame
```

Out[77]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40
thirteenth	Italy	Milan	1350680	181.80
fourteenth	India	Hyderabad	15000000	700.00

In [78]:

```
city_frame.drop('area', axis=1)
```

Out[78]:

	country	cityname	population
first	England	London	8615246
second	Germany	Berlin	3562166
third	Spain	Madrid	3165235
fourth	Italy	Rome	2874038
fifth	France	Paris	2273305
sixth	Austria	Vienna	1805681
eigth	Romania	Bucharest	1803425
eigth	Germany	Hamburg	1760433
ninth	Hungary	Budapest	1754000
tenth	Poland	Warsaw	1805681
eleventh	Spain	Barcelona	1602386
twelvth	Germany	Munich	1805681
thirteenth	Italy	Milan	1350680
fourteenth	India	Hyderabad	15000000

In [79]:

```
city_frame
```

Out[79]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40
thirteenth	Italy	Milan	1350680	181.80
fourteenth	India	Hyderabad	15000000	700.00

In [80]:

```
city_frame.drop('fourteenth')
```

Out[80]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40
thirteenth	Italy	Milan	1350680	181.80

In [81]:

```
city_frame
```

Out[81]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40
thirteenth	Italy	Milan	1350680	181.80
fourteenth	India	Hyderabad	15000000	700.00

In [82]:

```
city_frame.drop(['fourteenth', 'thirteenth'])
```

Out[82]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40

Permenently removing a row:

In [83]:

```
city_frame.drop('fourteenth', inplace=True)
```

In [84]:

```
city_frame
```

Out[84]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40
thirteenth	Italy	Milan	1350680	181.80

at(), iat()

In [85]:

```
city_frame.at['sixth', 'cityname'] = 'Vienna'
```

In [86]:

```
city_frame.iat[0, 2] = 8615246
```

In [87]:

```
city_frame
```

Out[87]:

	country	cityname	population	area
first	England	London	8615246	1572.00
second	Germany	Berlin	3562166	891.85
third	Spain	Madrid	3165235	605.77
fourth	Italy	Rome	2874038	1285.00
fifth	France	Paris	2273305	105.40
sixth	Austria	Vienna	1805681	414.60
eigth	Romania	Bucharest	1803425	228.00
eigth	Germany	Hamburg	1760433	755.00
ninth	Hungary	Budapest	1754000	525.20
tenth	Poland	Warsaw	1805681	517.00
eleventh	Spain	Barcelona	1602386	101.90
twelvth	Germany	Munich	1805681	310.40
thirteenth	Italy	Milan	1350680	181.80

Sorting

In [88]:

```
import pandas as pd
cities = {"cityname": ["London", "Berlin", "Madrid", "Rome",
                       "Paris", "Vienna", "Bucharest", "Hamburg",
                       "Budapest", "Warsaw", "Barcelona",
                       "Munich", "Milan"],
          "population": [8615246, 3562166, 3165235, 2874038,
                         2273305, 1805681, 1803425, 1760433,
                         1754000, 1805681, 1602386, 1805681,
                         1350680],
          "country": ["England", "Germany", "Spain", "Italy",
                      "France", "Austria", "Romania",
                      "Germany", "Hungary", "Poland", "Spain",
                      "Germany", "Italy"],
          "area"    : [1572, 891.85, 605.77, 1285, 105.4, 414.6,
                       228, 755, 525.2, 517, 101.9, 310.4, 181.8]
         }

ordinals = ["first", "second", "third", "fourth",
            "fifth", "sixth", "seventh", "eigth",
            "ninth", "tenth", "eleventh", "twelvth",
            "thirteenth"]

city_frame = pd.DataFrame(cities, index=ordinals)

city_frame
```

Out[88]:

	area	cityname	country	population
first	1572.00	London	England	8615246
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fourth	1285.00	Rome	Italy	2874038
fifth	105.40	Paris	France	2273305
sixth	414.60	Vienna	Austria	1805681
seventh	228.00	Bucharest	Romania	1803425
eigth	755.00	Hamburg	Germany	1760433
ninth	525.20	Budapest	Hungary	1754000
tenth	517.00	Warsaw	Poland	1805681
eleventh	101.90	Barcelona	Spain	1602386
twelvth	310.40	Munich	Germany	1805681
thirteenth	181.80	Milan	Italy	1350680

Sorting DataFrame on column 'population':

In [89]:

```
city_frame = city_frame.sort_values("population", ascending=False)
city_frame
```

Out[89]:

	area	cityname	country	population
first	1572.00	London	England	8615246
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fourth	1285.00	Rome	Italy	2874038
fifth	105.40	Paris	France	2273305
sixth	414.60	Vienna	Austria	1805681
tenth	517.00	Warsaw	Poland	1805681
twelvth	310.40	Munich	Germany	1805681
seventh	228.00	Bucharest	Romania	1803425
eigth	755.00	Hamburg	Germany	1760433

Sorting DataFrame on multiple columns:

In [90]:

```
city_frame = city_frame.sort_values(["population", 'area'], ascending=False)
city_frame
```

Out[90]:

	area	cityname	country	population
first	1572.00	London	England	8615246
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fourth	1285.00	Rome	Italy	2874038
fifth	105.40	Paris	France	2273305
tenth	517.00	Warsaw	Poland	1805681
sixth	414.60	Vienna	Austria	1805681
twelvth	310.40	Munich	Germany	1805681
seventh	228.00	Bucharest	Romania	1803425
eigth	755.00	Hamburg	Germany	1760433
ninth	525.20	Budapest	Hungary	1754000
eleventh	101.90	Barcelona	Spain	1602386
thirteenth	181.80	Milan	Italy	1350680

In [91]:

```
# Sorting DataFrame on multiple columns but different sorting orders
city_frame = city_frame.sort_values(['population', 'area'], ascending=[False, True])
city_frame
```

Out[91]:

	area	cityname	country	population
first	1572.00	London	England	8615246
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fourth	1285.00	Rome	Italy	2874038
fifth	105.40	Paris	France	2273305
twelvth	310.40	Munich	Germany	1805681
sixth	414.60	Vienna	Austria	1805681
tenth	517.00	Warsaw	Poland	1805681
seventh	228.00	Bucharest	Romania	1803425
eigth	755.00	Hamburg	Germany	1760433
ninth	525.20	Budapest	Hungary	1754000
eleventh	101.90	Barcelona	Spain	1602386
thirteenth	181.80	Milan	Italy	1350680

In [92]:

```
city_frame.head()
```

Out[92]:

	area	cityname	country	population
first	1572.00	London	England	8615246
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fourth	1285.00	Rome	Italy	2874038
fifth	105.40	Paris	France	2273305

In [93]:

```
city_frame.tail()
```

Out[93]:

	area	cityname	country	population
seventh	228.0	Bucharest	Romania	1803425
eigth	755.0	Hamburg	Germany	1760433
ninth	525.2	Budapest	Hungary	1754000
eleventh	101.9	Barcelona	Spain	1602386
thirteenth	181.8	Milan	Italy	1350680

In [94]:

```
# A nested dictionary of dicts can be passed to a DataFrame as well.
# The indices of the outer dictionary are taken as the the columns
# and the inner keys. i.e. the keys of the nested dictionaries,
# are used as the row indices:

growth = {"Switzerland": {"2010": 3.0, "2011": 1.8, "2012": 1.1, "2013": 1.9},
          "Germany": {"2010": 4.1, "2011": 3.6, "2012": 0.4, "2013": 0.1},
          "France": {"2010":2.0,  "2011":2.1, "2012": 0.3, "2013": 0.3},
          "Greece": {"2010":-5.4, "2011":-8.9, "2012":-6.6, "2013": -3.3},
          "Italy": {"2010":1.7, "2011": 0.6, "2012":-2.3, "2013":-1.9}
          }
growth_frame = pd.DataFrame(growth)
growth_frame
```

Out[94]:

	France	Germany	Greece	Italy	Switzerland
2010	2.0	4.1	-5.4	1.7	3.0
2011	2.1	3.6	-8.9	0.6	1.8
2012	0.3	0.4	-6.6	-2.3	1.1
2013	0.3	0.1	-3.3	-1.9	1.9

In [95]:

```
# Transposing
growth_frame.T
```

Out[95]:

	2010	2011	2012	2013
France	2.0	2.1	0.3	0.3
Germany	4.1	3.6	0.4	0.1
Greece	-5.4	-8.9	-6.6	-3.3
Italy	1.7	0.6	-2.3	-1.9
Switzerland	3.0	1.8	1.1	1.9

In [96]:

```
growth_frame
```

Out[96]:

	France	Germany	Greece	Italy	Switzerland
2010	2.0	4.1	-5.4	1.7	3.0
2011	2.1	3.6	-8.9	0.6	1.8
2012	0.3	0.4	-6.6	-2.3	1.1
2013	0.3	0.1	-3.3	-1.9	1.9

Querying

All the rows which are having population greater than 2 million

In [97]:

```
city_frame[city_frame['population'] > 2000000]
```

Out[97]:

	area	cityname	country	population
first	1572.00	London	England	8615246
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fourth	1285.00	Rome	Italy	2874038
fifth	105.40	Paris	France	2273305

Filtering with mutlitple conditions using

- and - &
- or - |

In [98]:

```
city_frame[ (city_frame['population'] > 2000000) & (city_frame['area'] < 1000)]
```

Out[98]:

	area	cityname	country	population
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fifth	105.40	Paris	France	2273305

In [99]:

```
city_frame[(city_frame['population'] > 2000000) & (city_frame['area'] < 1000)]
```

Out[99]:

	area	cityname	country	population
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fifth	105.40	Paris	France	2273305

In [100]:

```
city_frame
```

Out[100]:

	area	cityname	country	population
first	1572.00	London	England	8615246
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fourth	1285.00	Rome	Italy	2874038
fifth	105.40	Paris	France	2273305
twelvth	310.40	Munich	Germany	1805681
sixth	414.60	Vienna	Austria	1805681
tenth	517.00	Warsaw	Poland	1805681
seventh	228.00	Bucharest	Romania	1803425
eigth	755.00	Hamburg	Germany	1760433
ninth	525.20	Budapest	Hungary	1754000
eleventh	101.90	Barcelona	Spain	1602386
thirteenth	181.80	Milan	Italy	1350680

setting custom index from a column

In [101]:

```
d = city_frame.set_index('cityname')
d
```

Out[101]:

cityname	area	country	population
London	1572.00	England	8615246
Berlin	891.85	Germany	3562166
Madrid	605.77	Spain	3165235
Rome	1285.00	Italy	2874038
Paris	105.40	France	2273305
Munich	310.40	Germany	1805681
Vienna	414.60	Austria	1805681
Warsaw	517.00	Poland	1805681
Bucharest	228.00	Romania	1803425
Hamburg	755.00	Germany	1760433
Budapest	525.20	Hungary	1754000
Barcelona	101.90	Spain	1602386
Milan	181.80	Italy	1350680

In [102]:

```
d.loc['Warsaw']
```

Out[102]:

```
area              517
country        Poland
population    1805681
Name: Warsaw, dtype: object
```

In [103]:

```
d.loc[['London', 'Hamburg']]
```

Out[103]:

cityname	area	country	population
London	1572.0	England	8615246
Hamburg	755.0	Germany	1760433

In [104]:

```
city_frame
```

Out[104]:

	area	cityname	country	population
first	1572.00	London	England	8615246
second	891.85	Berlin	Germany	3562166
third	605.77	Madrid	Spain	3165235
fourth	1285.00	Rome	Italy	2874038
fifth	105.40	Paris	France	2273305
twelvth	310.40	Munich	Germany	1805681
sixth	414.60	Vienna	Austria	1805681
tenth	517.00	Warsaw	Poland	1805681
seventh	228.00	Bucharest	Romania	1803425
eigth	755.00	Hamburg	Germany	1760433
ninth	525.20	Budapest	Hungary	1754000
eleventh	101.90	Barcelona	Spain	1602386
thirteenth	181.80	Milan	Italy	1350680

Multiple columns as index

In [105]:

```
d1 = city_frame.set_index(['cityname', 'country'])
d1
```

Out[105]:

		area	population
cityname	**country**		
London	**England**	1572.00	8615246
Berlin	**Germany**	891.85	3562166
Madrid	**Spain**	605.77	3165235
Rome	**Italy**	1285.00	2874038
Paris	**France**	105.40	2273305
Munich	**Germany**	310.40	1805681
Vienna	**Austria**	414.60	1805681
Warsaw	**Poland**	517.00	1805681
Bucharest	**Romania**	228.00	1803425
Hamburg	**Germany**	755.00	1760433
Budapest	**Hungary**	525.20	1754000
Barcelona	**Spain**	101.90	1602386
Milan	**Italy**	181.80	1350680

In [106]:

```
d1.loc[('Warsaw', 'Poland')]
```

Out[106]:

```
area              517.0
population    1805681.0
Name: (Warsaw, Poland), dtype: float64
```

In [107]:

```
d1.loc[[('Warsaw', 'Poland'), ('Milan', 'Italy')]]
```

Out[107]:

		area	population
cityname	**country**		
Warsaw	**Poland**	517.0	1805681
Milan	**Italy**	181.8	1350680

In [108]:

```
d1.sort_index(ascending=[True, False])
```

Out[108]:

		area	population
cityname	**country**		
Barcelona	**Spain**	101.90	1602386
Berlin	**Germany**	891.85	3562166
Bucharest	**Romania**	228.00	1803425
Budapest	**Hungary**	525.20	1754000
Hamburg	**Germany**	755.00	1760433
London	**England**	1572.00	8615246
Madrid	**Spain**	605.77	3165235
Milan	**Italy**	181.80	1350680
Munich	**Germany**	310.40	1805681
Paris	**France**	105.40	2273305
Rome	**Italy**	1285.00	2874038
Vienna	**Austria**	414.60	1805681
Warsaw	**Poland**	517.00	1805681

Concatenate, Merge, Join

Concatenate

The concat function (in the main pandas namespace) does all of the heavy lifting of performing concatenation

operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

Note that I say “if any” because there is only a single possible axis of concatenation for Series.

```
pd.concat(objs, axis=0, join='outer', join_axes=None, ignore_index=False, ke
ys=None, levels=None, names=None, verify_integrity=False, copy=True)
```

In [109]:

```
import pandas as pd
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
'B': ['B0', 'B1', 'B2', 'B3'],
'C': ['C0', 'C1', 'C2', 'C3'],
'D': ['D0', 'D1', 'D2', 'D3']},
index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
'B': ['B4', 'B5', 'B6', 'B7'],
'C': ['C4', 'C5', 'C6', 'C7'],
'D': ['D4', 'D5', 'D6', 'D7']},
index=[4, 5, 6, 7])

df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
'B': ['B8', 'B9', 'B10', 'B11'],
'C': ['C8', 'C9', 'C10', 'C11'],
'D': ['D8', 'D9', 'D10', 'D11']},
index=[8, 9, 10, 11])

df1
```

Out[109]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [110]:

```
df2
```

Out[110]:

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

In [111]:

```
df3
```

Out[111]:

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

In [112]:

```
frames = [df1, df2, df3]
result = pd.concat(frames)
result
```

Out[112]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

In [113]:

```
result = pd.concat(frames, axis=1)
result
```

Out[113]:

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

In [114]:

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
'B': ['B0', 'B1', 'B2', 'B3'],
'C': ['C0', 'C1', 'C2', 'C3'],
'D': ['D0', 'D1', 'D2', 'D3']},
index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
'B': ['B4', 'B5', 'B6', 'B7'],
'C': ['C4', 'C5', 'C6', 'C7'],
'F': ['D4', 'D5', 'D6', 'D7']},
index=[2, 3, 4, 5])
```

In [115]:

```
df1
```

Out[115]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [116]:

```
df2
```

Out[116]:

	A	B	C	F
2	A4	B4	C4	D4
3	A5	B5	C5	D5
4	A6	B6	C6	D6
5	A7	B7	C7	D7

In [117]:

```
df = pd.concat([df1, df2]) # default axis=0, join='outer'
```

In [118]:

```
df
```

Out[118]:

	A	B	C	D	F
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2	NaN
3	A3	B3	C3	D3	NaN
2	A4	B4	C4	NaN	D4
3	A5	B5	C5	NaN	D5
4	A6	B6	C6	NaN	D6
5	A7	B7	C7	NaN	D7

In [119]:

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

Out[119]:

	A	B	C	D	A	B	C	F
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	A4	B4	C4	D4
3	A3	B3	C3	D3	A5	B5	C5	D5
4	NaN	NaN	NaN	NaN	A6	B6	C6	D6
5	NaN	NaN	NaN	NaN	A7	B7	C7	D7

In [120]:

```
pd.concat([df1, df2],axis=0, join='outer')
```

Out[120]:

	A	B	C	D	F
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2	NaN
3	A3	B3	C3	D3	NaN
2	A4	B4	C4	NaN	D4
3	A5	B5	C5	NaN	D5
4	A6	B6	C6	NaN	D6
5	A7	B7	C7	NaN	D7

In [121]:

```
df1
```

Out[121]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [122]:

```
df2
```

Out[122]:

	A	B	C	F
2	A4	B4	C4	D4
3	A5	B5	C5	D5
4	A6	B6	C6	D6
5	A7	B7	C7	D7

In [123]:

```
pd.concat([df1, df2],axis=0, join='inner')
```

Out[123]:

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3
2	A4	B4	C4
3	A5	B5	C5
4	A6	B6	C6
5	A7	B7	C7

In [124]:

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

Out[124]:

	A	B	C	D	A	B	C	F
2	A2	B2	C2	D2	A4	B4	C4	D4
3	A3	B3	C3	D3	A5	B5	C5	D5

In [125]:

```
pd.concat([df1, df2],axis=0, join='inner', ignore_index=True)
```

Out[125]:

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3
4	A4	B4	C4
5	A5	B5	C5
6	A6	B6	C6
7	A7	B7	C7

MERGE

pandas has full-featured, high performance in-memory join operations idiomatically very similar to relational databases like SQL. Users who are familiar with SQL but new to pandas might be interested in a comparison with SQL.

pandas provides a single function, merge, as the entry point for all standard database join operations between DataFrame objects.

Syntax:

```
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True, suffixes=('_x', '_y'), copy=True, indicator=False)
```

In [126]:

```
df1 = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
'key2': ['K0', 'K1', 'K0', 'K1'],
'A': ['A0', 'A1', 'A2', 'A3'],
'B': ['B0', 'B1', 'B2', 'B3']})

df2 = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
'key2': ['K0', 'K0', 'K0', 'K0'],
'C': ['C0', 'C1', 'C2', 'C3'],
'D': ['D0', 'D1', 'D2', 'D3']})
```

In [127]:

```
df1
```

Out[127]:

	A	B	key1	key2
0	A0	B0	K0	K0
1	A1	B1	K0	K1
2	A2	B2	K1	K0
3	A3	B3	K2	K1

In [128]:

```
df2
```

Out[128]:

	C	D	key1	key2
0	C0	D0	K0	K0
1	C1	D1	K1	K0
2	C2	D2	K1	K0
3	C3	D3	K2	K0

In [129]:

```
pd.merge(df1, df2, how='outer', on=['key1', 'key2'])
```

Out[129]:

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	NaN	NaN
2	A2	B2	K1	K0	C1	D1
3	A2	B2	K1	K0	C2	D2
4	A3	B3	K2	K1	NaN	NaN
5	NaN	NaN	K2	K0	C3	D3

In [130]:

```
pd.merge(df1, df2, how='inner', on=['key1', 'key2'])
```

Out[130]:

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	C2	D2

In [131]:

```
df1
```

Out[131]:

	A	B	key1	key2
0	A0	B0	K0	K0
1	A1	B1	K0	K1
2	A2	B2	K1	K0
3	A3	B3	K2	K1

In [132]:

```
df2
```

Out[132]:

	C	D	key1	key2
0	C0	D0	K0	K0
1	C1	D1	K1	K0
2	C2	D2	K1	K0
3	C3	D3	K2	K0

In [133]:

```
df = pd.merge(df1, df2, how='left', on=['key1', 'key2'])
df
```

Out[133]:

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	NaN	NaN
2	A2	B2	K1	K0	C1	D1
3	A2	B2	K1	K0	C2	D2
4	A3	B3	K2	K1	NaN	NaN

In [134]:

```
df = pd.merge(df1, df2, how='right', on=['key1', 'key2'])
df
```

Out[134]:

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	C2	D2
3	NaN	NaN	K2	K0	C3	D3

In [135]:

```
result = pd.merge(df1, df2, on='key1', suffixes=('_1', '_2'))
result
```

Out[135]:

	A	B	key1	key2_1	C	D	key2_2
0	A0	B0	K0	K0	C0	D0	K0
1	A1	B1	K0	K1	C0	D0	K0
2	A2	B2	K1	K0	C1	D1	K0
3	A2	B2	K1	K0	C2	D2	K0
4	A3	B3	K2	K1	C3	D3	K0

In [136]:

```
result = pd.merge(df1, df2, on='key1', suffixes=('_df1', '_df2'))
result
```

Out[136]:

	A	B	key1	key2_df1	C	D	key2_df2
0	A0	B0	K0	K0	C0	D0	K0
1	A1	B1	K0	K1	C0	D0	K0
2	A2	B2	K1	K0	C1	D1	K0
3	A2	B2	K1	K0	C2	D2	K0
4	A3	B3	K2	K1	C3	D3	K0

JOIN

DataFrame.join is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame

In [137]:

```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
'B': ['B0', 'B1', 'B2']},
index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
'D': ['D0', 'D2', 'D3']},
index=['K0', 'K2', 'K3'])

left.join(right, how='inner')
```

Out[137]:

	A	B	C	D
K0	A0	B0	C0	D0
K2	A2	B2	C2	D2

In [138]:

```
left.join(right, how='outer')
```

Out[138]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

In [139]:

```
left.join(right, how='left')
```

Out[139]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

In [140]:

```
df = pd.DataFrame({'Alpha':   ['A', 'B', 'A', 'A', 'C', 'B', 'C', 'A', 'C', 'B'],
                   'Value1': [1,    5,   7,   3,   2,   5,   9,   1,   5,   3],
                   'Value2': [3,    4,   2,   1,   7,   7,   2,   6,   2,   4]})
df
```

Out[140]:

	Alpha	Value1	Value2
0	A	1	3
1	B	5	4
2	A	7	2
3	A	3	1
4	C	2	7
5	B	5	7
6	C	9	2
7	A	1	6
8	C	5	2
9	B	3	4

In [141]:

```
df.groupby('Alpha').max()
```

Out[141]:

Alpha	Value1	Value2
A	7	6
B	5	7
C	9	7

In [142]:

```
df.groupby('Alpha').min()
```

Out[142]:

Alpha	Value1	Value2
A	1	1
B	3	4
C	2	2

In [143]:

```
df.groupby('Alpha').count()
```

Out[143]:

Alpha	Value1	Value2
A	4	4
B	3	3
C	3	3

In [144]:

```
df.groupby(['Alpha', 'Value1']).count()
```

Out[144]:

Alpha	Value1	Value2
A	**1**	2
	3	1
	7	1
B	**3**	1
	5	2
C	**2**	1
	5	1
	9	1

Importing Exporting CSV, EXCEL

In [145]:

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.randn(10, 5),
                  columns=['a', 'b', 'c', 'd', 'e'])
df
```

Out[145]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [146]:

```
df.to_csv('random_data.csv', sep=',', index=False)
```

In [147]:

```
df = pd.read_csv('random_data.csv')
df
```

Out[147]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [148]:

```
df.to_excel('random_data.xlsx', sheet_name='first_sheet')
```

In [149]:

```
pd.read_excel('random_data.xlsx', 'first_sheet')
```

Out[149]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [152]:

```
pd.read_excel('random_data.xlsx', 'first_sheet', use_cols=2)
```

Out[152]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [153]:

```
import pandas as pd
pd.read_excel('random_data.xlsx', 'first_sheet', use_cols=[0, 2, 4])
```

Out[153]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [154]:

```
pd.read_excel('random_data.xlsx', 'first_sheet', converters={'b': bool})
```

Out[154]:

	a	b	c	d	e
0	0.498639	True	0.899336	-0.962712	1.491917
1	-0.500365	True	-0.118741	0.218319	-0.089533
2	-0.261705	True	-1.336572	-0.127524	0.121660
3	0.065880	True	1.595831	-0.005255	0.069725
4	0.454241	True	0.200486	1.376159	-0.511748
5	-0.062856	True	1.751692	0.534487	0.286025
6	-2.330983	True	-1.436557	0.046854	-0.924108
7	0.563965	True	0.670361	-0.379618	0.673365
8	-0.277680	True	1.001299	-0.414218	0.818630
9	-2.170096	True	-0.805533	0.681785	-0.298607

In [155]:

```
import pandas as pd
cfun = lambda x: x if x > 0 else 0

pd.read_excel('random_data.xlsx', 'first_sheet', converters={'b': cfun})
```

Out[155]:

	a	b	c	d	e
0	0.498639	0.000000	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	0.000000	-1.336572	-0.127524	0.121660
3	0.065880	0.000000	1.595831	-0.005255	0.069725
4	0.454241	0.000000	0.200486	1.376159	-0.511748
5	-0.062856	0.000000	1.751692	0.534487	0.286025
6	-2.330983	0.000000	-1.436557	0.046854	-0.924108
7	0.563965	0.000000	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

Writing data to sql databses(MySQL):

In [158]:

```
import numpy as np
import pandas as pd
import sqlalchemy

df = pd.DataFrame(np.random.randn(10, 5),
                  columns=['a', 'b', 'c', 'd', 'e'])

engine = sqlalchemy.create_engine('mysql+mysqlconnector://naren:Python@7@localhost/s
con = engine.connect()
df.to_sql('sample_table', con)
con.close()
```

Reading data from sql databses(MySQL):

Reading SQL table:

In [159]:

```
import sqlalchemy
import pandas as pd
engine = sqlalchemy.create_engine('mysql+mysqlconnector://naren:Python@7@localhost/s
con = engine.connect()
df = pd.read_sql_table('sample_table', con)
con.close()
df
```

Out[159]:

	index	a	b	c	d	e
0	0	0.258138	1.280365	1.791044	-0.232482	-0.680663
1	1	1.806833	0.500571	0.021831	-1.633286	-0.673087
2	2	-0.834157	-1.794341	0.044068	-0.080284	-0.200498
3	3	-0.118295	-0.459052	0.605709	-0.129225	-0.426980
4	4	0.966566	0.283201	-0.297906	1.146471	1.900071
5	5	-0.506705	0.741484	-1.978576	-0.793495	0.511310
6	6	-1.974932	0.926235	0.391037	0.765724	0.540563
7	7	-1.044598	-0.463041	-0.893861	-0.426781	-0.749401
8	8	0.744666	-0.676293	0.689323	-0.719577	-0.141944
9	9	-1.271759	-1.608812	-2.400918	0.536314	0.088221

Running SQL query:

In [160]:

```
import sqlalchemy
import pandas as pd
engine = sqlalchemy.create_engine('mysql+mysqlconnector://naren:Python@7@localhost/s
con = engine.connect()
df = pd.read_sql_query('select a, c, e from sample_table limit 5', con)
con.close()
df
```

Out[160]:

	a	c	e
0	0.258138	1.791044	-0.680663
1	1.806833	0.021831	-0.673087
2	-0.834157	0.044068	-0.200498
3	-0.118295	0.605709	-0.426980
4	0.966566	-0.297906	1.900071

In [161]:

```
import sqlalchemy
import pandas as pd
engine = sqlalchemy.create_engine('mysql+mysqlconnector://naren:Python@7@localhost/s
con = engine.connect()
df = pd.read_sql('select a, c, e from sample_table limit 5', con)
con.close()
df
```

Out[161]:

	a	c	e
0	0.258138	1.791044	-0.680663
1	1.806833	0.021831	-0.673087
2	-0.834157	0.044068	-0.200498
3	-0.118295	0.605709	-0.426980
4	0.966566	-0.297906	1.900071

Handling missing data

In [162]:

```
import pandas as pd
df = pd.read_csv('random_data.csv')
```

In [163]:

```
df
```

Out[163]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [164]:

```
df.fillna(method='ffill')
```

Out[164]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [165]:

```
df.fillna(method='ffill', limit=2)
```

Out[165]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [166]:

```
df.fillna(method='bfill')
```

Out[166]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [167]:

```
df.fillna(df.mean())
```

Out[167]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

Matplotlib

In [168]:

```
%matplotlib inline
import matplotlib.pyplot as plt

# only for macbook retina
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('retina')

print (plt.style.available)
```

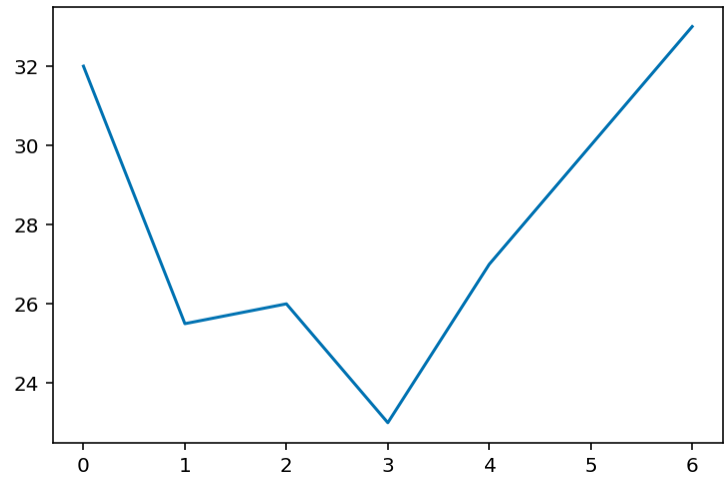
```
['seaborn-dark', 'seaborn-darkgrid', 'seaborn-ticks', 'fivethirtyeigh
t', 'seaborn-whitegrid', 'classic', '_classic_test', 'fast', 'seaborn-
talk', 'seaborn-dark-palette', 'seaborn-bright', 'seaborn-pastel', 'gr
ayscale', 'seaborn-notebook', 'ggplot', 'seaborn-colorblind', 'seaborn
-muted', 'seaborn', 'Solarize_Light2', 'seaborn-paper', 'bmh', 'tablea
u-colorblind10', 'seaborn-white', 'dark_background', 'seaborn-poster',
'seaborn-deep']
```

In [169]:

```
plt.style.use('seaborn-colorblind')
```

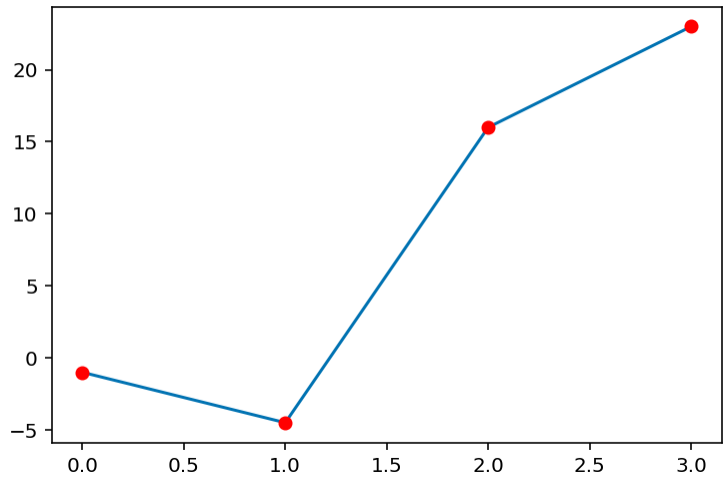
In [170]:

```
plt.plot([32, 25.5,  26, 23, 27, 30, 33])
plt.show()
```



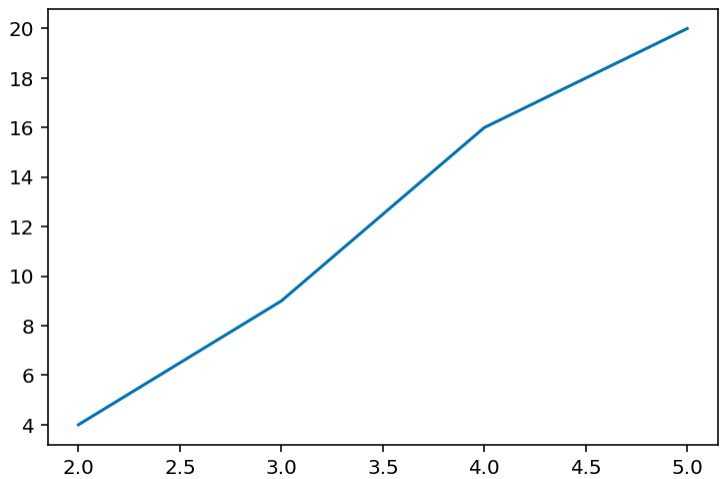
In [171]:

```
import matplotlib.pyplot as plt
plt.plot([-1, -4.5, 16, 23], "-")
plt.plot([-1, -4.5, 16, 23], "or")
plt.show()
```



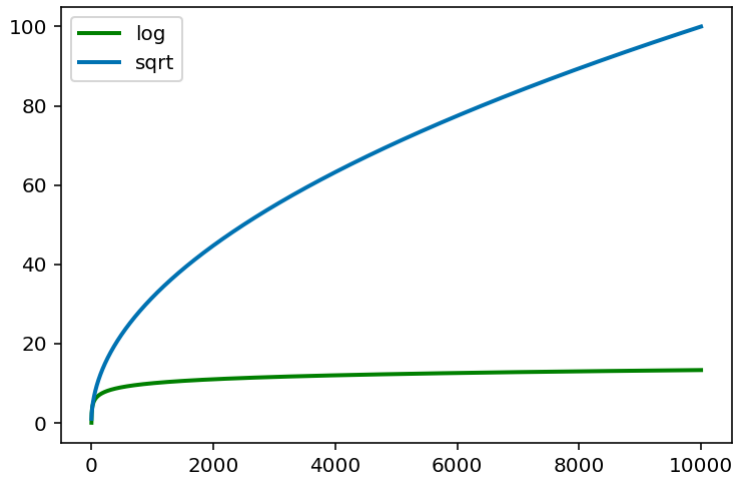
In [172]:

```
import matplotlib.pyplot as plt
x = [2, 3, 4, 5]
y = [4, 9, 16, 20]

plt.plot(x, y)
plt.show()
```



In [173]:

```
import matplotlib.pyplot as plt
import math

x = range(1, 10000)
y1 = [math.log(i, 2) for i in x]
y2 = [math.sqrt(i) for i in x]

plt.plot(x, y1, '-g', label='log', linewidth=2)
plt.plot(x, y2, label= 'sqrt', linewidth=2)

plt.legend()
plt.show()
```



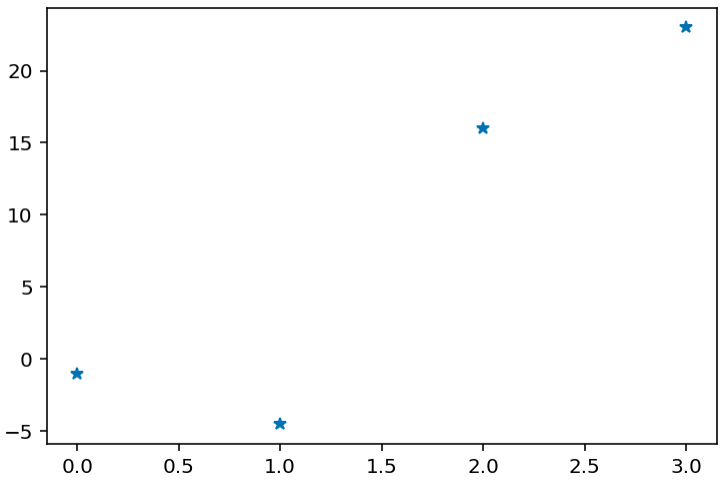
The format parameter of pyplot.plot

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker

```
'H'             hexagon2 marker
'+'             plus marker
'x'             x marker
'D'             diamond marker
'd'             thin_diamond marker
'|'             vline marker
'_'             hline marker
================================================
Colors:
==========  ========
'b'         blue
'g'         green
'r'         red
'c'         cyan
'm'         magenta
'y'         yellow
'k'         black
'w'         white
==========  ========
```

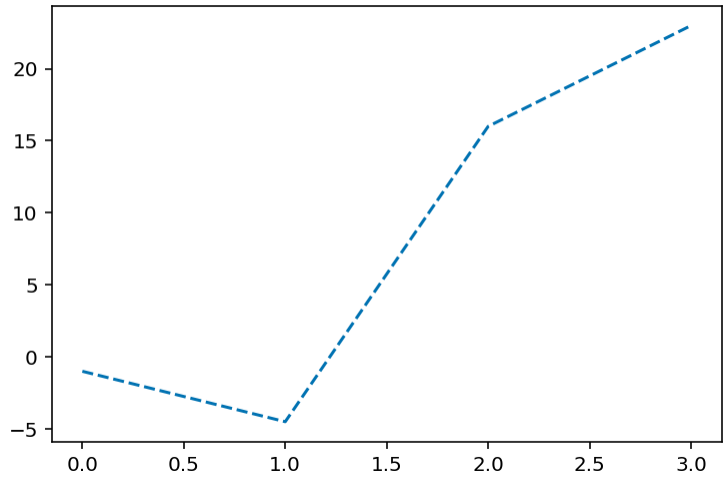
In [174]:

```
import matplotlib.pyplot as plt
plt.plot([-1, -4.5, 16, 23], "*")
plt.show()
```



In [175]:

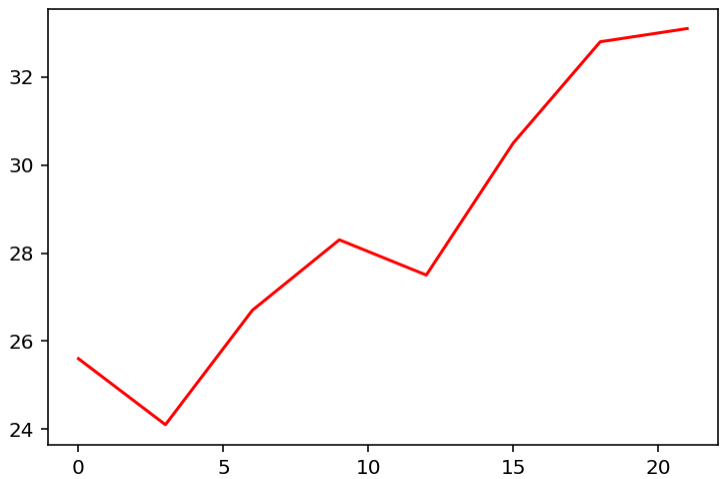
```
import matplotlib.pyplot as plt
plt.plot([-1, -4.5, 16, 23], "--")
plt.show()
```



In [176]:

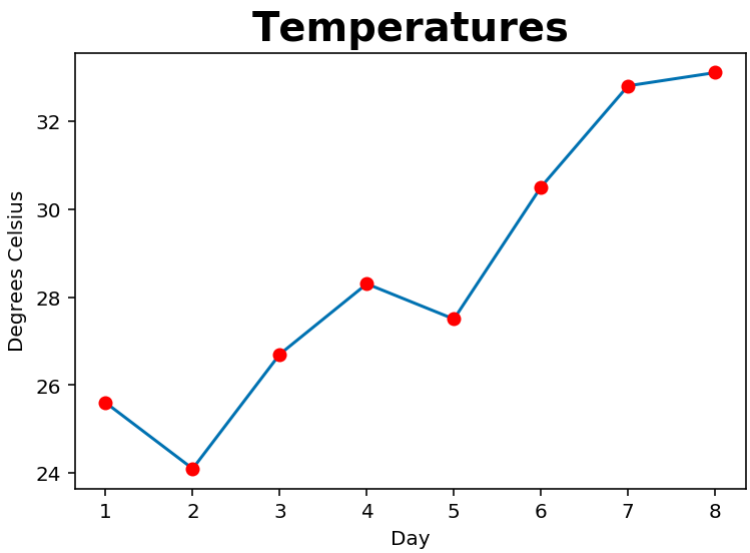
```
# our X values:
days = list(range(0, 22, 3))
print(days)
# our Y values:
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
plt.plot(days, celsius_values, 'r')
plt.show()
```

```
[0, 3, 6, 9, 12, 15, 18, 21]
```



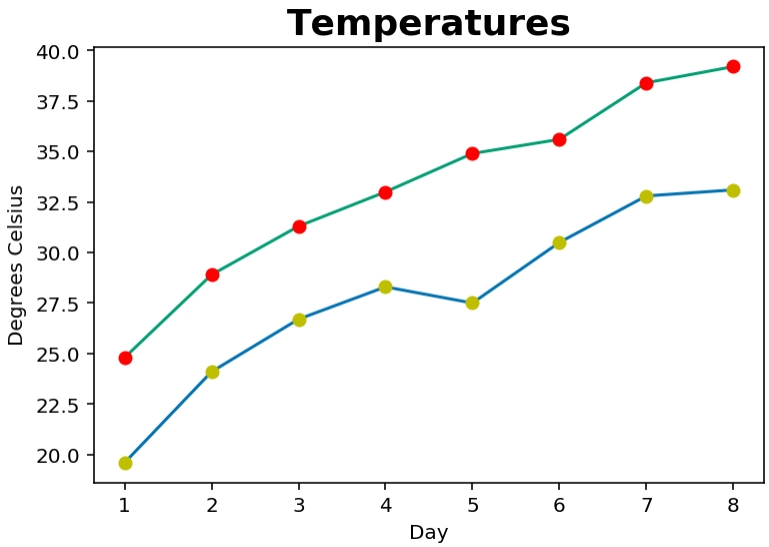
In [177]:

```
# Labels
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]

plt.plot(days, celsius_values)
plt.plot(days, celsius_values, "or")

plt.xlabel('Day')
plt.ylabel('Degrees Celsius')
plt.title('Temperatures', fontsize=20, loc='center', fontweight='bold')
plt.show()
```



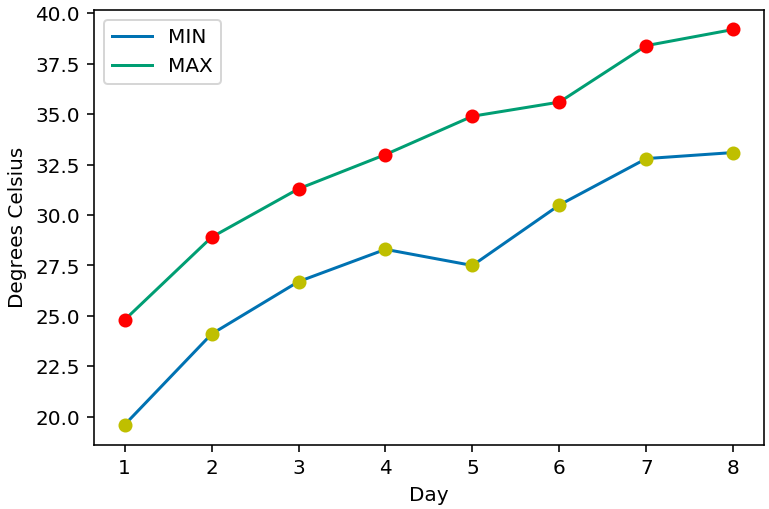
In [178]:

```
# Orbitary number of
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]
plt.xlabel('Day')
plt.ylabel('Degrees Celsius')
plt.title('Temperatures', fontsize=18, loc='center', fontweight='bold')
plt.plot(days, celsius_min,
         days, celsius_min, "oy",
         days, celsius_max,
         days, celsius_max, "or")
plt.show()
```



In [179]:

```
# Orbitary number of
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]
plt.xlabel('Day')
plt.ylabel('Degrees Celsius')
plt.plot(days, celsius_min, label='MIN')
plt.plot(days, celsius_min, "oy")
plt.plot(days, celsius_max, label='MAX')
plt.plot(days, celsius_max, "or")
plt.legend(loc='best')
plt.show()
```



Legend positions:

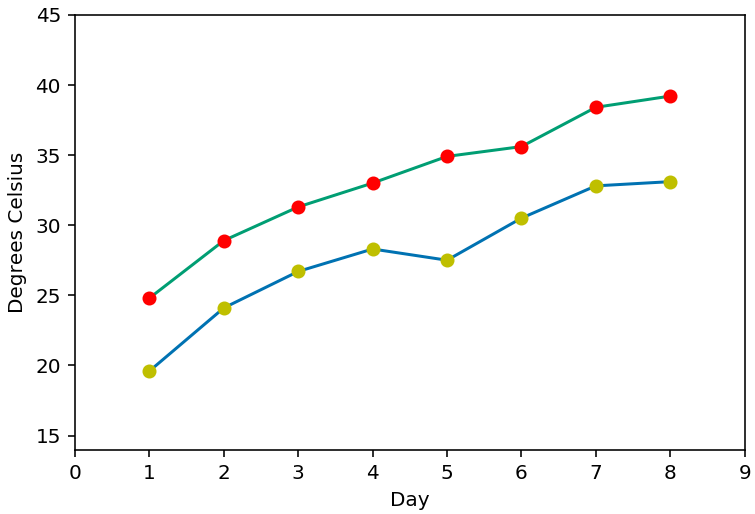
Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

axis(): function

In [180]:

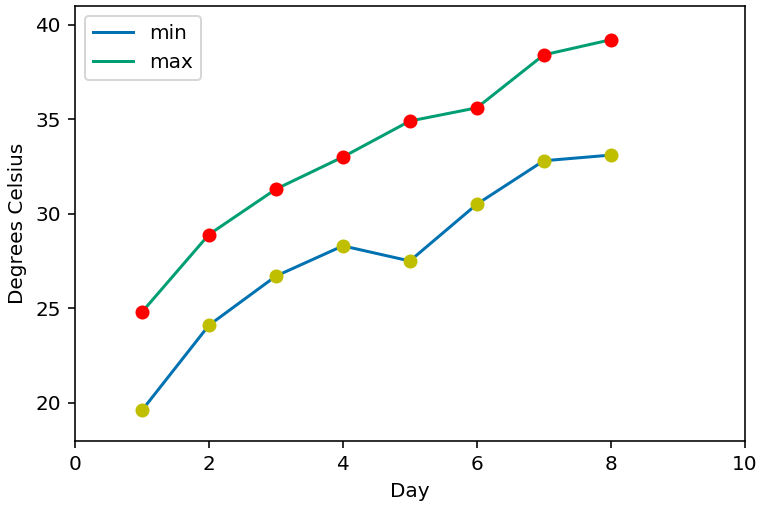
```
days = list(range(1,9))
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]
plt.xlabel('Day')
plt.ylabel('Degrees Celsius')
plt.plot(days, celsius_min,
         days, celsius_min, "oy",
         days, celsius_max,
         days, celsius_max, "or")

print("The current limits for the axes are:")
print(plt.axis())

print("We set the axes to the following values:")
xmin, xmax, ymin, ymax = 0, 9, 14, 45
print(xmin, xmax, ymin, ymax)
# Setting min, max limits
plt.axis([xmin, xmax, ymin, ymax])
plt.show()
```

```
The current limits for the axes are:
(0.6499999999999999, 8.35, 18.62, 40.18)
We set the axes to the following values:
0 9 14 45
```



In [181]:

```
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]
plt.xlabel('Day')
plt.ylabel('Degrees Celsius')
plt.plot(days, celsius_min, label='min')
plt.plot(days, celsius_min, "oy")

plt.plot(days, celsius_max, label='max')
plt.plot(days, celsius_max, "or")

plt.axis([0, 10, 18, 41])
plt.legend(loc='upper left')
plt.show()
```



"linspace" to Define X Values

In [182]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [183]:

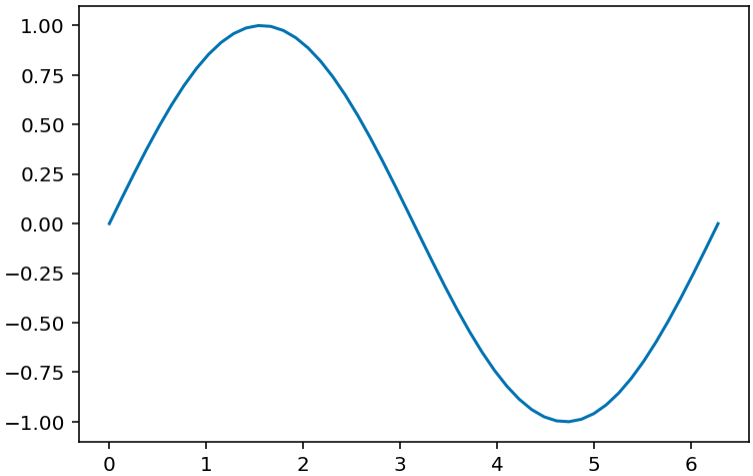
```
np.linspace(0, 15, 100)
```

Out[183]:

```
array([ 0.        ,  0.15151515,  0.3030303 ,  0.45454545,  0.6060606
1,
        0.75757576,  0.90909091,  1.06060606,  1.21212121,  1.3636363
6,
        1.51515152,  1.66666667,  1.81818182,  1.96969697,  2.1212121
2,
        2.27272727,  2.42424242,  2.57575758,  2.72727273,  2.8787878
8,
        3.03030303,  3.18181818,  3.33333333,  3.48484848,  3.6363636
4,
        3.78787879,  3.93939394,  4.09090909,  4.24242424,  4.3939393
9,
        4.54545455,  4.6969697 ,  4.84848485,  5.        ,  5.1515151
5,
        5.3030303 ,  5.45454545,  5.60606061,  5.75757576,  5.9090909
1,
        6.06060606,  6.21212121,  6.36363636,  6.51515152,  6.6666666
7,
        6.81818182,  6.96969697,  7.12121212,  7.27272727,  7.4242424
2,
        7.57575758,  7.72727273,  7.87878788,  8.03030303,  8.1818181
8,
        8.33333333,  8.48484848,  8.63636364,  8.78787879,  8.9393939
4,
        9.09090909,  9.24242424,  9.39393939,  9.54545455,  9.6969697
,
        9.84848485, 10.        , 10.15151515, 10.3030303 , 10.4545454
5,
       10.60606061, 10.75757576, 10.90909091, 11.06060606, 11.2121212
1,
       11.36363636, 11.51515152, 11.66666667, 11.81818182, 11.9696969
7,
       12.12121212, 12.27272727, 12.42424242, 12.57575758, 12.7272727
3,
       12.87878788, 13.03030303, 13.18181818, 13.33333333, 13.4848484
8,
       13.63636364, 13.78787879, 13.93939394, 14.09090909, 14.2424242
4,
       14.39393939, 14.54545455, 14.6969697 , 14.84848485, 15.
])
```

In [184]:

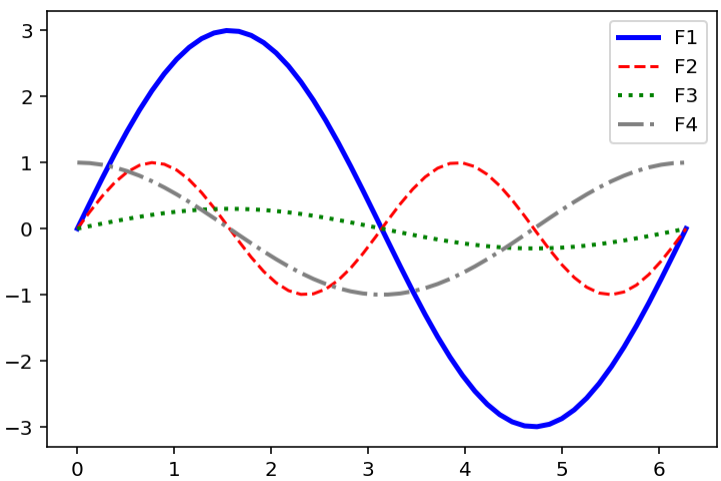
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(0, 2 * np.pi, 50)
Y = np.sin(X)
plt.plot(X,Y)

plt.show()
```



Changing the Line Style

In [185]:

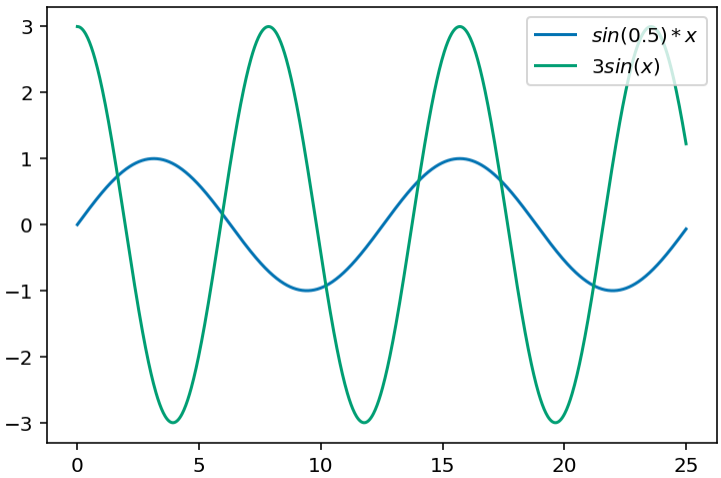
```
import matplotlib.pyplot as plt
X = np.linspace(0, 2 * np.pi, 50, endpoint=True)
F1 = 3 * np.sin(X)
F2 = np.sin(2*X)
F3 = 0.3 * np.sin(X)
F4 = np.cos(X)
plt.plot(X, F1, color="blue", linewidth=2.5, linestyle="-", label='F1')
plt.plot(X, F2, color="red", linewidth=1.5, linestyle="--", label='F2')
plt.plot(X, F3, color="green", linewidth=2, linestyle=":", label='F3')
plt.plot(X, F4, color="grey", linewidth=2, linestyle="-.", label='F4')
plt.legend(loc='best')
plt.show()
```



Legends

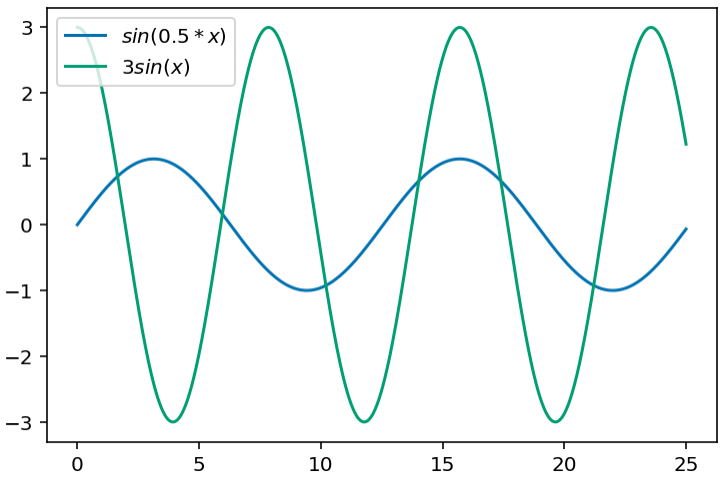
In [186]:

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 25, 1000)
F1 = np.sin(0.5 * X)
F2 = 3 * np.cos(0.8*X)
plt.plot(X, F1, label="$sin(0.5) * x$")
plt.plot(X, F2, label="$3 sin(x)$")
plt.legend(loc='upper right')
plt.show()
```



In [187]:

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 25, 1000)
F1 = np.sin(0.5 * X)
F2 = 3 * np.cos(0.8*X)
plt.plot(X, F1, label="$sin(0.5 * x)$")
plt.plot(X, F2, label="$3 sin(x)$")
plt.legend(loc='best')
plt.show()
```

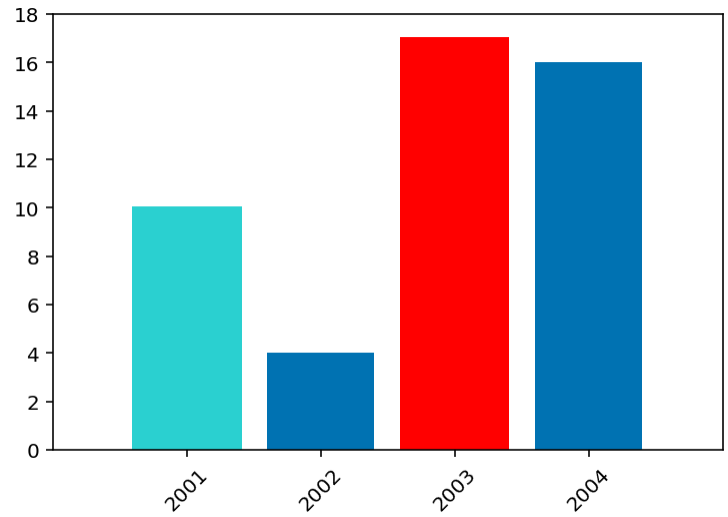


Bar Charts and Histograms

Histograms are used to show distributions of variables while bar charts are used to compare variables. Histograms plot quantitative data with ranges of the data grouped into bins or intervals while bar charts plot categorical data.

In [188]:

```
# Grid lines
import numpy as np
import matplotlib.pyplot as plt

y = [10,4,17,16]
x = range(1, len(y)+1)
bars = plt.bar(x, y)

bars[0].set_color('#2AD0D0')
bars[2].set_color('red')

plt.xticks(x, ('2001', '2002', '2003', '2004'), rotation=45)

plt.axis([0, 5, 0, 18])
plt.show()
```



In [189]:

```
import pandas as pd
cities = {"cityname": ["London", "Berlin", "Madrid", "Rome",
                   "Paris", "Vienna", "Bucharest", "Hamburg",
                   "Budapest", "Warsaw", "Barcelona",
                   "Munich", "Milan"],

          "population": [1615246,1803425, 3165235, 2874038,
                         1805681,  1760433, 1602386, 1805681,

                         1754000, 1805681, 2562166, 1350680, 1803425]


         }
df = pd.DataFrame(cities, index=range(1, len(cities['cityname'])+1))
df
```

Out[189]:

	cityname	population
1	London	1615246
2	Berlin	1803425
3	Madrid	3165235
4	Rome	2874038
5	Paris	1805681
6	Vienna	1760433
7	Bucharest	1602386
8	Hamburg	1805681
9	Budapest	1754000
10	Warsaw	1805681
11	Barcelona	2562166
12	Munich	1350680
13	Milan	1803425

In [190]:

```
hex(0x234567)[2:]
```

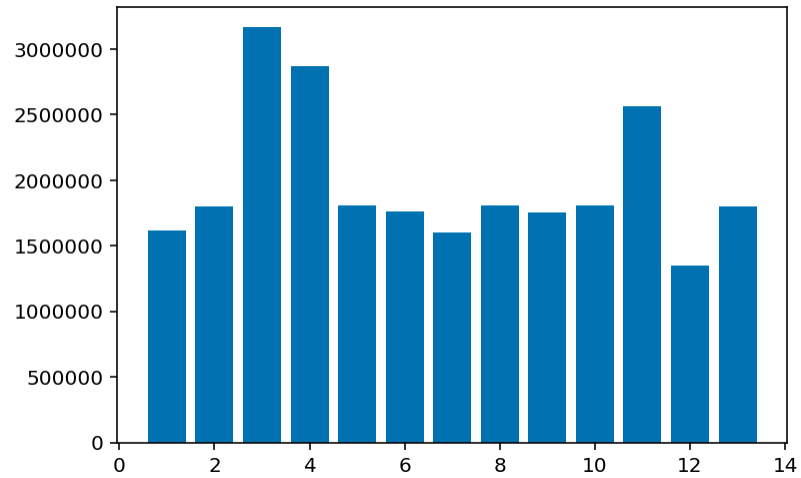
Out[190]:

```
'234567'
```

In [191]:

```
import pandas as pd

bars = plt.bar(df.index.values, df['population'].values)
base = 0x2AD0D0
for bar in bars:
    base += random.choice(range(1000, 2000, 100))
    bar.set_color('#' + hex(base)[2:])

plt.xticks(df.index.values, df['cityname'].values , rotation=45)

#plt.yticks(df['population'].values, df['population'].values)
plt.title('Pupulation')
plt.show()
```

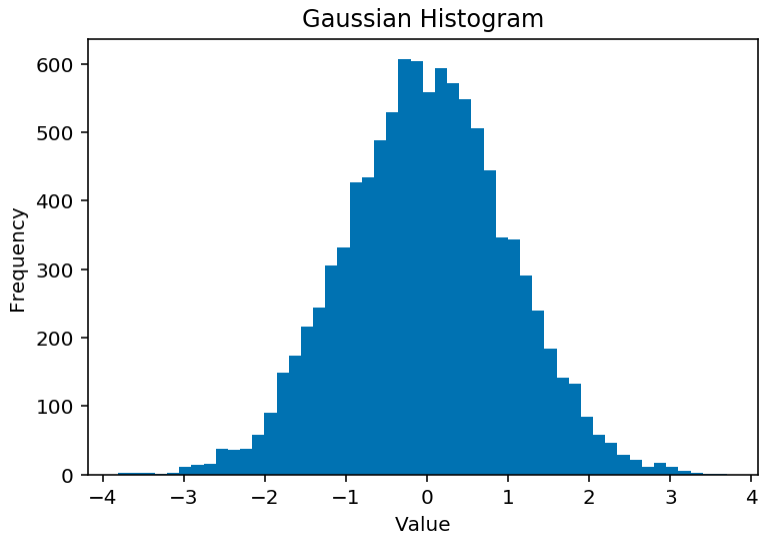
```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-191-d5cad64bb3ce> in <module>()
      4 base = 0x2AD0D0
      5 for bar in bars:
----> 6     base += random.choice(range(1000, 2000, 100))
      7     bar.set_color('#' + hex(base)[2:])
      8

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



Histograms

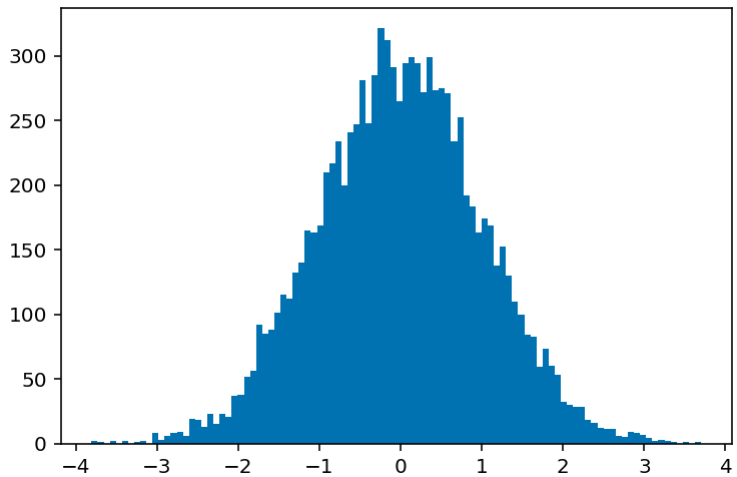
In [192]:

```
# the next "inline" statement is only needed,
# if you are working with "ipython notebook"
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
gaussian_numbers = np.random.normal(size=10000)
plt.hist(gaussian_numbers, bins=50)

plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```



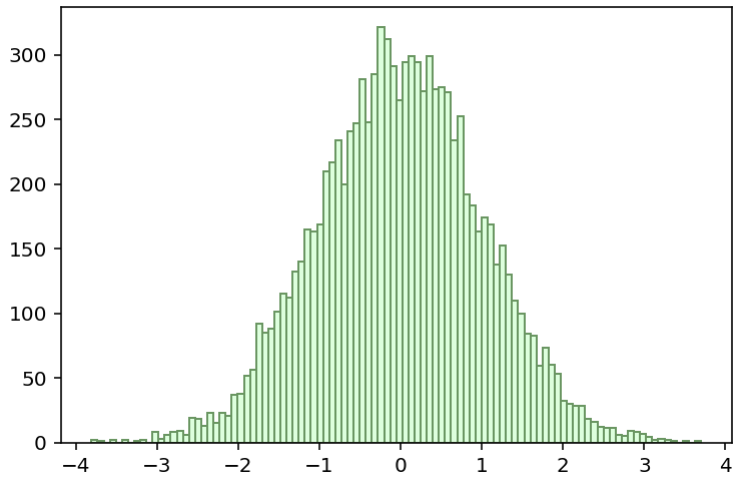
In [193]:

```
plt.hist(gaussian_numbers, bins=100)
plt.show()
```



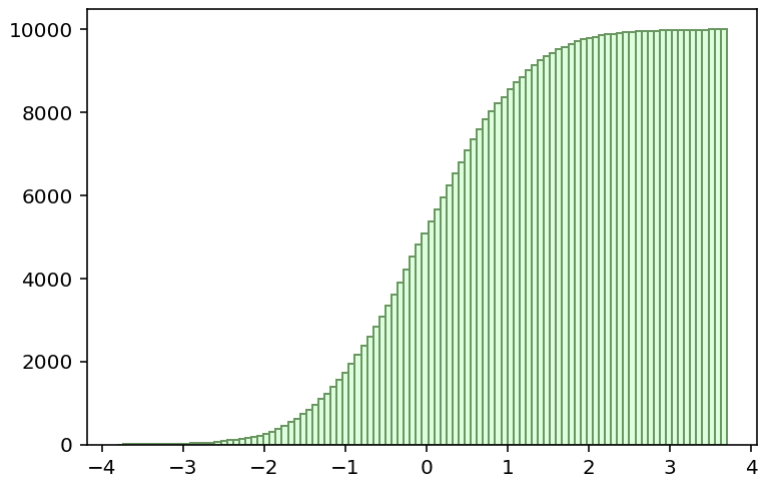
In [194]:

```
plt.hist(gaussian_numbers,
         bins=100,
         edgecolor="#6A9662",
         color="#DDFFDD")
plt.show()
```



In [195]:

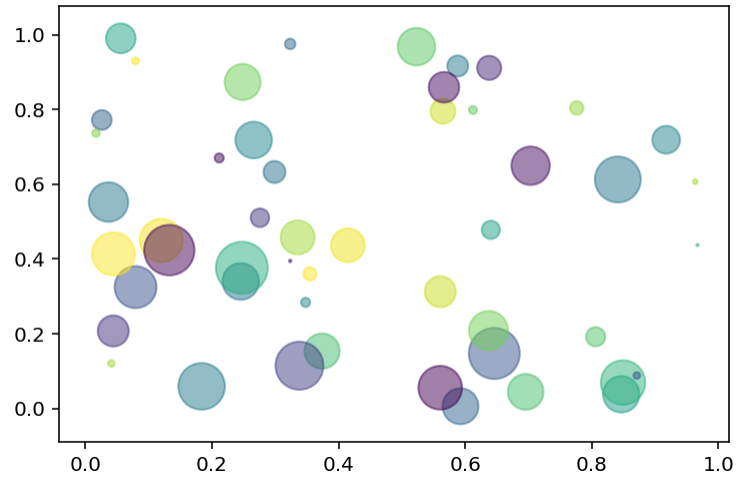
```
plt.hist(gaussian_numbers,
         bins=100,
         edgecolor="#6A9662",
         color="#DDFFDD",
         cumulative=True)
plt.show()
```



Scatter plot

In [196]:

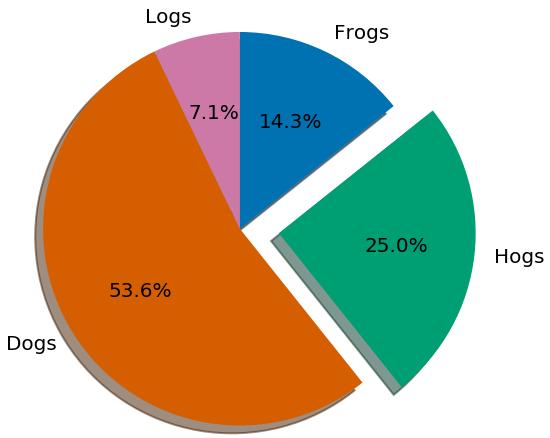
```
"""
Simple demo of a scatter plot.
"""
import numpy as np
import matplotlib.pyplot as plt

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2  # 0 to 15 point radii

# plt.scatter(x, y)
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



pie chart

In [197]:

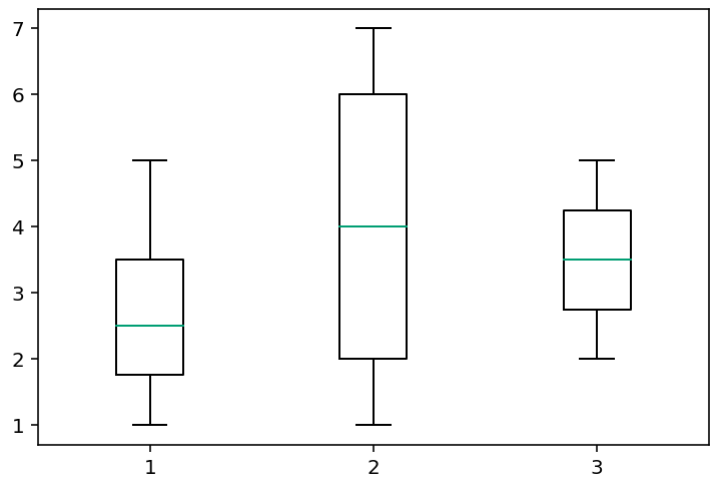
```
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [200, 350, 750, 100]
explode = (0, 0.2, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90, counterclock=False)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



Box plot

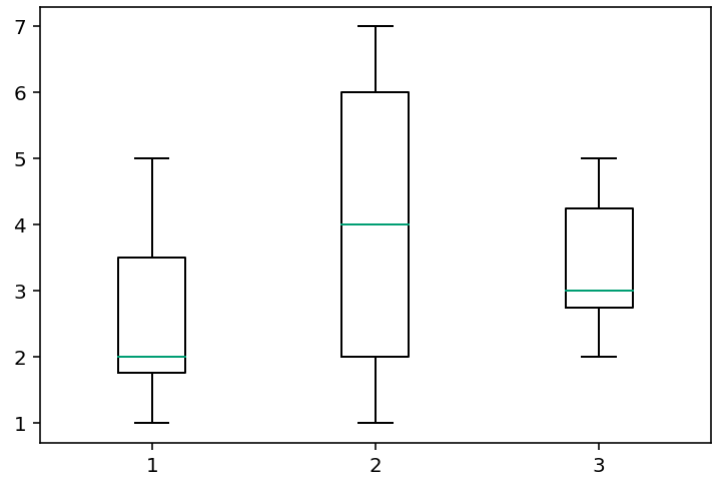
In [198]:

```
import matplotlib.pyplot as plt
import numpy as np

data2 = [[2, 3, 5, 1], [4, 6, 2, 1, 7], [3, 4, 2, 5]]
bp = plt.boxplot(data2)
plt.show()
```



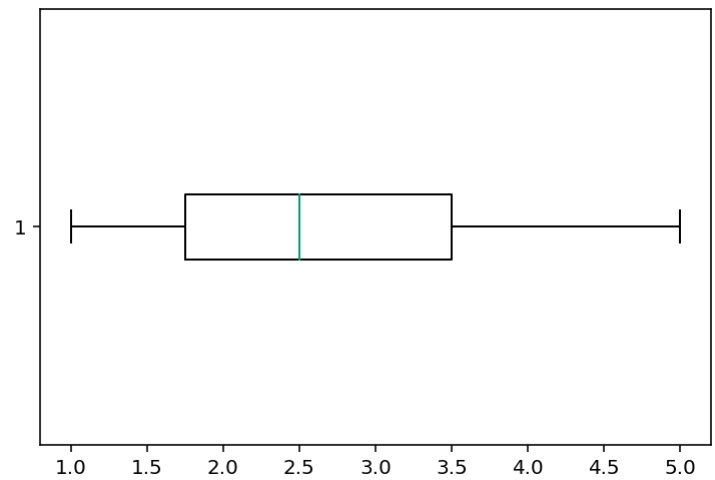
In [199]:

```
import matplotlib.pyplot as plt
import numpy as np

data2 = [[2, 3, 5, 1], [4, 6, 2, 1, 7], [3, 4, 2, 5]]
bp = plt.boxplot(data2, usermedians=[2, 4, 3])
plt.show()
```



In [200]:

```
import matplotlib.pyplot as plt
import numpy as np

data2 = [[2, 3, 5, 1], [3, 6, 2, 1, 7], [2, 4, 3, 1]]
bp = plt.boxplot([2, 3, 5, 1], vert=False)
plt.show()
```



In [201]:

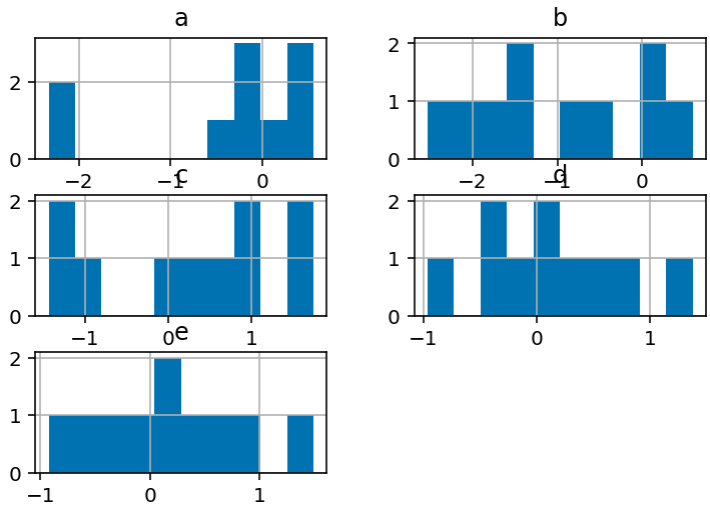
```
df = pd.read_excel('random_data.xlsx', 'first_sheet')
df
```

Out[201]:

	a	b	c	d	e
0	0.498639	-0.842658	0.899336	-0.962712	1.491917
1	-0.500365	0.597967	-0.118741	0.218319	-0.089533
2	-0.261705	-2.533009	-1.336572	-0.127524	0.121660
3	0.065880	-1.746893	1.595831	-0.005255	0.069725
4	0.454241	-1.924589	0.200486	1.376159	-0.511748
5	-0.062856	-1.415372	1.751692	0.534487	0.286025
6	-2.330983	-0.475604	-1.436557	0.046854	-0.924108
7	0.563965	-1.369186	0.670361	-0.379618	0.673365
8	-0.277680	0.034277	1.001299	-0.414218	0.818630
9	-2.170096	0.181913	-0.805533	0.681785	-0.298607

In [202]:

```
import matplotlib.pyplot as plt
import numpy as np
#data2 = [[2, 3, 5, 1], [3, 6, 2, 1, 7], [2, 4, 3, 1]]
df.hist()
plt.show()
```



Performance

Creating a large data frame:

In [ ]:

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(10**6, 26),
                  columns=[chr(x) for x in range(65, 65+26)])
```

Calculate time taken to save it ti csv:

In [ ]:

```
%%timeit
df.to_csv('large_data.csv')
```

Reading and writing Chunk by chunk is faster than reading at a time

In [ ]:

```
# Reading large csv
tp = pd.read_csv('large_data.csv', iterator=True, chunksize=10000, engine='c', index
df = pd.concat(tp, ignore_index=True)
df.head()
```

In [ ]:

```
# Writing large csv
```

In [ ]:

```
df.to_csv('processed_data.csv', sep=',', chunksize=10000, index=False)
```