In [11]:

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

Generate randomly distributed (x, y) pairs. Then, generate a copy of this data and shift each element by a random amount.

In [12]:

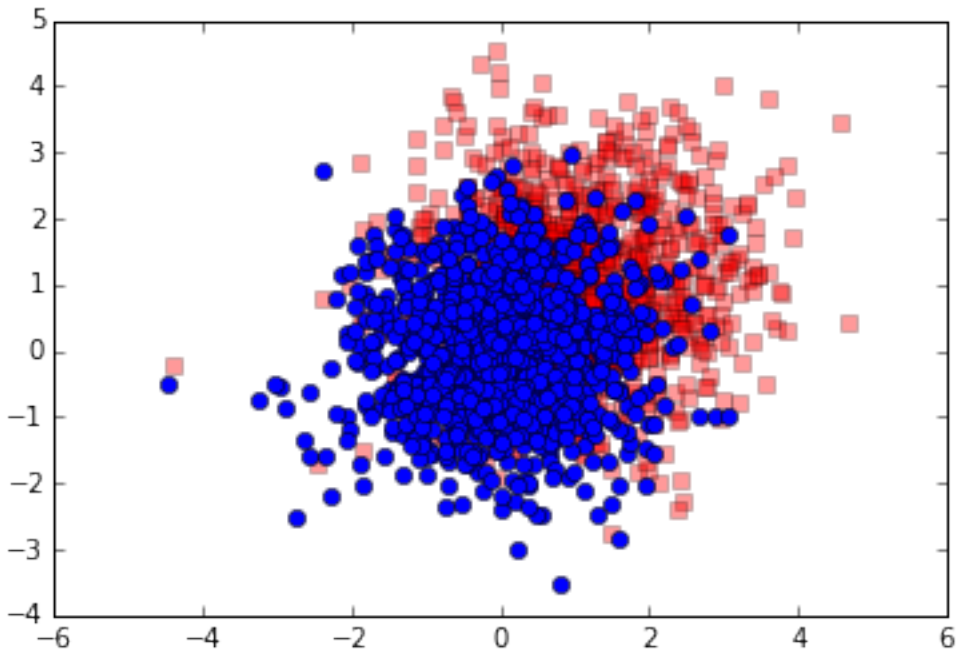
```
xy = np.random.randn(1000,2)
xshift = np.zeros(xy.shape)
xshift = 2*np.random.rand(xshift.shape[0],2)
xyhat = xy + xshift
```

In [13]:

```
plt.plot(xyhat[:,0], xyhat[:,1], 'rs', alpha=0.4)
plt.plot(xy[:,0], xy[:,1], 'o')
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x7f3b584eacf8>]
```



Find the average (scalar) distance that each point moved. To do so, calculate the (vector) differences between points in the original data and corresponding points in the shifted data, and then take the two-norms of those differences. Finally, take the average over all of the two-norms.

This is what it looks like if we loop through each point individually:

In [14]:

```
n = xy.shape[0]

m1 = 0.0
for i in range(n):
    m1 += np.sqrt((xy[i,0]-xyhat[i,0])**2 + (xy[i,1]-xyhat[i,1])**2)

print(m1/n)
```

```
1.54746284303
```

Since we are working with arrays, we can use numpy to be more efficient. Notice that this code gives the exact same answer:

In [15]:

```
err = (xy-xyhat)**2
err = err.sum(axis=1)
err = np.sqrt(err)
m1 = err.sum()
print(m1/n)
```

```
1.54746284303
```

The code can be written more compactly using just one line.

In [16]:

```
err = np.sqrt(((xy - xyhat)**2).sum(axis=1)).sum()/n
print(err)
```

```
1.54746284303
```

What happens if we just take the norm of the entire difference vector? Is it the same as what we did above?

In [17]:

```
print(np.linalg.norm(xy-xyhat))
```

```
52.2236429011
```

We can also pass an argument to the norm function to get a different norm.

In [18]:

```
dist = np.sqrt(((xy - xyhat)**2).sum(axis=1))
print(np.linalg.norm(dist, ord=np.inf))
```

```
2.76144343403
```

Examine the documentation for numpy's norm routine.

In [19]:

```
np.linalg.norm?
```

In [ ]:

```

```