Assignment 1 - Normalizing Continuous Features

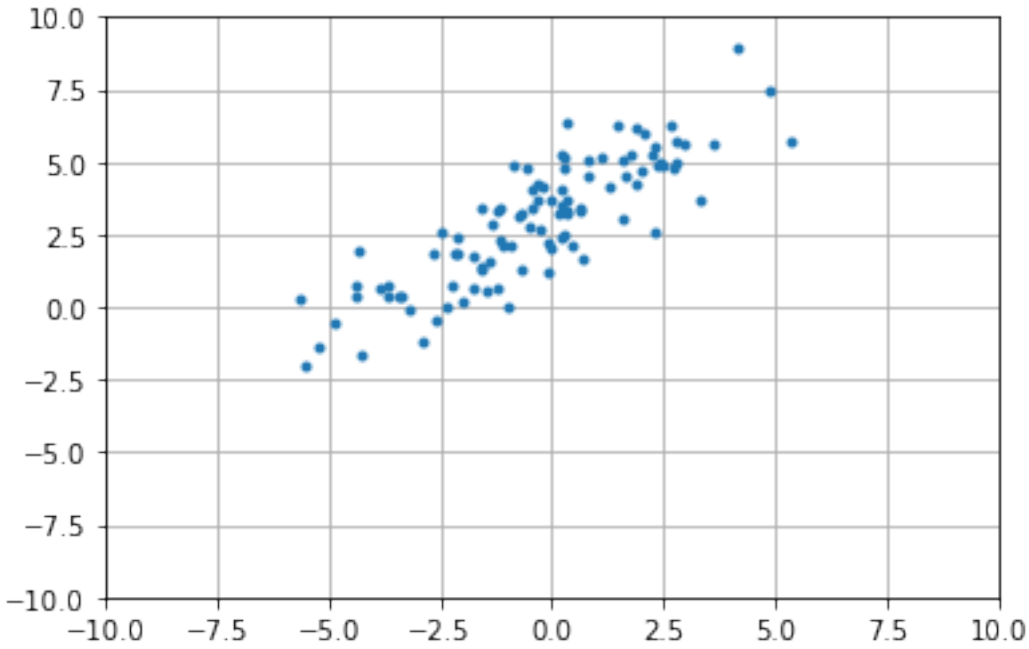
Consider data drawn from a 2 dimensional Normal distribution. Normalize the data by first subtracting the mean from each dimension and then divide the result by its respective standard deviation.

```
import matplotlib.pylab as plt
import numpy as np
from numpy.random import multivariate_normal as mvn
%matplotlib inline

# generates some toy data
mu = np.array([0,3])
C = np.array([[5.,4.],[4.,5.]])
X = mvn(mu,C,100)

# plot the data
plt.plot(X[:,0], X[:,1], '.')
plt.grid()
lim = [-10, 10]
plt.xlim(lim)
plt.ylim(lim)
```

```
(-10.0, 10.0)
```



```
mu = X.mean(axis=0)
mu
```

```
array([-0.34669059,  2.97853925])
```

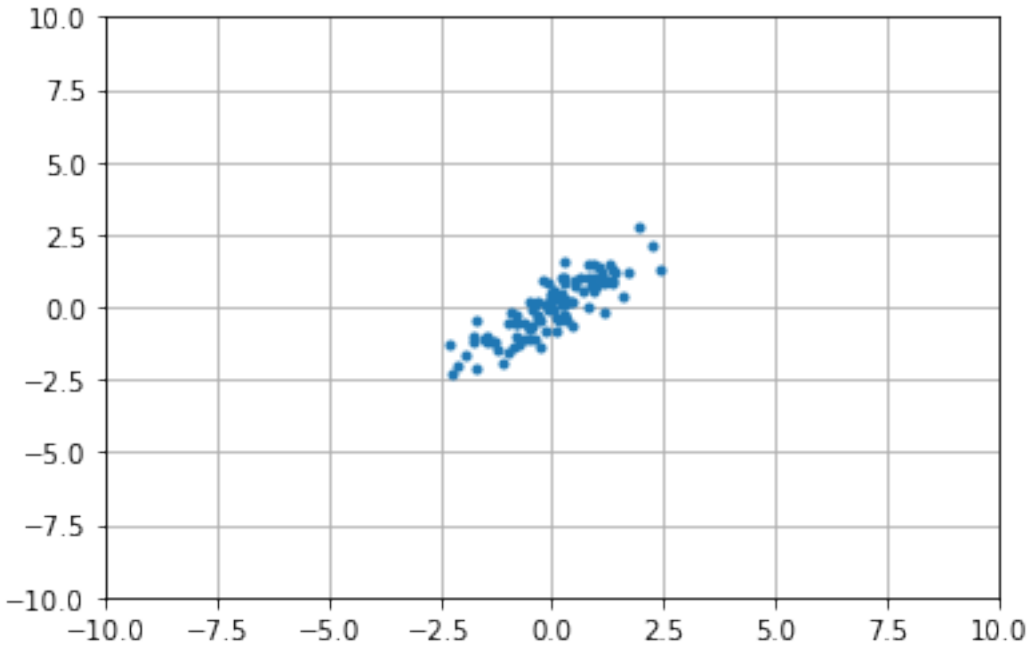
```
stdev = X.std(axis=0)
stdev
```

```
array([2.32629073, 2.15692281])
```

```
# Xnormalized = ...
Xnormalized = (X-mu)/stdev

# plot the data
plt.plot(Xnormalized[:,0], Xnormalized[:,1], '.')
plt.grid()
lim = [-10, 10]
plt.xlim(lim)
plt.ylim(lim)
```

```
(-10.0, 10.0)
```



Assignment 2 - One-Hot Encoding

Consider the data set ['blue', 'yellow', 'blue', 'green', 'red', 'yellow']

Write a function `one_hot_encoding` that takes a list of strings like the above and returns a samples-by-unique-items numpy array in which each row corresponds to the one-hot-encoded version of the respective data point in the original list.

```
import numpy as np
data = ['blue', 'yellow', 'blue', 'green', 'red', 'yellow']

# def one_hot_encoding(string_list):
#for p in range(len(string_list)):
#     return

def one_hot_encoding(string_list):
    uniqueValues = np.unique(string_list)
    encode = np.full((len(string_list), len(uniqueValues)), 0)

    for p in range(len(string_list)):
        occurances = np.where(uniqueValues == string_list[p])
        encode[p, occurances] = 1

    return encode

one_hot_encoded_data = one_hot_encoding(data)
one_hot_encoded_data
```

```
array([[1, 0, 0, 0],
       [0, 0, 0, 1],
       [1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```

```
# tests whether the first row encoding is the same as the third
assert (one_hot_encoded_data[0,:] - one_hot_encoded_data[2,:]).sum()
== 0
# tests whether the second row encoding is the same as the last
assert (one_hot_encoded_data[1,:] - one_hot_encoded_data[-1,:]).sum()
== 0
# tests whether each row has only one non-zero entry
assert (one_hot_encoded_data.sum(axis=1) -
np.ones(one_hot_encoded_data.shape[0])).sum() == 0
```

Assignment 3 - Bag-of-Words Features

Consider the following data set

```
corpus = [
     'This is the first document.',
     'This document is the second document.',
     'And this is the third one.',
     'Is this the first document?']
```

Implement a function `bag_of_words` that takes a list of sentences as strings and transforms them into a (preferably sparse) numpy array of size number-of-data-points-by-number-of-words-in-the-corpus.

Compare your result with the result from sklearn.feature_extraction.text.CountVectorizer

```
corpus = [
     'This is the first document.',
     'This document is the second document.',
     'And this is the third one.',
     'Is this the first document?']

import string

def bag_of_words(corpus):
    features = []
    newCorpus = []
    concatted = []

    for p in corpus:
        p = p.lower()
        p = p.translate(str.maketrans('', '', string.punctuation))
        newCorpus.append(p)
        concatted = np.concatenate((concatted, p.split()))

    features = np.unique(concatted)

    bow = np.zeros((len(newCorpus),len(features)), dtype=np.int8)

    for i, p in enumerate(newCorpus):
        for word in p.lower().split():
            occurance = np.where(features == word)
            bow[i][occurance] += 1

    print(bow)
    return bow

bag_of_words(corpus)
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

```
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
       [0, 2, 0, 1, 0, 1, 1, 0, 1],
       [1, 0, 0, 1, 1, 0, 1, 1, 1],
       [0, 1, 1, 1, 0, 0, 1, 0, 1]], dtype=int8)
```