

Data 301 Data Analytics

Python for Data Analytics

Dr. Irene Vrbik

University of British Columbia Okanagan
irene.vrbik@ubc.ca

Introduction

Recall:

- ▶ A Python *module* or *library* is code written by others for a specific purpose.
- ▶ Whenever coding, make sure to look for modules that are already written for you to make your development faster!
- ▶ Modules are imported using the import command:

```
import <modulename>
```

Useful Python Modules

- ▶ Today we be discuss some useful Python modules for data analytics:
 - ▶ NumPy (<http://www.numpy.org/>) - N-dimensional arrays, integrating C/C++ and Fortran code, linear algebra, Fourier transform, and random numbers
 - ▶ SciPy (<http://www.scipy.org/>) - numerical integration and optimization
 - ▶ pandas (<http://pandas.pydata.org/>) - data structures, reading/writing data, data merging/joining/slicing/grouping, time series
 - ▶ scikit-learn (<https://scikit-learn.org/stable/>) - machine learning (built on NumPy, SciPy, and matplotlib)

Charts

In addition, we will also explore a useful module for creating visualizations in Python.

- ▶ matplotlib <http://matplotlib.org/> - foundational 2D plotting

N.B. here are numerous graphing and chart libraries:

- ▶ ggplot <http://ggplot.yhathq.com/> - based on R's ggplot2
- ▶ pygal <http://pygal.org/en/stable/> - dynamic chart library
- ▶ Bokeh <http://bokeh.pydata.org/> - goal is to produce charts similar to D3.js for browsers
- ▶ Seaborn <http://stanford.edu/~mwaskom/software/seaborn/> - based on matplotlib and designed for statistical graphics
- ▶ NVD3 <https://github.com/areski/python-nvd3>

- ▶ **SciPy** (pronounced "Sigh Pie") is group of Python libraries for scientific computing.
- ▶ SciPy includes functions for some advanced math problems as listed on this slide, as well as data visualization.
- ▶ Using data visualization methods is the best way to communicate with others. Showing them meaningful results of analysis. These libraries enable you to create graphs, charts and maps.

matplotlib

- ▶ `matplotlib` is the most widely used plotting library in Python.
- ▶ N.B. to have plots appear in Jupyter Notebook will use the command:

```
%matplotlib inline
```

(if you are not using Jupyter notebook, you don't need this)

pyplot

- ▶ `matplotlib.pyplot` is a commonly used sub-library.
- ▶ It is standard practice to shorten `matplotlib.pyplot` to `plt`:

```
import matplotlib.pyplot as plt
```

This will save us some typing (eg. we can type `plt.bar`) instead of `matplotlib.pyplot.bar`)

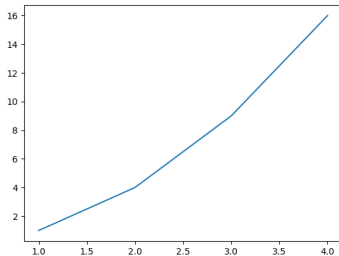
- ▶ See a helpful [pyplot tutorial](#)

dotted module names

Packages are a way of structuring Python's module namespace by using *dotted module names*. For example, the module name `A.B` designates a submodule named `B` in a package named `A`.

As a very simple first example (from [pyplot tutorial](#)), lets create a line chart:

```
# x-values = 1,2,3,4  
# y-values = 1,4,9,16  
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])  
# plot wont be shown until we call:  
plt.show()
```



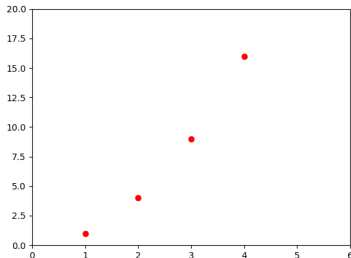
We can change the plot from lines - to points o and from blue (b) to red (r).

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
```

Additionally we can change the axis ranges:

```
# x-axis from 0--6, y-axis from 0--20  
plt.axis([0, 6, 0, 20])  
plt.show()
```

See [here](#) for more plotting options.



- ▶ In matplotlib were limited to working with lists.
- ▶ To get over this limitation, we will generally also import the numpy module (pronounced *num-pie*) to make use of numpy arrays.
- ▶ These arrays can be used for both input and output objects and perform fast array calculations.
- ▶ Common practice is to shorten this module name to np using `import numpy as np`
- ▶ See a helpful [numpy tutorial](#)

numpy arrays

A **numpy array** is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.

rank The number of dimensions is the rank of the array

shape the shape of an array is a tuple of integers giving the size of the array along each dimension.

Consider the following examples taken from the **pyplot tutorial**

```
# Create a rank 1 array
>>> a = np.array([1, 2, 3])
>>> a[0] = 5    # Change an element of the array
>>> print(a.shape) # Prints "(3,)"
>>> a
array([5, 2, 3])
```

numpy arrays

Create a rank 2 array using Python nested lists, and access elements using square brackets:

```
>>> b = np.array([[1,2,3],[4,5,6]])
>>> print(b.shape)  # Prints "(2, 3)"
>>> b
array([[1, 2, 3],
       [4, 5, 6]])
# extract element in row 1 (index 1) and column 3 (index 2)
>>> b[1,2]
6
>>> b[(1,2)] # can also index using a tuple
6
# more examples:
>>> print(b[0,2],b[1,0])
3 4
```

numpy arrays indexing

Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

```
>>> b
array([[1, 2, 3],
       [4, 5, 6]])
>>> # first row:
>>> b[0, :]
array([1, 2, 3])
>>> # second column
>>> b[:,1]
array([2, 5])
>>> # first 2 rows and columns 1 and 2
b[:2, 1:3]
```

numpy arrays

np.arange return evenly spaced values within a given interval.
Eg, evenly sampled points between 0 and 5 at 0.2 intervals

```
>>> t = np.arange(0, 1, 0.2)
# array([0. , 0.2, 0.4, 0.6, 0.8])
```

General usage: `np.arange(start, stop, step)`.

- ▶ as before start and step defaults to 0 and 1, resp.

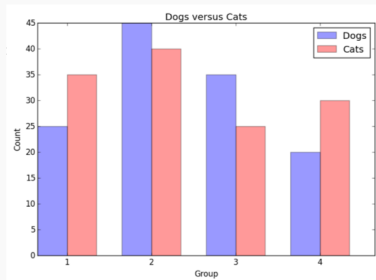
There are a number of **other methods** for array creation as well:

```
>>> a = np.zeros((2,2))    # Create a 2x2 array of zeros
>>> b = np.ones((1,2))     # Create a 1x2 array of ones
>>> c = np.full((2,2), 7)  # Creates 2x2 a array of 7s
```

matplotlib - Bar Chart Example

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

```
# number of dogs/group  
data1 = [25,45,35,20]  
# number of cats/group  
data2 = [35,40,25,30]  
index = np.arange(len(data1))  
bar_width = 0.35  
opacity = 0.4  
# cont'd on next slide
```



```
>>> # to create arrays for rect position  
>>> index = np.arange(len(data1))  
>>> index  
array([0, 1, 2, 3])
```

matplotlib - Bar Chart Example

```
# continued from previous slide
rects1 = plt.bar(index, data1, bar_width, alpha=opacity,
                  color='b', label='Dogs')
rects2 = plt.bar(index + bar_width, data2, bar_width,
                  alpha=opacity, color='r', label='Cats')
plt.xlabel('Group')
plt.ylabel('Count')
plt.title('Dogs versus Cats')
plt.xticks(index + bar_width, ('1', '2', '3', '4'))
plt.legend()
plt.tight_layout() # to remove whitespace
plt.show()
```

N.B.:

```
>>> index + bar_width
array([0.35, 1.35, 2.35, 3.35])
```


Some comments

- ▶ Python starts plotting everything in the background and won't display the graph until `plot.show()` is called.
- ▶ matplotlib colours are given [here](#)
 - ▶ eg. `r` for red `c` for cyan `k` for black
 - ▶ you can also use full names (`'green'`), hex strings (`'#008000'`), or RGB/RGBA tuples (`(0,1,0,1)`).
- ▶ The `label` argument will be used when we call `plt.legend()`
 - ▶ the colours, and labels will be generated automatically (using the `plt.x/ylabel`)

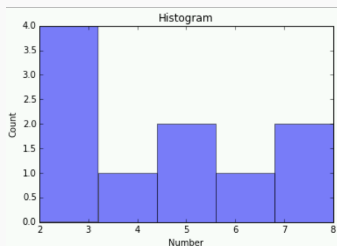
matplotlib - Histograms

Python histograms

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

```
num_bins = 5
x = [5,3,8,5,2,7,2,4,6,2]
# saves output to variables
# (eg of tuple unpacking!)
n, bins, patches = plt.hist(x, num_bins,
                             density=False, facecolor='blue', alpha=0.5)
```

```
plt.xlabel('Number')
plt.ylabel('Count')
plt.title('Histogram')
plt.show()
```



matplotlib - Histograms comments

`bins` partition our range of values into a subsets

Histograms count the number of observations falling into each bin

`n`: is the number of counts in each bin of the histogram

`bins`: is the left hand edge of each bin, example:

```
>>> bins  
array([2. , 3.2, 4.4, 5.6, 6.8, 8. ])
```

`patches`: is the individual patches used to create the histogram,
e.g a collection of rectangles:

```
>>> print(patches[1])  
Rectangle(xy=(3.2, 0), width=1.2, height=1, angle=0)
```

`density` is a logical object that is False (or 0) if you want the
y-axis to be in terms of counts, and True (or 1) when plotting
the probability densities

SciPy stats

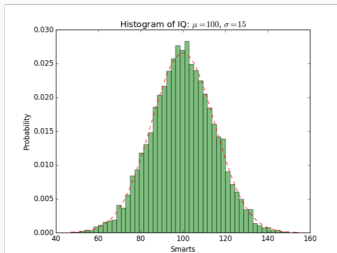
- ▶ Another helpful module is `scipy.stats`
- ▶ This module contains a large number of probability distributions as well as a growing library of statistical functions.
- ▶ We will use the `norm.pdf` compare the probability density function (pdf) or a normal distribution with random data generated from a normal distribution.
 - ▶ The **normal distribution** is a bell shaped distribution with a mean (i.e. center) given by μ (or `mu` in the code) and a standard deviation of σ (or `sigma` in the code).
 - ▶ We generate random normal data using the `numpy.random.randn` function (which return samples from the "standard normal"¹ distribution).
- ▶ N.B. we use **TeX markup** in the title eg. `μ` appears μ

¹Normal distribution with a $\mu=0$, $\sigma = 1$

matplotlib - Histogram Example (2)

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

mu = 100
sigma = 15
x = mu+sigma*np.random.randn(10000)
num_bins = 50
n, bins, patches = plt.hist(x, num_bins, edgecolor="k",
                             density=1, facecolor='green', alpha=0.5)
y = scipy.stats.norm.pdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'Histogram of IQ:  $\mu=100$ ,  $\sigma=15$ ')
plt.subplots_adjust(left=0.15)
plt.show()
```



Exercise

1. How does your plot change when you update the number of bins from the previous example from 50 to 10.
2. How does your plot change when you delete the `edgecolor = "k"`? (remember k stands for black)
3. How does your plot change when you set `density` to `False`?
4. How does your plot change when you set the `facecolor` to `'g'`?
5. How does your plot change when you set `alpha` to `0.1`?

Try in Charts

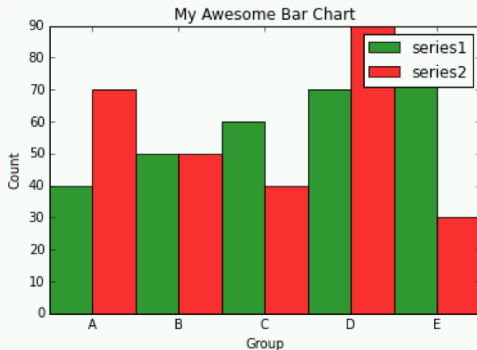
Example 16

Write a program to create a bar chart for this data:

```
series1 = [40, 50, 60, 70, 80]
```

```
series2 = [70, 50, 40, 90, 30]
```

Output:



SciPy stats

- ▶ In addition the `scipy.stats` module has the `linregress` function for performing linear regression.

- ▶ The general syntax is:

```
scipy.stats.linregress(x, y)
```

- ▶ The above calculate a linear least-squares regression for two sets of measurements: the explanatory variable `x` and the response variable `y`.

SciPy Linear Regression Example

```
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
x = np.array([5, 7, 9, 11, 13, 15])
y = np.array([11, 14, 20, 24, 29, 31])
slope, intercept, r_value, p_value,
... slope_std_error = stats.linregress(x, y)
```

Save the resulting tuple to variables that we can use later,...

SciPy Linear Regression Example

Find the residuals, (observed y values minus the predicted y values produced by our fitted model):

```
predict_y = intercept + slope * x
print("Predicted y-values:",predict_y)
pred_error = y - predict_y
print("Prediction error:",pred_error)
```

Compute the **residual standard error** (a measure of prediction error)

```
degr_freedom = len(x) - 2
residual_std_error = np.sqrt(np.sum(pred_error**2)/degr_freedom)
print("Residual error:",residual_std_error)
```

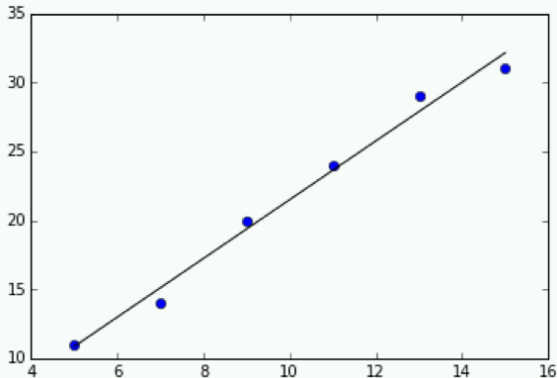
Output:

```
Predicted y-values: [10.85714286 15.11428571 ....
Prediction error: [ 0.14285714 -1.11428571  0.62857143  ...
Residual error: 1.041976144503454
```

SciPy Linear Regression Example

Plot the raw data with the fitted regression line:

```
plt.plot(x, y, 'o') # o for points  
plt.plot(x, predict_y, 'k-') # - for line, k for black  
plt.show()
```



SciPy - K-means Clustering

Scipy also provides routines for conducting k -means clustering and vector quantization.

The input of k -means algorithm:

- k the number of clusters to generate
- x the set of observations to cluster

The output of k -means algorithm:

- ▶ a set of centroids, one for each of the k clusters.
- ▶ a classification vector: an observation is assigned to the cluster whose centroid is closest to it.

SciPy - K-means Clustering

```
from numpy import vstack,array
from numpy.random import rand
from scipy.cluster.vq import kmeans,vq

# creates a 300x2 array of data
data = vstack((rand(150,2) + array([.5,.5]),rand(150,2)))
# computing K-Means with K = 2 (2 clusters)
centroids,_ = kmeans(data,2)
# assign each sample to a cluster
idx,_ = vq(data,centroids)
```

Some comments

`rand(150,2)` generates an `np.array` with 150 rows and 2 columns with random numbers from the interval `[0,1)` more [here](#)

Alternatively, we could have used a loop and See `numpy.asarray`

```
import random as rnd
import numpy as np
data = []
for i in range(0,100):
    data.append([rnd.random(), rnd.random()])

# need to convert to array for kmeans
data = np.asarray(data)
# looks like:
# array([[0.78056987, 0.62426341],
#        [0.96726103, 0.9859364 ],
#        [1.35816411, 1.4966473 ],
#        ....
```

Some comments

vstack stack arrays in sequence vertically (row wise).

```
>>> a = np.array([1, 2, 3])
>>> b = np.array([[2, 3, 4], [5,6,7]])
>>> np.vstack((a,b))
array([[1, 2, 3],
       [2, 3, 4],
       [5, 6, 7]])
```

Tip:

When you import a function named `myfunction` from a module named `mymodule` *via* `import mymodule` you need to use the dot notation, i.e. `mymodule.myfunction` to use it. If you import the function *via* `from mymodule import myfunction`, you can access the function by calling it directly, i.e. `myfunction`

Some comments

The notation `,_` means we'll be taking the first element of the tuple outputted from `kmeans()` (i.e. the $k \times N$ centroid array) and ignore the second argument.

```
# a simple example (2 gets thrown away)
>>> A,_ = [1,2] # A = 1
>>> _,B = [1,2] # B=2
>>> A,_,_ = [1,2,3] # A = 1
```

Alternatively, we may think of this notation as doing the following:

```
>>> km = kmeans(data,2)
>>> type(km)
<class 'tuple'>
>>> centroids = km[0]
>>> centroids
array([[0.45706156, 0.53454601],
       [1.00664659, 1.02451566]])
```


SciPy - K means Clustering (2)

```
>>> # Move data into individual lists based on clustering
... clusters = []
>>> for i in range(0, numclusters):
...     clusters.append([], [])
...
>>> clusters
[[[]], [], [], []]
```

Creates a nested Python list. First element of clusters contains the data points from members assigned to group 0, the second element contains the data pairs assigned to group 1.

```
# run after the code from the next slide:
clusters[0][0] # x values from cluster 0 (blue)
clusters[0][1] # y values from cluster 0 (blue)
clusters[1][0] # x values from cluster 1 (red)
clusters[1][1] # y values from cluster 1 (red)
```

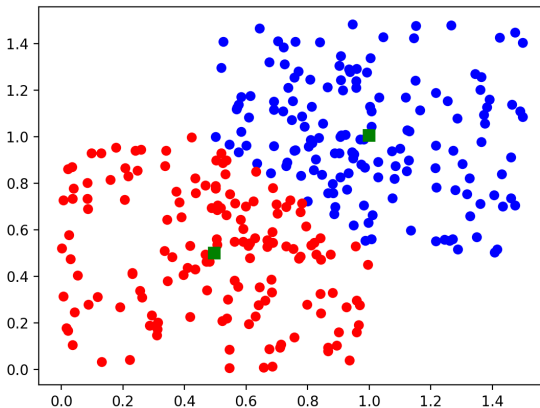
SciPy - K means Clustering (3)

```
for i in range(0,len(idx)):
    clusterIdx = idx[i]
    # saving the x values for cluster = clusterIdx
    clusters[clusterIdx][0].append(data[i][0])
    # saving the y values for cluster = clusterIdx
    clusters[clusterIdx][1].append(data[i][1])

# Plot data points and cluster centroids
# 'ob' for blue points (group index 0)
# 'or' for red points (group index 1)
plt.plot(clusters[0][0],clusters[0][1],'ob',
          clusters[1][0],clusters[1][1],'or')
# centroids[:,0] = first column for centroid of group 0
# centroids[:,1] = second column for centroid of group 1
plt.plot(centroids[:,0],centroids[:,1],'sg',markersize=8)
plt.show()
```

SciPy - K means Clustering

The plot argument `ob/or` stands for `blue` and `red` circle markers, resp. See more on plot formatting [here](#).



Example 17

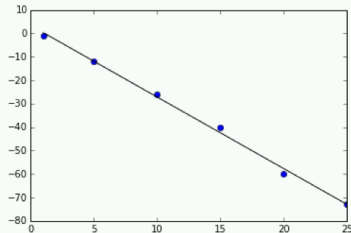
Write a program that uses SciPy to perform a linear regression on this data set:

$x = [1, 5, 10, 15, 20, 25]$

$y = [-1, -12, -26, -40, -60, -73]$

Output:

```
Formula: -3.05 * x + 3.3 = y  
Predicted y-values: [ 0.25 -11.95 -27.2 -42.45 -57.7 -72.95]  
Prediction error: [-1.25 -0.05 1.2 2.45 -2.3 -0.05]  
Residual error: 1.89076704012
```



scikit-learn and BeautifulSoup

scikit-learn (<http://scikit-learn.org/>) is a machine learning library for Python.

Features:

- ▶ classification
- ▶ regression
- ▶ clustering
- ▶ dimensionality reduction

BeautifulSoup

(<http://www.crummy.com/software/BeautifulSoup/>) is a library to make it easy to search, navigate, and extract data from HTML and XML documents.

Python - Databases

- ▶ **MySQL** is a popular open source relational database management system.
- ▶ We can access this data via Python using the `mysql.connector` module. See the **developer guide** and **W3schools tutorial** for more details
 - ▶ **notes** on connecting
 - ▶ **notes** on creating tables
 - ▶ **notes** on querying
- ▶ Note that this code will require that we are on UBC secure wifi (off campus we must connect with VPN).

Try it - Databases

Example 18

Write a program that queries the WorksOn database and returns the employees grouped by title where the employee name is after 'J'. The output should display their title and the average salary for that title. Connection info:

```
cnx = mysql.connector.connect(user='data301',  
password='ubc', host='cosc304.ok.ubc.ca',  
database='WorksOn')
```

Output:

```
EE 30000.000000  
ME 40000.000000  
PR 20000.000000  
SA 50000.000000
```

answer

```
import mysql.connector
try:
    cnx=mysql.connector.connect(user='data301', password='ubc',
    host='cosc304.ok.ubc.ca',database='WorksOn')
    cursor = cnx.cursor()
    query = ("SELECT title, AVG(salary) as avgSalary
    FROM emp WHERE ename >= 'J' GROUP BY title")
    cursor.execute(query)
    for (title, avgSalary) in cursor:
        print(title, avgSalary)
    cursor.close()
except mysql.connector.Error as err:
    print(err)
finally:
    cnx.close()
```


MapReduce

Map-Reduce is a technique for processing large data sets in a functional manner. See the [Wikipedia article](#)

The technique was invented by Google and is implemented in a variety of systems including Python, NoSQL databases, and a Big Data system called Hadoop.

In Hadoop, map takes as input key-value pairs and outputs key-value pairs. The shuffle step will move pairs to particular machines based on keys. The reduce step takes a list of key-value pairs (with same key) and reduces to one value.

It is possible to code map/reduce functions in Python for use in Hadoop cluster.

MapReduce

Simpler version of Map-Reduce in Python without a cluster:

Map function takes as input a list and a function then applies function to each element of the list to produce a new list as output

Filter function only keeps list elements where filter function is True

Reduce function takes result of map/filter and produces single value from list

N.B. The reduce function has been moved to the `functools` module for Python3 with creator noting *"99% of the time an explicit for loop is more readable"*

map

General usage: `map(function_to_apply, list_of_inputs)`

Data: a_1, a_2, \dots, a_n (could be elements in a tuple or a list)

Function: $f(x)$

Calling `map($f(x)$, Data)` would return

$$f(a_1), f(a_2), \dots f(a_n)$$

filter

General usage: `filter(function_to_apply, list_of_inputs)`

Data: a_1, a_2, \dots, a_n (could be elements in a tuple or a list)

Function: $f(x)$ (this function should return True or False)

Calling `map($f(x)$, Data)` would return:

all a_i s for which $f(a_i)$ returns True

reduce

General usage: `reduce(function_to_apply, list_of_inputs)`

Data: a_1, a_2, \dots, a_n (could be elements in a tuple or a list)

Function: $f(x, y)$

Calling `reduce(f(x, y), Data)` would perform:

Step 1: $val_1 = f(a_1, a_2)$

Step 2: $val_2 = f(val_1, a_3)$

Step 3: $val_3 = f(val_2, a_4)$

...

Step n-1 $val_{n-1} = f(val_{n-2}, a_n)$

and return: val_{n-1}

Python - MapReduce Example

```
import functools      # For Reduce

data = [1, 2, 3, 4, 5, 6]

def triple(x): # function to be used with map
    return x*3

def myfilter(x): # function to be used with filter
    if x % 2 == 0:
        return True
    return False

def sum(x, y):      # function to be used with reduce
    return x+y
```

Python - map Example

```
>>> result = list(map(triple, data))
>>> print("Result after map:",result)
Result after map: [3, 6, 9, 12, 15, 18]
```

To see this using a for loop:

```
>>> items = [1, 2, 3, 4, 5, 6]
>>> output = []
>>> for i in items:
...     val = triple(i)
...     output.append(val)
...
>>> print(output)
[3, 6, 9, 12, 15, 18]
```

Python - filter Example

```
>>> result = list(filter(myfilter, result))
>>> print("Result after filter:",result)
Result after filter: [6, 12, 18]
```

Alternatively using a for loop:

```
>>> items = [3, 6, 9, 12, 15, 18]
>>> output = []
>>> for i in items:
...     val = myfilter(i)
...     if val:
...         output.append(i)
...
>>> print(output) # returns all the even numbers
[6, 12, 18]
```


Python - reduce Example

```
>>> result = functools.reduce(sum, result)
>>> print("Result after reduce:",result)
Result after reduce: 36
```

To see this using a loop:

```
>>> items = [6, 12, 18]
>>> val = items[0]
>>> item2 = items[1:]
>>> for a in item2:
...     val = sum(val,a)
...
>>> print(val)
36
```

-
- ▶ Notice that when using `map`, the items in the result are values returned by the function `triple`.
 - ▶ In contrast, the values returned by `myfilter` in `filter(f, ...)` are not outputted results themselves, but rather an index of the values that returned `True`.
 - ▶ The values returned by the `reduce` function returns a *single* result.

Try It - MapReduce

Example 19

Write a map-reduce program that during the map step will subtract 2 from each element. The reduce step should return the product of all the elements in the list.

Pandas

- ▶ Pandas is a very popular Python library for data analysis.
- ▶ The library provides “high-performance, easy-to-use data analysis tools for the Python programming language” [Source: pandas.pydata.org]
- ▶ By importing this module, you can create *data frame* which are two dimensional tables consisting of column and row labels (like a spreadsheet).
- ▶ It is designed to provided easy indexing functionality.

Installing pandas

- ▶ The easiest way to install pandas is to use the **Anaconda** distribution.
- ▶ With Anaconda you can install pandas and any dependencies by typing the following in Terminal/Command Prompt:

```
$ conda install pandas
```

- ▶ Check to see if you have installed pandas properly in Python:

```
>>> import pandas  
>>> pandas.__version__  
'0.24.2'
```

- ▶ Read more on pandas install using other methods **[here](#)**.

-
- ▶ At the core of pandas are three data structures:

Conclusion

Python has many libraries to help with data analysis tasks:

- ▶ reading and write to files
- ▶ csv module for processing CSV files
- ▶ Biopython for bioinformatics
- ▶ numerous chart libraries including matplotlib and ggplot
- ▶ SciPy - collection of libraries for scientific computing
- ▶ libraries for web access and parsing (BeautifulSoup)
- ▶ database access libraries and connectors

The `try-except` statement is used to handle exceptions so that the program may continue when an error condition occurs.

Objectives

- ▶ Open, read, write, and close text files
- ▶ Process CSV files including using the csv module
- ▶ Define: IPv4/IPv6 address, domain, domain name, URL
- ▶ Read URLs using urllib.request.
- ▶ Define: exception, exception handling
- ▶ Use try-except statement to handle exceptions and understand how each of try, except, else, finally blocks are used
- ▶ Import Python modules
- ▶ Use Biopython module to retrieve NCBI data and perform BLAST
- ▶ Build charts using matplotlib
- ▶ Perform linear regression and k-means clustering using SciPy
- ▶ Connect to and query the MySQL database using Python
- ▶ Write simple Map-Reduce programs