Data Visualization with Python

All Versions

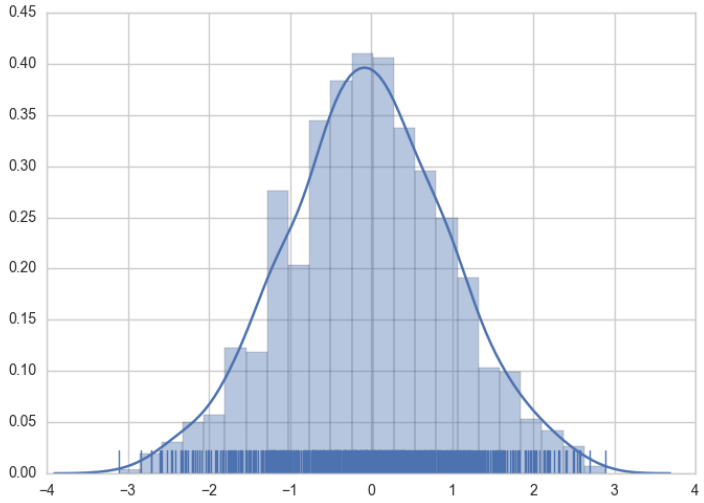
Examples

Seaborn

Seaborn is a wrapper around Matplotlib that makes creating common statistical plots easy. The list of supported plots includes univariate and bivariate distribution plots, regression plots, and a number of methods for plotting categorical variables. The full list of plots Seaborn provides is in their API reference .

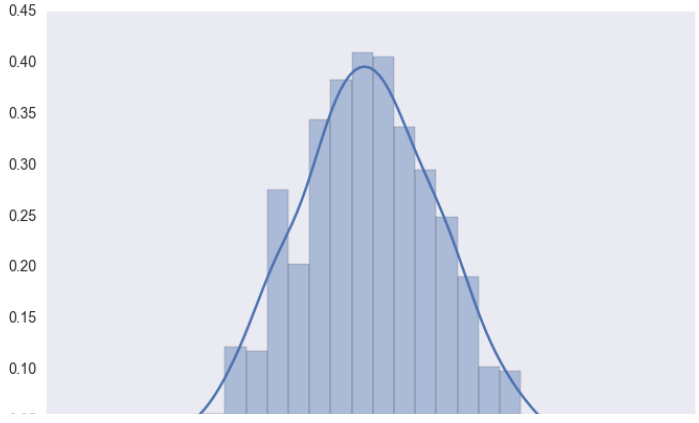
Creating graphs in Seaborn is as simple as calling the appropriate graphing function. Here is an example of creating a histogram, kernel density estimation, and rug plot for randomly generated data.

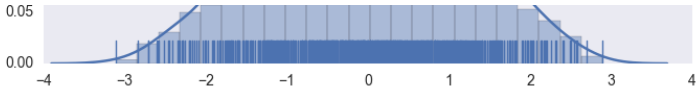
```
import numpy as np  # numpy used to create data from plotting
import seaborn as sns  # common form of importing seaborn

# Generate normally distributed data
data = np.random.randn(1000)

# Plot a histogram with both a rugplot and kde graph superimposed
sns.distplot(data, kde=True, rug=True)
```



The style of the plot can also be controled using a declarative syntax.

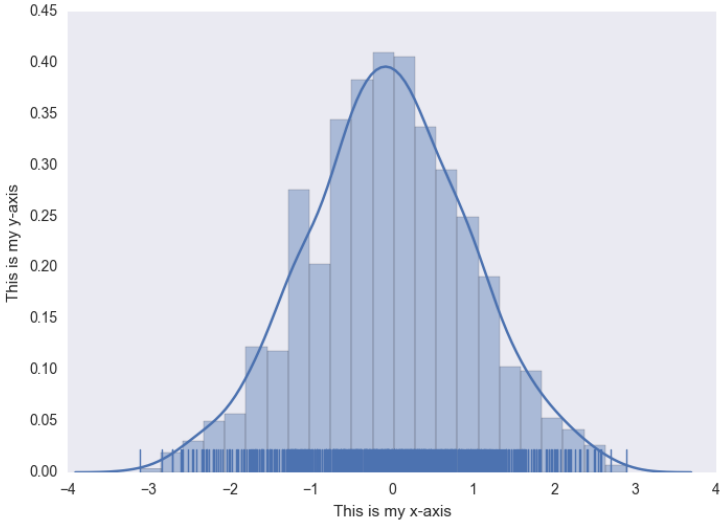
```
# Using previously created imports and data.

# Use a dark background with no grid.
sns.set_style('dark')
# Create the plot again
sns.distplot(data, kde=True, rug=True)
```





As an added bonus, normal matplotlib commands can still be applied to Seaborn plots. Here's an example of adding axis titles to our previously created histogram.

```
# Using previously created data and style

# Access to matplotlib commands
import matplotlib.pyplot as plt

# Previously created plot.
sns.distplot(data, kde=True, rug=True)
# Set the axis labels.
plt.xlabel('This is my x-axis')
plt.ylabel('This is my y-axis')
```



Matplotlib

Matplotlib is a mathematical plotting library for Python that provides a variety of different plotting functionality.

The matplotlib documentation can be found here , with the SO Docs being available here .

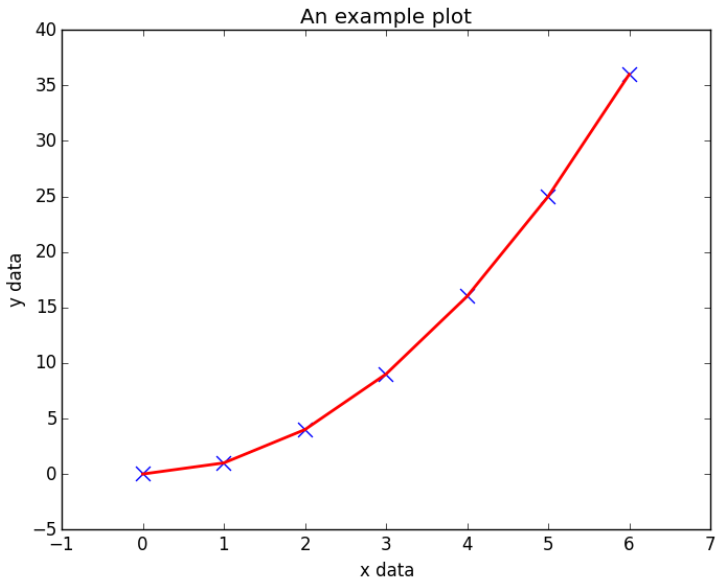
Matplotlib provides two distinct methods for plotting, though they are interchangable for the most part:

- Firstly, matplotlib provides the `pyplot` interface, direct and simple-to-use interface that allows plotting of complex graphs in a MATLAB-like style.
- Secondly, matplotlib allows the user to control the different aspects (axes, lines, ticks, etc) directly using an object-based system. This is more difficult but allows complete control over the entire plot.

Below is an example of using the `pyplot` interface to plot some generated data:

```
import matplotlib.pyplot as plt

# Generate some data for plotting.
x = [0, 1, 2, 3, 4, 5, 6]
y = [i**2 for i in x]

# Plot the data x, y with some keyword arguments that control the plot style.
# Use two different plot commands to plot both points (scatter) and a line (plot).

plt.scatter(x, y, c='blue', marker='x', s=100) # Create blue markers of shape "x" and size 100
plt.plot(x, y, color='red', linewidth=2) # Create a red line with linewidth 2.

# Add some text to the axes and a title.
plt.xlabel('x data')
plt.ylabel('y data')
plt.title('An example plot')

# Generate the plot and show to the user.
plt.show()
```

```
plt.show()
```



Note that `plt.show()` is known to be problematic in some environments due to running `matplotlib.pyplot` in interactive mode, and if so, the blocking behaviour can be overridden explicitly by passing in an optional argument, `plt.show(block=True)` , to alleviate the issue.

Syntax

Parameters

Remarks