In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sps
import pandas as pd
from statsmodels.sandbox.stats.multicomp import multipletests
from sklearn.cross_validation import train_test_split



%matplotlib inline
```

```
/home/riv/.local/lib/python3.5/site-packages/sklearn/cross_validation.py:4
4: DeprecationWarning: This module was deprecated in version 0.18 in favor
of the model_selection module into which all the refactored classes and fu
nctions are moved. Also note that the interface of the new CV iterators ar
e different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [3]:
```
slump_test = pd.read_csv('slump_test.data', header = 0, sep=',')
slump_test.head()
```

Out[3]:

	No	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	SLUMP(cm)	FLOW(cm)	Compressive Strength (28-day)(Mpa)
0	1	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	34.99
1	2	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.0	20.0	41.14
2	3	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	41.81
3	4	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.0	21.5	42.08
4	5	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	26.82

In [4]:
```
sample = []

sample.append(np.array(slump_test.values[:, 4]))
sample.append(np.array(slump_test.values[:, 5]))
sample.append(np.array(slump_test.values[:, 7]))
sample.append(np.array(slump_test.values[:, 8]))
sample.append(np.array(slump_test.values[:, 9]))
sample.append(np.array(slump_test.values[:,-1]))

sample = np.array(sample).T
```

In [5]:
```
train, test = train_test_split(sample, test_size=0.5, random_state = 1)
```

In [6]:
```
from scipy.stats import multivariate_normal
from statsmodels.distributions.empirical_distribution import ECDF

mean = np.mean(train, axis=0)
cov = np.cov(train, rowvar=0)

theory = multivariate_normal(mean, cov)

print(mean)
```

```
[ 194.1254902     8.38039216  747.18823529   16.29901961   45.57254902
   35.94607843]
```

In [8]:
```python
#https://gist.github.com/vmonaco/e9ff0ac61fcb3b1b60ba
import sys
import numpy as np
import pandas as pd
import scipy.stats as stats
from sklearn.neighbors import kneighbors_graph
from scipy.sparse.csgraph import minimum_spanning_tree

SIGNIFICANCE = 0.05


def mst_edges(V, k):
    """
    Construct the approximate minimum spanning tree from vectors V
    :param: V: 2D array, sequence of vectors
    :param: k: int the number of neighbor to consider for each vector
    :return: V ndarray of edges forming the MST
    """

    # k = len(X)-1 gives the exact MST
    k = min(len(V) - 1, k)

    # generate a sparse graph using the k nearest neighbors of each point
    G = kneighbors_graph(V, n_neighbors=k, mode='distance')

    # Compute the minimum spanning tree of this graph
    full_tree = minimum_spanning_tree(G, overwrite=True)

    return np.array(full_tree.nonzero()).T


def ww_test(X, Y, k=10):
    """
    Multi-dimensional Wald-Wolfowitz test
    :param X: multivariate sample X as a numpy ndarray
    :param Y: multivariate sample Y as a numpy ndarray
    :param k: number of neighbors to consider for each vector
    :return: W the WW test statistic, R the number of runs
    """
    m, n = len(X), len(Y)
    N = m + n

    XY = np.concatenate([X, Y]).astype(np.float)

    # XY += np.random.normal(0, noise_scale, XY.shape)

    edges = mst_edges(XY, k)

    labels = np.array([0] * m + [1] * n)

    c = labels[edges]
    runs_edges = edges[c[:, 0] == c[:, 1]]

    # number of runs is the total number of observations minus edges within
    R = N - len(runs_edges)

    # expected value of R
    e_R = ((2.0 * m * n) / N) + 1

    # variance of R is _numer/_denom
    _numer = 2 * m * n * (2 * m * n - N)
    _denom = N ** 2 * (N - 1)

    # see Eq. 1 in Friedman 1979
    # W approaches a standard normal distribution
    W = (R - e_R) / np.sqrt(_numer/_denom)

    return W, R
```

In [12]:
```
teoretical_sample = theory.rvs(size = 100)
```

In [13]:
```
W, R = (ww_test(train, teoretical_sample))
pvalue = stats.norm.cdf(W)  # one sided test
reject = pvalue <= SIGNIFICANCE

print('W = %.3f, %d runs' % (W, R))
print('p = %.4f' % pvalue)
print('%s H_0 at 0.05 significance level' % ('Reject' if reject else 'Fail t
print('The samples appear to have %s distribution' % ('different' if reject
```

```
W = 1.178, 75 runs
p = 0.8807
Fail to reject H_0 at 0.05 significance level
The samples appear to have similar distribution
```