第15章 生成数据

数据可视化指的是通过可视化表示来探索数据。 它与数据分析紧密相关，而数据分析指的是使用代码来探索数据集的规律和关联。 数据集可以是用一行代码就能表示的小型数字列表，也可以是数千兆字节的数据。

数据分析的目的是把隐藏在一大批看来杂乱无章的数据中的信息集中和提炼出来，从而找出所研究对象的内在规律。在实际应用中，数据分析可帮助人们做出判断，以便采取适当行动。数据分析是有组织有目的地收集数据、分析数据，使之成为信息的过程。

漂亮地呈现数据并非仅仅关乎漂亮的图片。 通过以引人注目的简单方式呈现数据，能让观看者明白其含义：发现数据集中原本未知的规律和意义。

所幸即便没有超级计算机，你也能够可视化复杂的数据。 鉴于Python的高效性，使用它在笔记本电脑上就能快速地探索由数百万个数据点组成的数据集。数据点并非必须是数。 利用本书前半部分介绍的基本知识，也可对非数值数据进行分析。

在基因研究、天气研究、政治经济分析等众多领域，人们常常使用Python来完成数据密集型工作。 数据科学家使用Python编写了一系列优秀的可视化和分析工具，其中很多可供你使用。 最流行的工具之一是Matplotlib，它是一个数学绘图库，我们将使用它来制作简单的图表，如折线图和散点图。 然后，我们将基于随机漫步概念生成一个更有趣的数据集——根据一系列随机决策生成的图表。

本章还将使用Plotly包，它生成的图表非常适合在数字设备上显示。 Plotly生成的图表可根据显示设备的尺寸自动调整大小，还具备众多交互特性，如在用户将鼠标指向图表的不同部分时突出数据集的特定方面。 本章将使用Plotly来分析掷骰子的结果。

15.1 安装Matplotlib

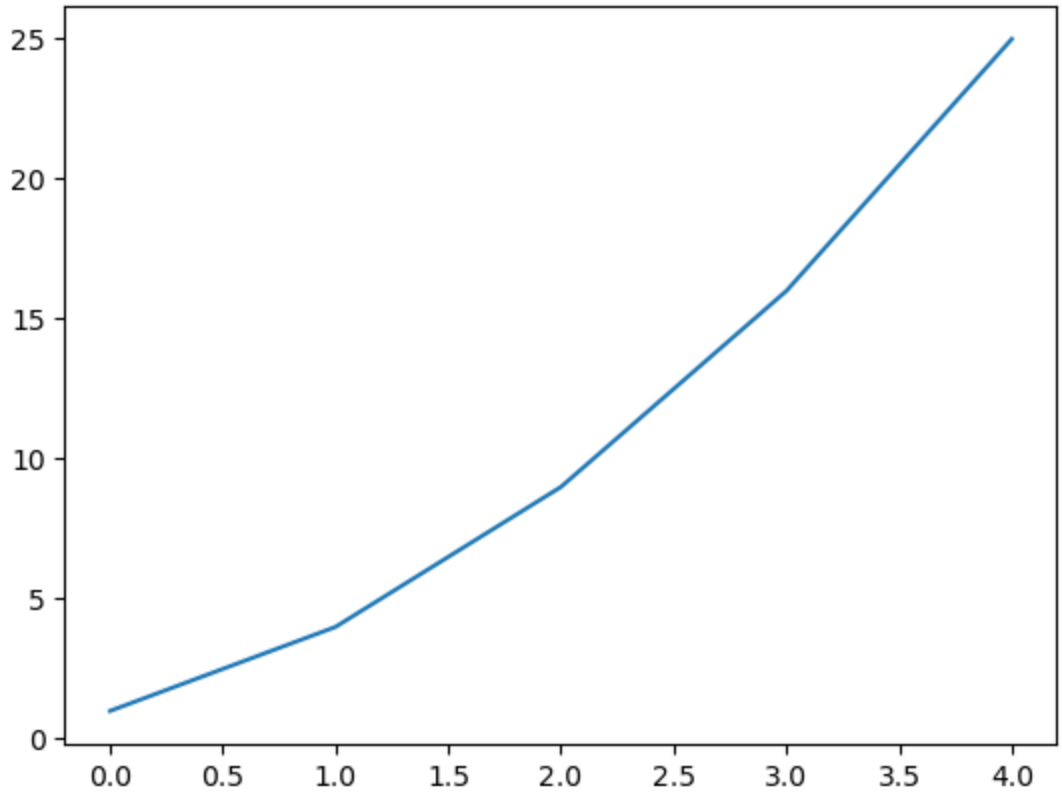
```
$pip install matplotlib
```

Matplotlib, Seaborn, Plotly, Dash

15.2 绘制简单的折线图

下面使用Matplotlib绘制一个简单的折线图，再对其进行定制，以实现信息更丰富的数据可视化效果。

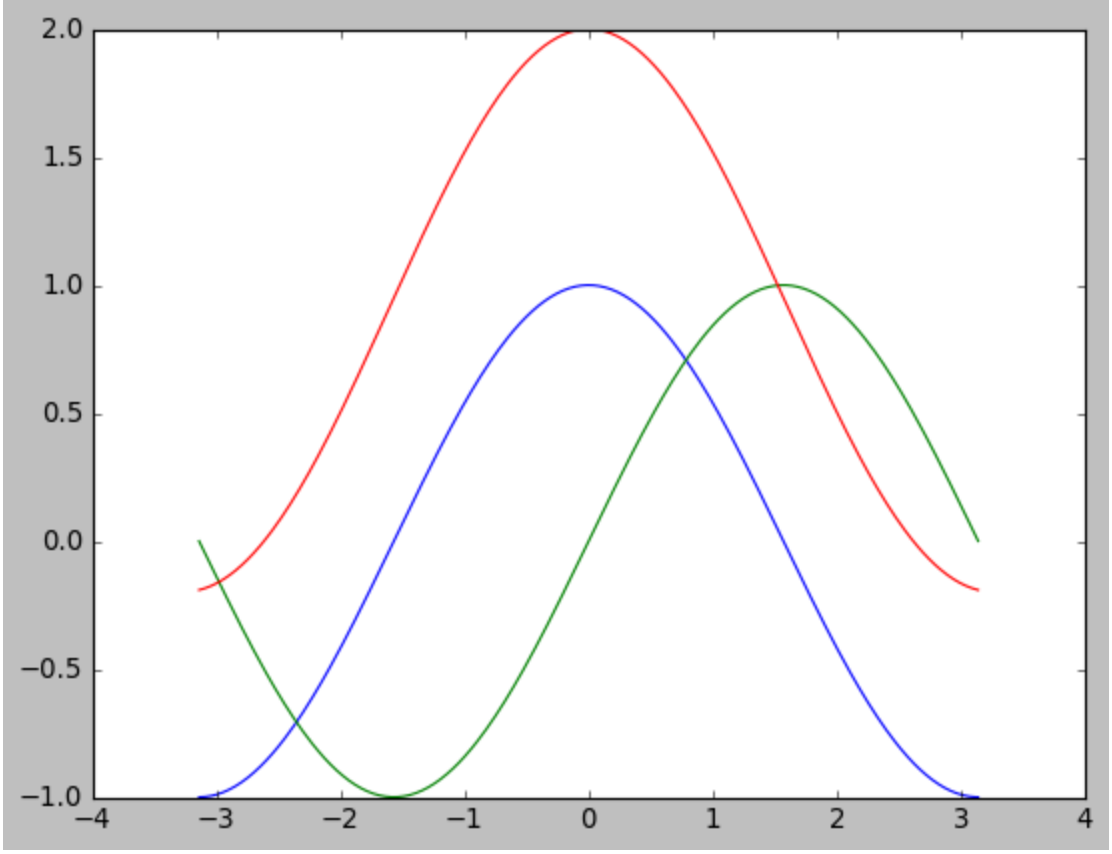
In [3]:
```
import matplotlib.pyplot as plt

squares = [1, 4, 9, 16, 25]
plt.plot(squares)
plt.show()
```



首先导入模块pyplot，并为其指定别名plt，以免反复输入pyplot。 模块pyplot包含很多用于生成图表的函数。

我们创建了一个名为squares的列表，在其中存储要用来制作图表的数据。 接下来调用方法plot()，它尝试根据给定的数据绘制图表。 函数plt.show()打开Matplotlib查看器并显示绘制的图表。

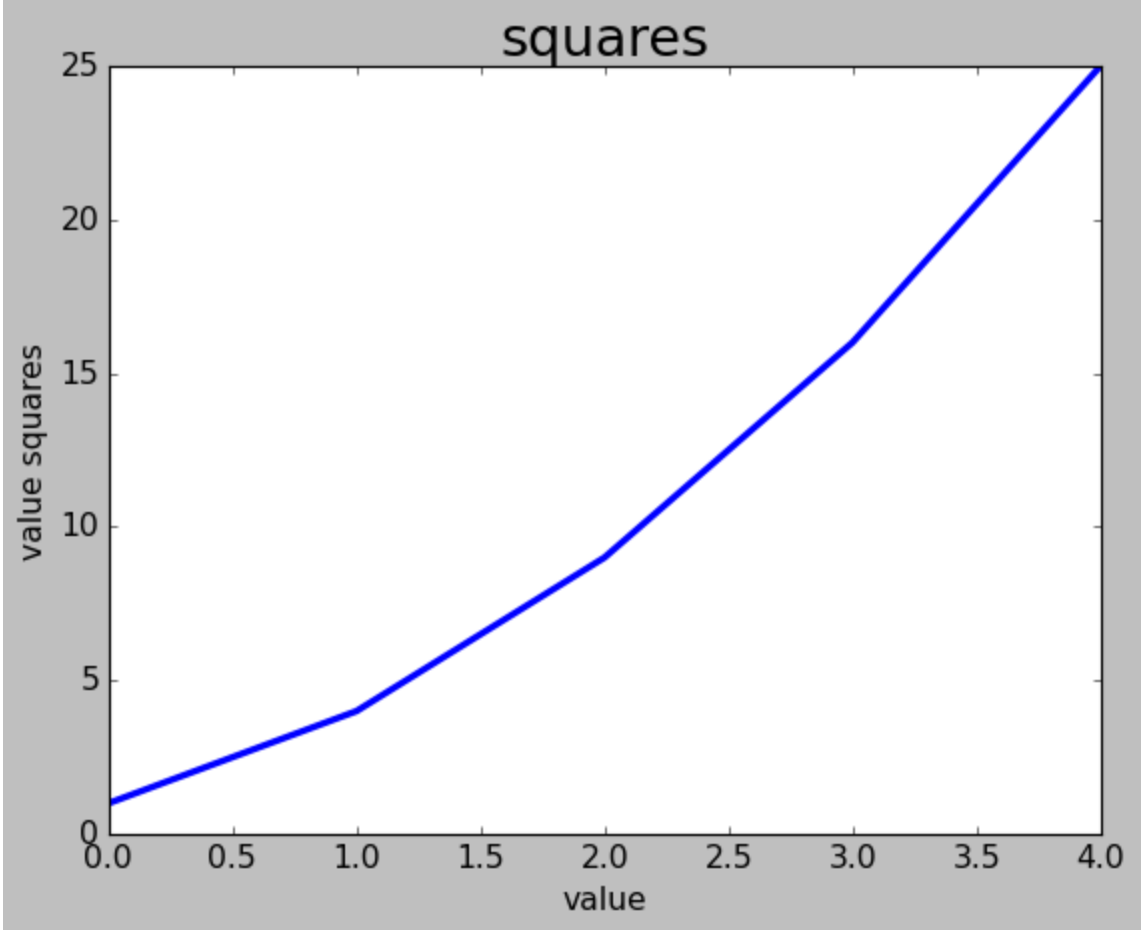
In [91]:
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C,S = np.cos(X), np.sin(X)

plt.plot(X,C)
plt.plot(X,S)
plt.plot(X,C+ np.cos(0.2*X))
plt.show()
```



15.2.1 修改标签文字和线条粗细

如图所示的图形标签文字太小、线条太细，难以看清楚。 所幸Matplotlib让你能够调整可视化的各个方面。下面通过一些定制来改善这个图表的可读性，如下所示：

In [92]:
```
import matplotlib.pyplot as plt
squares = [1, 4, 9, 16, 25]
fig, ax = plt.subplots()
ax.plot(squares, linewidth=3)

# 设置图表标题并给坐标轴加上标签。
ax.set_title("squares", fontsize=24)
ax.set_xlabel("value", fontsize=14)
ax.set_ylabel("value squares", fontsize=14)
# 设置刻度标记的大小2
ax.tick_params(axis='both', labelsize=14)
plt.show()
```

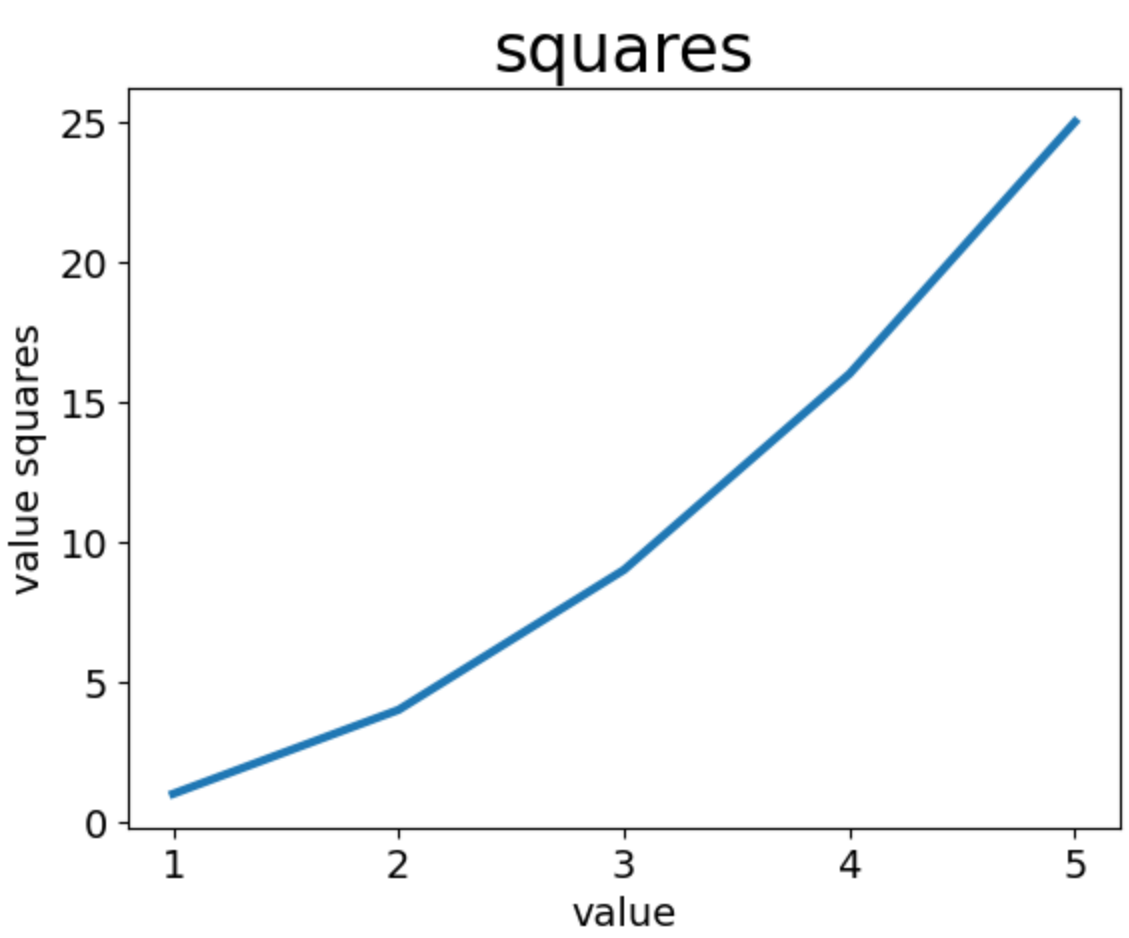


set_title()给图表指定标题 方法set_xlabel()和set_ylabel()让你能够为每条轴设置标题 方法tick_params() 设置刻度的样式,其中指定的实参将影响轴和轴上的刻度（axes='both' ），并将刻度标记的字号设置为14（labelsize=14 ）。

15.2.2 校正图形

图形更容易看清后，我们发现没有正确地绘制数据：折线图的终点指出4.0的平方为25！下面来修复这个问题。 向plot()提供一系列数时，它假设第一个数据点对应的坐标值为0，但这里第一个点对应的值为1。 为改变这种默认行为，可向plot()同时提供输入值和输出值：

In [6]:
```
input_values = [1, 2, 3, 4, 5]
squares = [1, 4, 9, 16, 25]
fig, ax = plt.subplots()
ax.plot(input_values, squares, linewidth=3)
ax.set_title("squares", fontsize=24)
ax.set_xlabel("value", fontsize=14)
ax.set_ylabel("value squares", fontsize=14)
ax.tick_params(axis='both', labelsize=14)
plt.show()
```



15.2.3 使用内置样式

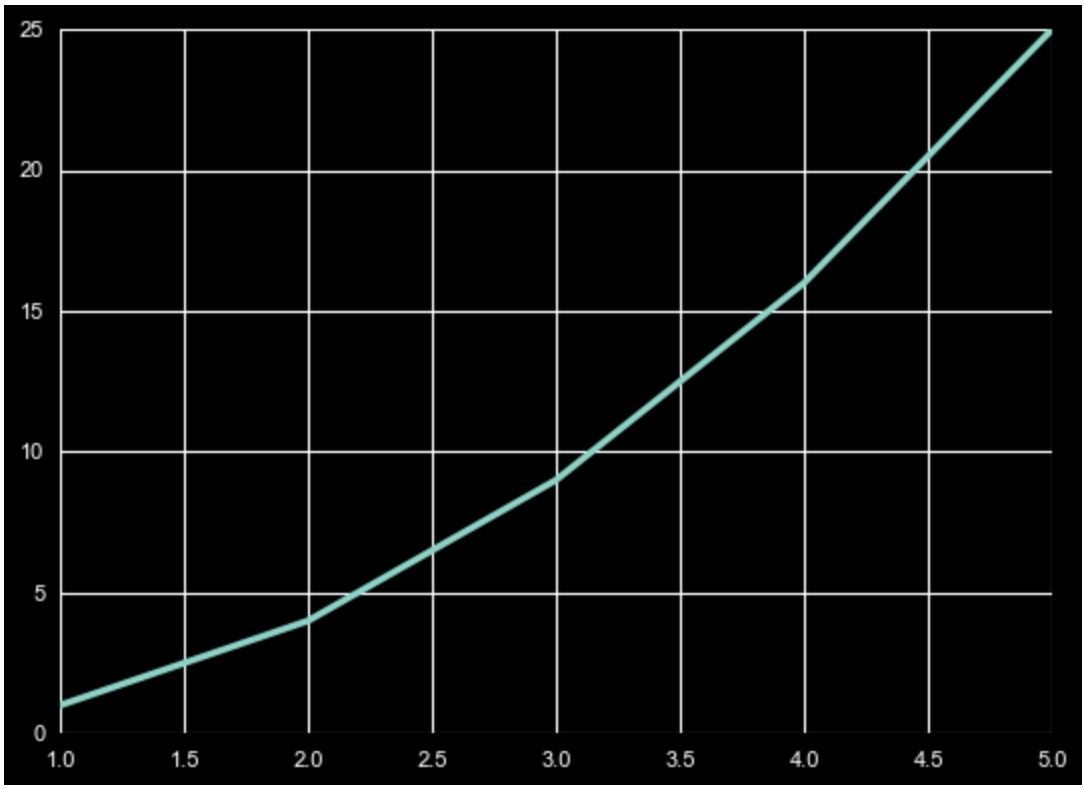
Matplotlib提供了很多已经定义好的样式，它们使用的背景色、网格线、线条粗细、字体、字号等设置很不错，让你无须做太多定制就可生成引人瞩目的可视化效果。 要获悉在你的系统中可使用哪些样式，可在终端会话中执行如下命令：

In [21]:
```
plt.style.available
```

Out[21]:
```
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-v0_8',
 'seaborn-v0_8-bright',
 'seaborn-v0_8-colorblind',
 'seaborn-v0_8-dark',
 'seaborn-v0_8-dark-palette',
 'seaborn-v0_8-darkgrid',
 'seaborn-v0_8-deep',
 'seaborn-v0_8-muted',
 'seaborn-v0_8-notebook',
 'seaborn-v0_8-paper',
 'seaborn-v0_8-pastel',
 'seaborn-v0_8-poster',
 'seaborn-v0_8-talk',
```

```
 'seaborn-v0_8-ticks',
 'seaborn-v0_8-white',
 'seaborn-v0_8-whitegrid',
 'tableau-colorblind10']
```

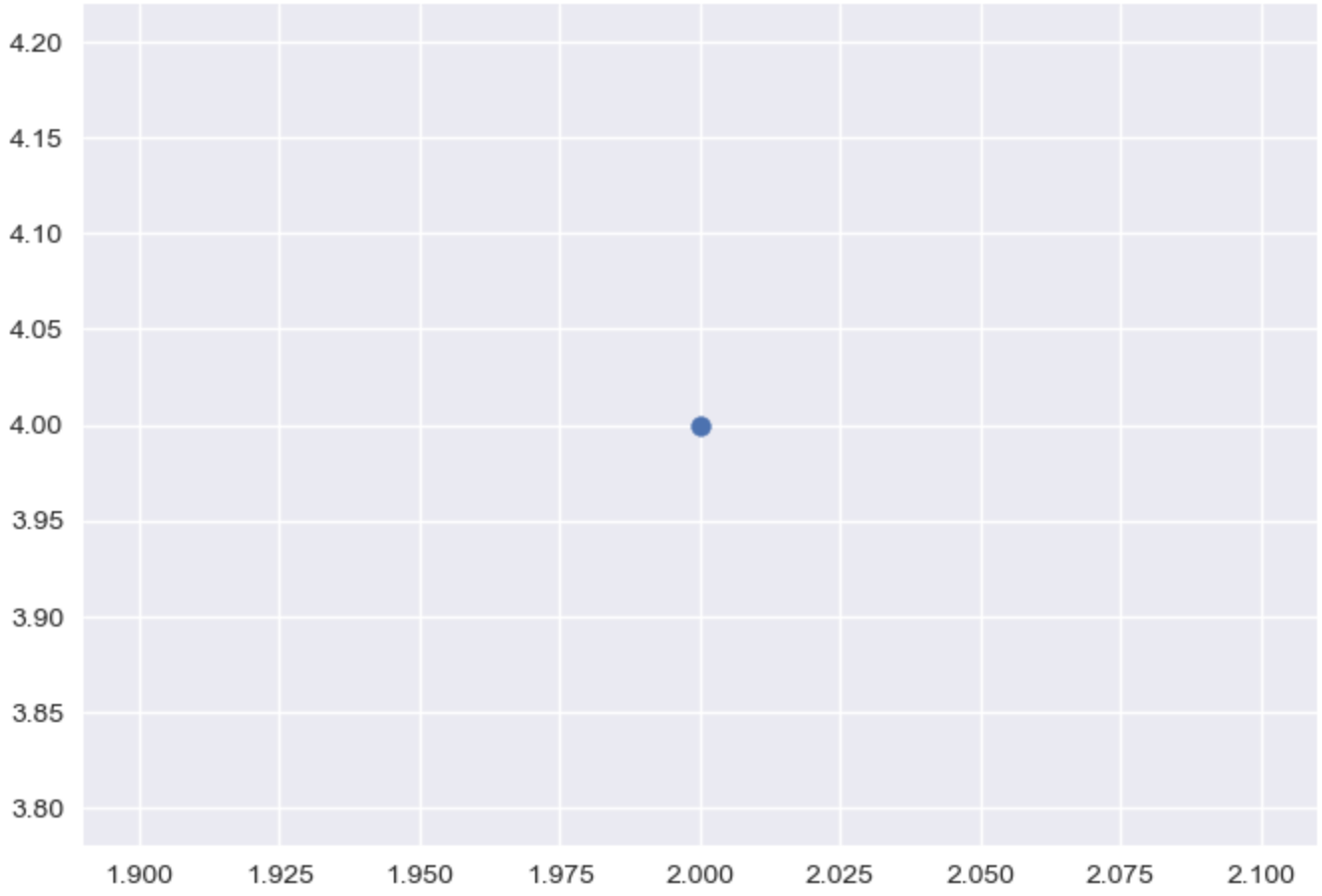
In [96]:
```
input_values = [1, 2, 3, 4, 5]
squares = [1, 4, 9, 16, 25]
#plt.style.use('seaborn-v0_8')
#plt.style.use('seaborn-v0_8-colorblind')
#plt.style.use('seaborn-v0_8-poster')
#plt.style.use('seaborn-v0_8-notebook')
#plt.style.use('seaborn-v0_8-darkgrid')
#plt.style.use('seaborn-v0_8-dark-palette')
plt.style.use('dark_background')

fig, ax = plt.subplots()
ax.plot(input_values, squares, linewidth=3)
#ax.set_title('seaborn style')
plt.show()
```



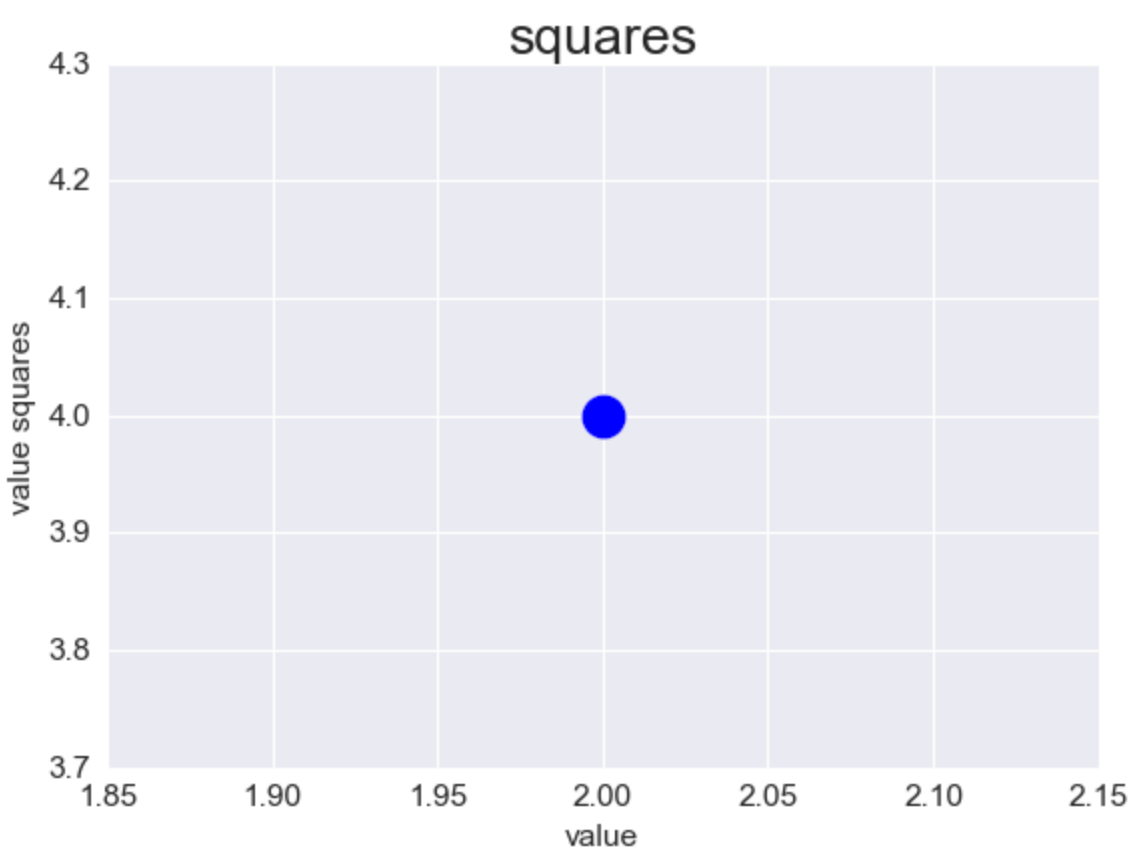
15.2.4使用scatter()绘制散点图并设置样式

有时候，绘制散点图并设置各个数据点的样式很有用。 例如，你可能想以一种颜色显示较小的值，用另一种颜色显示较大的值。 绘制大型数据集时，还可对每个点都设置同样的样式，再使用不同的样式选项重新绘制某些点以示突出。

In [35]:
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()
ax.scatter(2, 4)
plt.show()
```

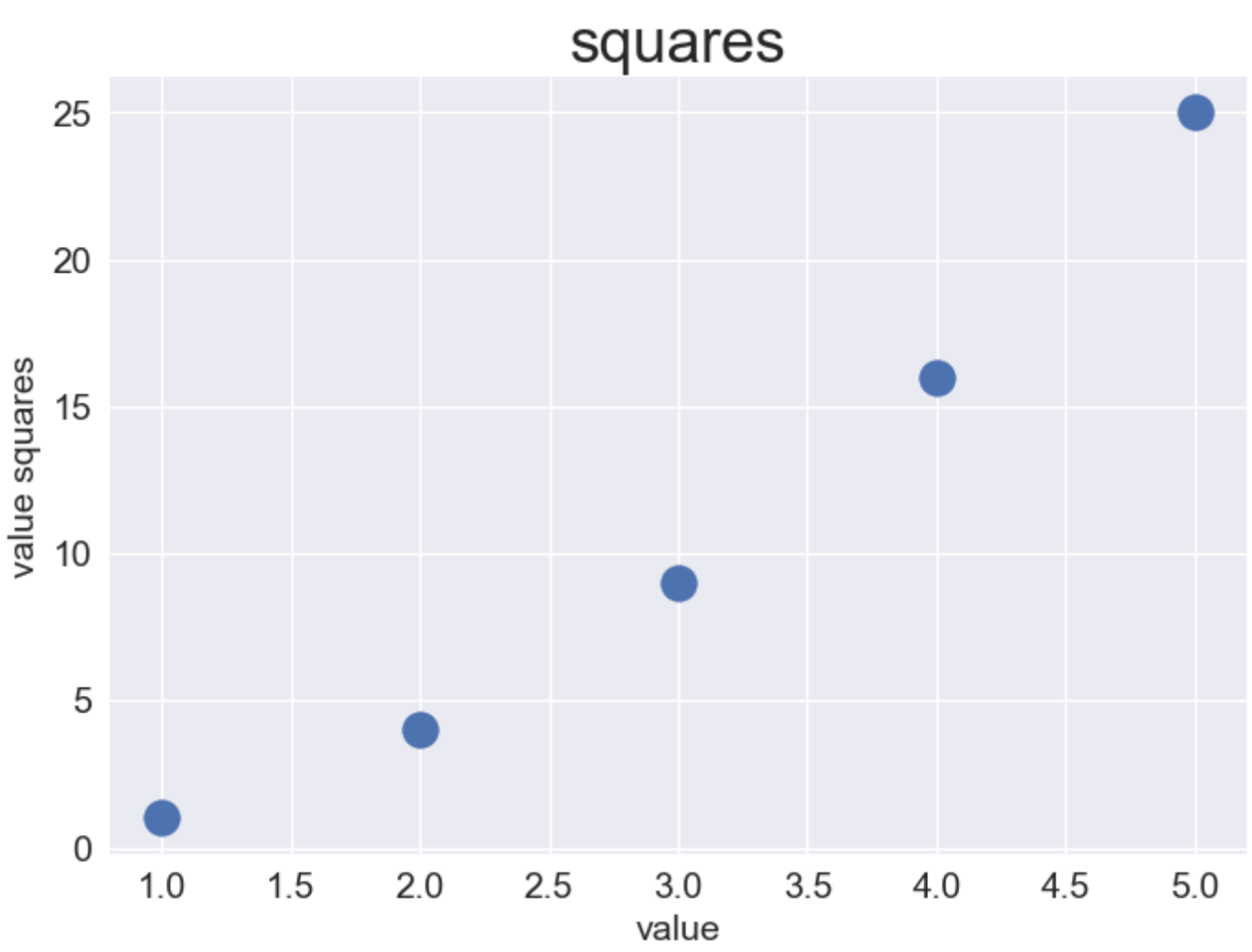


In [97]:
```
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()
ax.scatter(2, 4, s=400)
ax.set_title("squares", fontsize=24)
ax.set_xlabel("value", fontsize=14)
ax.set_ylabel("value squares", fontsize=14)
ax.tick_params(axis='both', which='major', labelsize=14)
plt.show()
```

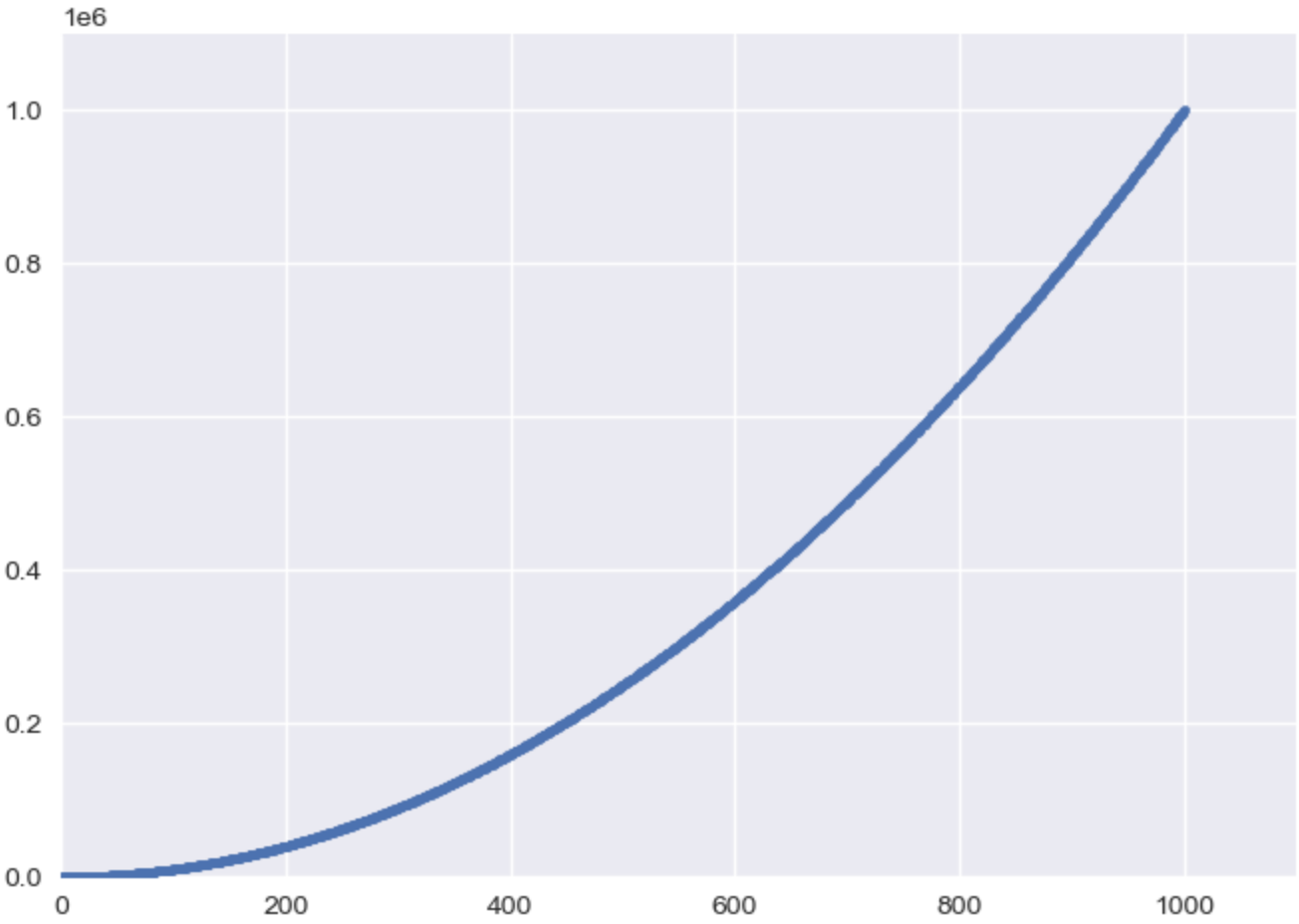


15.2.5 使用scatter()绘制一系列点

In [37]:
```
fig, ax = plt.subplots()
x_values = [1, 2, 3, 4, 5]
y_values = [1, 4, 9, 16, 25]
ax.scatter(x_values, y_values, s=200)
ax.set_title("squares", fontsize=24)
ax.set_xlabel("value", fontsize=14)
ax.set_ylabel("value squares", fontsize=14)
ax.tick_params(axis='both', which='major', labelsize=14)
plt.show()
```



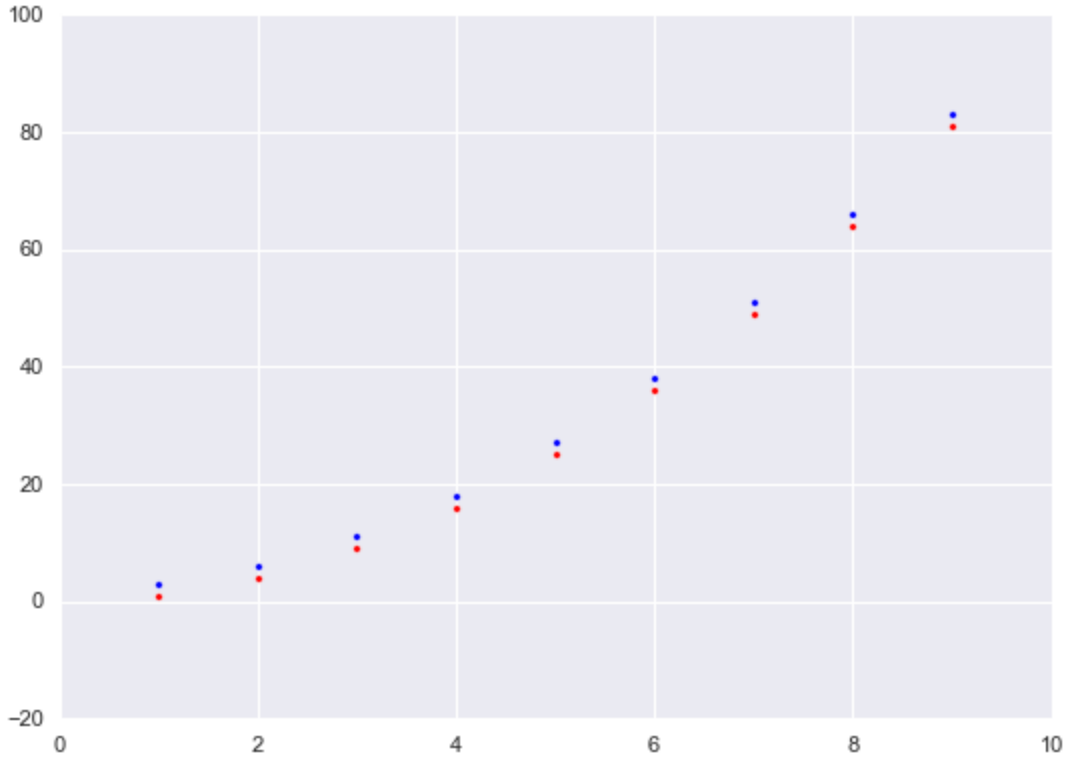
In [38]:
```
x_values = range(1, 1001)
y_values = [x**2 for x in x_values]
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, s=10)
ax.axis([0, 1100, 0, 1_100_000])
plt.show()
```



15.2.7 自定义颜色

In [100…

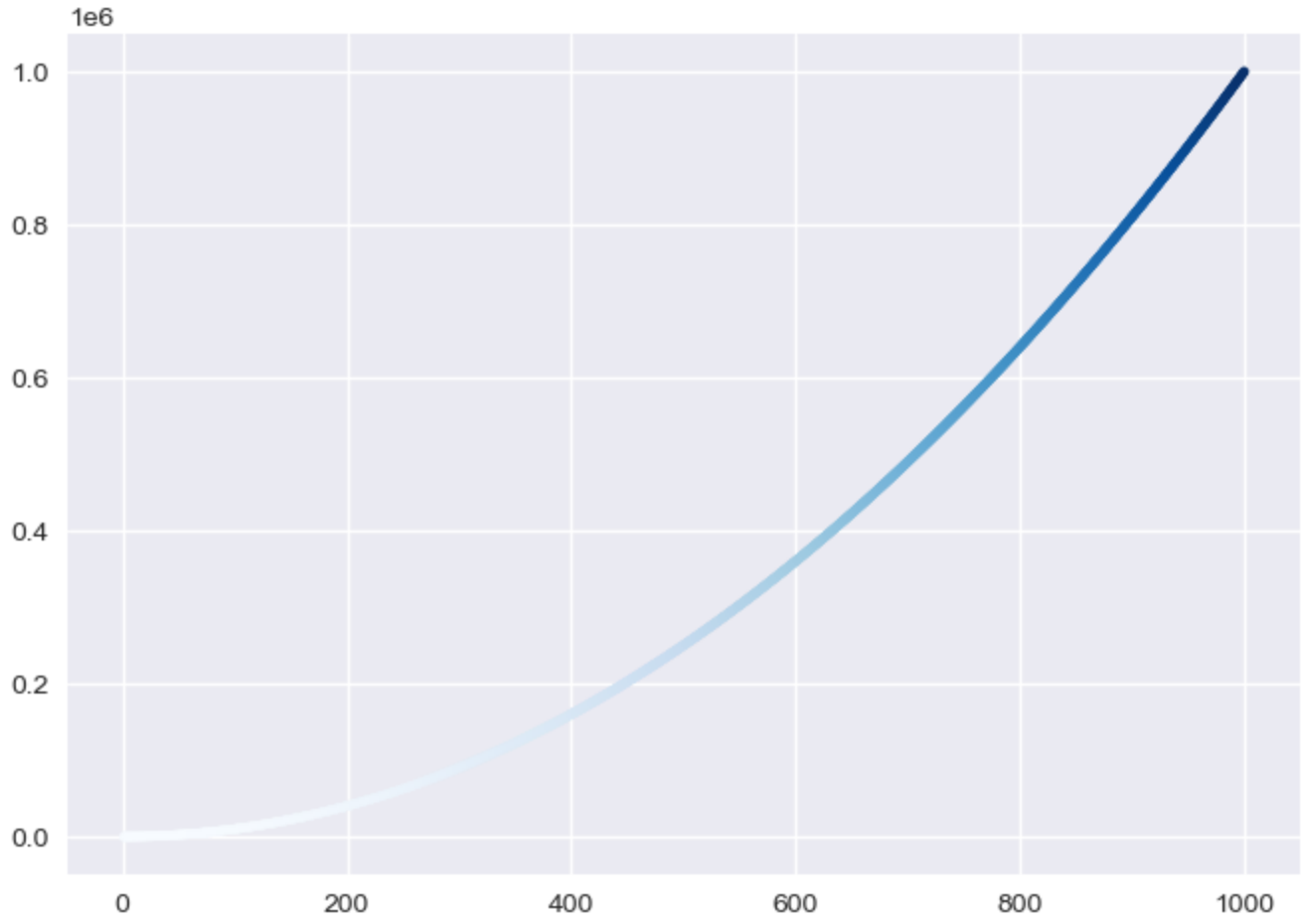
```
x_values = range(1, 10)
y_values = [x**2 for x in x_values]
y_values2 = [x**2+2 for x in x_values]
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, c='red', s=10)
ax.scatter(x_values, y_values2, c='blue', s=10)
#ax.scatter(x_values, y_values, color=(1, 0, 1), s=10)
plt.show()
```



15.2.8 使用颜色映射

颜色映射（colormap）是一系列颜色，从起始颜色渐变到结束颜色。 在可视化中，颜色映射用于突出数据的规律。 例如，你可能用较浅的颜色来显示较小的值，并使用较深的颜色来显示较大的值。

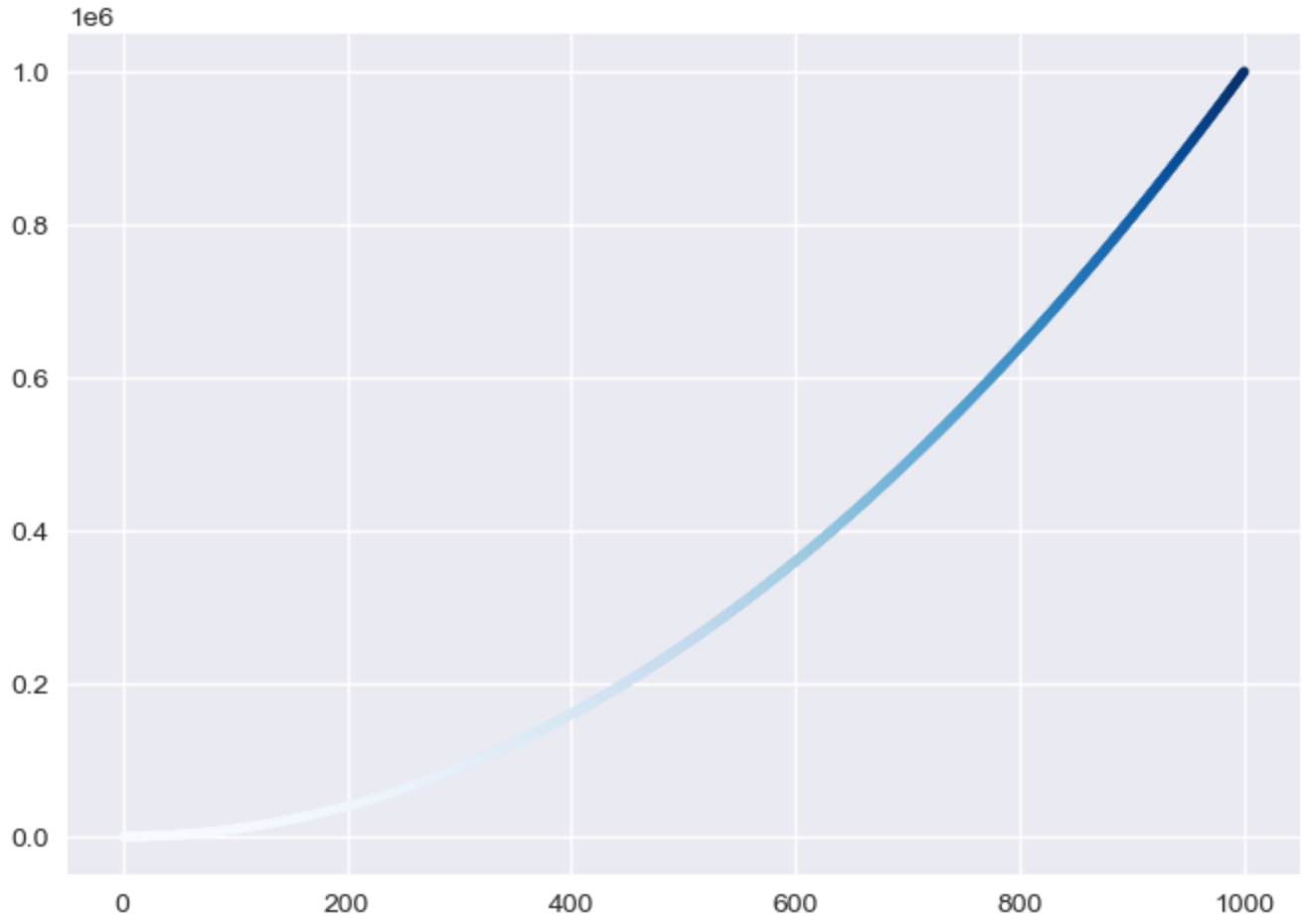
In [44]:
```
import matplotlib.pyplot as plt
x_values = range(1, 1001)
y_values = [x**2 for x in x_values]
plt.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, s=10)
plt.show()
```



15.2.9 保存图表

要让程序自动将图表保存到文件中，可将调用plt.savefig()

In [45]:
```
import matplotlib.pyplot as plt
x_values = range(1, 1001)
y_values = [x**2 for x in x_values]
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, s=10)
plt.savefig('squares_plot.png', bbox_inches='tight')
plt.show()
```



15.3 随机漫步

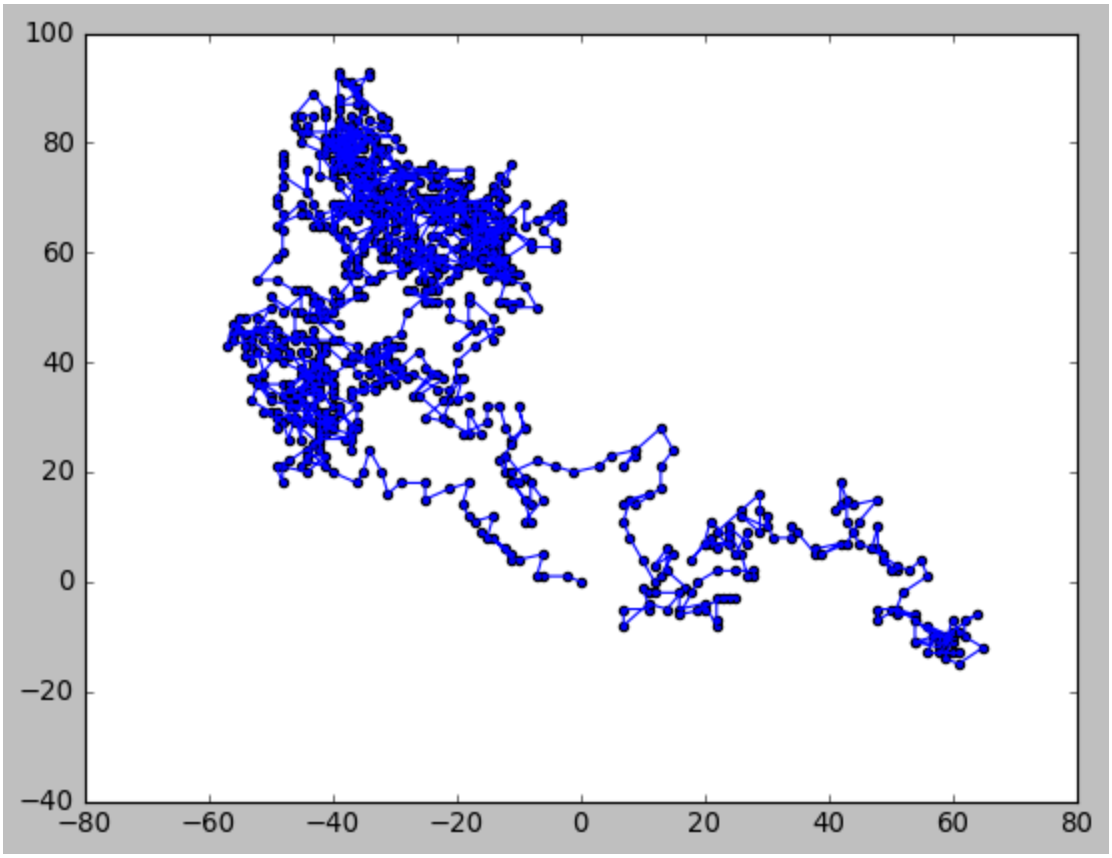
使用Python来生成随机漫步数据，再使用Matplotlib将这些数据呈现出来。 随机漫步是这样行走得到的路径： 每次行走都是完全随机的、没有明确的方向，结果是由一系列随机决策决定的。

In [46]:
```
from random import choice
class RandomWalk:
    """一个生成随机漫步数据的类。"""

    def __init__(self, num_points=5000):
        """初始化随机漫步的属性。"""
        self.num_points = num_points

        #所有随机漫步都始于(0, 0)。
        self.x_values = [0]
        self.y_values = [0]


    def fill_walk(self):
        """计算随机漫步包含的所有点。"""
        # 不断漫步，直到列表达到指定的长度。
        while len(self.x_values) < self.num_points:
            # 决定前进方向以及沿这个方向前进的距离。
            x_direction = choice([1, -1])
            x_distance = choice([0, 1, 2, 3, 4])
            x_step = x_direction * x_distance

            y_direction = choice([1, -1])
            y_distance = choice([0, 1, 2, 3, 4])
            y_step = y_direction * y_distance

            # 拒绝原地踏步。
```

```
            if x_step == 0 and y_step == 0:
                continue

            # 计算下一个点的x值和y值。
            x = self.x_values[-1] + x_step
            y = self.y_values[-1] + y_step
            self.x_values.append(x)
            self.y_values.append(y)
```

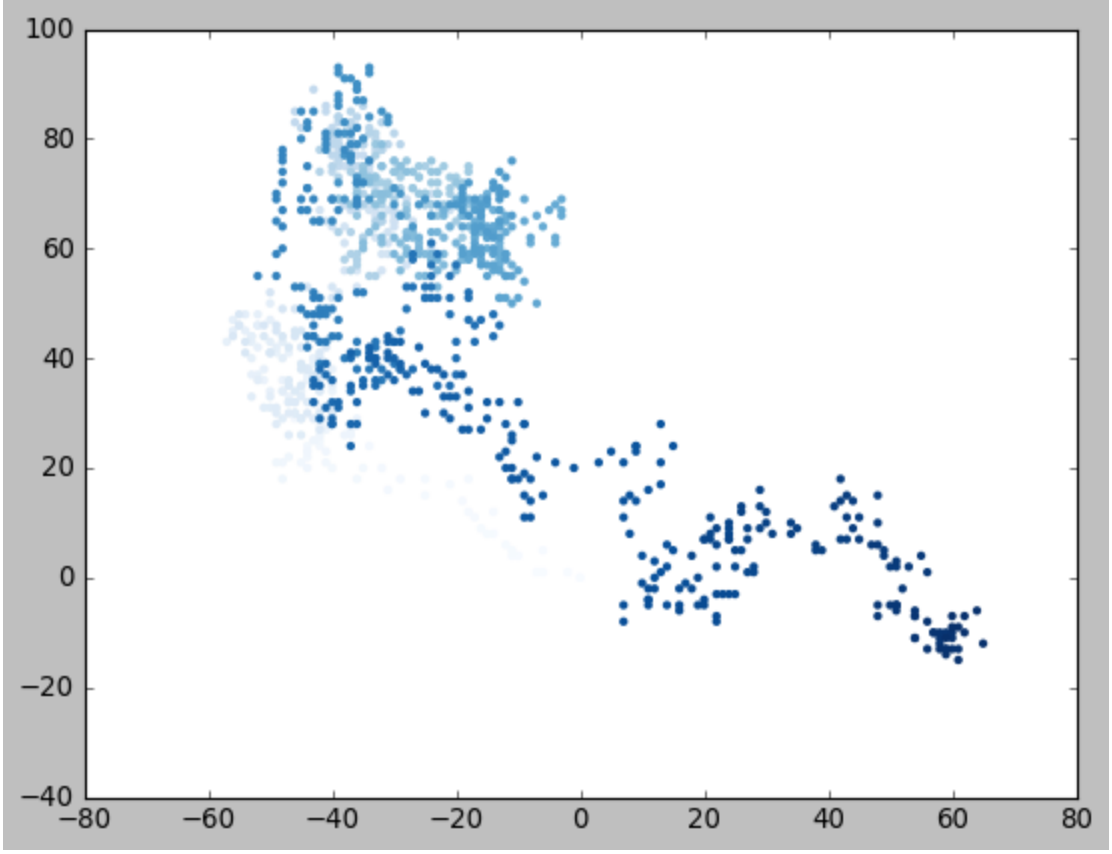
In [112…

```
rw = RandomWalk(num_points=1000)
rw.fill_walk()
# 将所有的点都绘制出来。
plt.style.use('classic')
fig, ax = plt.subplots()
ax.scatter(rw.x_values, rw.y_values, s=15)
ax.plot(rw.x_values, rw.y_values)
plt.show()
```



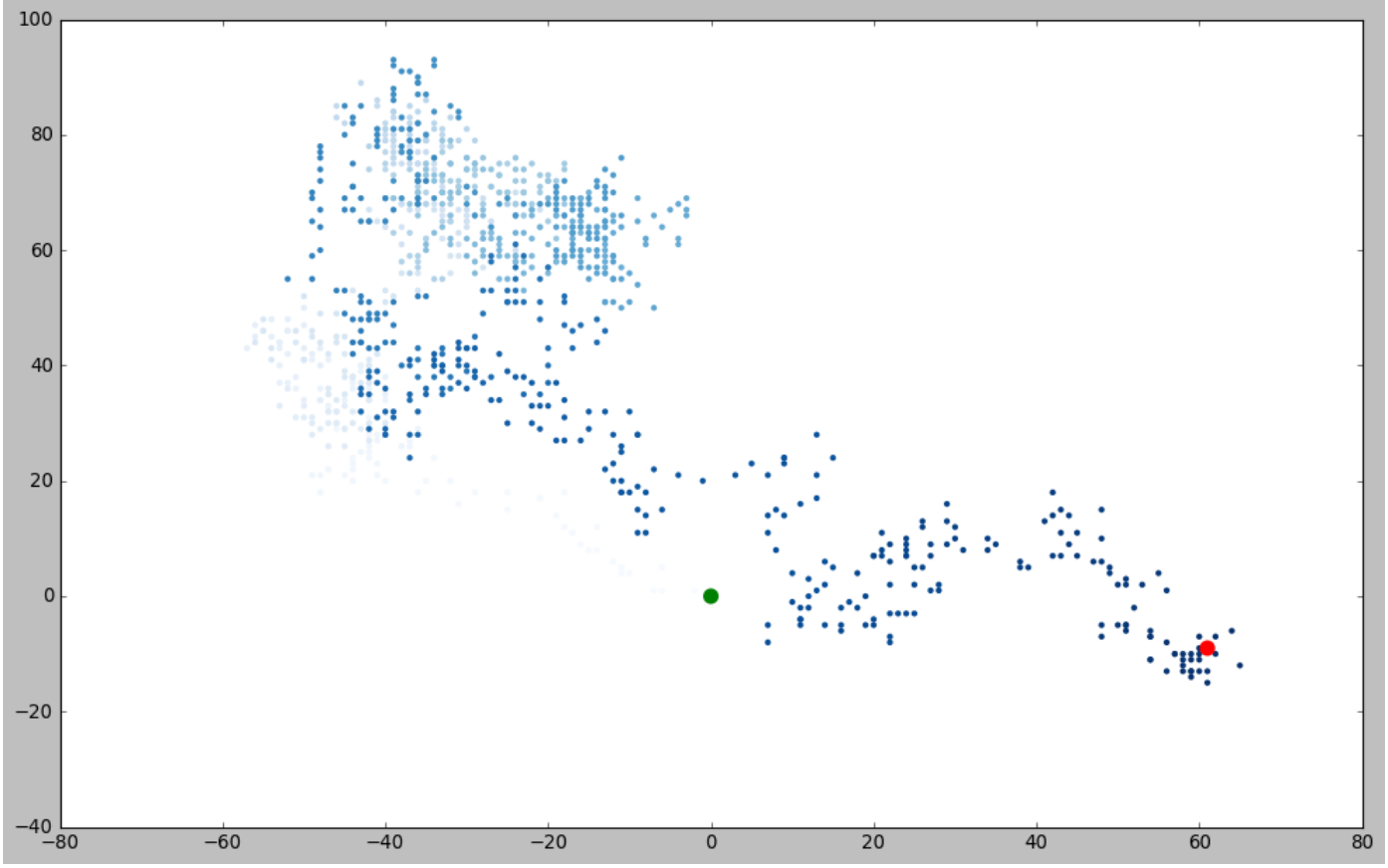
In [113…

```
plt.style.use('classic')
fig, ax = plt.subplots()

point_numbers = range(rw.num_points)

ax.scatter(rw.x_values, rw.y_values, c=point_numbers,
            cmap=plt.cm.Blues,
            edgecolors='none', s=15)
plt.show()
```



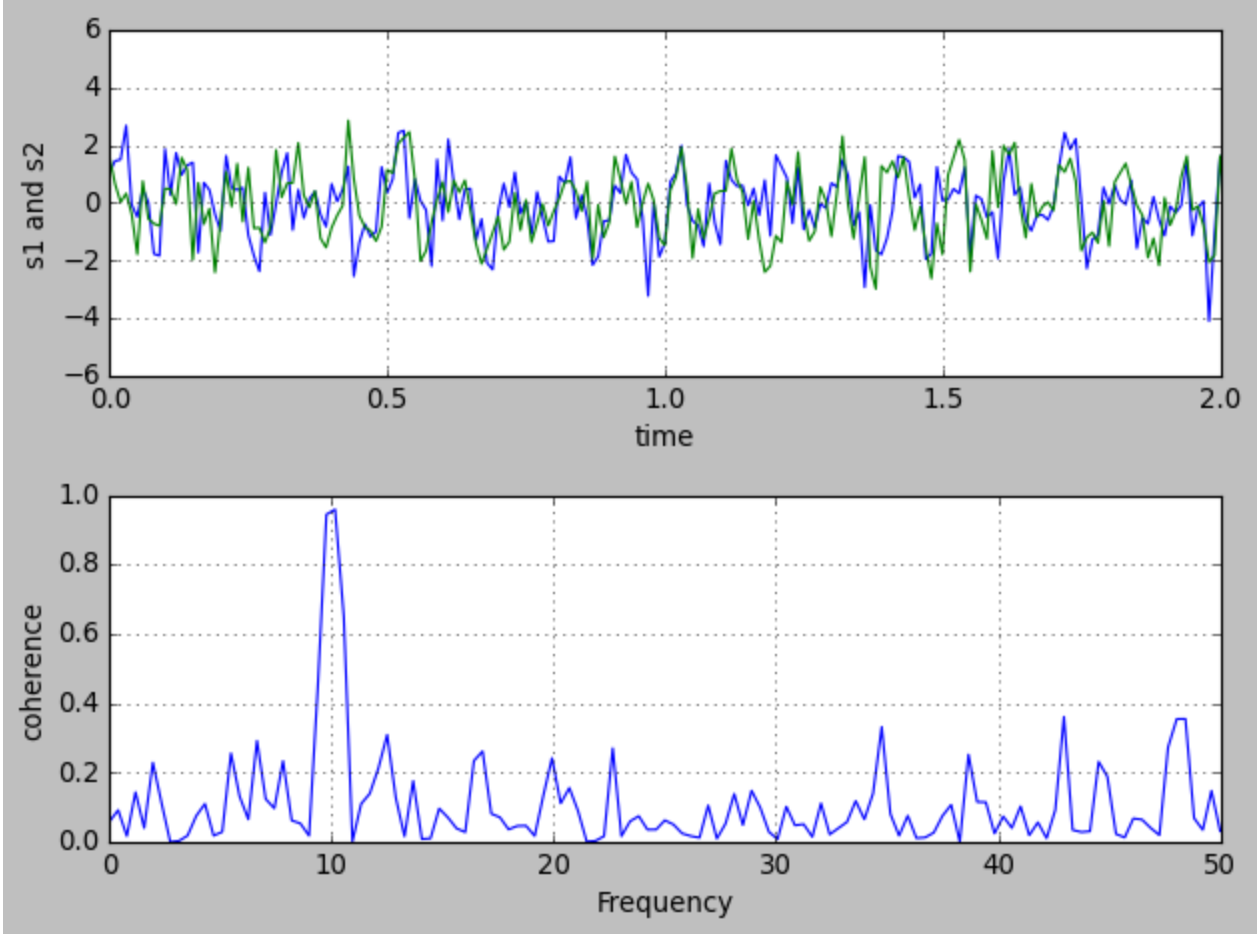
In [114…

```
#set figure size:
fig, ax = plt.subplots(figsize=(15, 9))

# set resolution:
# fig, ax = plt.subplots(figsize=(15, 9), dpi=128)

point_numbers = range(rw.num_points)
ax.scatter(rw.x_values, rw.y_values, c=point_numbers,
           cmap=plt.cm.Blues,
           edgecolors='none', s=15)

# plot start and end points
ax.scatter(0, 0, c='green', edgecolors='none', s=100)
ax.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',s=100)

# hide axes
#ax.get_xaxis().set_visible(False)
#ax.get_yaxis().set_visible(False)

plt.show()
```



In [58]:

```
import numpy as np
np.random.seed(19680801)

dt = 0.01
t = np.arange(0, 30, dt)
nse1 = np.random.randn(len(t))                 # white noise 1
nse2 = np.random.randn(len(t))                 # white noise 2

# Two signals with a coherent part at 10Hz and a random part
s1 = np.sin(2 * np.pi * 10 * t) + nse1
s2 = np.sin(2 * np.pi * 10 * t) + nse2

fig, axs = plt.subplots(2, 1)
axs[0].plot(t, s1, t, s2)
axs[0].set_xlim(0, 2)
axs[0].set_xlabel('time')
axs[0].set_ylabel('s1 and s2')
axs[0].grid(True)

# matplotlib库的axis模块中的Axes.cohere()函数用于绘制x和y之间的相干性
cxy, f = axs[1].cohere(s1, s2, 256, 1. / dt)
axs[1].set_ylabel('coherence')

fig.tight_layout()
plt.show()

# https://vimsky.com/examples/usage/matplotlib-axes-axes-cohere-in-python.html
```

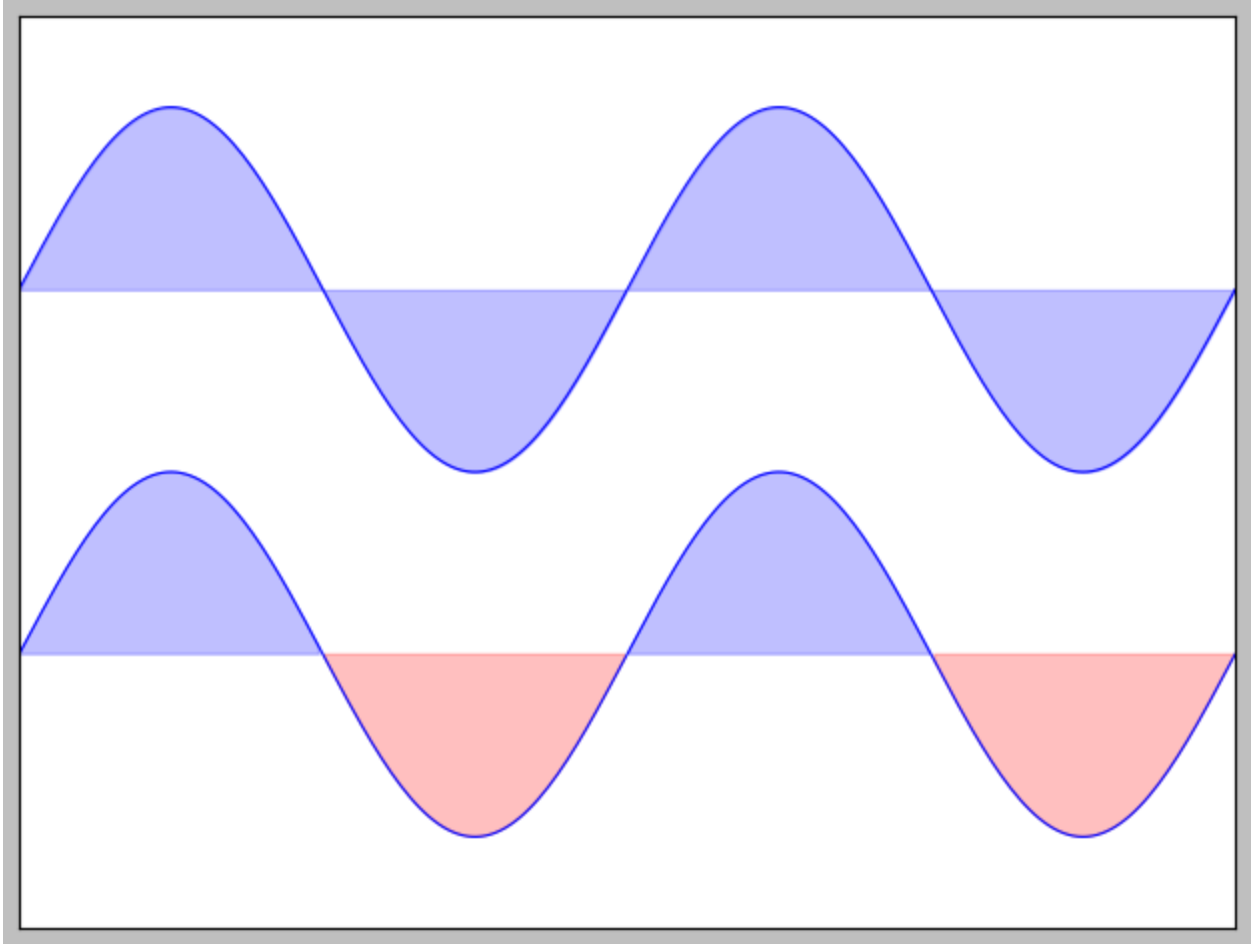


In [59]:

```
import matplotlib.gridspec as gridspec

G = gridspec.GridSpec(3, 3)

axes_1 = plt.subplot(G[0, :])
plt.xticks([]), plt.yticks([])
plt.text(0.5,0.5, 'Axes 1',ha='center',va='center',size=24,alpha=.5)

axes_2 = plt.subplot(G[1,:-1])
plt.xticks([]), plt.yticks([])
plt.text(0.5,0.5, 'Axes 2',ha='center',va='center',size=24,alpha=.5)

axes_3 = plt.subplot(G[1:, -1])
plt.xticks([]), plt.yticks([])
plt.text(0.5,0.5, 'Axes 3',ha='center',va='center',size=24,alpha=.5)

axes_4 = plt.subplot(G[-1,0])
plt.xticks([]),plt.yticks([])
plt.text(0.5,0.5, 'Axes 4',ha='center',va='center',size=24,alpha=.5)

axes_5 = plt.subplot(G[-1,-2])
plt.xticks([]),plt.yticks([])
plt.text(0.5,0.5, 'Axes 5',ha='center',va='center',size=24,alpha=.5)

#plt.savefig('../figures/gridspec.png', dpi=64)
plt.show()
```

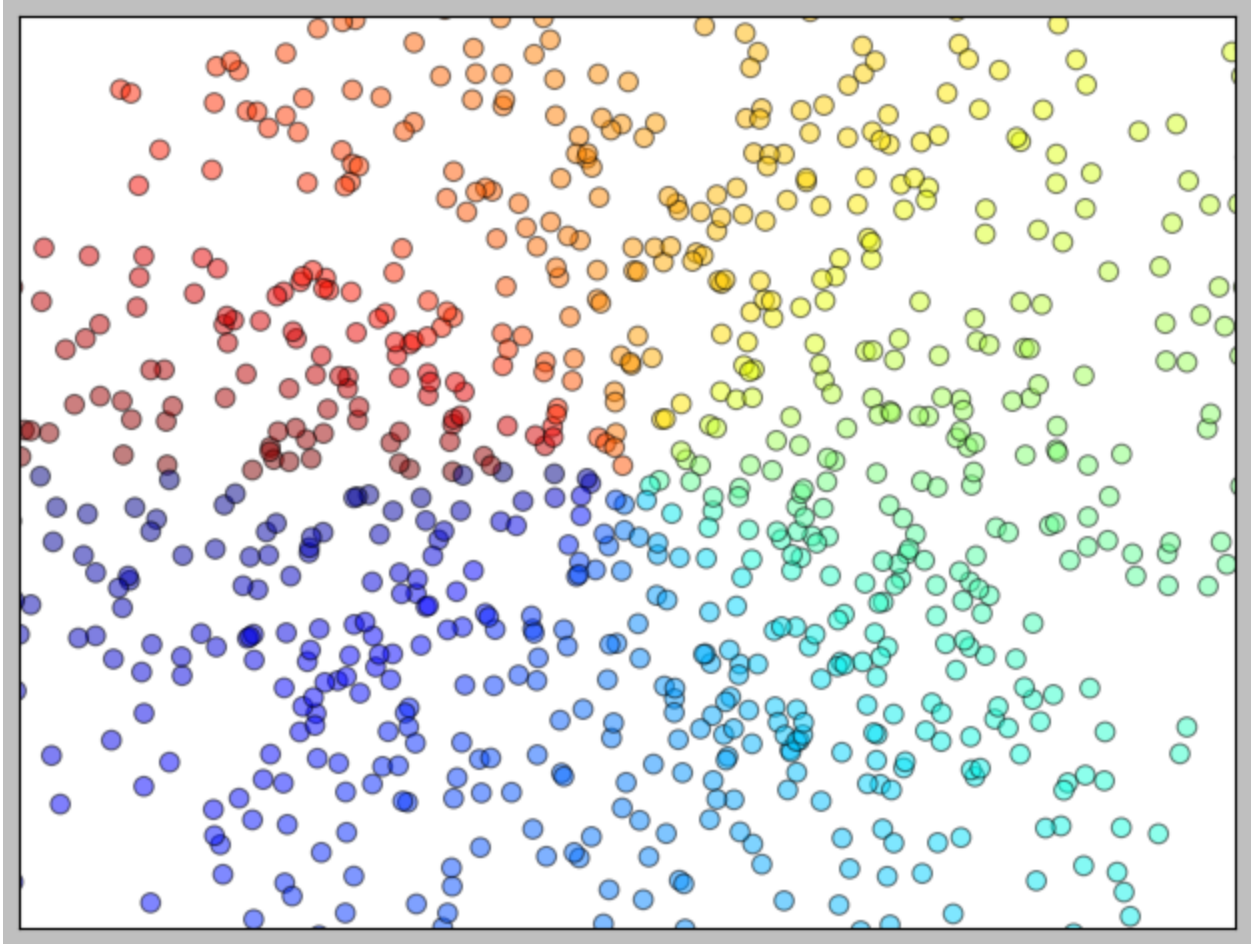
In [60]:

```
n = 256
X = np.linspace(-np.pi,np.pi,n,endpoint=True)
Y = np.sin(2*X)

plt.axes([0.025,0.025,0.95,0.95])

plt.plot (X, Y+1, color='blue', alpha=1.00)
plt.fill_between(X, 1, Y+1, color='blue', alpha=.25)

plt.plot (X, Y-1, color='blue', alpha=1.00)
plt.fill_between(X, -1, Y-1, (Y-1) > -1, color='blue', alpha=.25)
plt.fill_between(X, -1, Y-1, (Y-1) < -1, color='red',  alpha=.25)

plt.xlim(-np.pi,np.pi), plt.xticks([])
plt.ylim(-2.5,2.5), plt.yticks([])
# savefig('../figures/plot_ex.png',dpi=48)
plt.show()
```



In [115…

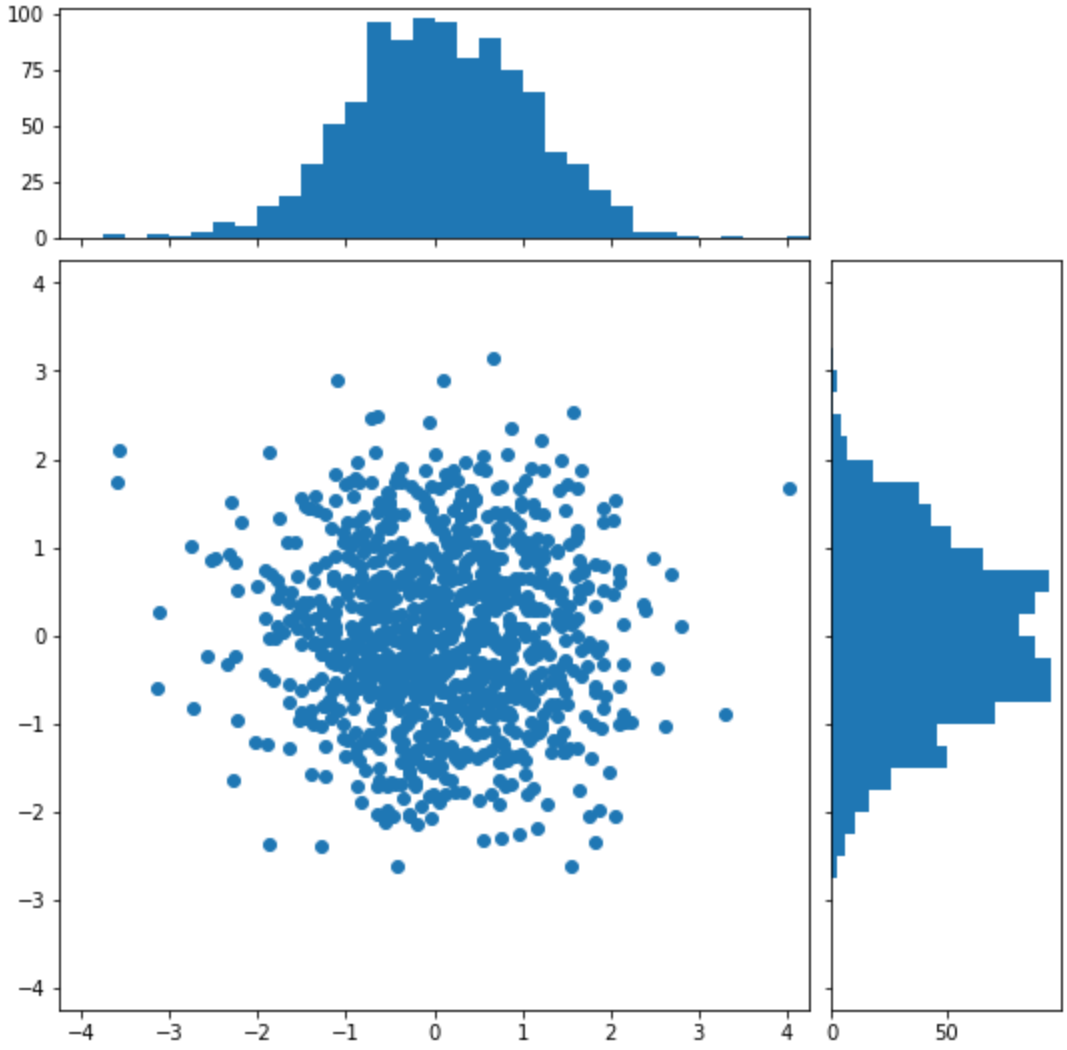
```
n = 1024
X = np.random.normal(0,1,n)
Y = np.random.normal(0,1,n)
T = np.arctan2(Y,X)

plt.axes([0.025,0.025,0.95,0.95])
plt.scatter(X,Y, s=75, c=T, alpha=.5)

plt.xlim(-1.5,1.5), plt.xticks([])
plt.ylim(-1.5,1.5), plt.yticks([])
# savefig('../figures/scatter_ex.png',dpi=48)
plt.show()
```



In [33]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter

# Fixing random state for reproducibility
np.random.seed(19680801)


# the random data
x = np.random.randn(1000)
y = np.random.randn(1000)

nullfmt = NullFormatter()         # no labels

# definitions for the axes
left, width = 0.1, 0.65
bottom, height = 0.1, 0.65
bottom_h = left_h = left + width + 0.02

rect_scatter = [left, bottom, width, height]
rect_histx = [left, bottom_h, width, 0.2]
rect_histy = [left_h, bottom, 0.2, height]

# start with a rectangular Figure
plt.figure(1, figsize=(8, 8))

axScatter = plt.axes(rect_scatter)
axHistx = plt.axes(rect_histx)
axHisty = plt.axes(rect_histy)

# no labels
axHistx.xaxis.set_major_formatter(nullfmt)
axHisty.yaxis.set_major_formatter(nullfmt)

# the scatter plot:
```

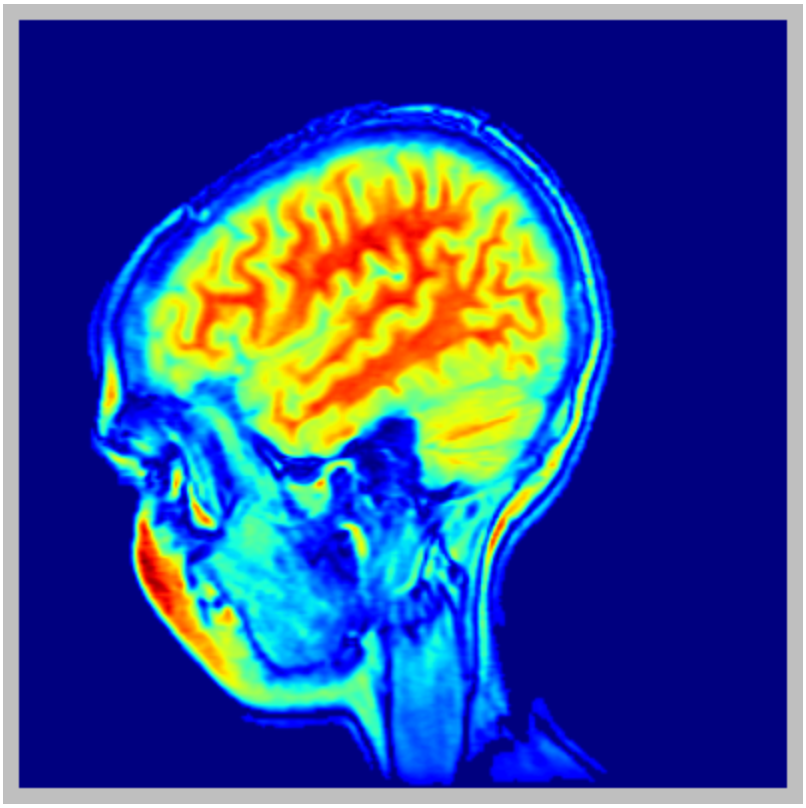
```
axScatter.scatter(x, y)

# now determine nice limits by hand:
binwidth = 0.25
xymax = max(np.max(np.abs(x)), np.max(np.abs(y)))
lim = (int(xymax/binwidth) + 1) * binwidth

axScatter.set_xlim((-lim, lim))
axScatter.set_ylim((-lim, lim))

bins = np.arange(-lim, lim + binwidth, binwidth)
axHistx.hist(x, bins=bins)
axHisty.hist(y, bins=bins, orientation='horizontal')

axHistx.set_xlim(axScatter.get_xlim())
axHisty.set_ylim(axScatter.get_ylim())

plt.show()
```



In [116…

```
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
import matplotlib.cm as cm
import numpy as np
import cv2


# Data are 256x256 16 bit integers
with cbook.get_sample_data('s1045.ima.gz') as dfile:
    im = np.frombuffer (dfile.read(), np.uint16).reshape((256, 256))

    fig, ax = plt.subplots(num="MRI_demo")
```

```
    ax.imshow(im)
    ax.axis('off')

    plt.show()
```



15.4 使用Plotly模拟掷骰子

本节将使用Python包Plotly来生成交互式图表。 需要创建在浏览器中显示的图表时，Plotly很有用，因为它生成的图表将自动缩放以适合观看者的屏幕。 Plotly生成的图表还是交互式的：用户将鼠标指向特定元素时，将突出显示有关该元素的信息。

在这个例子中，将对掷骰子的结果进行分析。 抛掷一个6面的常规骰子时，可能出现的结果为1～6点，且出现每种结果的可能性相同。 然而，如果同时掷两个骰子，某些点数出现的可能性将比其他点数大。 为确定哪些点数出现的可能性最大，将生成一个表示掷骰子结果的数据集，并根据结果绘制一个图形。

In [80]:
```
from random import randint

class Die:
    """表示一个骰子的类。"""

    def __init__(self, num_sides=6):
        """骰子默认为6面。"""
        self.num_sides = num_sides
    def roll(self):
        """返回一个位于1和骰子面数之间的随机值。"""
        return randint(1, self.num_sides)
```

In [119…
```
# 创建一个骰子实例。
die = Die()
# 掷几次骰子并将结果存储在一个列表中。
results = []
for roll_num in range(100):
    result = die.roll()
    results.append(result)
```

```
#print(results)
```

15.4.4 分析结果

为分析掷一个骰子的结果，计算每个点数出现的次数：

In [120…
```
# 分析结果。
frequencies = []
for value in range(1, die.num_sides+1):
    frequency = results.count(value)
    frequencies.append(frequency)

print(frequencies)
#print(sorted(frequencies))
```

```
[17, 21, 15, 18, 15, 14]
```

15.4.5 绘制直方图

有了频率列表，就可以绘制一个表示结果的直方图了。 直方图是一种条形图，指出了各种结果出现的频率。创建这种直方图的代码如下：

In [121…
```
from plotly.graph_objs import Bar, Layout
from plotly import offline

x_values = list(range(1, die.num_sides+1))
data = [Bar(x=x_values, y=frequencies)]

x_axis_config = {'title': '结果'}
y_axis_config = {'title': '结果的频率'}

my_layout = Layout(title='掷一个D6 1000次的结果',
xaxis=x_axis_config, yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='d6.html')
```

Out[121]:
```
'd6.html'
```

15.4.6 同时掷两个骰子

In [122…
```
die_1 = Die()
die_2 = Die()


results = []
for roll_num in range(1000):
    result = die_1.roll() + die_2.roll()
    results.append(result)

# 分析结果。
frequencies = []
max_result = die_1.num_sides + die_2.num_sides

for value in range(2, max_result+1):
    frequency = results.count(value)
    frequencies.append(frequency)

x_values = list(range(2, max_result+1))
data = [Bar(x=x_values, y=frequencies)]
```

```
x_axis_config = {'title': '结果', 'dtick': 1, }
y_axis_config = {'title': '结果的频率'}

my_layout = Layout(title='掷两个D6 1000次的结果',
xaxis=x_axis_config,
yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='d6_d6.html')
```

Out[122]:
```
'd6_d6.html'
```

15.5 小结

在本章中，学习了：

- 如何生成数据集以及如何对其进行可视化；
- 如何使用Matplotlib创建简单的图表，以及如何使用散点图来探索随机漫步过程；
- 如何使用Plotly来创建直方图，以及如何使用直方图来探索同时掷两个面数不同的骰子的结果。

在第16章中，将从网上下载数据，并继续使用Matplotlib和Plotly来探索这些数据。

In [99]:
```
# The end
```