NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment.

**import** **numpy** **as** **np**

a = np.arange(15).reshape(3, 5) # 3rows and 5 columns

x = np.arange(10,20,2)

a.shape # (3,5)

a.ndim #2

a.dtype.name #int64

a.size # 15

##array

b = np.array([6, 7, 8]) #create one dimension array

##two d array

b = np.array([(1.5,2,3), (4,5,6)])

##

np.zeros( (3,4) ) ##3 row and 4 cols with all 0 value

np.ones( (2,3,4), dtype=np.int16 )

c **=** np**.**full((2,2), 7) *# Create a constant array*

##index

a **=** np**.**array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

*# Use slicing to pull out the subarray consisting of the first 2 rows*

*# and columns 1 and 2; b is the following array of shape (2, 2):*

*# [[2 3]*

*# [6 7]]*

b **=** a[:2, 1:3]

*# A slice of an array is a view into the same data, so modifying it*

*# will modify the original array.*

**print**(a[0, 1]) *# Prints "2"*

b[0, 0] **=** 77 *# b[0, 0] is the same piece of data as a[0, 1]*

**print**(a[0, 1]) *# Prints "77"*

##reshape

a = np.arange(8)

b = a.reshape(4,2)

##transpose

a = np.arange(12).reshape(3,4)

print np.transpose(a)

###split

a = np.arange(9)

print 'Split the array in 3 equal-sized subarrays:'

b = np.split(a,3)

print 'Split the array at positions indicated in 1-D array:'

b = np.split(a,[4,7])

print b

a = np.arange(16).reshape(4,4)

print 'Horizontal splitting:'

b = np.hsplit(a,2)

print b

print '\n'

###append

print np.append(a, [7,8,9])

print np.append(a, [[7,8,9]],axis = 0)

print '\n'

print 'Append elements along axis 1:'

print np.append(a, [[5,5,5],[7,8,9]],axis = 1)

###insert

a = np.array([[1,2],[3,4],[5,6]])

print 'First array:'

print a

print '\n'

print 'Axis parameter not passed. The input array is flattened before insertion.'

print np.insert(a,3,[11,12])

print '\n'

print 'Axis parameter passed. The values array is broadcast to match input array.'

print 'Broadcast along axis 0:'

print np.insert(a,1,[11],axis = 0)

print '\n'

print 'Broadcast along axis 1:'

print np.insert(a,1,11,axis = 1)

##delete

a = np.arange(12).reshape(3,4)

print 'First array:'

print a

print '\n'

print 'Array flattened before delete operation as axis not used:'

print np.delete(a,5)

print '\n'

print 'Column 2 deleted:'

print np.delete(a,1,axis = 1)

print '\n'

print 'A slice containing alternate values from array deleted:'

a = np.array([1,2,3,4,5,6,7,8,9,10])

print np.delete(a, np.s\_[::2])

##unique

a = np.array([5,2,6,2,7,5,6,8,2,9])

print 'First array:'

print a

print '\n'

print 'Unique values of first array:'

u = np.unique(a)

print u

print '\n'

print 'Unique array and Indices array:'

u,indices = np.unique(a, return\_index = True)

print indices

print '\n'

print 'We can see each number corresponds to index in original array:'

print a

print '\n'

print 'Indices of unique array:'

u,indices = np.unique(a,return\_inverse = True)

print u

print '\n'

print 'Indices are:'

print indices

print '\n'

print 'Reconstruct the original array using indices:'

print u[indices]

print '\n'

print 'Return the count of repetitions of unique elements:'

u,indices = np.unique(a,return\_counts = True)

print u

print indices

import random

print random.randint(0, 5)

import random

random.random() \* 100

random.choice( ['red', 'black', 'green'] ).

The choice function can often be used for choosing a random element from a list.

import random

myList = [2, 109, False, 10, "Lorem", 482, "Ipsum"]

random.choice(myList)

The shuffle function, shuffles the elements in list in place, so they are in a

random order.

random.shuffle(list)

df.groupby('A').sum()

df.groupby(['A','B']).sum()

df.groupby(['Country', 'Item\_Code'])[["Y1961", "Y1962", "Y1963"]].sum()

df["score"]

df [["score", "release\_year"]]

df["score"].mean()

Filter rows

df.loc[df['column\_name'] == some\_value]

print(df.loc[df['B'].isin(['one','three'])])

Combine multiple conditions with &:

df.loc[(df['column\_name'] == some\_value) & df['other\_column'].isin(some\_values)]

To select rows whose column value does not equal some\_value, use !=:

df.loc[df['column\_name'] != some\_value]

df[(df.foo == 222) | (df.bar == 444)]

sort

df.sort\_values(by=['col1'])

df.sort\_values(by=['col1', 'col2'])

df.sort\_values(by='col1', ascending=**False**)

Putting NAs first

df.sort\_values(by='col1', ascending=**False**, na\_position='first')

## correlation

**import pandas as pd**

**path = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data'**

**mpg\_data = pd.read\_csv(path, delim\_whitespace=True, header=None,**

**names = ['mpg', 'cylinders', 'displacement','horsepower',**

**'weight', 'acceleration', 'model\_year', 'origin', 'name'],**

**na\_values='?')**

Upon inspecting the dataset, we see that **horsepower** has six missing values, which pandas' correlation method will automatically drop. Since the number of missing values is small, this setting is acceptable for our illustrative example. However, always make sure that dropping missing values is appropriate for your use case. If that is not the case, there are many existing methods for filling in and handling missing values, such as simple mean imputation.

**mpg\_data.info()**

pandas provides a convenient one-line method **corr()** for calculating correlation between data frame columns. In our fuel efficiency example, we can check whether heavier vehicles tend to have lower **mpg** by passing the method to specific columns:

**mpg\_data['mpg'].corr(mpg\_data['weight'])**

**-0.8317409332443354**

As expected, there seems to be a strong negative correlation between vehicle**weight** and **mpg**. But what about **horsepower** or **displacement**? Conveniently, pandas can quickly calculate correlation between all columns in a dataframe. The user can also specify the correlation method: Spearman, Pearson, or Kendall. If no method is specified, Pearson is used by default. Here, we drop model year and origin variables and calculate Pearson correlation between all remaining columns of the data frame:

In [ ]:

**# pairwise correlation**

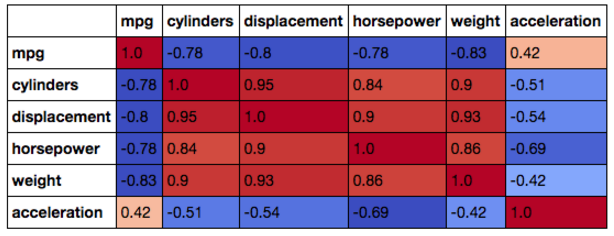
**mpg\_data.drop(['model\_year', 'origin'], axis=1).corr(method='spearman')**

Out[ ]:

|  | **mpg** | **cylinders** | **displacement** | **horsepower** | **weight** | **acceleration** |
| --- | --- | --- | --- | --- | --- | --- |
| **mpg** | 1.000000 | -0.821864 | -0.855692 | -0.853616 | -0.874947 | 0.438677 |
| **cylinders** | -0.821864 | 1.000000 | 0.911876 | 0.816188 | 0.873314 | -0.474189 |
| **displacement** | -0.855692 | 0.911876 | 1.000000 | 0.876171 | 0.945986 | -0.496512 |
| **horsepower** | -0.853616 | 0.816188 | 0.876171 | 1.000000 | 0.878819 | -0.658142 |
| **weight** | -0.874947 | 0.873314 | 0.945986 | 0.878819 | 1.000000 | -0.404550 |
| **acceleration** | 0.438677 | -0.474189 | -0.496512 | -0.658142 | -0.404550 | 1.000000 |

pandas also supports highlighting methods for tables, so it is easier to see high and low correlations. It is important to understand possible correlations in your data, especially when building a regression model. Strongly correlated predictors, phenomenon referred to as multicollinearity, will cause coefficient estimates to be less reliable. Below is an example of calculating Pearson correlation on our data and using a color gradient to format the resulting table:

**model\_year', 'origin'], axis=1).corr(method='pearson').style.format("{:.2}").background\_gradient(cmap=plt.get\_cmap('coolwarm'), axis=1)**



Finally, to visually inspect the relationship between **mpg**, **weight**, **horsepower**,and **acceleration**, we can plot these values and calculate Pearson and Spearman coefficients. The dataset at hand consists of less than 400 points, which can be easily displayed on a [scatter plot](https://www.datascience.com/blog/learn-data-science-intro-to-data-visualization-in-matplotlib). If you are dealing with much larger datasets, consider taking a sample of your data first to speed up the process and produce more readable plots.

In this case, Spearman's coefficient is higher than Pearson for **horsepower** and**weight**, since relationship is non-linear. For **acceleration**, both coefficients are close since the relationship is not as clearly defined:

**# plot correlated values**

**plt.rcParams['figure.figsize'] = [16, 6]**

**fig, ax = plt.subplots(nrows=1, ncols=3)**

**ax=ax.flatten()**

**cols = ['weight', 'horsepower', 'acceleration']**

**colors=['#415952', '#f35134', '#243AB5', '#243AB5']**

**j=0**

**for i in ax:**

**if j==0:**

**i.set\_ylabel('MPG')**

**i.scatter(mpg\_data[cols[j]], mpg\_data['mpg'], alpha=0.5, color=colors[j])**

**i.set\_xlabel(cols[j])**

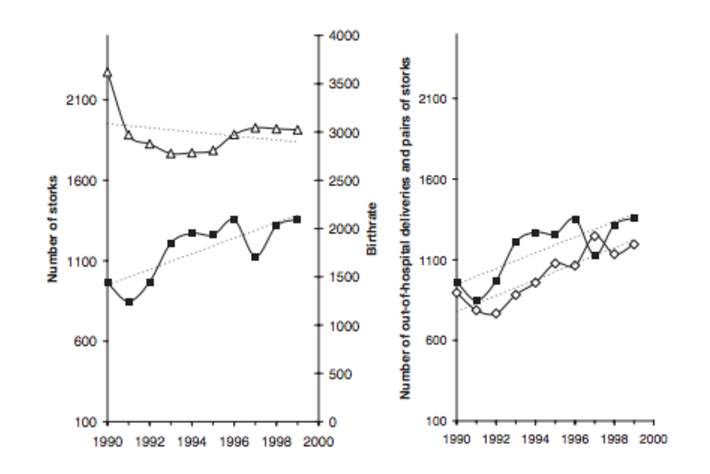
**i.set\_title('Pearson: %s'%mpg\_data.corr().loc[cols[j]]['mpg'].round(2)+' Spearman: %s'%mpg\_data.corr(method='spearman').loc[cols[j]]['mpg'].round(2))**

**j+=1**

**plt.show()**

Correlation and Causation

The relationships between variables in our fuel efficiency example were very intuitive and explainable through vehicle mechanics. However, things are not always this straightforward. It is a well-known fact that correlation does not imply causation, and therefore, any strong correlation should be thought of critically. For example, German researchers used the concept of correlation in [this humorous paper](http://web.stanford.edu/class/hrp259/2007/regression/storke.pdf) to support a theory that babies are delivered by storks. This figure shows the correlation between the number of storks and baby deliveries:



The chart on the left shows an increasing trend in the number of storks (black line) and a decreasing trend in the number of clinical deliveries. On the other hand, the chart on the right shows that a number of out-of-hospital deliveries (white square marks) follow the increasing pattern in the number of storks. Looking at the correlation between these series, the authors suggest that the increase in out-of-hospital deliveries paired with the increase in the number of storks and the simultaneous decrease in hospital deliveries suggest that more and more babies in Germany are being delivered by storks.