Lab 01 - Linear Algebra and Numpy

Stuart Harley

This lab is designed as an introduction back into Jupyter, Numpy, Linear Algebra, matplotlib, etc. In this lab we primarily follow along with the notebook stub that we were provided. I also completed this lab using Rosie as was instructed.

Welcome to the first lab in CS3400! If you can edit this you probably have a working instance of jupyter notebook (either locally or on ROSIE). If you are looking at this as an pdf, maybe you still need to get an instance of Jupyter running. Please follow the neccessary steps in Experiment 1.

Experiment 1

In this experiment you will be making sure that you can connect to ROSIE and run an interactive session (jupyter notebook session). You should have a username on ROSIE by the start of class, but you might have to reset your password. At the current time, to do this you will have to access the terminal on ROSIE - meaning you will have to ssh in. Once you have reset your password, you will be able to access ROSIE's web portal and initiate interactive session from there. The following steps and sections will give you what you need to start.

Accessing ROSIE

An objective of this class is to give you some more experinece using remote resources and ROSIE is a great resource to have. To help support our use of ROSIE we have a High Performance Computing (HPC) Admininstrator. Our admin is Gagan Daroach (daroachgb@msoe.edu). While Gagan is also a great resouce to have, your first stop should always be ROSIE's webpage (www.msoe.dev).

SSH Client

If you are on windows, you will have to download and install an ssh client. A common and free client is Putty (https://www.putty.org/). Please follow the link and install Putty on your machine.

On network or off

If you are doing these steps off-campus, you will need to use a VPN to access the network that ROSIE is on. To do this you can follow the written instruction on msoe.dev (https://gagandaroach.github.io/rosie//#web-access).

Starting an Interactive session

Once you have access to ROSIE's network (VPN) and you have a username and current password (done through the SSH client), you can now complete the steps for starting an interactive session. You should access ROSIE's web portal (http://dh-ood.hpc.msoe.edu/) and start a jupyter notebook session to run (and complete) this notebook.

Experiment 2 - Structuing your Data and Feature Matrices / Slicing

In this experiment you will refamiliarize yourself with python/numpy and use some of the common data manipulation techniques that you will need for the rest of the class.

What is Numpy?

- Matrix library
- Memory-efficient data structures -- arrays
  - Used in scikit-learn, matplotlib, and others
- Expressive API for indexing and operations
- Time-efficient algorithms
  - Calls C and Fortran libraries where possible

How Do I Import Libraries into my Jupyter Notebook working kernel?

- The following bit of code can be used to import libraries. The world is your oyster!

In [1]:
```
import numpy as np
import scipy
import scipy.stats as stats
import matplotlib.pyplot as plt
from IPython.display import Image
picturename = '/data/cs3400/misc/mb.gif'
datapath = '/data/cs3400/datasets/IRIS.csv'
```

How to read in files, organize data, and plot some features!

In the first step you will read the IRIS.csv file that you are given (which is also on our class's datashare on ROSIE) and put the features into a matrix. In machine learning the standard for organizing matrices is always observations in rows, and features that describe the observations as columns. Read in the data file and assign the data to a numpy matrix.

1. Use the function numpy.loadtxt.
   - You will want to use the proper delimiter for the file you have.
   - Make sure that you skip any text rows, numpy matrices can only be a single datatype.
   - Depending on the dataset you may need to specify what columns you want to use.
   - If you get stuck and don't want to head to the web, you can always use the the help command for more information e.g. `help(np.loadtext)`

With your data matrix you should explore the data a bit.

1. Use `data.shape` to find your dimensions
2. Plot the first two features your data using matplotlib. Label all of your axes and use legends!
   A. Make a figure with a line plot
   B. Make another figure with a scatter plot
   C. Make a third figure displaying both the same line and scatter plots.
3. Print all of the feature values for the 150th observation in your dataset.
4. Select observations 49-52 from your dataset and print them to the notebook.
5. Select all of the entries in your dataset that have their first feature <= 5 and print the first 5 results. (hint: do this in multiple steps. First make a boolean mask of your matrix)
6. Calculate the median, standard deviation, and mode of the entries selected in the previous step. (Hint 1: these should be done column by column. Hint 2: Don't forget about other packages like scipy!)

1) Load the IRIS.csv file into a numpy matrix

```
In [2]: data = np.loadtxt(datapath, delimiter=',', skiprows=1, usecols=(0,1,2,3))
```

2) Display its dimensions (`data.shape`)

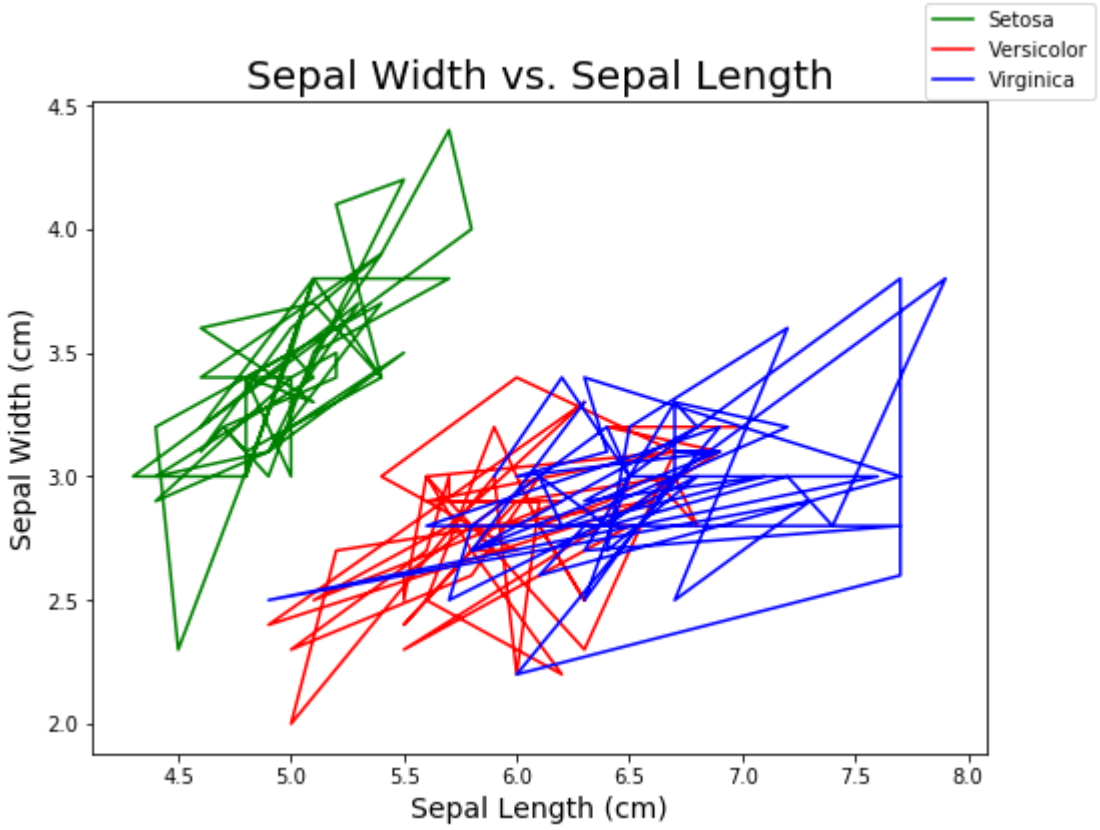
```
In [3]: data.shape
Out[3]: (150, 4)
```

Plot the first two features of your data using matplotlib. Label all of your axes and use legends!

3-A) Make a line plot of the first two dimensions using matplotlib

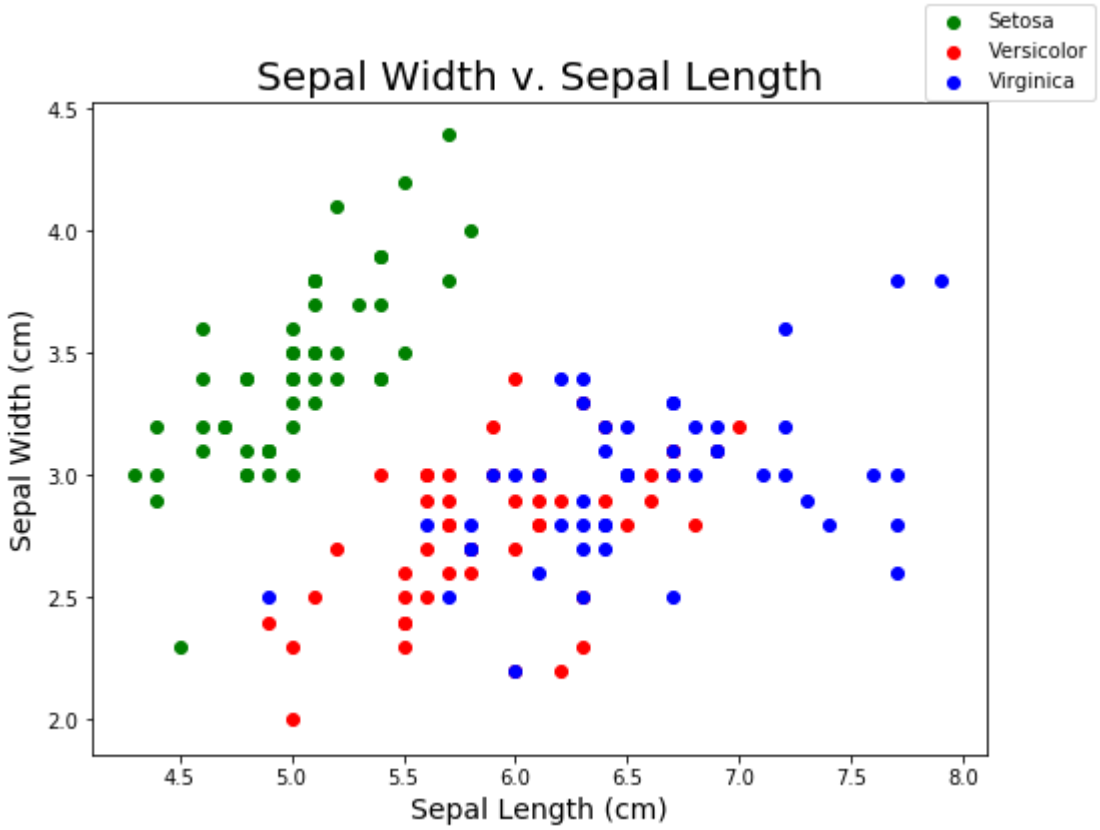
In [4]:
```
x = data[:,0]
y = data[:,1]
x_setosa = x[0:50]
y_setosa = y[0:50]
x_versicolor = x[50:100]
y_versicolor = y[50:100]
x_virginica = x[100:150]
y_virginica = y[100:150]

line_plot, line_axes = plt.subplots(figsize=(8, 6))
line_axes.plot(x_setosa, y_setosa, label='Setosa', color='g')
line_axes.plot(x_versicolor, y_versicolor, label='Versicolor', color='r')
line_axes.plot(x_virginica, y_virginica, label='Virginica', color='b')
line_axes.set_xlabel('Sepal Length (cm)', fontsize=14)
line_axes.set_ylabel('Sepal Width (cm)', fontsize=14)
line_axes.set_title('Sepal Width vs. Sepal Length', fontsize=20)
line_plot.legend();
```



3-B) Make a scatter plot of the first two dimensions using matplot lib

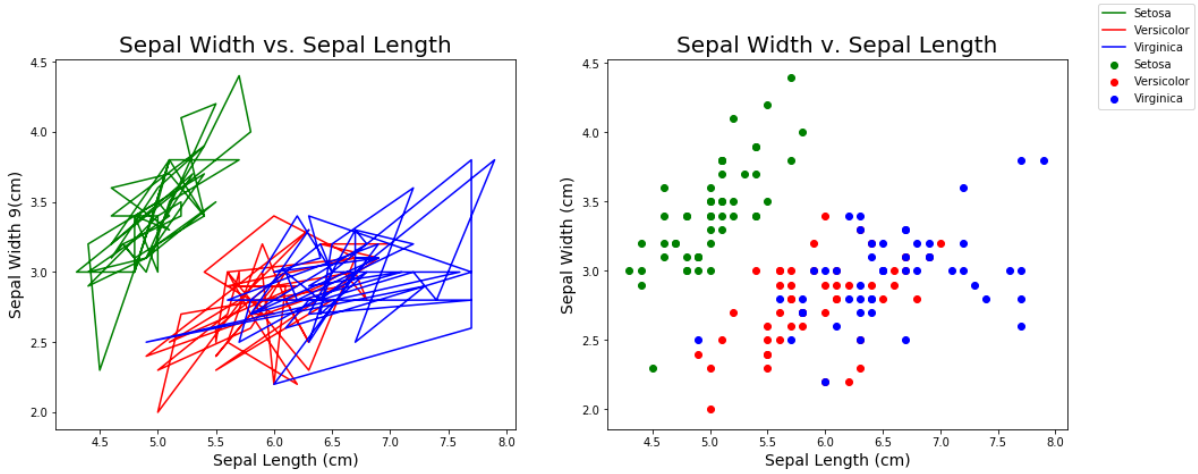
In [5]:
```
scatter_plot, scatter_axes = plt.subplots(figsize=(8, 6))
scatter_axes.scatter(x_setosa, y_setosa, label='Setosa', color='g')
scatter_axes.scatter(x_versicolor, y_versicolor, label='Versicolor', color='r'
)
scatter_axes.scatter(x_virginica, y_virginica, label='Virginica', color='b')
scatter_axes.set_xlabel('Sepal Length (cm)', fontsize=14)
scatter_axes.set_ylabel('Sepal Width (cm)', fontsize=14)
scatter_axes.set_title('Sepal Width v. Sepal Length', fontsize=20)
scatter_plot.legend();
```



3-C) Make a third figure displaying both the same line and scatter plots

In [6]:
```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
axes[0].plot(x_setosa, y_setosa, label='Setosa', color='g')
axes[0].plot(x_versicolor, y_versicolor, label='Versicolor', color='r')
axes[0].plot(x_virginica, y_virginica, label='Virginica', color='b')
axes[0].set_xlabel('Sepal Length (cm)', fontsize=14)
axes[0].set_ylabel('Sepal Width 9(cm)', fontsize=14)
axes[0].set_title('Sepal Width vs. Sepal Length', fontsize=20)

axes[1].scatter(x_setosa, y_setosa, label='Setosa', color='g')
axes[1].scatter(x_versicolor, y_versicolor, label='Versicolor', color='r')
axes[1].scatter(x_virginica, y_virginica, label='Virginica', color='b')
axes[1].set_xlabel('Sepal Length (cm)', fontsize=14)
axes[1].set_ylabel('Sepal Width (cm)', fontsize=14)
axes[1].set_title('Sepal Width v. Sepal Length', fontsize=20)
fig.legend();
```



4) Print all of the feature values for the 150th observation in your dataset.

In [7]:
```
print('For the 150th entry in the dataset...')
print('Sepal Length: ' + str(data[149, 0]))
print('Sepal Width: ' + str(data[149, 1]))
print('Petal Length: ' + str(data[149, 2]))
print('Petal Width: ' + str(data[149, 3]))
```

```
For the 150th entry in the dataset...
Sepal Length: 5.9
Sepal Width: 3.0
Petal Length: 5.1
Petal Width: 1.8
```

5) Select observations 49-52 from your dataset and print them to the notebook.

In [8]:
```
print(data[48:52, 0:4])
```

```
[[5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]]
```

6) Select all of the entries in your dataset that have their first feature <= 5 and print the first 5 results (Hint: Do this in multiple steps. First make a boolean mask of your matrix)

In [9]:
```
mask = data[:,0] <= 5
indexes = np.where(mask)[0]
num_true = np.count_nonzero(mask)
less_than_5 = np.empty((num_true,4))
for x in range(num_true):
    less_than_5[x] = data[indexes[x]]
print(less_than_5[0:5])
```

```
[[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [4.6 3.4 1.4 0.3]]
```

7) Calculate the median, standard deviation, and mode of the entries selected in the previous step. (Hint 1: these should be done column by column. Hint 2: Don't forget about other packages like scipy!)

In [10]:
```
print("For flowers whose first feature was <= 5...")
print("Median Sepal Length: " + str(np.median(less_than_5[:,0])))
print("Median Sepal Width: " + str(np.median(less_than_5[:,1])))
print("Median Petal Length: " + str(np.median(less_than_5[:,2])))
print("Median Petal Width: " + str(np.median(less_than_5[:,3])))
print("St. dev Sepal Length: " + str(np.std(less_than_5[:,0])))
print("St. dev Sepal Width: " + str(np.std(less_than_5[:,1])))
print("St. dev Petal Length: " + str(np.std(less_than_5[:,2])))
print("St. dev Petal Width: " + str(np.std(less_than_5[:,3])))
print("Mode Sepal Length: " + str(stats.mode(less_than_5[:,0])))
print("Mode Sepal Width: " + str(stats.mode(less_than_5[:,1])))
print("Mode Petal Length: " + str(stats.mode(less_than_5[:,2])))
print("Mode Petal Width: " + str(stats.mode(less_than_5[:,3])))
```

```
For flowers whose first feature was <= 5...
Median Sepal Length: 4.85
Median Sepal Width: 3.1
Median Petal Length: 1.45
Median Petal Width: 0.2
St. dev Sepal Length: 0.21323402636539976
St. dev Sepal Width: 0.38649062084350766
St. dev Petal Length: 0.7729559738432714
St. dev Petal Width: 0.3470860844228705
Mode Sepal Length: ModeResult(mode=array([5.]), count=array([10]))
Mode Sepal Width: ModeResult(mode=array([3.]), count=array([6]))
Mode Petal Length: ModeResult(mode=array([1.4]), count=array([8]))
Mode Petal Width: ModeResult(mode=array([0.2]), count=array([17]))
```

Experiment 3 - Linear Algebra in Numpy

In this experiment you will be performing a number of linear algebra operations in your jupyter notebook. Don't forget about the linalg module of numpy!

We have started by creating a few vectors and matrices for you.

In [11]:
```
array_1 = np.array([1, 2, 3, 4, 5], dtype=np.float32)
print(array_1)
array_2 = np.zeros(4, dtype=np.int32)
print(array_2)
matrix_1 = np.ones((4,5), dtype=np.float64)
print(matrix_1)
matrix_2 = np.eye(5,5)
print(matrix_2)
```

```
[1. 2. 3. 4. 5.]
[0 0 0 0]
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

You will:

1. Create a few more numpy vectors and matrices
2. Print the number of dimensions each of your numpy vectors and matrices
3. Print the shape (length and dimension) of each of your numpy vectors and matrices
4. Print the datatype used in each of your numpy vectors and matrices
5. Try to compute a dot product on two matrices of with disagreeable dimensions
6. Compute a dot product on two matrices with agreeable dimensions
7. Try to compute element-wise addition on two matrices with disagreeable dimensions
8. Compute an element-wise addition on two matrices with agreeable dimensions
9. Compute the norm (distance) between a vector and itself
10. Compute the norm (distance) between two different vectors
11. Apply a set of linear coefficients to a matrix of observations.

1) Create numpy vectors and matrices (we have done a few for you)

In [12]:
```
array_3 = np.array([10, 10, 10, 10, 10], dtype=np.int8)
matrix_3 = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
matrix_4 = np.array([[2, 4, 6],[8, 10, 12],[14, 16, 18]], dtype=np.int16)
```

2) Print the number of dimensions each of your numpy vectors and matrices

In [13]:
```
print(array_3.ndim)
print(matrix_3.ndim)
print(matrix_4.ndim)
```

```
1
2
2
```

3. Print the shape (length and dimension) of each of your numpy vectors and matrices

In [14]:
```
print(array_3.shape)
print(matrix_3.shape)
print(matrix_4.shape)
```

```
(5,)
(3, 3)
(3, 3)
```

4) Print the datatype used in each of your numpy vectors and matrices

In [15]:
```
print(array_3.dtype)
print(matrix_3.dtype)
print(matrix_4.dtype)
```

```
int8
int64
int16
```

5) Try to compute a dot product on two matrices with disagreeable dimensions

In [16]:
```
np.dot(array_2, array_3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-edb4f0ac12ed> in <module>
----> 1 np.dot(array_2, array_3)

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (4,) and (5,) not aligned: 4 (dim 0) != 5 (dim 0)
```

6) Compute a dot product on two matrices with agreeable dimensions

In [17]:
```
np.dot(array_1, array_3)
```

Out[17]:
```
150.0
```

7) Try to compute element-wise addition on two matrices with disagreeable dimensions

In [18]:
```
np.add(array_1, matrix_3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-4b1c6516b2b8> in <module>
----> 1 np.add(array_1, matrix_3)

ValueError: operands could not be broadcast together with shapes (5,) (3,3)
```

8) Compute an element-wise addition on two matrices with agreeable dimensions

In [19]:
```
np.add(matrix_3, matrix_4)
```

Out[19]:
```
array([[ 3,  6,  9],
       [12, 15, 18],
       [21, 24, 27]])
```

9) Compute the norm (distance) between a vector and itself

In [20]:
```
np.linalg.norm(array_3)
```

Out[20]:
```
22.360679774997898
```

10) Compute the norm (distance) between two different vectors

In [21]:
```
np.linalg.norm(array_1-array_3)
```

Out[21]:
```
15.9687195
```

11) Apply a set of linear coefficients to a matrix of observations.

From your problem set you can see the form of this model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

which can also be represented in vector notation as:

$$y = x^T \beta$$

Use the vectors that you created in problem 5 of problem set 1 and evaluate it here. Evaluate it twice, once using matrix multiplication and once with dot products

In [22]:
```
print('The vectors from problem 5 were x = [1, x1, x2, x3] and B = [B0, B1, B
2, B3].')
print('For x I will use the vector [1, 2, 3, 4] and for B I will use [2, 4, 6,
8].')
x = np.array([1, 2, 3, 4])
B = np.array([2, 4, 6, 8])
print('Dot product value: ' + str(np.dot(B, x)))
print('Matrix multiplication value: ' + str(np.matmul(x.T, B)))
print('As expected, the result from both calculations was 60')
```

```
The vectors from problem 5 were x = [1, x1, x2, x3] and B = [B0, B1, B2, B3].
For x I will use the vector [1, 2, 3, 4] and for B I will use [2, 4, 6, 8].
Dot product value: 60
Matrix multiplication value: 60
As expected, the result from both calculations was 60
```

Bonus Material: Additional Indexing Topics

Before considering the following indexing procedures, think about the following question. Can I index a vector (nx1) using a matrix (nxm)? What would happen if I try?

In [23]:
```
X = np.random.randint(10, size=(10, 3))
y = np.expand_dims(np.array([1, 0, 1, 1, 0, 0, 2, 2, 1, 0], dtype=np.int32),ax
is=1)
```

Think of the above matrix, X, as a feature matrix (10x3) and the above vector, y, as a response vector/matrix (10x1). How can I index and get the first index of X or y?

In [24]:
```
y[0,0]
```

Out[24]: 1

In [25]:
```
X[0,0]
```

Out[25]: 6

What if I want multiple elements from this array that are not sequential? Such as element 0 and element 7?

In [26]:
```
print(y[0,0])
print(y[7,0])
```

```
1
2
```

Pretty straighforward, eh? Can I do this in one go?

In [27]:
```
print(y[[0,7],[0,0]])
```

```
[1 2]
```

Not too shabby! Now, is there anything preventing me from re-indexing the same element? Let's try!

In [28]:
```
print(y[[7,7],[0,0]])
```

```
[2 2]
```

woah

Finally, lets take this to a rediculous conclusion... What happens if I supply more index calls (as a matrix) than the variable has in shape?

In [29]:
```
print(y[[7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
]])
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

In [30]:
```
Image(picturename)
```

Out[30]:
```
<IPython.core.display.Image object>
```