Python Programming

by Narendra Allam

Chapter 11

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[] import pandas as pd
```

```
In[] 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[] S.index
```

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

```
In[] S.values
```

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

```
In[] S[4]
```

```
Output: 25.0
```

In[]
```
S = pd.Series([36, 32, 45, 30, 25, 40, 42], index=['Sunday', 'Monda
y',
                                                   'Tuesday', 'Wedn
essday',
                                                   'Thursday', 'Fri
day',
                                                   'Saturday'])
print S
```

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

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

Output: 40

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

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

In [ ]:

In[]
```
expected_dates = [1, 3, 4, 6, 8, 9, 10]

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

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

In[]
```
None == None
```

Output: True

In[]
```
import numpy as np
np.nan == np.nan
```

Output: False

In[]
```
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= 6.34428877022
```

In[]
```
s1.describe()
```

Output:
```
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[]
```
s1.isnull()
```

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

In[]
```
s1
```

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

In[]
```
s1[s1.isnull()] = 0
```

In[]
```
s1.describe()
```

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

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

```
dtype: int64
```

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

Output: 40

In[]
```
import numpy as np
print((S + 3) * 4)
print("======================")
```

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

In[]
```
np.sin(S)
```

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

In[]
```
S[S > 30]
```

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

In[]
```
S[S > 30] = [30, 40, 50]
```

In[]
```
S
```

Output:
```
apples       20
oranges      30
cherries     40
pears        10
Mango        50
dtype: int64
```

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

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

```
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
```

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

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

In[]
```
my_cities = ["London", "Paris", "Zurich", "Berlin",
             "Stuttgart", "Hamburg"]

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

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

In[]
```
my_city_series.isnull()
```

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

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

In[]
```
my_city_series
```

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

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

```
In[] city_series
```

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

```
In[] 
s = pd.Series(my_city_series, dtype='uint32')
s
```

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

np.nan is not zero

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

```
In[] s1.describe()
```

```
Output: 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[] s2.describe()
```

```
Output: 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[]
```
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"]}

city_frame = pd.DataFrame(cities)

city_frame
```

Output:

	cityname	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[]

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

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

Output:

	cityname	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[]
```
city_frame = pd.DataFrame(city_frame,
                          columns=[ "country",
                                    "cityname",
                                    "population"])
city_frame
```

Output:

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

In[]
```
city_frame
```

Output:

	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[]
```
city_frame.rename(index = {'eigth':'seventh'})
```

Output:

	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[]
```
grp = city_frame.groupby(city_frame.index)
```

In[]
```
city_frame
```

Output:

	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[]
```
city_frame['cityname']
```

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

In[]
```
city_frame['cityname']['eigth']
```

Output: `'Hamburg'`

In[]
```
city_frame.cityname['eigth']
```

Output: 'Hamburg'

In[]
```
city_frame.set_value('fourth', 'cityname', 'ROME_MODIFIED')
```

Output:

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

Output:

	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

Slicing and views

loc(), iloc()

In[]
```
city_frame
```

Output:

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

Output:

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

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

Output:

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

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

Output:
```
country          Romania
cityname       Bucharest
population       1803425
Name: seventh, dtype: object
```

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

Output:

	country	cityname
third	Spain	Madrid
fifth	France	Paris
seventh	Romania	Bucharest
ninth	Hungary	Budapest

Accessing Specific columns and rows

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

Output:

	country	population
first	England	8615246
sixth	Austria	1805681
tenth	Poland	1805681

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

Output:

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

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

Output:

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

In[]
```
city_frame.iloc[2:9, [0, 2]]
```

Output:

	country	population
third	Spain	3165235
fourth	Italy	2874038
fifth	France	2273305
sixth	Austria	1805681
seventh	Romania	1803425
eigth	Germany	1760433
ninth	Hungary	1754000

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

Output:

	country	cityname	population
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

In[] `city_frame`

Output:

	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[] `city_frame.sum()`

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

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

Output: `34177957L`

In[] `city_frame.any()`

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

In[]
```
x = city_frame["population"].cumsum()
print(x)
```

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

Adding a new column

In[]
```
import numpy as np

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

In[]
```
city_frame
```

Output:

	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
seventh	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[]
```
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[]
```
city_frame
```

Output:

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

In[]
```
df
```

Output:

	country	cityname	population	area
fifteenth	India	Hyderabad	15000000	700

In[]
```
city_frame = city_frame.append(df)
```

In[]
```
city_frame
```

Output:

	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
seventh	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
fifteenth	India	Hyderabad	15000000	700.00

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

In[]
```
city_frame
```

Output:

	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
seventh	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
eleventh	Spain	Barcelona	1602386	101.90	NaN
twelvth	Germany	Munich	1805681	310.40	NaN
thirteenth	Italy	Milan	1350680	181.80	NaN
fifteenth	India	Hyderabad	15000000	700.00	NaN

Deleting a column

In[]
```
city_frame.pop('extra')
```

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

In[] `city_frame`

Output:

	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
seventh	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
fifteenth	India	Hyderabad	15000000	700.00

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

Output:

	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
fifteenth	India	Hyderabad	15000000

In[] `city_frame`

Output:

	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
seventh	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
fifteenth	India	Hyderabad	15000000	700.00

In[] `city_frame.drop('fifteenth')`

Output:

	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
seventh	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[] `city_frame`

Output:

	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
seventh	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
fifteenth	India	Hyderabad	15000000	700.00

```
In[] city_frame.drop(['fifteenth', 'thirteenth'])
```

Output:

	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
seventh	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[] city_frame.drop('fifteenth', inplace=True)
```

In[]
```
city_frame
```

Output:

	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
seventh	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[]
```
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
```

Output:

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

Output:

	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
ninth	525.20	Budapest	Hungary	1754000
eleventh	101.90	Barcelona	Spain	1602386
thirteenth	181.80	Milan	Italy	1350680

Sorting DataFrame on multiple columns:

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

Output:

	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[]

```
city_frame = city_frame.sort_values(['population', 'area'], ascendi
ng=[False, True])
city_frame
```

Output:

	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[]

```
city_frame.head()
```

Output:

	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[]
```
city_frame.tail()
```

Output:

	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[]
```
growth = {"Switzerland": {"2010": 3.0, "2011": 1.8, "2012": 1.1, "2
013": 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
```

Output:

	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[]
```
growth_frame.T
```

Output:

	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[]
```
growth_frame
```

Output:

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

Output:

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

Output:

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

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

Output:

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

In[]
```
city_frame
```

Output:

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

Output:

	area	country	population
cityname			
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[]
```
d.loc['Warsaw']
```

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

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

Output:

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

In[]
```
city_frame
```

Output:

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

Output:

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

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

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

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

Output:

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

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

Output:

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

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=Fals
e, keys=None, levels=None, names=None, verify_integrity=False, copy=Tru
e)
```

In[]
```
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
```

Output:

	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[]
```
df2
```

Output:

	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[]
```
df3
```

Output:

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

Output:

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

Output:

	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[]
```
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[]
```
df1
```

Output:

	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[]
```
df2
```

Output:

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

In[]
```
df
```

Output:

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

Output:

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

Output:

	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[] `df1`

Output:

	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[] `df2`

Output:

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

Output:

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

Output:

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

Output:

	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[]
```
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[]
```
df1
```

Output:

	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[]
```
df2
```

Output:

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

Output:

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

Output:

	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[]
```
df1
```

Output:

	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[]
```
df2
```

Output:

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

Output:

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

Output:

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

Output:

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

Output:

	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[]
```
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')
```

Output:

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

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

Output:

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

Output:

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

In[]
```
df = pd.DataFrame({'Alpha': ['A', 'B', 'A', 'A', 'C'],
                   'Value': [1,   1,   2,   1,   2],
                   'Value2': [3,   4,   2,   1,   2]})
df
```

Output:

	Alpha	Value	Value2
0	A	1	3
1	B	1	4
2	A	2	2
3	A	1	1
4	C	2	2

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

Output:

		Value2
Alpha	**Value**	
A	**1**	2
	2	1
B	**1**	1
C	**2**	1

In[]
```
df.groupby('Alpha').max()
```

Output:

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

In[] `df.groupby('Alpha').min()`

Output:

	Value	Value2
Alpha		
A	1	1
B	1	4
C	2	2

In[] `df.groupby('Alpha').count()`

Output:

	Value	Value2
Alpha		
A	3	3
B	1	1
C	1	1

Importing Exporting CSV, EXCEL

In[]
```
import numpy as np
import pandas as pd

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

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	-2.343989	-0.993894	-0.280460
4	-2.052502	-0.045687	-1.964213	-0.615022	0.674743
5	-1.115725	-1.325247	-0.325938	0.435674	-0.737618
6	-0.635687	0.018955	-0.440329	1.020786	-1.096965
7	0.586991	-1.234016	-0.230980	0.765827	-1.439112
8	0.402842	-1.075903	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

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

In[]
```
df = pd.read_csv('random_data.csv')
df
```

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	-2.343989	-0.993894	-0.280460
4	-2.052502	-0.045687	-1.964213	-0.615022	0.674743
5	-1.115725	-1.325247	-0.325938	0.435674	-0.737618
6	-0.635687	0.018955	-0.440329	1.020786	-1.096965
7	0.586991	-1.234016	-0.230980	0.765827	-1.439112
8	0.402842	-1.075903	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

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

In[]
```
pd.read_excel('random_data.xlsx', 'first_sheet')
```

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	-2.343989	-0.993894	-0.280460
4	-2.052502	-0.045687	-1.964213	-0.615022	0.674743
5	-1.115725	-1.325247	-0.325938	0.435674	-0.737618
6	-0.635687	0.018955	-0.440329	1.020786	-1.096965
7	0.586991	-1.234016	-0.230980	0.765827	-1.439112
8	0.402842	-1.075903	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

In[]
```
pd.read_excel('random_data.xlsx', 'first_sheet', parse_cols=2)
```

Output:

	a	b
0	-0.185424	0.556467
1	-0.092305	0.519260
2	-0.643201	-1.094444
3	0.101593	0.211654
4	-2.052502	-0.045687
5	-1.115725	-1.325247
6	-0.635687	0.018955
7	0.586991	-1.234016
8	0.402842	-1.075903
9	-0.338593	1.632160

In[]
```
import pandas as pd
pd.read_excel('random_data.xlsx', 'first_sheet', parse_cols=[0, 2,
3])
```

Output:

	b	c
0	0.556467	-0.018397
1	0.519260	-0.810838
2	-1.094444	0.770276
3	0.211654	-2.343989
4	-0.045687	-1.964213
5	-1.325247	-0.325938
6	0.018955	-0.440329
7	-1.234016	-0.230980
8	-1.075903	-0.191814
9	1.632160	0.936806

In[]
```
pd.read_excel('random_data.xlsx', 'first_sheet', converters={'b': b
ool})
```

Output:

	a	b	c	d	e
0	-0.185424	True	-0.018397	1.271560	-0.712240
1	-0.092305	True	-0.810838	0.395051	0.478624
2	-0.643201	True	0.770276	1.685257	1.018772
3	0.101593	True	-2.343989	-0.993894	-0.280460
4	-2.052502	True	-1.964213	-0.615022	0.674743
5	-1.115725	True	-0.325938	0.435674	-0.737618
6	-0.635687	True	-0.440329	1.020786	-1.096965
7	0.586991	True	-0.230980	0.765827	-1.439112
8	0.402842	True	-0.191814	0.295246	0.497974
9	-0.338593	True	0.936806	-1.414648	-1.417063

In[]
```
import pandas as pd
cfun = lambda x: x if x > 0 else 0
pd.read_excel('random_data.xlsx', 'first_sheet', converters={'b': c
fun})
```

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	0.000000	0.770276	1.685257	1.018772
3	0.101593	0.211654	-2.343989	-0.993894	-0.280460
4	-2.052502	0.000000	-1.964213	-0.615022	0.674743
5	-1.115725	0.000000	-0.325938	0.435674	-0.737618
6	-0.635687	0.018955	-0.440329	1.020786	-1.096965
7	0.586991	0.000000	-0.230980	0.765827	-1.439112
8	0.402842	0.000000	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

Writing data to sql databses(MySQL):

In[]
```
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:Pyt
hon@7@localhost/sampledb')
con = engine.connect()
df.to_sql('table_1', con)
con.close()
```

Reading data from sql databses(MySQL):

Reading SQL table:

In[]
```
import sqlalchemy
import pandas as pd
engine = sqlalchemy.create_engine('mysql+mysqlconnector://naren:Pyt
hon@7@localhost/sampledb')
con = engine.connect()
df = pd.read_sql_table('table_1', con)
con.close()
df
```

Output:

	index	a	b	c	d	e
0	0	0.610789	0.518444	-0.058651	0.114676	0.269575
1	1	-1.367908	0.815131	-1.468489	-0.865035	0.524990
2	2	1.835877	-0.133482	1.241036	0.873051	-0.345534
3	3	-0.028590	-1.748423	-0.871569	-0.767376	1.146337
4	4	-0.349349	-0.231934	-0.151872	1.325395	-0.397325
5	5	0.484683	0.582593	0.751083	-1.544639	-0.693485
6	6	0.368602	-0.062292	0.944895	1.147730	0.136469
7	7	-0.044510	-0.699011	-1.436375	-0.838213	-1.065825
8	8	-0.895608	-0.413882	0.483729	0.603788	1.195901
9	9	1.168927	0.402929	-0.618149	-0.756853	1.703624

Running SQL query:

In[]
```
import sqlalchemy
import pandas as pd
engine = sqlalchemy.create_engine('mysql+mysqlconnector://naren:Pyt
hon@7@localhost/sampledb')
con = engine.connect()
df = pd.read_sql_query('select a, c, e from table_1 limit 5', con)
con.close()
df
```

Output:

	a	c	e
0	0.610789	-0.058651	0.269575
1	-1.367908	-1.468489	0.524990
2	1.835877	1.241036	-0.345534
3	-0.028590	-0.871569	1.146337
4	-0.349349	-0.151872	-0.397325

In[]
```
import sqlalchemy
import pandas as pd
engine = sqlalchemy.create_engine('mysql+mysqlconnector://naren:Pyt
hon@7@localhost/sampledb')
con = engine.connect()
df = pd.read_sql('select a, c, e from table_1 limit 5', con)
con.close()
df
```

Output:

	a	c	e
0	0.610789	-0.058651	0.269575
1	-1.367908	-1.468489	0.524990
2	1.835877	1.241036	-0.345534
3	-0.028590	-0.871569	1.146337
4	-0.349349	-0.151872	-0.397325

Handling missing data

In[]
```
import pandas as pd
df = pd.read_csv('random_data.csv')
df.loc[3:7, 'c'] = np.nan
```

In[]
```
df
```

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	NaN	-0.993894	-0.280460
4	-2.052502	-0.045687	NaN	-0.615022	0.674743
5	-1.115725	-1.325247	NaN	0.435674	-0.737618
6	-0.635687	0.018955	NaN	1.020786	-1.096965
7	0.586991	-1.234016	NaN	0.765827	-1.439112
8	0.402842	-1.075903	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

In[]
```
df.fillna(method='ffill')
```

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	0.770276	-0.993894	-0.280460
4	-2.052502	-0.045687	0.770276	-0.615022	0.674743
5	-1.115725	-1.325247	0.770276	0.435674	-0.737618
6	-0.635687	0.018955	0.770276	1.020786	-1.096965
7	0.586991	-1.234016	0.770276	0.765827	-1.439112
8	0.402842	-1.075903	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

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

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	0.770276	-0.993894	-0.280460
4	-2.052502	-0.045687	0.770276	-0.615022	0.674743
5	-1.115725	-1.325247	NaN	0.435674	-0.737618
6	-0.635687	0.018955	NaN	1.020786	-1.096965
7	0.586991	-1.234016	NaN	0.765827	-1.439112
8	0.402842	-1.075903	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

In[] `df.fillna(method='bfill')`

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	-0.191814	-0.993894	-0.280460
4	-2.052502	-0.045687	-0.191814	-0.615022	0.674743
5	-1.115725	-1.325247	-0.191814	0.435674	-0.737618
6	-0.635687	0.018955	-0.191814	1.020786	-1.096965
7	0.586991	-1.234016	-0.191814	0.765827	-1.439112
8	0.402842	-1.075903	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

In[]
```
df.fillna(df.mean())
```

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	0.137207	-0.993894	-0.280460
4	-2.052502	-0.045687	0.137207	-0.615022	0.674743
5	-1.115725	-1.325247	0.137207	0.435674	-0.737618
6	-0.635687	0.018955	0.137207	1.020786	-1.096965
7	0.586991	-1.234016	0.137207	0.765827	-1.439112
8	0.402842	-1.075903	-0.191814	0.295246	0.497974
9	-0.338593	1.632160	0.936806	-1.414648	-1.417063

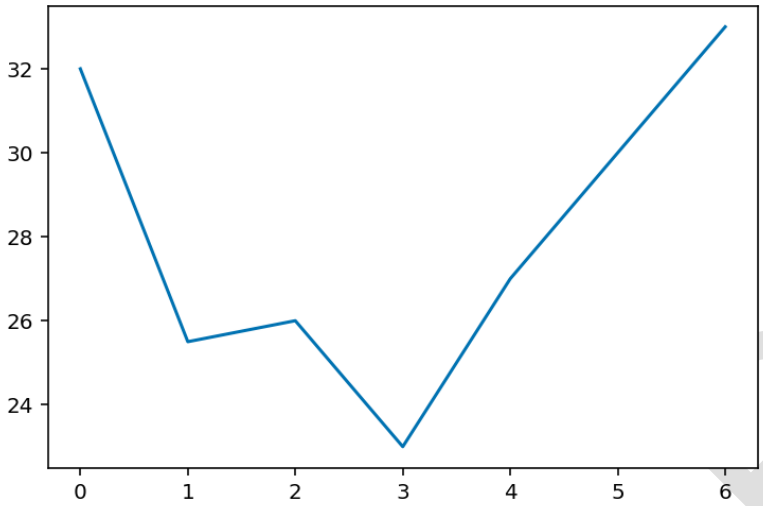
Matplotlib

In[]
```
%matplotlib inline
import matplotlib.pyplot as  plt


from IPython.display import  set_matplotlib_formats
set_matplotlib_formats('retina')

print plt.style.available
```

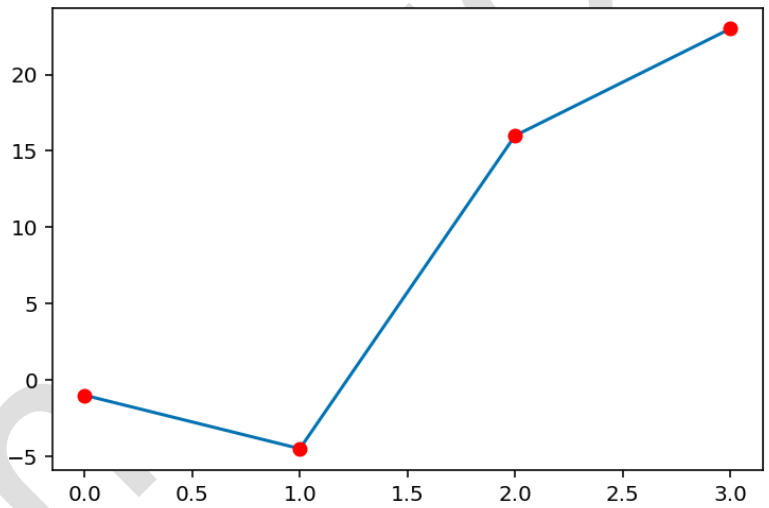
```
[u'seaborn-darkgrid', u'seaborn-notebook', u'classic', u'seaborn-t
icks', u'grayscale', u'bmh', u'seaborn-talk', u'dark_background',
u'ggplot', u'fivethirtyeight', u'_classic_test', u'seaborn-colorbl
ind', u'seaborn-deep', u'seaborn-whitegrid', u'seaborn-bright', u'
seaborn-poster', u'seaborn-muted', u'seaborn-paper', u'seaborn-whi
te', u'seaborn-pastel', u'seaborn-dark', u'seaborn', u'seaborn-dar
k-palette']
```

In[]
```
plt.style.use('seaborn-colorblind')
```

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

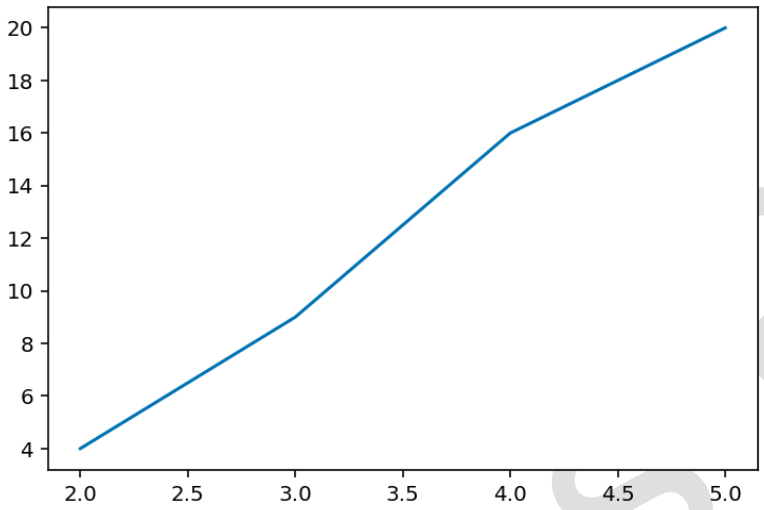


In[]
```
import matplotlib.pyplot as plt
plt.plot([-1, -4.5, 16, 23], "-")
plt.plot([-1, -4.5, 16, 23], "or")
plt.show()
```

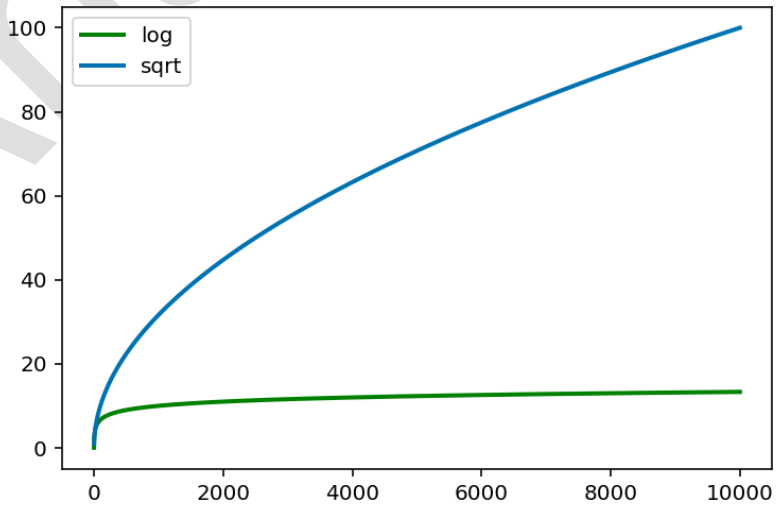


In[]
```
plt.plot?
```

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

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



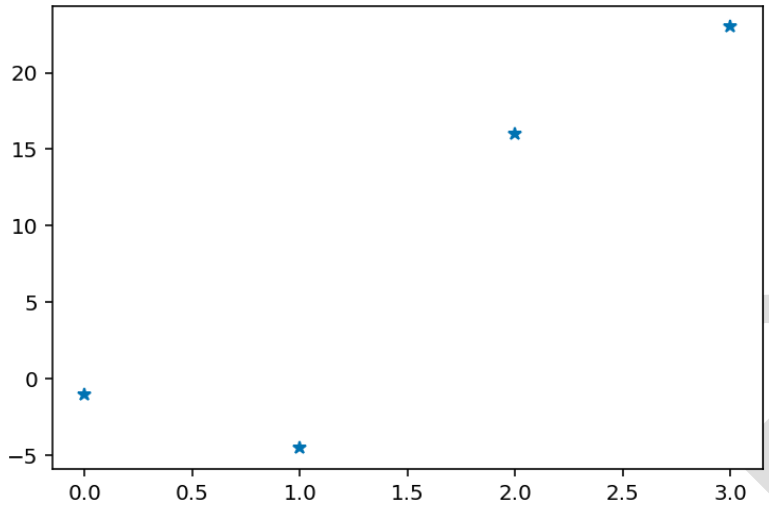
In[]
```
import matplotlib.pyplot as plt
import math

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()
```



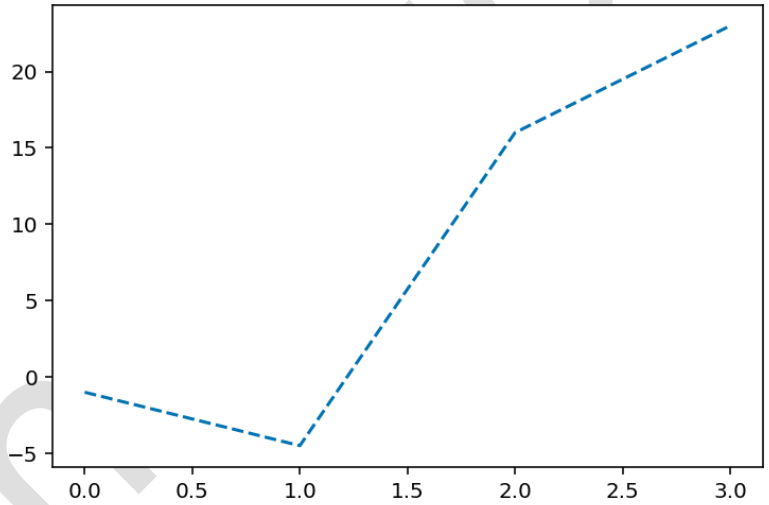
The format parameter of pyplot.plot

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



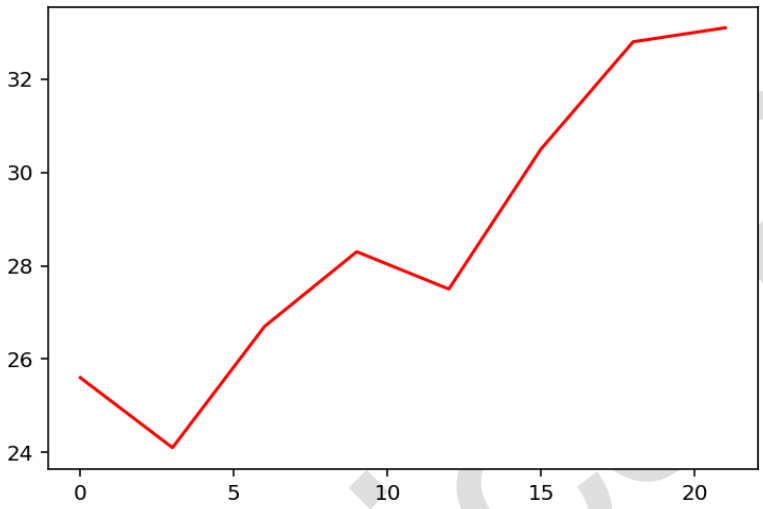
In[]
```
import matplotlib.pyplot as plt
plt.plot([-1, -4.5, 16, 23], "--")
plt.show()
```



In[]

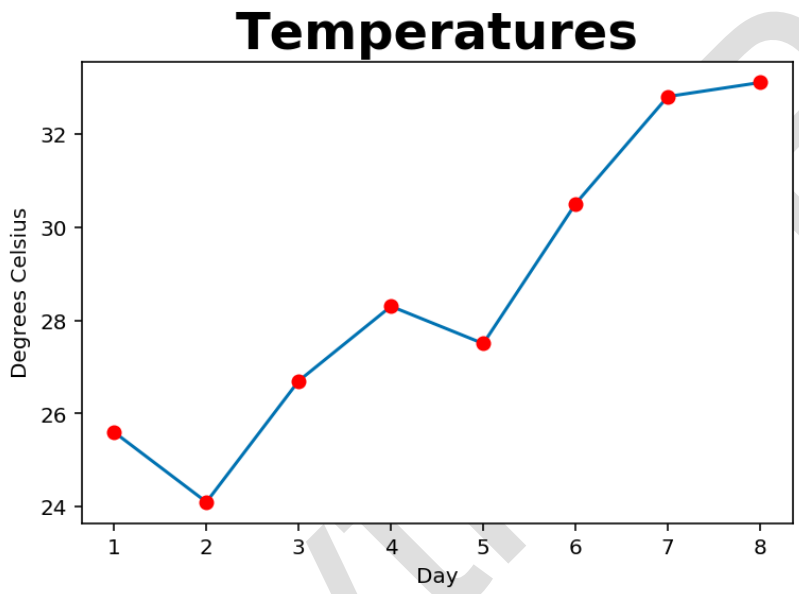
```
days = list(range(0, 22, 3))
print(days)

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]
```



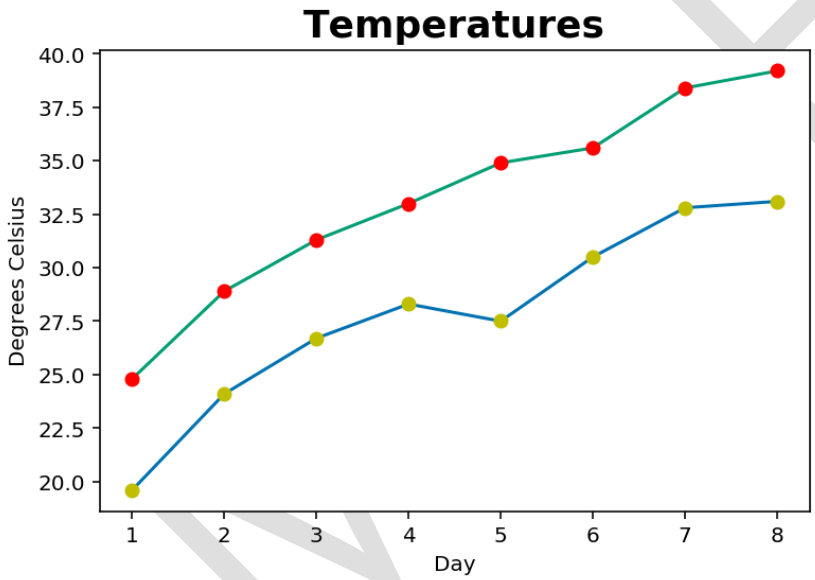
In[]

```
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=24, loc='center', fontweight='bo
ld')
plt.show()
```



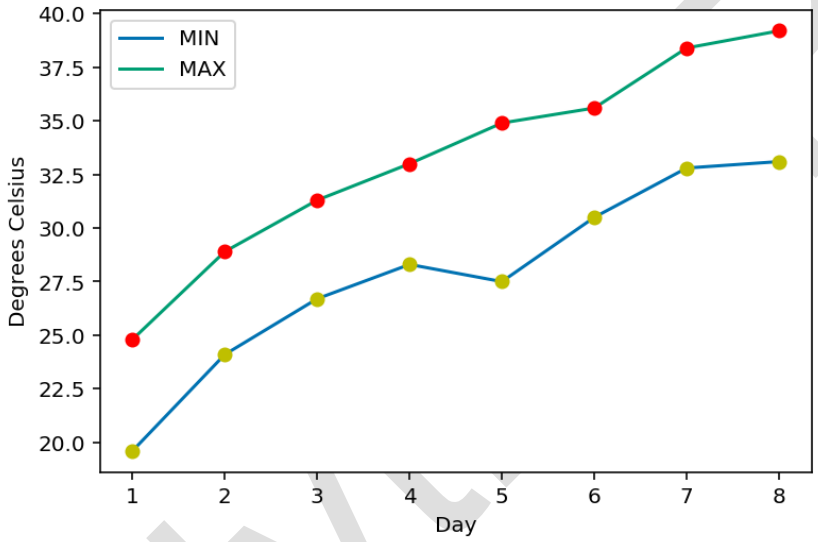
In[]

```
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='bo
ld')
plt.plot(days,  celsius_min,
         days, celsius_min, "oy",
         days, celsius_max,
         days, celsius_max, "or")
plt.show()
```



In[]

```
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()
```

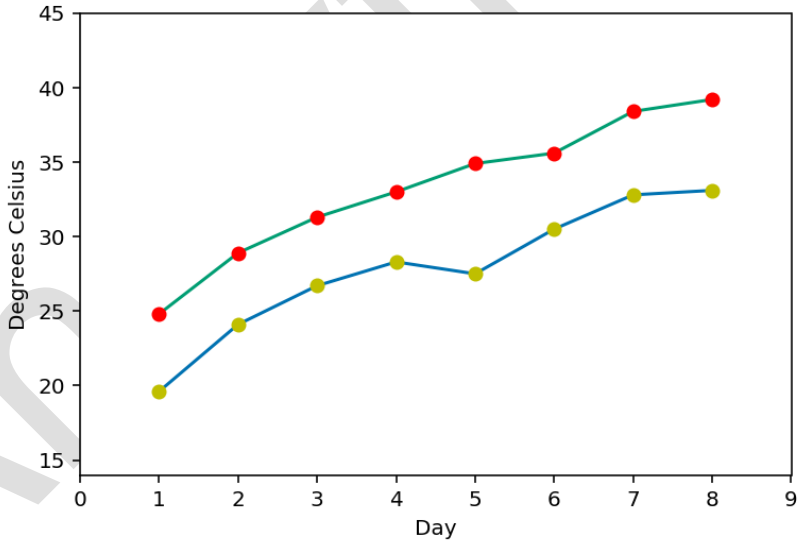


Legend positions:

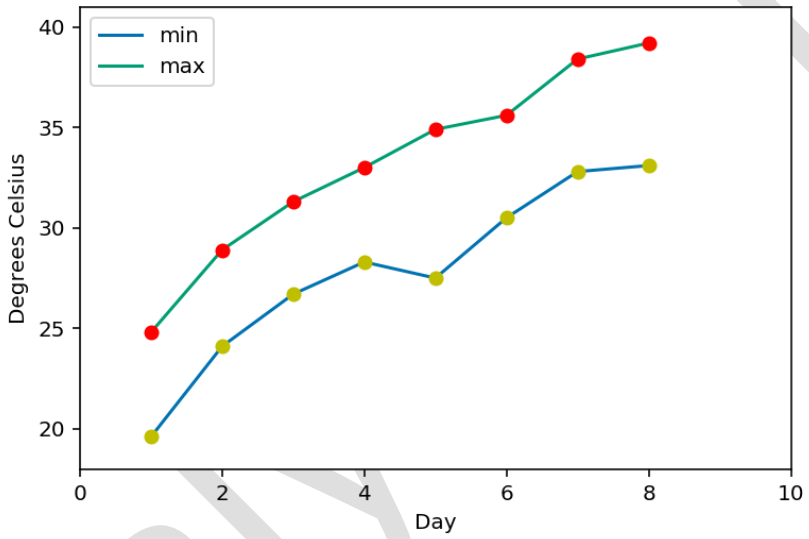
Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

In[]
```
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)

plt.axis([xmin, xmax, ymin, ymax])
plt.show()
```

```
The current limits for the axes are:
(0.6499999999999991, 8.3499999999999996, 18.620000000000001, 40.1
8)
We set the axes to the following values:
(0, 9, 14, 45)
```



In[]
```
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.plot(days, celsius_min, label='min')
plt.plot(days, celsius_min, "oy")
plt.xlabel('Day')
plt.ylabel('Degrees Celsius')
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()
```



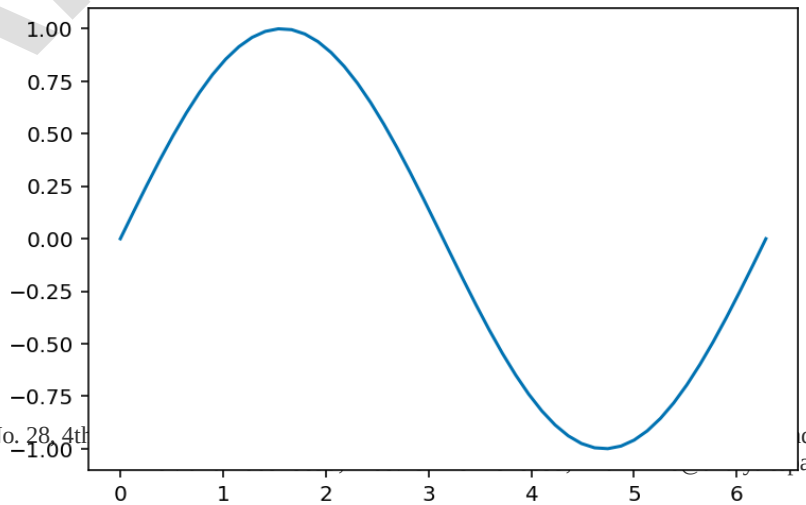
"linspace" to Define X Values

In[]
```
import numpy as np
import matplotlib.pyplot as plt
```

In[]
```
np.linspace(0, 15, 100)
```

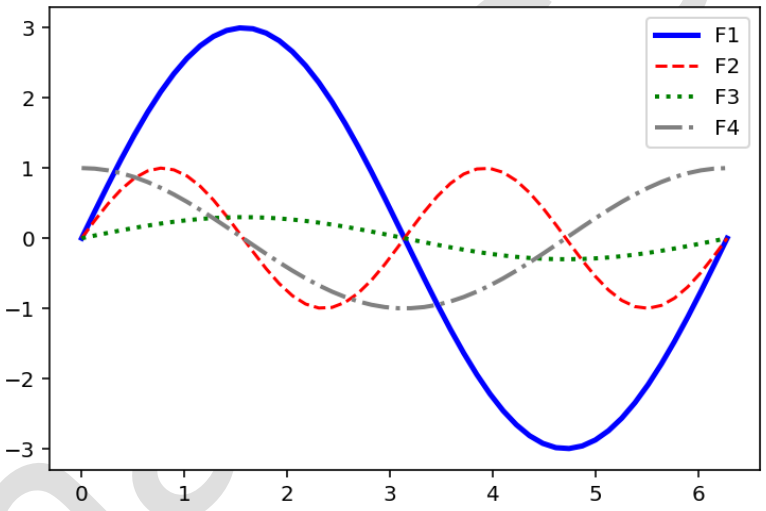
Output:
```
array([  0.        ,   0.15151515,   0.3030303 ,   0.45454545,
         0.60606061,   0.75757576,   0.90909091,   1.06060606,
         1.21212121,   1.36363636,   1.51515152,   1.66666667,
         1.81818182,   1.96969697,   2.12121212,   2.27272727,
         2.42424242,   2.57575758,   2.72727273,   2.87878788,
         3.03030303,   3.18181818,   3.33333333,   3.48484848,
         3.63636364,   3.78787879,   3.93939394,   4.09090909,
         4.24242424,   4.39393939,   4.54545455,   4.6969697 ,
         4.84848485,   5.        ,   5.15151515,   5.3030303 ,
         5.45454545,   5.60606061,   5.75757576,   5.90909091,
         6.06060606,   6.21212121,   6.36363636,   6.51515152,
         6.66666667,   6.81818182,   6.96969697,   7.12121212,
         7.27272727,   7.42424242,   7.57575758,   7.72727273,
         7.87878788,   8.03030303,   8.18181818,   8.33333333,
         8.48484848,   8.63636364,   8.78787879,   8.93939394,
         9.09090909,   9.24242424,   9.39393939,   9.54545455,
         9.6969697 ,   9.84848485,  10.        ,  10.15151515,
        10.3030303 ,  10.45454545,  10.60606061,  10.75757576,
        10.90909091,  11.06060606,  11.21212121,  11.36363636,
        11.51515152,  11.66666667,  11.81818182,  11.96969697,
        12.12121212,  12.27272727,  12.42424242,  12.57575758,
        12.72727273,  12.87878788,  13.03030303,  13.18181818,
        13.33333333,  13.48484848,  13.63636364,  13.78787879,
        13.93939394,  14.09090909,  14.24242424,  14.39393939,
        14.54545455,  14.6969697 ,  14.84848485,  15.        ])
```

In[]
```
%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()
```



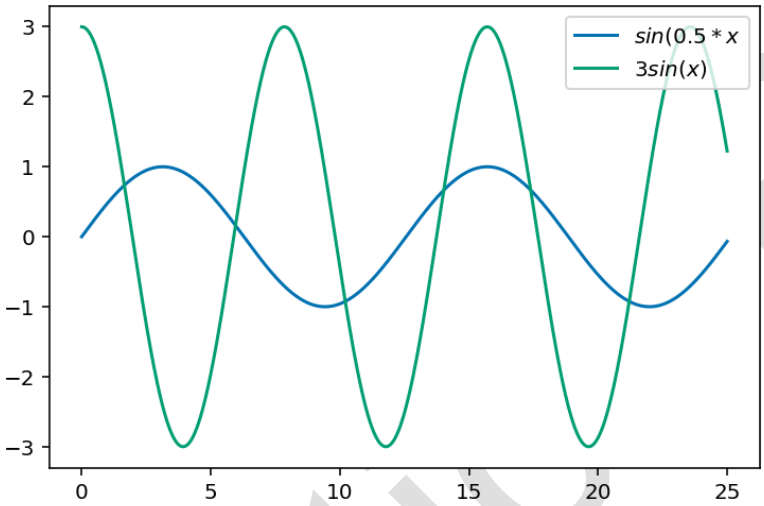
Changing the Line Style

In[]
```
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='F
3')
plt.plot(X, F4, color="grey", linewidth=2, linestyle="-.", label='F
4')
plt.legend(loc='best')
plt.show()
```

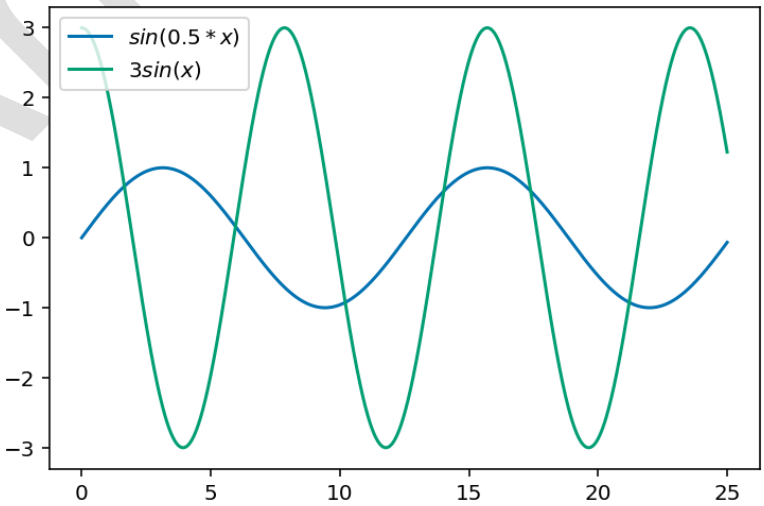


Legends

In[]
```
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[]
```
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()
```

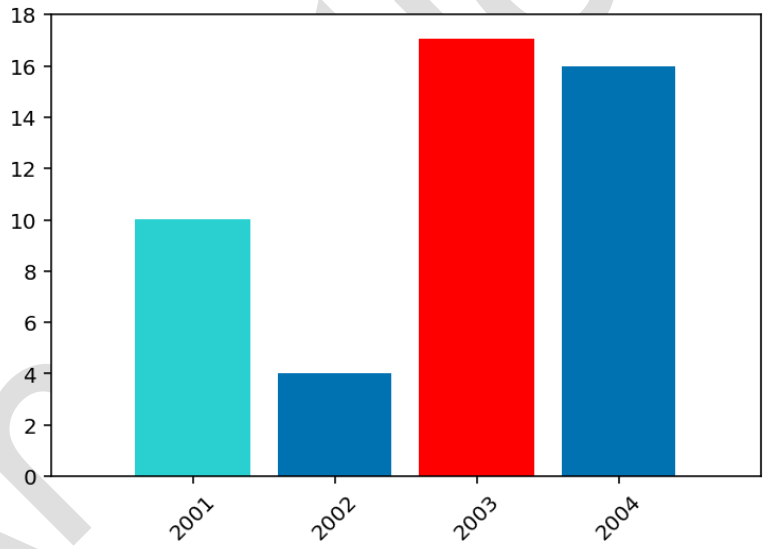


Bar Charts and Histograms

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[]

```
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')
bar_width = 2.0

plt.xticks(x, ('2001', '2002', '2003', '2004'), rotation=45)
plt.axis([0, 5, 0, 18])
plt.show()
```

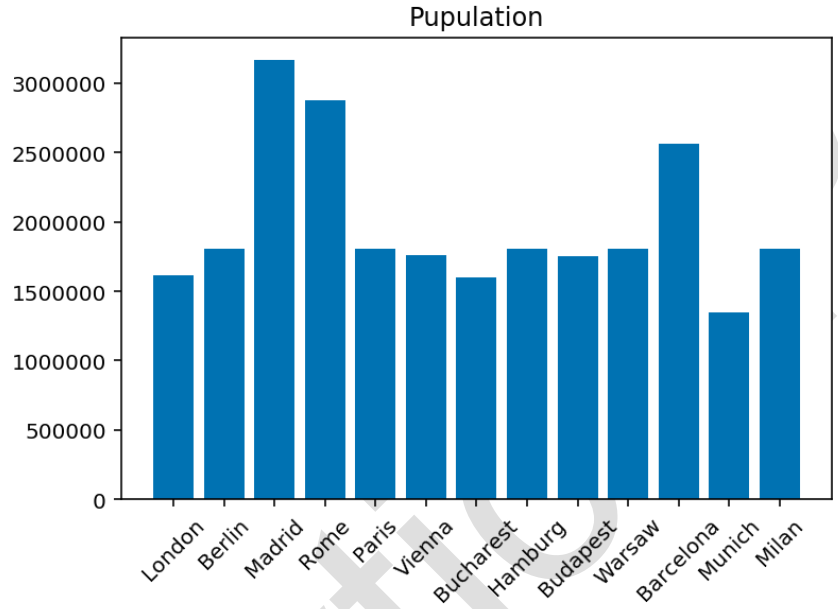


In[]
```
import pandas as pd
cities = {"cityname": ["London", "Berlin", "Madrid",  "Rome",
                    "Paris", "Vienna", "Bucharest", "Hamburg",
                    "Budapest", "Warsaw", "Barcelona",
                    "Munich", "Milan"],

          "population": [1615246,1803425, 3165235, 2874038,
                         1805681,  1760433, 1602386, 1805681,
                         1754000, 1805681, 2562166, 1350680,  180342
5]


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

Output:

	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

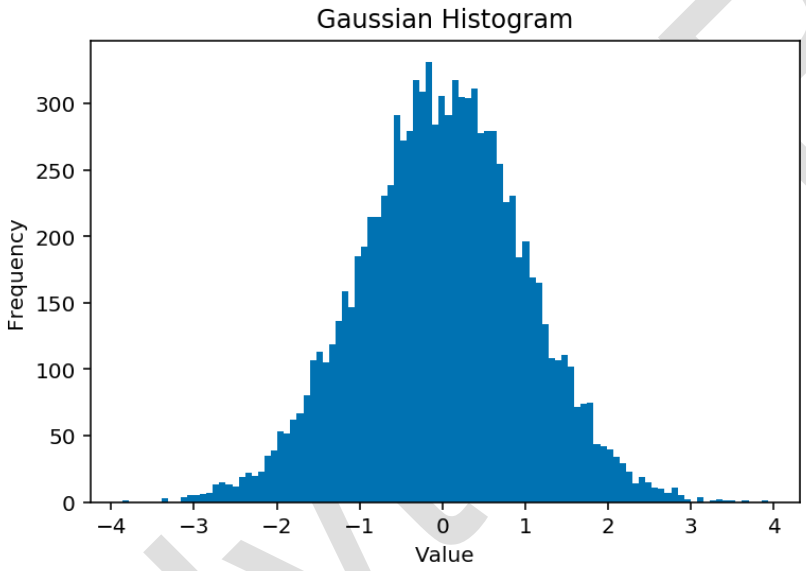
In[]
```
import pandas as pd

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


plt.title('Pupulation')
plt.show()
```



Histograms

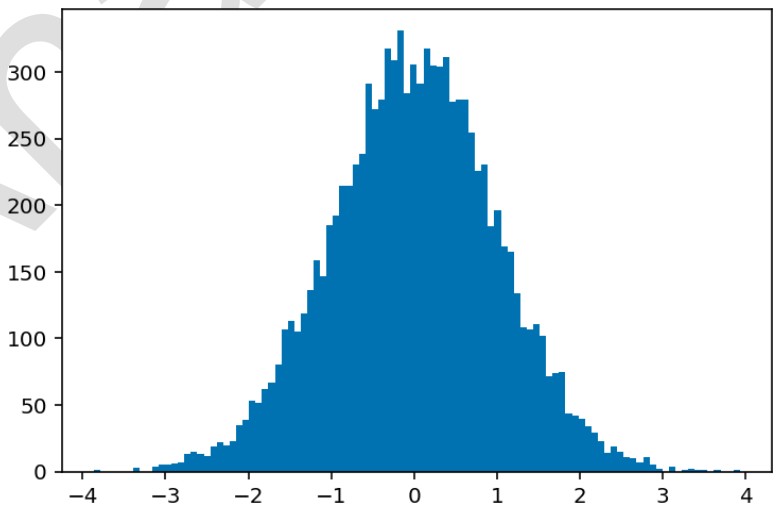
In[]

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
gaussian_numbers = np.random.normal(size=10000)
plt.hist(gaussian_numbers, bins=100)

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

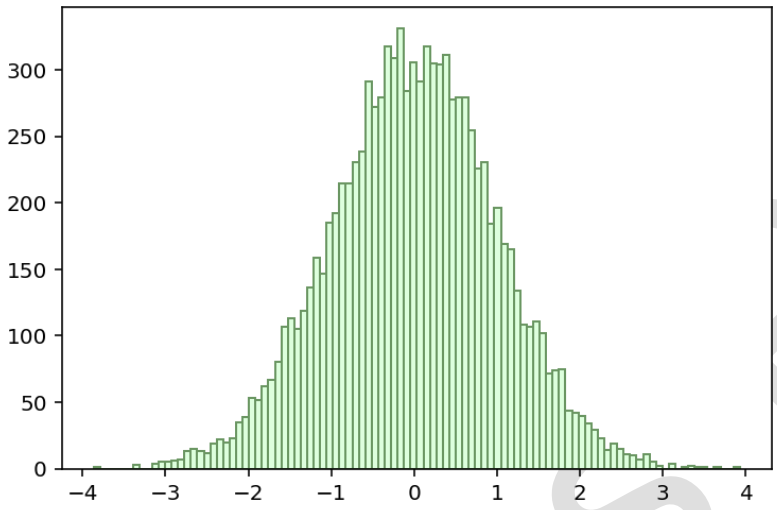


In[]

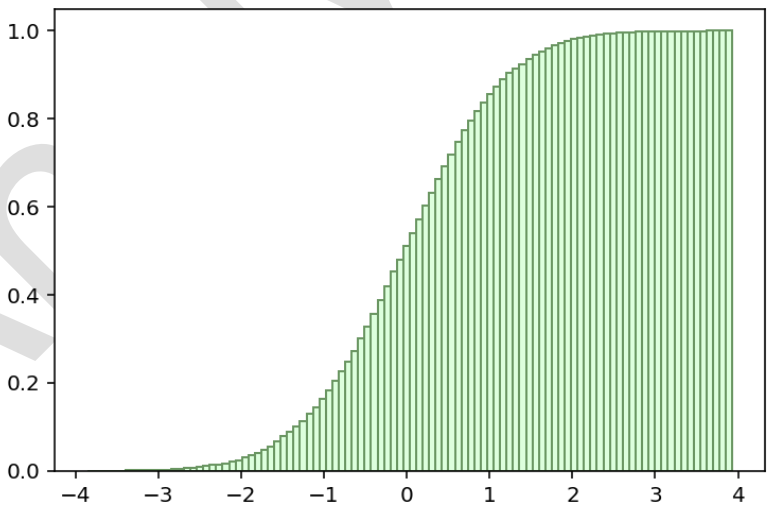
```
plt.hist(gaussian_numbers, bins=100)
plt.show()
```



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



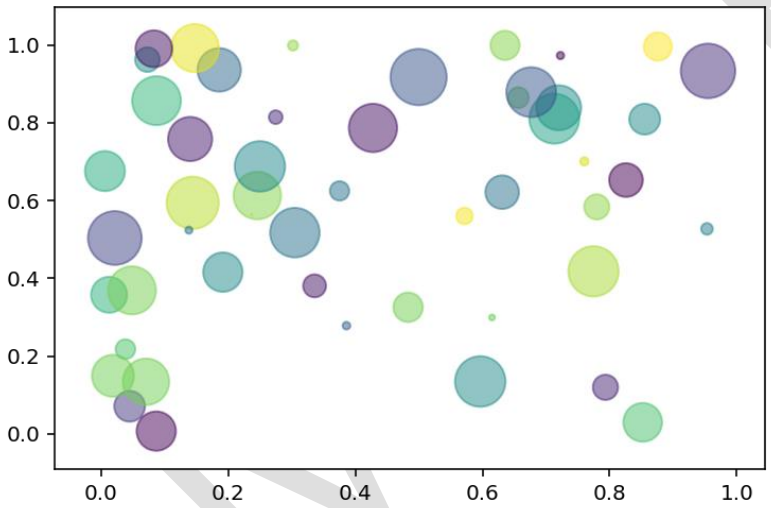
In[]
```
plt.hist(gaussian_numbers,
         bins=100,
         normed=True,
         edgecolor="#6A9662",
         color="#DDFFDD",
         cumulative=True)
plt.show()
```



Scatter plot

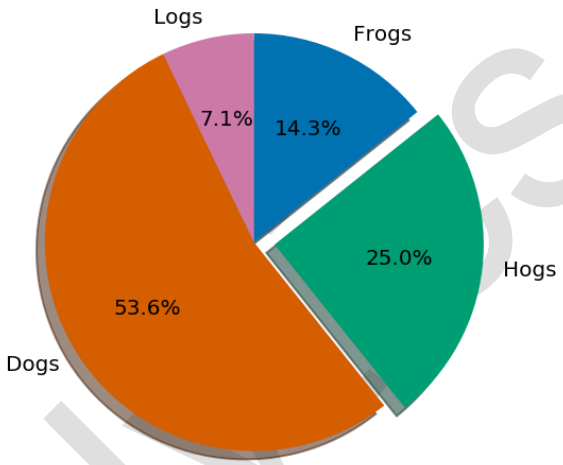
In[]

```
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


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



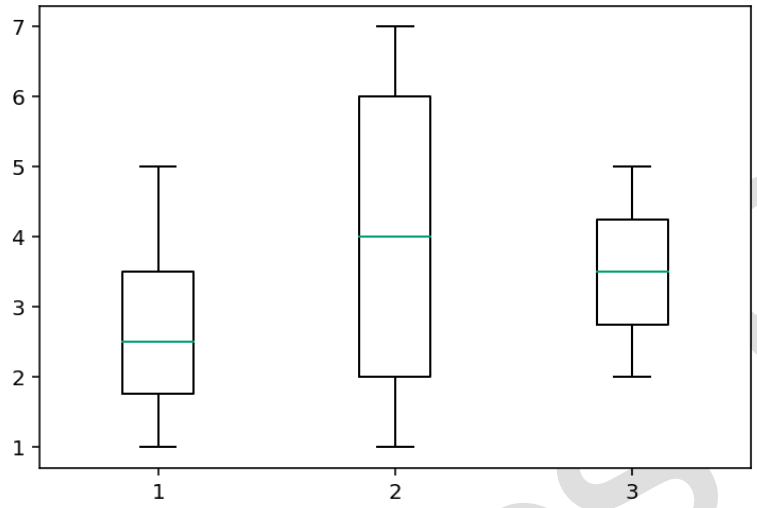
pie chart

In[]
```
import matplotlib.pyplot as  plt



labels = 'Frogs', 'Hogs', 'Dogs',  'Logs'
sizes = [200, 350, 750, 100]
explode = (0, 0.1, 0, 0)


fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90, counterclock=False)
ax1.axis('equal')


plt.show()
```

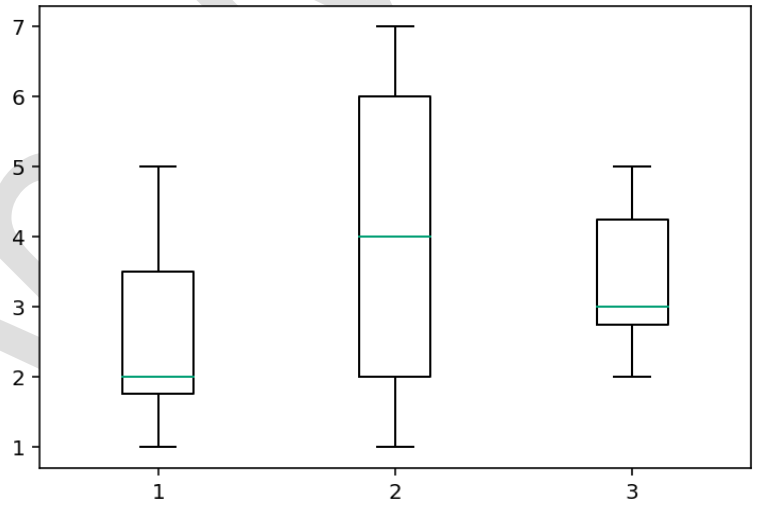


Box plot

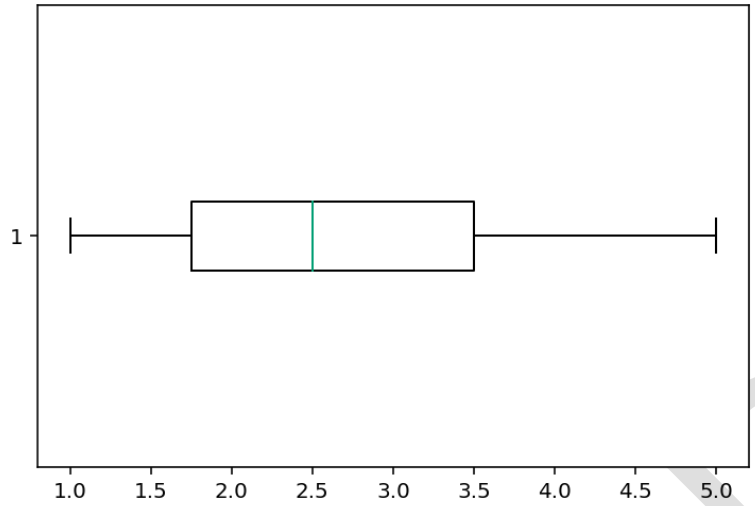
In[]
```
import matplotlib.pyplot as plt
import numpy as np

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



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

Output:

	a	b	c	d	e
0	-0.185424	0.556467	-0.018397	1.271560	-0.712240
1	-0.092305	0.519260	-0.810838	0.395051	0.478624
2	-0.643201	-1.094444	0.770276	1.685257	1.018772
3	0.101593	0.211654	-2.343989	-0.993894	-0.280460
4	-2.052502	-0.045687	-1.964213	-0.615022	0.674743
5	-1.115725	-1.325247	-0.325938	0.435674	-0.737618
6	-0.635687	0.018955			