In [30]:

```
# The normal imports
import numpy as np
from numpy.random import randn
import pandas as pd

# Import the stats library from numpy
from scipy import stats

# These are the plotting modules adn libraries we'll use:
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# Command so that plots appear in the iPython Notebook
%matplotlib inline
```

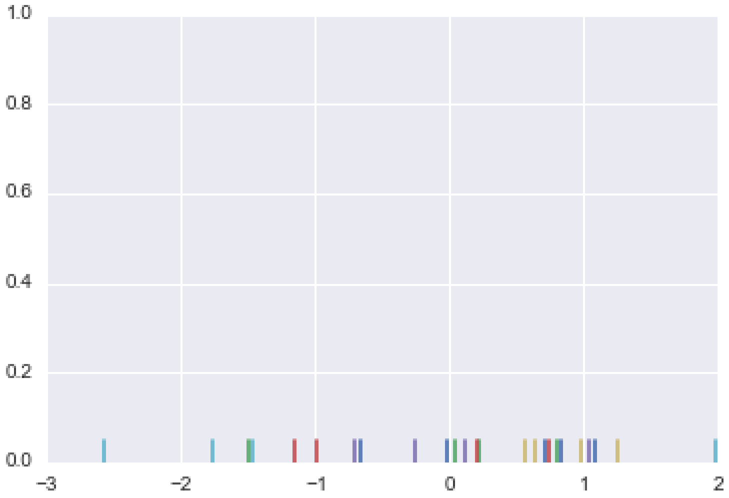
First we'll learn how to understand and make a KDE plot manually, and then we'll see how to do it quickly with seaborn!

In [31]:

```
# Let's start off with a carpet/rug plot
# A rug plot simpot puts ticks wherever a value occured

#Create dataset
dataset = randn(25)
#Create rugplot
sns.rugplot(dataset)
#Set y-axis limit
plt.ylim(0,1)
```

Out[31]:

```
(0, 1)
```

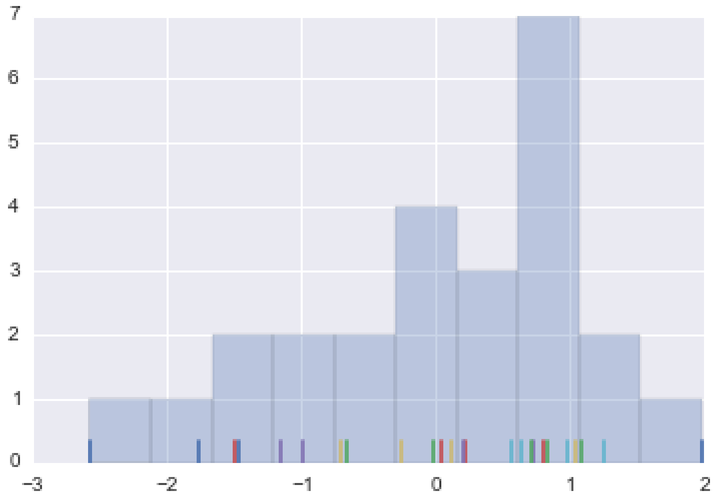


In [32]:

```
# Plot a histogram on top of
plt.hist(dataset,alpha=0.3)
sns.rugplot(dataset)
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x209891d0>
```



The histogram sets up 10 bins and then just count how many ticks appeared in each bin, setting the height of each bar

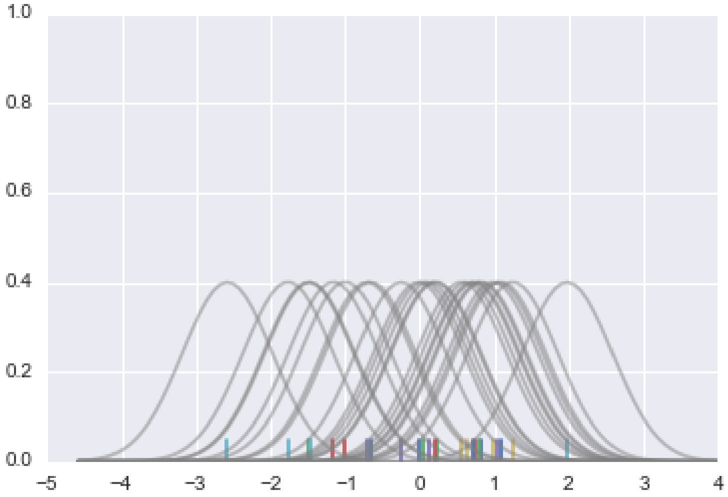
The kernel density plot will represent each tick mark with a gaussian basis function. Let's see how we would do this manually

In [39]:

```
# Create another rugplot
sns.rugplot(dataset);

# Set up the x-axis for the plot
x_min = dataset.min() - 2
x_max = dataset.max() + 2

# 100 equally spaced points from x_min to x_max
x_axis = np.linspace(x_min,x_max,100)

# Set up the bandwidth, for info on this:
url = 'http://en.wikipedia.org/wiki/Kernel_density_estimation#Practical_estimatio

bandwidth = ((4*dataset.std()**5)/(3*len(dataset)))**.2


# Create an empty kernel list
kernel_list = []

# Plot each basis function
for data_point in dataset:

    # Create a kernel for each point and append to list
    kernel = stats.norm(data_point,bandwidth).pdf(x_axis)
    kernel_list.append(kernel)

    #Scale for plotting
    kernel = kernel / kernel.max()
    kernel = kernel * .4
    plt.plot(x_axis,kernel,color = 'grey',alpha=0.5)

plt.ylim(0,1)
```

Out[39]:

```
(0, 1)
```

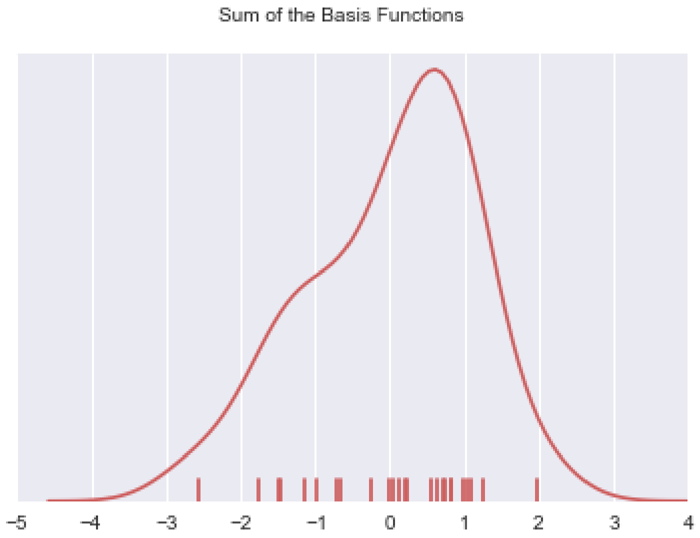


In [55]:

```
# To get the kde plot we can sum these basis functions.



# Plot the sum of the basis function
sum_of_kde = np.sum(kernels,axis=0)

# Plot figure
fig = plt.plot(x_axis,sum_of_kde,color='indianred')

# Add the initial rugplot
sns.rugplot(dataset,c = 'indianred')

# Get rid of y-tick marks
plt.yticks([])

# Set title
plt.suptitle("Sum of the Basis Functions")
```

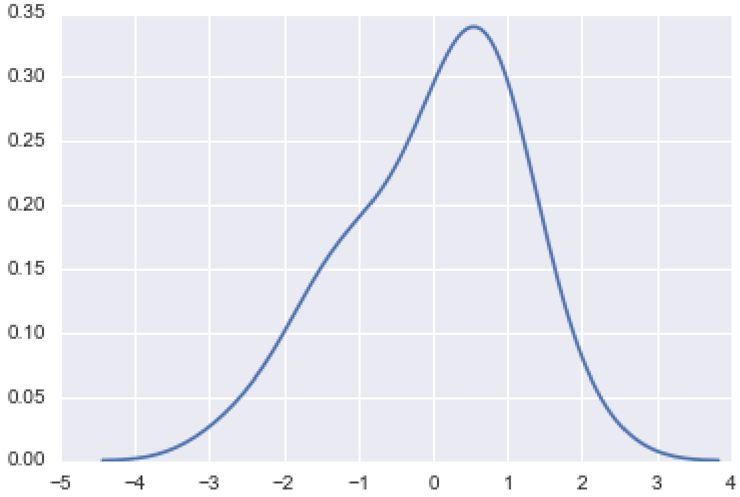
Out[55]:

```
<matplotlib.text.Text at 0x20e56a20>
```

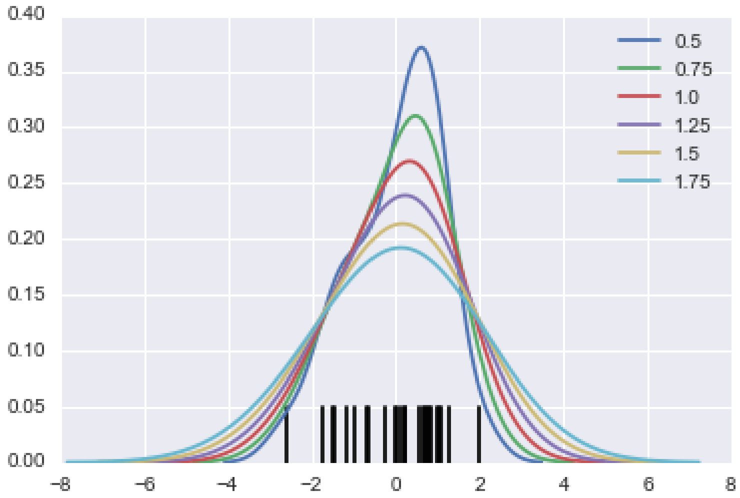


```
In [58]: # Now we can see how to do it in one step with seaborn! Awesome!
         sns.kdeplot(dataset)
```

```
Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x20d86e10>
```

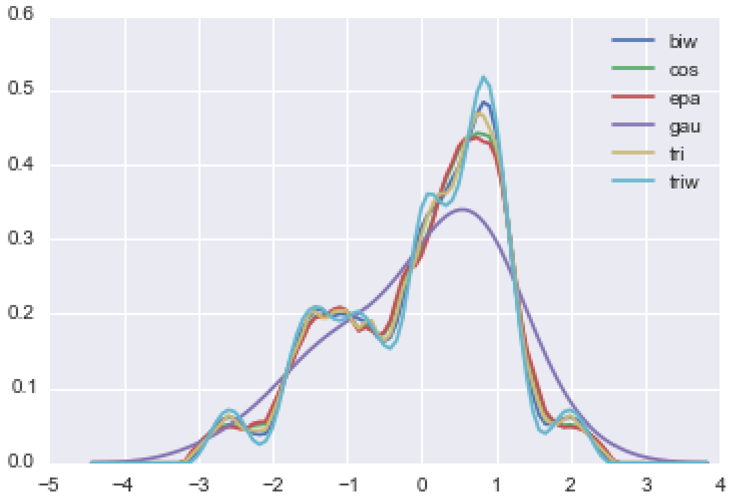


```
In [73]: # We can adjust the bandwidth of the sns kde to make the kde plot more or less se

         # Rugplot
         sns.rugplot(dataset,color='black')

         # Plot various bandwidths
         for bw in np.arange(0.5,2,0.25):
             sns.kdeplot(dataset,bw=bw,lw=1.8,label=bw)
```



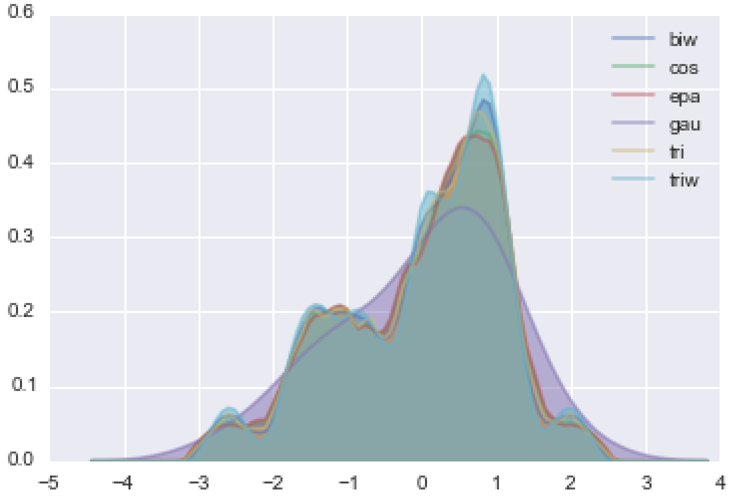
In [75]:

```
# We can also choose different kernels

kernel_options = ["biw", "cos", "epa", "gau", "tri", "triw"]

# More info on types
url = 'http://en.wikipedia.org/wiki/Kernel_(statistics)'

# Use label to set legend
for kern in kernel_options:
    sns.kdeplot(dataset,kernel=kern,label=kern)
```



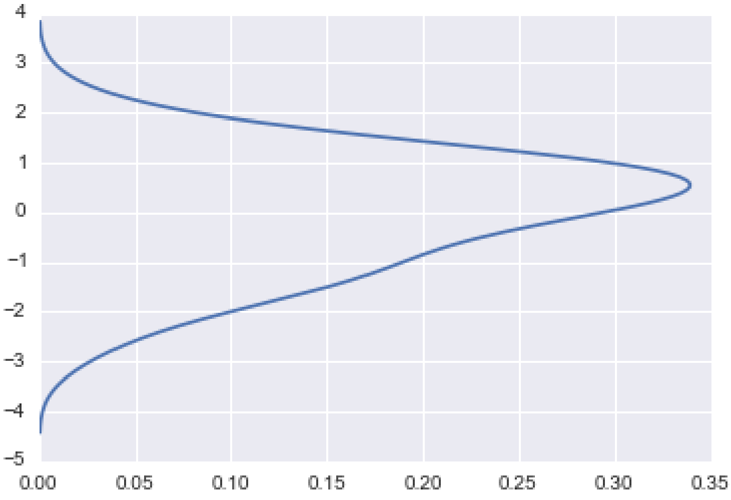
In [79]:

```
# We can also shade if desired
for kern in kernel_options:
    sns.kdeplot(dataset,kernel=kern,label=kern,shade=True,alpha=0.5)
```



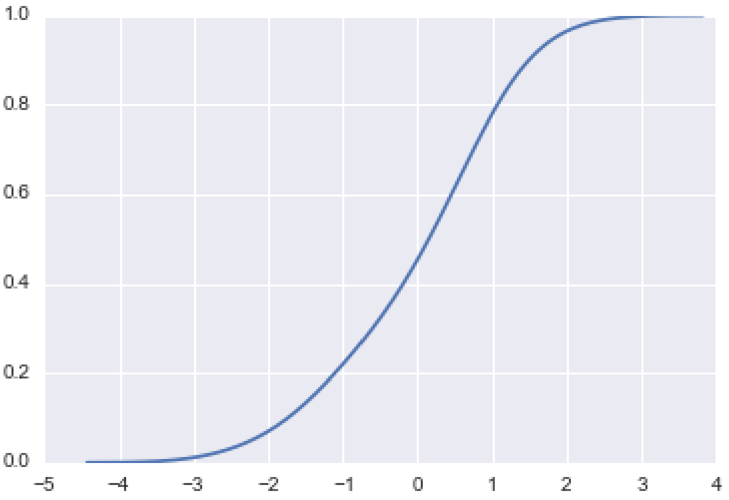
In [76]:
```
# For vertical axis, use the vertical keyword
sns.kdeplot(dataset,vertical=True)
```

Out[76]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21f660f0>
```



In [81]:
```
# Finally we can also use kde plot to create a cumulative distribution function (

# URL for info on CDF
url = 'http://en.wikipedia.org/wiki/Cumulative_distribution_function'

sns.kdeplot(dataset,cumulative=True)
```

Out[81]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2004f358>
```



Multivariate Density Estimation using kdeplot

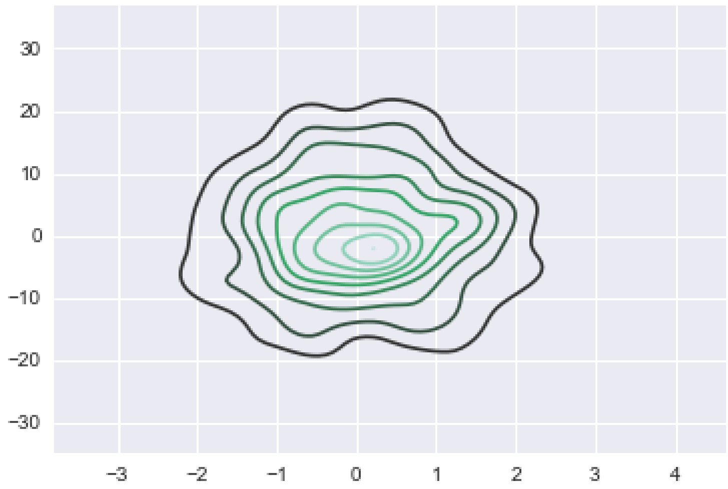
We can also use kdeplot for multidimensional data. Lets see how it works!

In [88]:

```
# Let's create a new dataset

# Mean center of data
mean = [0,0]

# Diagonal covariance
cov = [[1,0],[0,100]]

# Create dataset using numpy
dataset2 = np.random.multivariate_normal(mean,cov,1000)

# Bring back our old friend pandas
dframe = pd.DataFrame(dataset2,columns=['X','Y'])

# Plot our dataframe
sns.kdeplot(dframe)
```

Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20777240>
```

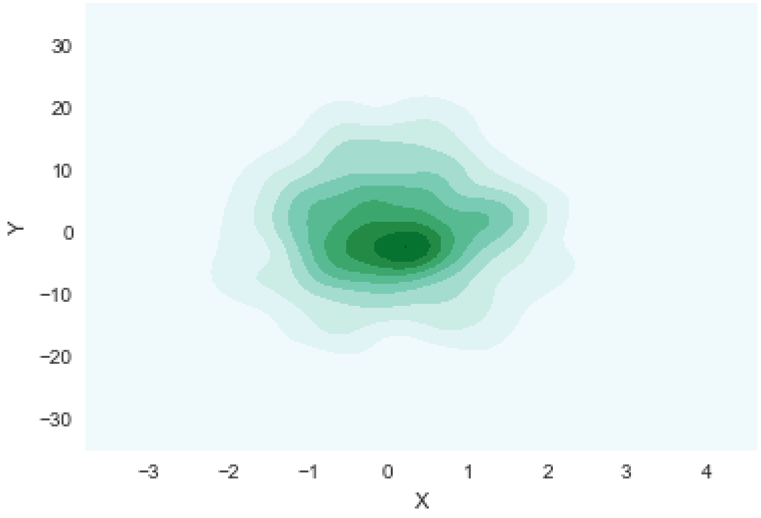


In [89]:

```
# We could have also passed two vectors seperately, and shade
sns.kdeplot(dframe.X,dframe.Y,shade=True)
```

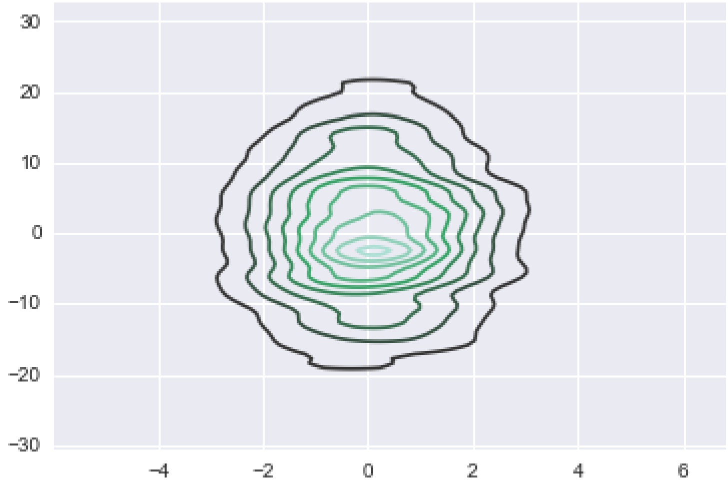
Out[89]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21708668>
```



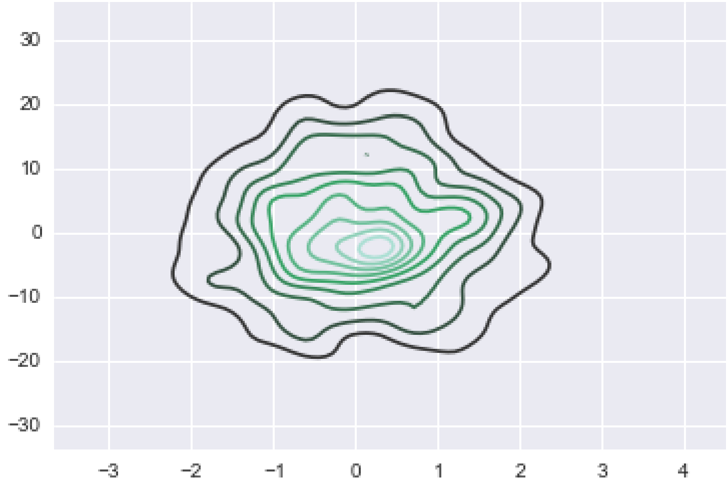
In [90]:
```
# Can specify a particualr bandwidth
ns.kdeplot(dframe,bw=1)
```

Out[90]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20b15c88>
```

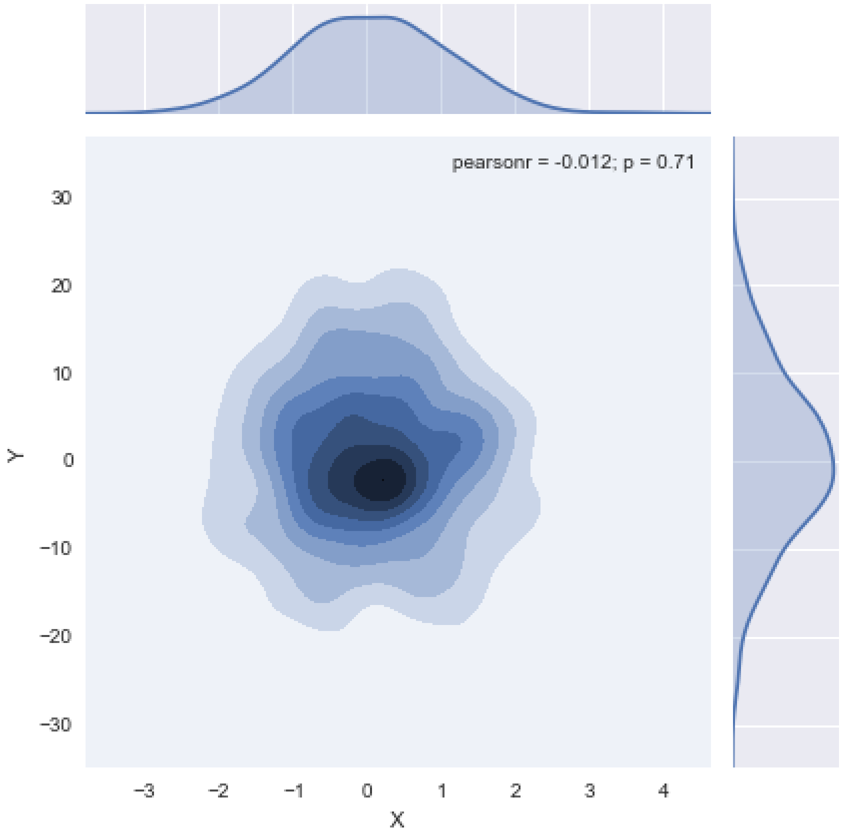


In [92]:
```
# Or just use silverman again
sns.kdeplot(dframe,bw='silverman')
```

Out[92]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f5d1fd0>
```



```
In [94]: # We can also create a kde joint plot, simliar to the hexbin plots we saw before

         sns.jointplot('X','Y',dframe,kind='kde')

Out[94]: <seaborn.axisgrid.JointGrid at 0x22115630>
```



```
In [97]: # Next up: Combingign plot styles using distplot!

In [ ]:
```