External libraries

BB1000 Programming in Python KTH

Essential Python libraries

- NumPy: 'Numerical python' - package for scientific computing in Python. Ment primarily to sort, reshape, and index array types...
- pandas: data structures and functions to work with structured data. The main object in pandas is the `DataFrame`, which is a two-dimentional tabular.
- matplotlib: producing plots; the basic functions handled in this course are all in the matplotlib.pyplot module.

```
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

NumPy

One of the key fatures of NumPy is its N-dimensional array object: `ndarray`.
They enable to perform mathematical operations on blocks of data.

```
>>> import numpy as np
>>> data1 = [6, 7.5, 8, 0, 1]
>>> arr1 = np.array(data1)

>>> print(arr1)
[ 6.   7.5  8.   0.   1. ]

>>> data2 = [[1, 2, 3, 4], [ 5, 6, 7, 8]]
>>> arr2 = np.array(data2)

>>> print(arr2)
[[1 2 3 4]
 [5 6 7 8]]

>>> print(arr2.ndim)
2
>>> print(arr2.shape)
(2, 4)
```

NumPy - Default arrays

```
>>> print(np.zeros(10))
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]

>>> print(np.zeros((3,6)))
[[ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]]

>>> print(np.arange(15))
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

NumPy - Default arrays

```
>>> print(np.empty((2,3,2)))
[[[  2.35558336e-310   2.02731498e-316]
  [  2.35558575e-310   2.35558575e-310]
  [  2.35558575e-310   2.35558575e-310]]

 [[  2.35558575e-310   2.35558575e-310]
  [  2.35558575e-310   2.35558575e-310]
  [  2.35558575e-310   2.42092166e-322]]]

>>> print(np.eye(3))
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

`empty` creates an array without initializing its values to any particular value. It does the ideal recipe to return garbage…

NumPy - Operations between arrays and scalars

```
>>> arr = np.array([[1., 2., 3.], [4., 5., 6.]])
>>> print(arr*arr)
[[  1.   4.   9.]
 [ 16.  25.  36.]]

>>> print(arr-arr)
[[ 0.  0.  0.]
 [ 0.  0.  0.]]

>>> print(1/arr)
[[ 1.          0.5         0.33333333]
 [ 0.25        0.2         0.16666667]]

>>> print(arr**0.5)
[[ 1.          1.41421356  1.73205081]
 [ 2.          2.23606798  2.44948974]]
```

NumPy - Basic indexing and slicing

```
>>> arr= np.arange(10)
>>> print(arr[5])
5
>>> print(arr[5:8])
[5 6 7]
>>> arr[5:8] = 12
>>> print(arr)
[ 0  1  2  3  4 12 12 12  8  9]

>>> arr_slice = arr[5:8]
>>> arr_slice[:] = 64
>>> print(arr)
[ 0  1  2  3  4 64 64 64  8  9]
```

NumPy - Basic indexing and slicing

```
>>> arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> print(arr2d[2])
[7 8 9]
>>> print(arr2d[0][2])
3
```

```
>>> print(arr2d[:2, 1:])
[[2 3]
 [5 6]]
```

Entries up till (but not including) the second row are kept, as well as the column starting from (and including) the first one.

NumPy - Boolean indexing

```
>>> names = np.array(['Asterix', 'Obelix', 'Idefix', 'Asterix', 'Idefix', 'Obelix', 'Obelix
>>> names
array(['Asterix', 'Obelix', 'Idefix', 'Asterix', 'Idefix', 'Obelix',
       'Obelix'],
      dtype='|S7')
```

```
>>> data = np.random.randn(7,4)
>>> data
array([[ 0.02062421, -0.1369847 ,  0.90160195,  0.75181516],
       [-1.1268401 , -0.41237719, -0.21513891,  0.2190537 ],
       [-0.00535594,  0.15848914, -0.99522448,  0.93785222],
       [ 0.84553696, -1.7851311 ,  0.74135975,  0.36109035],
       [ 1.22254501, -0.68403217,  0.39343747,  1.59037781],
       [ 0.02684093, -0.62523998,  0.06727077, -1.3981326 ],
       [ 0.70864672, -1.46741426, -1.69648987, -0.47846134]])
>>> names == 'Asterix'
array([ True, False, False,  True, False, False, False], dtype=bool)
```

NumPy - Boolean indexing

Those rows in data indexed with 'True' can be selected:

```
>>> data[names == 'Asterix']
array([[ 0.02062421, -0.1369847 ,  0.90160195,  0.75181516],
       [ 0.84553696, -1.7851311 ,  0.74135975,  0.36109035]])
```

And also slicing is possible:

```
>>> data[names == 'Asterix', 2:]
array([[ 0.90160195,  0.75181516],
       [ 0.74135975,  0.36109035]])
```

For negation `!=` can be used as wel as `-`

```
>>> names != 'Asterix'
array([False,  True,  True, False,  True,  True,  True], dtype=bool)
>>> data[-(names == 'Asterix')]
array([[-1.1268401 , -0.41237719, -0.21513891,  0.2190537 ],
       [-0.00535594,  0.15848914, -0.99522448,  0.93785222],
       [ 1.22254501, -0.68403217,  0.39343747,  1.59037781],
       [ 0.02684093, -0.62523998,  0.06727077, -1.3981326 ],
       [ 0.70864672, -1.46741426, -1.69648987, -0.47846134]])
```

NumPy - Boolean indexing

To select two of the three names to combine multiple boolean conditions, use boolean arithmetic operators like `&` (and) and `|` (or):

```
>>> mask = (names == 'Asterix') | (names == 'Idefix')
>>> mask
array([ True, False,  True,  True,  True, False, False], dtype=bool)
```

In this way, it is possible to set data to whole rows:

```
>>> data[names != 'Asterix'] = 7
>>> data
array([[ 0.02062421, -0.1369847 ,  0.90160195,  0.75181516],
       [ 7.        ,  7.        ,  7.        ,  7.        ],
       [ 7.        ,  7.        ,  7.        ,  7.        ],
       [ 0.84553696, -1.7851311 ,  0.74135975,  0.36109035],
       [ 7.        ,  7.        ,  7.        ,  7.        ],
       [ 7.        ,  7.        ,  7.        ,  7.        ],
       [ 7.        ,  7.        ,  7.        ,  7.        ]])
```

NumPy - Transposing arrays

Arrays have the `transpose` method and also the special `T` attribute:

```
>>> arr = np.arange(15).reshape((3,5))
>>> arr
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> arr.T
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

The method is very interesting in linear algebra, as the inner matrix product 'X^TX' can be easily calculated:

```
>>> np.dot(arr.T, arr)
array([[125, 140, 155, 170, 185],
       [140, 158, 176, 194, 212],
       [155, 176, 197, 218, 239],
       [170, 194, 218, 242, 266],
       [185, 212, 239, 266, 293]])
```

NumPy - Unary and binary universal functions

A universal function ('ufunction') is a function that performs elementwise operations on data in ndarrays. A unary one only focusses upon one array, while binary ones require 2 arrays.

Examples of unary ufunctions are `sqrt`, `exp`, `abs`, `log`, `sign`, `floor` (largest integer less than or equal to the element), `ceil` (analogon for `floor` but then higher or equal to the element), `cos`,…

Examples of binary ufunctions are `add`, `subtract`, `multiply`, `divide`, `power`, `max`, `min`, `mod` (remainder of division), `greater`, `less`, `less_equal`, …

NumPy - Unary and binary universal functions

A few examples...

```
>>> arr = np.arange(10)
>>> np.sqrt(arr)
array([ 0.        ,  1.        ,  1.41421356,  1.73205081,  2.        ,
        2.23606798,  2.44948974,  2.64575131,  2.82842712,  3.        ])
```

```
>>> arr2 = np.random.randn(10)
>>> arr2
array([-1.32672421, -2.02196629, -1.87814963,  1.3586335 ,  0.66869694,
        1.64577817,  0.01575116,  0.09529667, -0.32427566,  0.73408638])
>>> arr3 = np.random.randn(10)
>>> arr3
array([-0.09059814, -0.05915682,  1.39919745, -0.96167955, -2.70897768,
       -1.44743637,  0.47766619, -0.18136026,  0.87246909, -0.43929249])
>>> np.maximum(arr2, arr3)
array([-0.09059814, -0.05915682,  1.39919745,  1.3586335 ,  0.66869694,
        1.64577817,  0.47766619,  0.09529667,  0.87246909,  0.73408638])
```

NumPy - Conditions and arrays

The `numpy.where(condition,firstargument,secondargument)` function reduces the expression `x if condition else y` for arrays: if the `condition` is true, then the `firstargument` is executed, else the `secondargument` is done.

```
>>> xarr = np.array([1.1, 1.2, 1.3 , 1.4, 1.5])
>>> yarr = np.array([2.1, 2.2, 2.3 , 2.4, 2.5])
>>> cond = np.array([True, False, True, True, False])
>>> result = np.where(cond, xarr, yarr)
>>> result
array([ 1.1,  2.2,  1.3,  1.4,  2.5])
```

For boolean arrays, `any` tests whether one or more values in an array is `True`, while `all` checks if every value is `True`.

```
>>> bools = np.array([False, False, True, False])
>>> np.any(bools)
True
>>> np.all(bools)
False
```

NumPy - Sorting and Unique

```
>>> arr= np.random.randn(8)
>>> arr
array([ 0.71176752,  0.24762018, -0.61990769,  0.77071301,  0.67810754,
        1.92071058,  1.01916251,  1.06109087])
>>> np.sort(arr)
array([-0.61990769,  0.24762018,  0.67810754,  0.71176752,  0.77071301,
        1.01916251,  1.06109087,  1.92071058])
```

```
>>> names = np.array(['Asterix', 'Kanalltix', 'Kaningentix', 'Asterix', 'Kaningentix', 'Ast
>>> np.unique(names)
array(['Asterix', 'Kanalltix', 'Kaningentix'],
      dtype='|S11')
```

NumPy - Storing array, saving and loading

`np.save` and `np.load` allow to save and load data on disk. It will be in raw binary format and have file extension `.npy`.

```
>>> arr = np.arange(10)
>>> np.save('some_array', arr)
>>> np.load('some_array.npy')
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Heroix finds a paper with the service numbers of the roman soldiers which attacked his village yesterday. How can he manipulate this using arrays?

```
125481,568937,428937,585667,889375,442568,558937,557934,554723,258649,34582
>>> arr = np.loadtxt('servicenumbers.txt', delimiter=',')
>>> arr
array([ 125481.,  568937.,  428937.,  585667.,  889375.,  442568.,
        558937.,  557934.,  554723.,  258649.,   34582.])
```

When Heroix wants to write the service numbers of his Celtic warriors on a file, he uses `np.savetxt`:

```
>>> arr2=np.random.randn(8)
>>> np.savetxt('CelticWarr.txt',arr2)
```

NumPy - Linear algebra

Attention has to be paid at `*` which is an element-wsie product instead of a matrix dot product. The function `dot` is used in numpy (see 'Transposing arrays').

To do calculations on matrices, `numpy.linalg` has a standard set of functions, like `diag` (return the diagonal elements of a square matrix), `trace`, `det` (matrix determinant), `eig` (eigenvalues and eigenvectors of a square matrix), `inv` (inverse),...

```
>>> X = np.random.randn(2,2)
>>> mat = X.T.dot(X)
>>> np.linalg.inv(mat)
array([[  1.70182387,   3.91458018],
       [  3.91458018,  10.22597796]])
```

NumPy - cumsum()

The `cumsum()` function gives out the cumulative sum of the numbers in the array.

```
>>> a = np.array([[1,2,3], [4,5,6]])
>>> np.cumsum(a)
array([ 1,  3,  6, 10, 15, 21])
>>> np.cumsum(a,axis=0)   #sum over columns
array([[1, 2, 3],
       [5, 7, 9]])
>>> np.cumsum(b,axis=1)   #sum over rows
array([[ 1,  3,  6],
       [ 4,  9, 15]])
```

Pandas - Series

A `Series` is a one-dimensional object containing an array of data and an associated array of data labels, called its index.

```
>>> obj = pd.Series([4, 7, -5, 3])
>>> obj
0    4
1    7
2   -5
3    3
dtype: int64
>>> obj.values
array([ 4,  7, -5,  3])
>>> obj.index
RangeIndex(start=0, stop=4, step=1)
>>> for j in obj.index : print j
0
1
2
3
```

Pandas - Series

It is also possible to define the index.

```
>>> obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
>>> obj2
d    4
b    7
a   -5
c    3
```

Single values can be selected and assigned.

```
>>> obj2['a']
-5
>>> obj2['d'] = 6
```

Numpy array operations are possible - the index value does not change.

```
>>> np.exp(obj2)
d     403.428793
b    1096.633158
a       0.006738
c      20.085537
dtype: float64
```

Pandas - DataFrame

A DataFrame represents a tabular, spreadsheet-like data structure which contains an ordered collection of columns, which can have a different value type (numeric, boolean,...). A DataFrame has both a row and a column index.

```
>>> data = {'Celt': ['Asterix', 'Asterix', 'Asterix', 'Obelix', 'Obelix'], 'age' :
[18, 19, 20, 19, 20 ], 'numberofromans': [1.5, 1.7, 3.6, 2.4, 2.9]}
>>> frame = pd.DataFrame(data)
>>> frame
      Celt  age  numberofromans
0  Asterix   18             1.5
1  Asterix   19             1.7
2  Asterix   20             3.6
3   Obelix   19             2.4
4   Obelix   20             2.9
```

Pandas - DataFrame

The sequence of columns can be specified and the index can be redefined.

```
>>> frame2 = pd.DataFrame(data, columns=['numberofromans','Celt'], index=['one',
'two', 'three', 'four', 'five'])
>>> frame2
       numberofromans     Celt
one               1.5  Asterix
two               1.7  Asterix
three             3.6  Asterix
four              2.4   Obelix
five              2.9   Obelix
```

In order to write out the DataFrame, `to_csv` is used.

```
>>> frame2.to_csv('out_frame2.csv')
```

Pandas - DataFrame

Columns can be retrieved in two different ways.

```
>>> frame2['Celt']
one      Asterix
two      Asterix
three    Asterix
four      Obelix
five      Obelix
Name: Celt, dtype: object
>>> frame2.numberofromans
one      1.5
two      1.7
three    3.6
four     2.4
five     2.9
Name: numberofromans, dtype: float64
```

Rows can be retrieved by using `ix` and the index.

```
>>> frame2.ix['three']
numberofromans        3.6
Celt              Asterix
Name: three, dtype: object
```

Pandas - DataFrame

Assigning a column that doesn't exist will create a new column.

```
>>> frame2['thick'] = frame2.Celt == 'Obelix'
>>> frame2
              numberofromans     Celt  thick
       one               1.5  Asterix  False
       two               1.7  Asterix  False
       three             3.6  Asterix  False
       four              2.4   Obelix   True
       five              2.9   Obelix   True
```

`del` removes columns

```
>>> del frame2['thick']
>>> frame2
       numberofromans     Celt
       one               1.5  Asterix
       two               1.7  Asterix
       three             3.6  Asterix
       four              2.4   Obelix
       five              2.9   Obelix
```

Pandas - Dataframe

Use can be made of nested structures:

```
>>> punch = { 'Asterix': {18: 1.5, 19: 1.7, 20: 3.6}, 'Obelix': { 19: 2.4,
20: 2.9}}
>>> frame = pd.DataFrame(punch)
>>> frame
        Asterix  Obelix
    18      1.5     NaN
    19      1.7     2.4
    20      3.6     2.9
```

... and it can be transposed

```
>>> frame.T
    18   19   20
    Asterix  1.5  1.7  3.6
    Obelix   NaN  2.4  2.9
```

Pandas - Data alignment

```
>>> A = pd.Series([7.3, -2.5, 3.4, 1.5], index = ['a', 'c', 'd', 'e'])
>>> B = pd.Series([-2.1, 3.6, -1.5, 4, 3.1], index = ['a', 'c', 'e', 'f', 'g'])
>>> A+B
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

Remark that d is missing in B, while f and g are absent in A.

Pandas - Data alignment

```
>>> df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),
index=['Asterix', 'Obelix', 'Kanalltix'])
>>> df2 = pd.DataFrame(np.arange(12.).reshape((4,3)), columns=list('bde'),
index=['Miraculix', 'Asterix', 'Obelix', 'Kaningentix'])
>>> df1
            b    c    d
Asterix   0.0  1.0  2.0
Obelix    3.0  4.0  5.0
Kanalltix  6.0  7.0  8.0
>>> df2
               b     d     e
Miraculix    0.0   1.0   2.0
Asterix      3.0   4.0   5.0
Obelix       6.0   7.0   8.0
Kaningentix  9.0  10.0  11.0
>>> df1+df2
               b   c     d   e
Asterix      3.0 NaN   6.0 NaN
Kanaltix     NaN NaN   NaN NaN
Kaningentix  NaN NaN   NaN NaN
Miraculix    NaN NaN   NaN NaN
Obelix       9.0 NaN  12.0 NaN
```

Pandas - Sorting

```
>>> obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])
>>> obj.sort_index()
a    1
b    2
c    3
d    0
dtype: int64
>>> obj2 = pd.Series([ 4, 7, -3, 2], index=['d', 'a', 'b', 'c'])
>>> obj2.sort_values()
b   -3
c    2
d    4
a    7
dtype: int64
```

Pandas - Sorting

```
>>> frame = pd.DataFrame(np.arange(8).reshape((2,4)), index=['three', 'one'],
columns=['d', 'a', 'b', 'c'])
>>> frame.sort_index()     #The same as frame.sort_index(axis=0)
       d  a  b  c
one    4  5  6  7
three  0  1  2  3
>>> frame.sort_index(axis=1)
       a  b  c  d
three  1  2  3  0
one    5  6  7  4
>>> frame.sort_index(axis=1, ascending=False)
       d  c  b  a
three  0  3  2  1
one    4  7  6  5
```

When Kanalltix wants to sort the values by a specified column, however, he cannot use `sort_index`, but has to revert to `sort_values(by= )`.

```
>>> frame = pd.DataFrame({'b': [4, 7, -3], 'a': [0, 1, 0]})
>>> frame.sort_values(by='b')
   a  b
2  0 -3
0  0  4
1  1  7
```

Pandas - Baby names 1880-2015

On http://www.ssa.gov/oact/babynames/limits.html the total number of births for each gender/name combination is given as a raw archive.

```
Mary,F,7065
Anna,F,2604
Emma,F,2003
Elizabeth,F,1939
Minnie,F,1746
```

Since this is a comma-separated form, use is made of `pandas.read_csv` to load the data

```
import pandas as pd
names1880 = pd.read_csv('Names/yob1880.txt', names=['name', 'sex', 'births'])
```

The data are printed as

```
>>> print(names1880)
         name sex  births
0        Mary   F    7065
1        Anna   F    2604
2        Emma   F    2003
3        Elizabeth   F    1939
4        Minnie   F    1746
...
1996     Worthy   M       5
1997     Wright   M       5
1998       York   M       5
1999  Zachariah   M       5

[2000 rows x 3 columns]
```

To get an overview over all births, we can use the sum of the births by sex:

```
>>> names1880.groupby('sex')['births'].sum()
sex
F     90992
M    110490
Name: births, dtype: int64
```

Pandas - Excel

On the internet, Kaningentix finds an excel sheet containing all herbs, grasses and vegetables which can be found in the forest. The list contains not only the names and the subsequent characterizations, but also where these are found and the time of the medicinal effect.

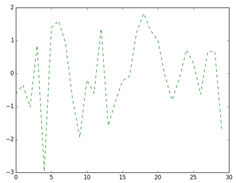
It is advisable to use pandas, making use of the `read_excel` function.

```
>>> table = pd.read_excel(data.xlsx)
>>> table
      herb  color  place          why?
0    grass   green  river  knee injury
1  mushroom  brown   tree     headache
```

Matplotlib

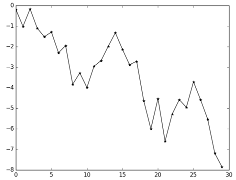
Basic syntax for a plot

```
>>> plt.plot(np.random.randn(30), linestyle='--', color='g')
```



Other styles and colors are available and can easily be searched.

```
>>> plt.plot(np.random.randn(30).cumsum(), color='k', linestyle='solid',
marker='*')
```



Matplotlib - Figures and subplots

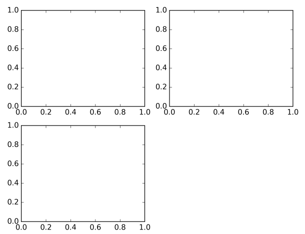
To manipulate figures, use is made of a `Figure` 'object'.

```
>>> fig = plt.figure()
```

Subplots are made using `add_subplot`.

```
>>> ax1 = fig.add_subplot(2,2,1)
>>> ax2 = fig.add_subplot(2,2,2)
>>> ax3 = fig.add_subplot(2,2,3)
```

The first two arguments of `subplot` point at the amount of pictures in one row and in one column. The last argument counts: left above is picture 1, right above is picture 2, left down is 3 etc.

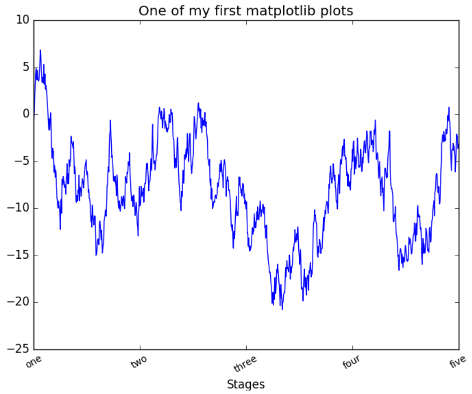


Techniques to fill these plots will be given in the following slides.

Matplotlib - title, axis labels, ticks, and ticklabels

To adjust the axes, it is a good idea to use `add_subplot` - even when there is only one plot in the figure.

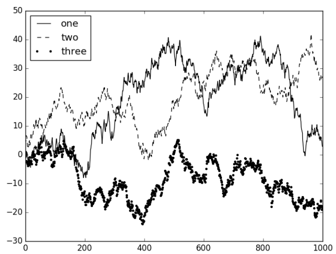
```
>>> fig= plt.figure(); ax = fig.add_subplot(1, 1, 1)
>>> ax.plot(np.random.randn(1000).cumsum())
[<matplotlib.lines.Line2D at 0x2ab9f8ee0fd0>]
>>> ticks = ax.set_xticks([0, 250, 500, 750, 1000])
>>> labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
rotation=30, fontsize='small')
>>> ax.set_title('One of my first matplotlib plots')
<matplotlib.text.Text at 0x2ab9f9648c50>
>>> ax.set_xlabel('Stages')
```



Matplotlib - adding legends

Adding a legend is only possible when more than one plot is included in the graphic.

```
>>> fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
>>> ax.plot(np.random.randn(1000).cumsum(), 'k', label='one')
[<matplotlib.lines.Line2D at 0x2ab9f9aaae50>]
>>> ax.plot(np.random.randn(1000).cumsum(), 'k--', label='two')
[<matplotlib.lines.Line2D at 0x2ab9f9dbd150>]
>>> ax.plot(np.random.randn(1000).cumsum(), 'k.', label='three')
[<matplotlib.lines.Line2D at 0x2ab9f9aa96d0>]
>>> ax.legend(loc='best')
```



Matplotlib - Saving plots to a file

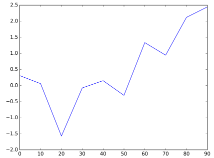
To save the figure on the last figure, `savefig` can be used. Pay attention to the extension - `.png`, `.jpg` and `.pdf` give the respective format of the pictures. The resolution can be indicated using `dpi`.

```
>>> plt.savefig('myfavouritelastpic.png', dpi=400)
```

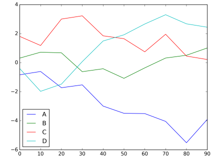
Matplotlib - Plotting functions in Pandas

Line plots

```
>>> s= pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
>>> s.plot()
```



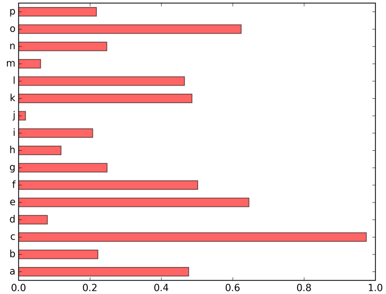

```
>>> df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
columns = ['A', 'B', 'C', 'D'], index=np.arange(0, 100, 10))
>>> df.plot()
```



Matplotlib - Plotting functions in Pandas

Bar plots

```
>>> data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnopq'))
>>> data.plot(kind='barh', color='r', alpha=0.5)
```

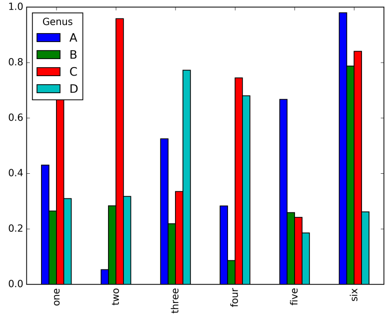


`alpha` points at the intensity of the red color.

Matplotlib - Plotting functions in Pandas

Bar plots

```
>>> df = pd.DataFrame(np.random.rand(6,4), index= ['one', 'two', 'three', 'four',
'five', 'six'], columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
>>> df
Genus         A         B         C         D
one    0.430194  0.264419  0.863249  0.309548
two    0.053039  0.283306  0.958429  0.317259
three  0.525248  0.218834  0.334936  0.772332
four   0.282869  0.086094  0.745032  0.680265
five   0.667275  0.258653  0.241644  0.185485
six    0.979554  0.787525  0.840468  0.261518
>>> df.plot(kind='bar')
```



Matplotlib - Total births by gender and year

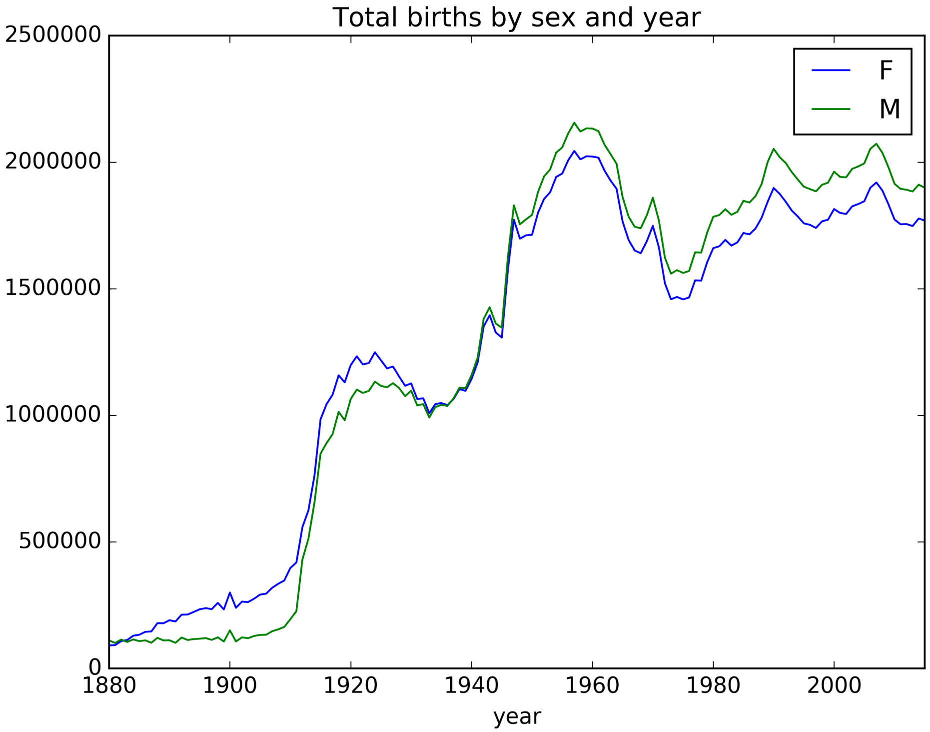
From 1880 to 2015, a file is available containing the year of birth, together with the amount of born females and males.

```
>>> tb=pd.read_csv('out_total_births.csv')
>>> tb
     year        F        M
0    1880    90992   110490
1    1881    91953   100743
2    1882   107848   113686
3    1883   112318   104627
..    ...      ...      ...
133  2013  1747544  1883945
134  2014  1777242  1910876
135  2015  1769325  1898858

[136 rows x 3 columns]

>>> tb.plot(title='Total births by sex and year', x='year',y=['F','M'])
```

Matplotlib - Total births by gender and year



PIP

Pip is a package management system used to install and manage software packages written in Python, taken from the `Python Package Index' (PyPI).

Use:

```
$ pip3 install package-name
$ pip3 uninstall package-name
```

Virtual environments

Miraculix has been coding a lot throughout his life. He has programs made from the early days of Python - and he has ones which he made yesterday. To avoid compatibility issues ("program Herbs1.py needs the module ColorGrass-1.0.2, while program IdefixIllness.py needs the module ColorGrass-12.1.15b"), Miraculix uses `virtual environments` in which he can load the exact packages he needs.

```
$ which python3
/usr/bin/python3
$ python3 -m venv ./venv
$ source venv/bin/activate
(venv)$ which python3
.../venv/bin/python3
(venv)$ which pip3
.../venv/bin/pip3
(venv)$ pip3 install ColorGrass-1.0.2
```

… and Miraculix makes sure he is aware of the employed packages:

```
(venv)$ pip3 freeze > requirements.txt
```

References

"Python for Data Analysis", Wes McKinney, O'Reilly Media, Sebastopol, CA: 2013

https://docs.python.org/3/

https://matplotlib.org/gallery.html