情节 – 介绍

Plotly 是一家总部位于蒙特利尔的技术计算公司，参与开发数据分析和可视化工具，如**Dash**和**Chart Studio**。它还为 Python、R、MATLAB、Javascript 和其他计算机编程语言开发了开源图形应用程序编程接口 (API) 库。

Plotly 的一些**重要功能**如下 –

- 它生成交互式图形。
- 这些图形以 JavaScript 对象表示法**(JSON) 数据格式**存储，以便可以使用其他编程语言（如 R、Julia、MATLAB 等）的脚本读取它们。
- 图形可以以各种光栅和矢量图像格式导出

Plotly – 环境设置

本章重点介绍如何借助 Plotly 在 Python 中进行环境设置。

安装 Python 包

始终建议使用 Python 的虚拟环境功能来安装新包。以下命令在指定的文件夹中创建一个虚拟环境。

```
python -m myenv
```

要激活如此创建的虚拟环境，请在**bin**子文件夹中运行**activate**脚本，如下所示。

```
source bin/activate
```

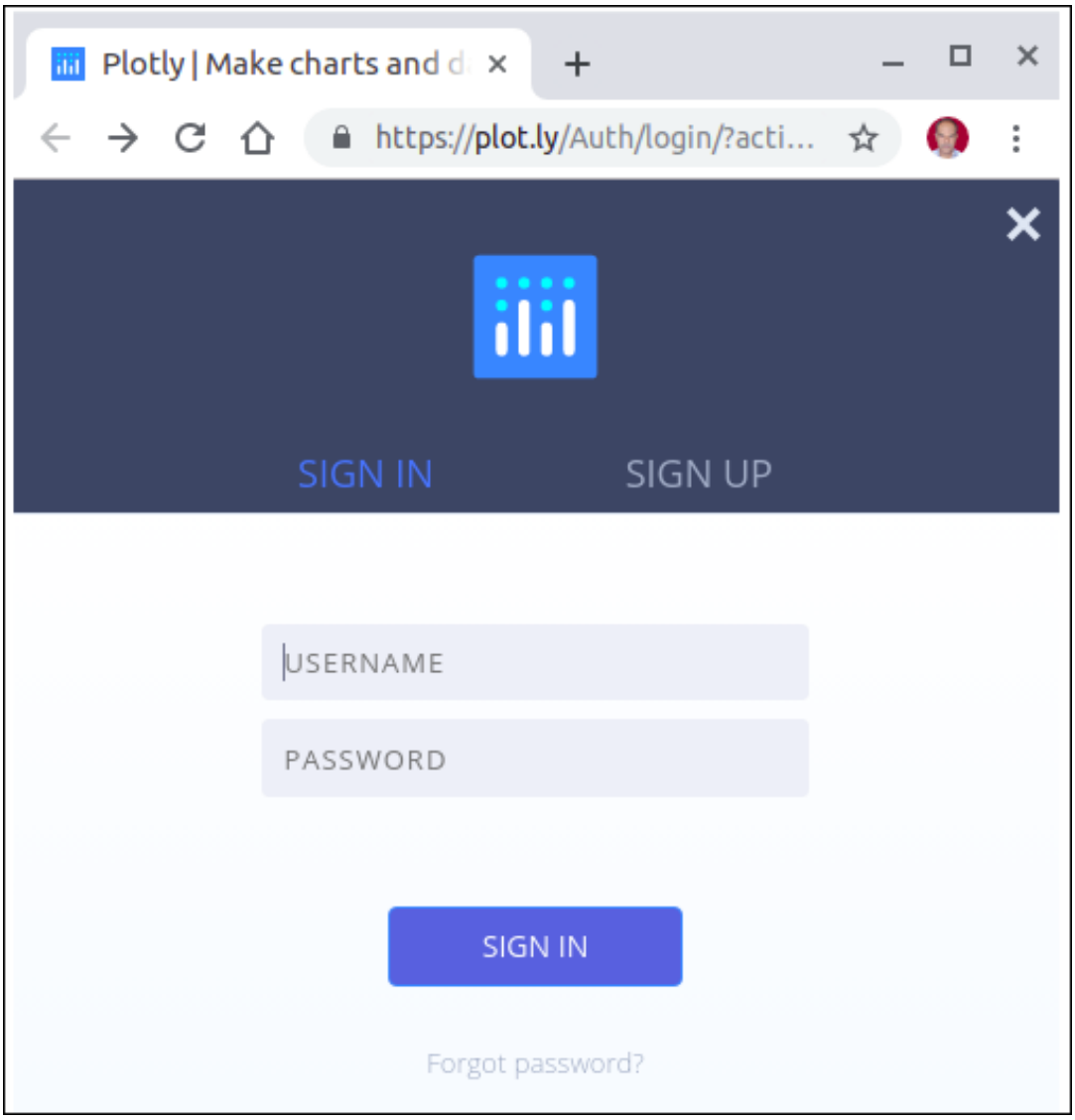
现在我们可以使用 pip 实用程序安装 plotly 的 Python 包，如下所示。

```
pip install plotly
```

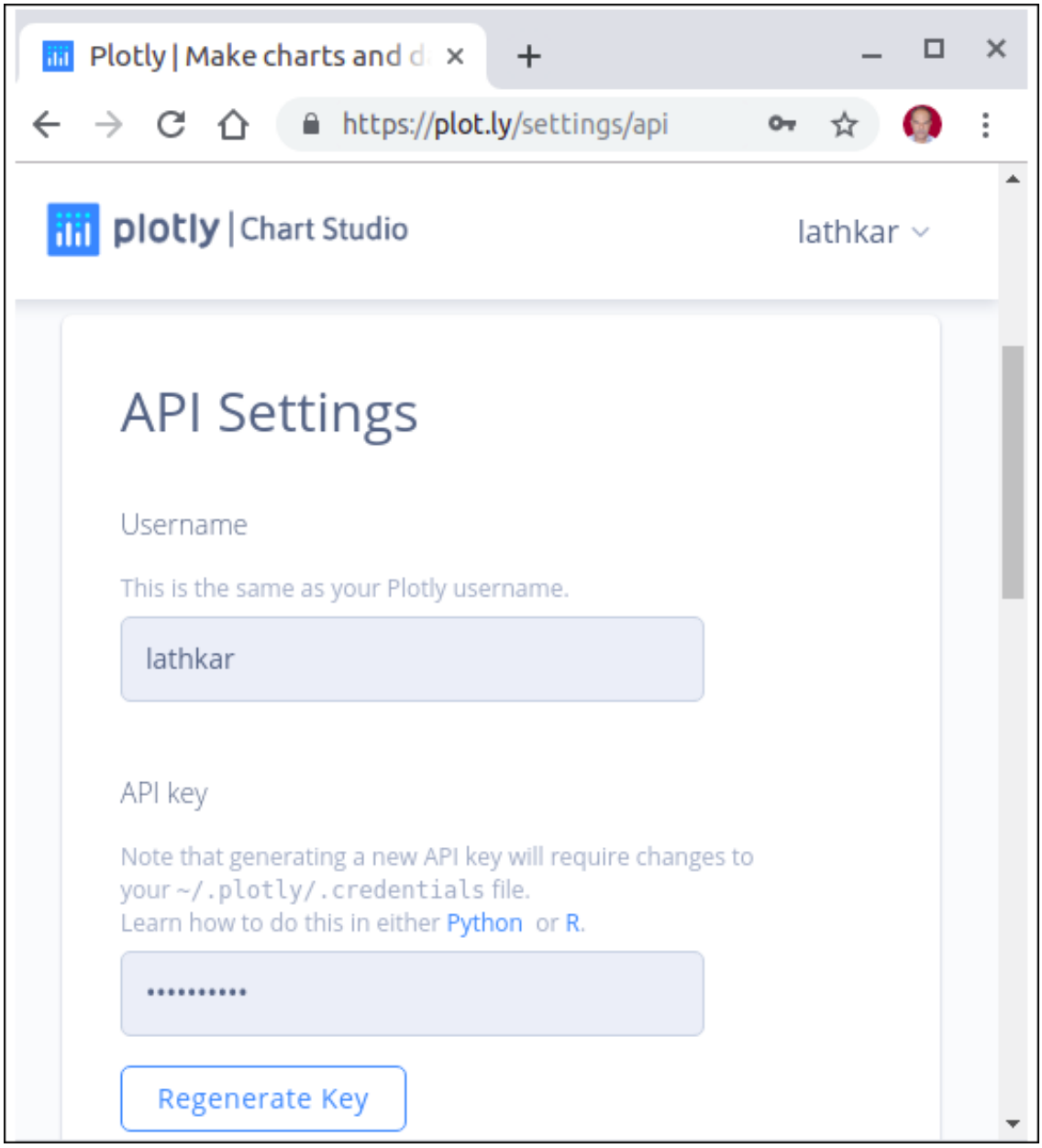
您可能还想安装**Jupyter 笔记本**应用程序，它是**Ipython**解释器的基于 Web 的界面。

```
pip install jupyter notebook
```

首先，您需要在https://plot.ly 的网站上创建一个帐户。您可以使用此处提到的链接https://plot.ly/api_signup 进行注册，然后登录成功。



接下来，从仪表板的设置页面获取 API 密钥。



使用您的用户名和 API 密钥在**Python 解释器**会话中设置凭据。

```
import plotly
plotly.tools.set_credentials_file(username='test',
api_key='********************')
```

在主目录下的.plotly**子文件夹中**创建了一个名为**凭据**的特殊文件。它看起来类似于以下内容 –

```
{
   "username": "test",
   "api_key": "********************",
   "proxy_username": "",
   "proxy_password": "",
   "stream_ids": []
}
```

为了生成图，我们需要从 plotly 包中导入以下模块 –

```
import plotly.plotly as py
import plotly.graph_objs as go
```

plotly.plotly 模块包含帮助我们与 Plotly 服务器通信的功能。**plotly.graph_objs 模块中的**函数生成图形对象

Plotly – 在线和离线绘图

下一章涉及在线和离线绘图的设置。让我们首先研究在线绘图的设置。

在线绘图的设置

在线绘图的**数据**和**图表**保存在您的**plot.ly 帐户中**。在线绘图由两种方法生成，这两种方法都会为绘图创建一个唯一的**url**并将其保存在您的 Plotly 帐户中。

- **py.plot()** – 返回唯一的 url 并可选择打开 url。
- **py.iplot()** – 在**Jupyter Notebook 中**工作以在**Notebook**中显示绘图时。

我们现在将显示以**弧度为单位的**简单角度**与其正弦值的关系图**。首先，使用numpy 库中的**arange()**函数获取角度在 0 到 2π 之间的 ndarray 对象。此 ndarray 对象用作图形**x 轴上**的值。必须在**y 轴上**显示的 x 角的相应正弦值通过以下语句获得 –

```
import numpy as np
import math #needed for definition of pi
xpoints = np.arange(0, math.pi*2, 0.05)
ypoints = np.sin(xpoints)
```

接下来，使用**graph_objs 模块**中的**Scatter()**函数创建散点跟踪。

```
trace0 = go.Scatter(
   x = xpoints,
   y = ypoints
)
data = [trace0]
```

使用上面的列表对象作为**plot()**函数的参数。

```
py.plot(data, filename = 'Sine wave', auto_open=True)
```

将以下脚本另存为**plotly1.py**

```
import plotly
plotly.tools.set_credentials_file(username='lathkar', api_key='********************')
import plotly.plotly as py
import plotly.graph_objs as go
import numpy as np
import math #needed for definition of pi
```

```
xpoints = np.arange(0, math.pi*2, 0.05)
ypoints = np.sin(xpoints)
trace0 = go.Scatter(
   x = xpoints, y = ypoints
)
data = [trace0]
py.plot(data, filename = 'Sine wave', auto_open=True)
```

从命令行执行上述脚本。结果图将显示在浏览器中指定的 URL 处，如下所述。

```
$ python plotly1.py
High five! You successfully sent some data to your account on plotly.
View your plot in your browser at https://plot.ly/~lathkar/0
```

在显示的图表上方，您会找到绘图、数据、Python 和 Rand 分叉历史选项卡。

当前，选择了**绘图选项卡**。数据选项卡显示包含 x 和**y**数据点的网格。从 Python & R 选项卡中，您可以查看与 Python、R、JSON、Matlab 等中当前绘图相对应的代码。以下快照显示了上面生成的绘图的 Python 代码 –

离线绘图的设置

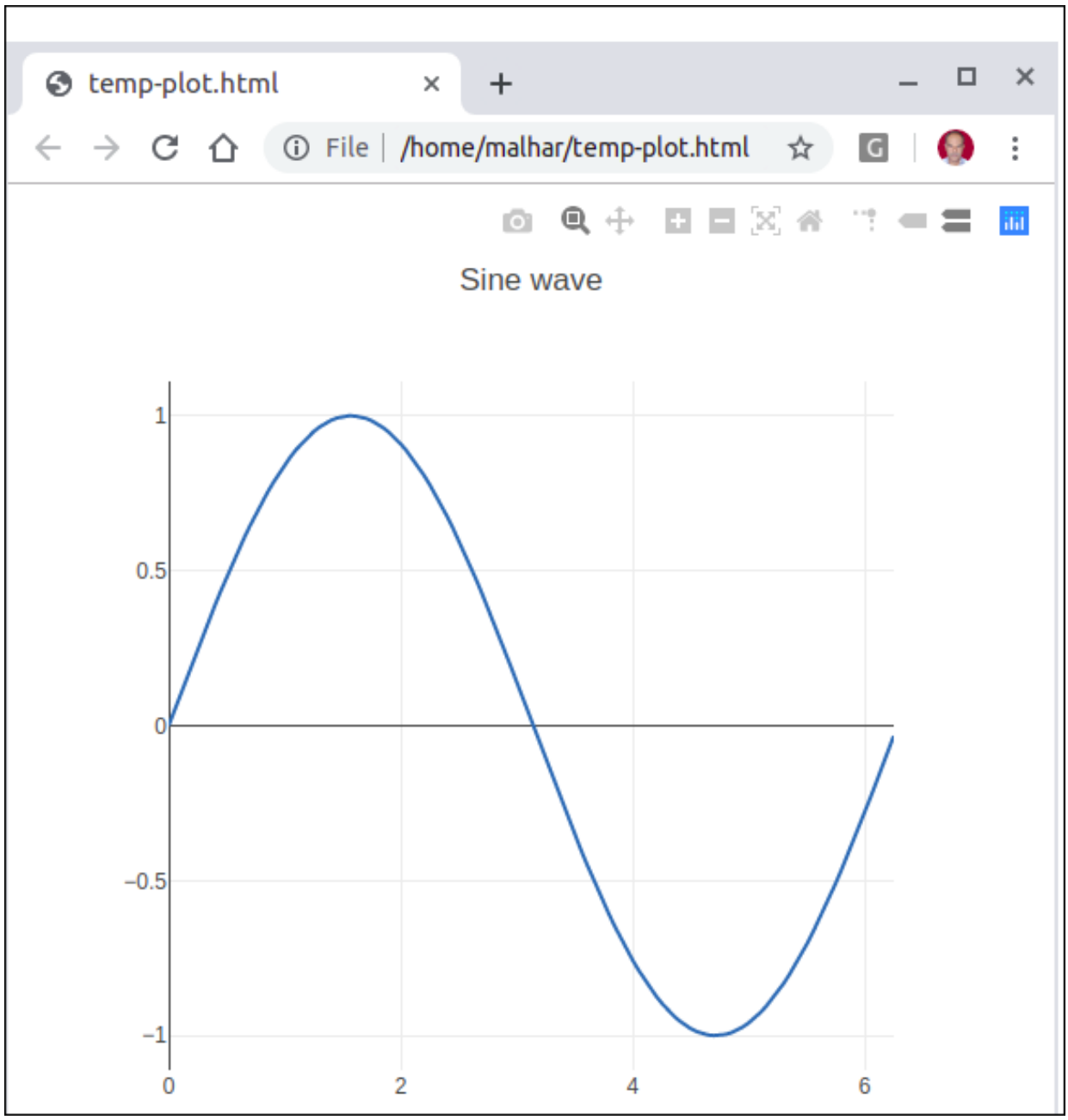
Plotly 允许您离线生成图形并将它们保存在本地机器中。该**plotly.offline.plot（）**函数创建一个保存在本地和Web浏览器里面打开一个独立的HTML。

在**Jupyter Notebook 中**离线工作时使用**plotly.offline.iplot()**以在**Notebook**中显示绘图。

注– Plotly的版本**1.9.4+**需要进行脱机打印。

更改脚本中的**plot() 函数**语句并运行。一个名为**temp-plot.html 的**HTML 文件将在本地创建并在 Web 浏览器中打开。

```
plotly.offline.plot(
   { "data": data,"layout": go.Layout(title = "hello world")}, auto_open = True)
```



Plotly – 使用 Jupyter Notebook 内联绘图

在本章中，我们将学习如何使用 Jupyter Notebook 进行内联绘图。

为了在笔记本内显示绘图，您需要按如下方式启动绘图的笔记本模式 –

```
from plotly.offline import init_notebook_mode
init_notebook_mode(connected = True)
```

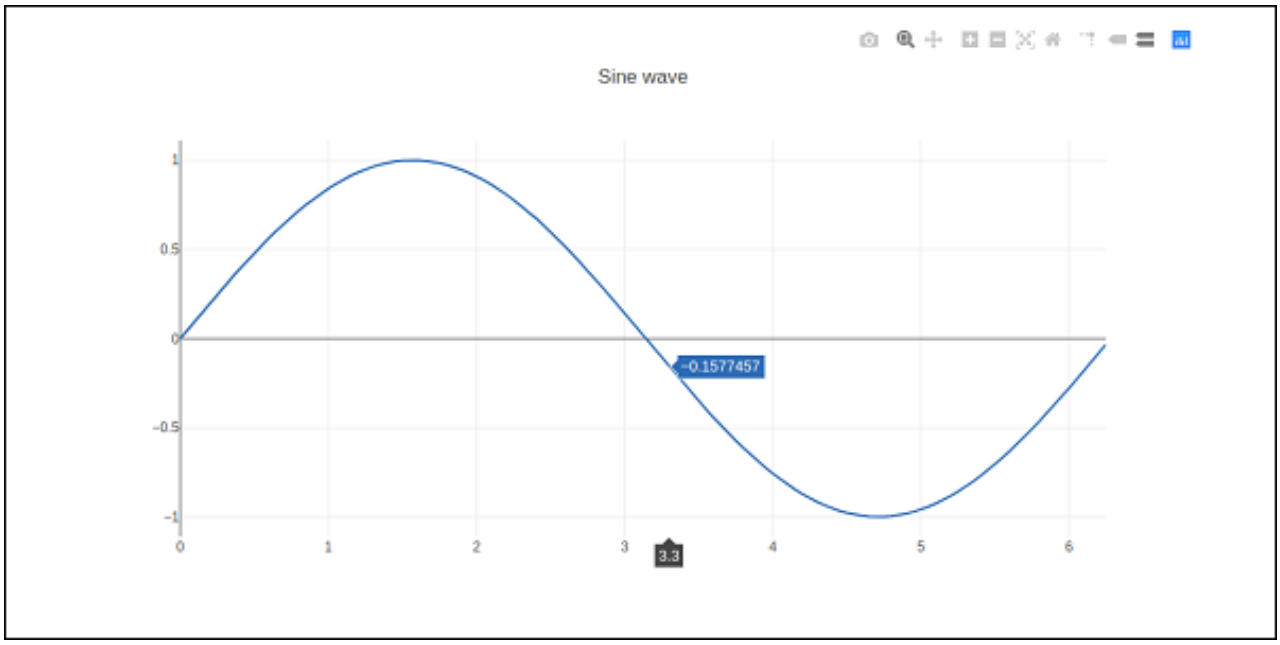
保持脚本的其余部分不变，然后按**Shift+Enter**运行笔记本单元。图表将在笔记本内部离线显示。

```
import plotly
plotly.tools.set_credentials_file(username = 'lathkar', api_key = '************')
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode(connected = True)
```

```
import plotly
import plotly.graph_objs as go
import numpy as np
import math #needed for definition of pi

xpoints = np.arange(0, math.pi*2, 0.05)
ypoints = np.sin(xpoints)
trace0 = go.Scatter(
   x = xpoints, y = ypoints
)
data = [trace0]
plotly.offline.iplot({ "data": data,"layout": go.Layout(title="Sine wave")})
```

Jupyter 笔记本输出如下所示 –



绘图输出在**右上角**显示一个**工具栏**。它包含用于下载为**png、放大和缩小、框和套索、选择和悬停的按钮。**

Plotly – 包结构

Plotly Python 包具有三个主要模块，如下所示 –

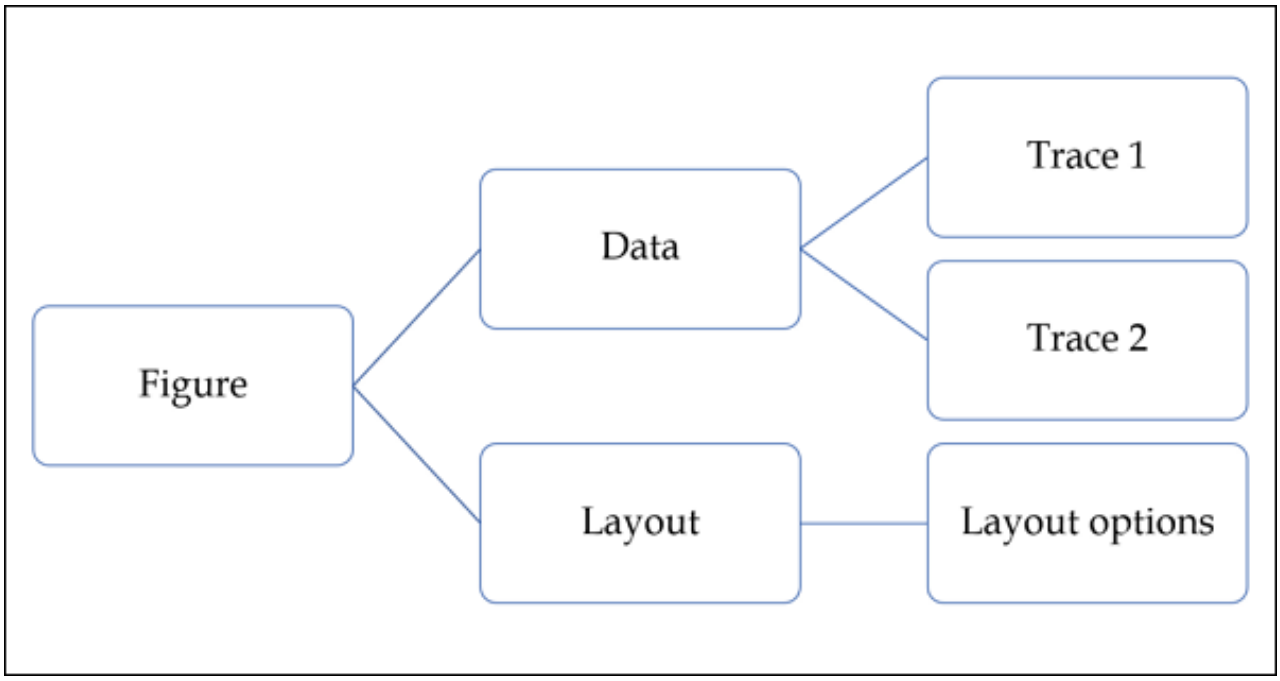
- plotly.plotly
- plotly.graph_objs
- 绘图工具

该**plotly.plotly模块**包含需要从Plotly的服务器的响应功能。该模块中的函数是本地机器和 Plotly 之间的接口。

该**plotly.graph_objs模块**是包含组成你看到图中的对象的所有类定义的最重要的模块。定义了以下图形对象 –

- 数字，
- 数据，

- 约，
- 不同的图形轨迹，如**Scatter**、**Box**、**Histogram**等。



所有图形对象都是类似字典和列表的对象，用于生成和/或修改 Plotly 图的每个特征。

该**plotly.tools模块**包含了许多有用的功能，促进和提高Plotly经验。此模块中定义了**子图生成**、在**IPython 笔记本中**嵌入 Plotly 图、保存和检索您的凭据的函数。

绘图由 Figure 对象表示，该对象表示在**plotly.graph_objs 模块中**定义的 Figure 类。它的构造函数需要以下参数 –

```
import plotly.graph_objs as go
fig = go.Figure(data, layout, frames)
```

该**数据**参数是一个Python列表对象。它是您希望绘制的所有轨迹的列表。轨迹只是我们给要绘制的数据集合的名称。一个**跟踪**对象是根据你想如何绘制表面上的数据显示命名。

Plotly 提供了许多跟踪对象，例如**scatter**、**bar**、**pie**、**heatmap**等，每个都由**graph_objs**函数中的各个函数返回。例如：**go.scatter()**返回散点跟踪。

```
import numpy as np
import math #needed for definition of pi

xpoints=np.arange(0, math.pi*2, 0.05)
ypoints=np.sin(xpoints)

trace0 = go.Scatter(
   x = xpoints, y = ypoints
)
data = [trace0]
```

的**布局**参数定义图的外观，以及情节特征，其是无关的数据。因此，我们将能够更改标题、轴标题、注释、图例、间距、字体甚至在绘图顶部绘制形状等内容。

```
layout = go.Layout(title = "Sine wave", xaxis = {'title':'angle'}, yaxis =
{'title':'sine'})
```

一个图可以有**图标题**和轴**标题**。它还可能有注释以指示其他描述。

最后，还有一个由**go.Figure() 函数**创建的**Figure 对象**。它是一个类似字典的对象，包含数据对象和布局对象。最终绘制图形对象。

```
py.iplot(fig)
```

Plotly – 导出到静态图像

离线图形的输出可以导出为各种光栅和矢量图像格式。为此，我们需要安装两个依赖项—— **orca**和**psutil**。

逆戟鲸

Orca 代表**开源报告创建者应用程序**。它是一个 Electron 应用程序，可从命令行生成绘图、破折号应用程序、仪表板的图像和报告。Orca 是 Plotly 图像服务器的支柱。

实用工具

psutil （**python 系统和进程实用程序**）是一个跨平台库，用于在 Python 中检索有关正在运行的进程和系统利用率的信息。它实现了**UNIX**命令行工具提供的许多功能，例如：**ps**、**top**、**netstat**、**ifconfig**、**who**等。 psutil 支持所有主要操作系统，例如 Linux、Windows 和 MacOs

Orca 和 psutil 的安装

如果您使用的是 Python 的 Anaconda 发行版，则 conda**包管理器**可以很容易地安装 orca 和 psutil ，如下所示 –

```
conda install -c plotly plotly-orca psutil
```

因为，orca 在 PyPi 存储库中不可用。您可以改为使用**npm 实用程序**来安装它。

```
npm install -g electron@1.8.4 orca
```

使用pip安装psutil

```
pip install psutil
```

如果您无法使用 npm 或 conda，也可以从以下网站下载 orca 的预构建二进制文件，该网站位于 https://github.com/plotly/orca/releases。

要将 Figure 对象导出为 png、jpg 或 WebP 格式，首先，导入**plotly.io**模块

```
import plotly.io as pio
```

现在，我们可以调用**write_image()**函数如下 –

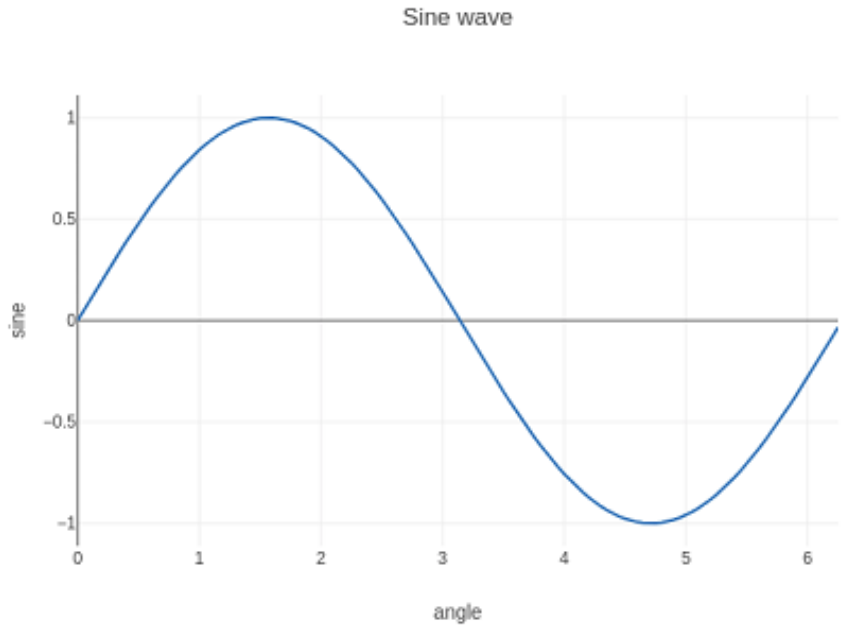
```
pio.write_image(fig, 'sinewave.png')
pio.write_image(fig, 'sinewave.jpeg')
pio.write_image(fig,'sinewave.webp)
```

orca 工具还支持导出为 svg、pdf 和 eps 格式。

```
Pio.write_image(fig, 'sinewave.svg')
pio.write_image(fig, 'sinewave.pdf')
```

在**Jupyter notebook 中**，通过**pio.to_image()**函数获得的图像对象可以内联显示如下 –

```
In [6]: from IPython.display import Image
        img_bytes = pio.to_image(fig, format='png')
        Image(img_bytes)
Out[6]:
```



情节 – 传奇

默认情况下，具有多条轨迹的 Plotly 图表会自动显示图例。如果它只有一条迹线，则不会自动显示。要显示，请将 Layout 对象的**showlegend**参数设置为 True。

```
layout = go.Layoyt(showlegend = True)
```

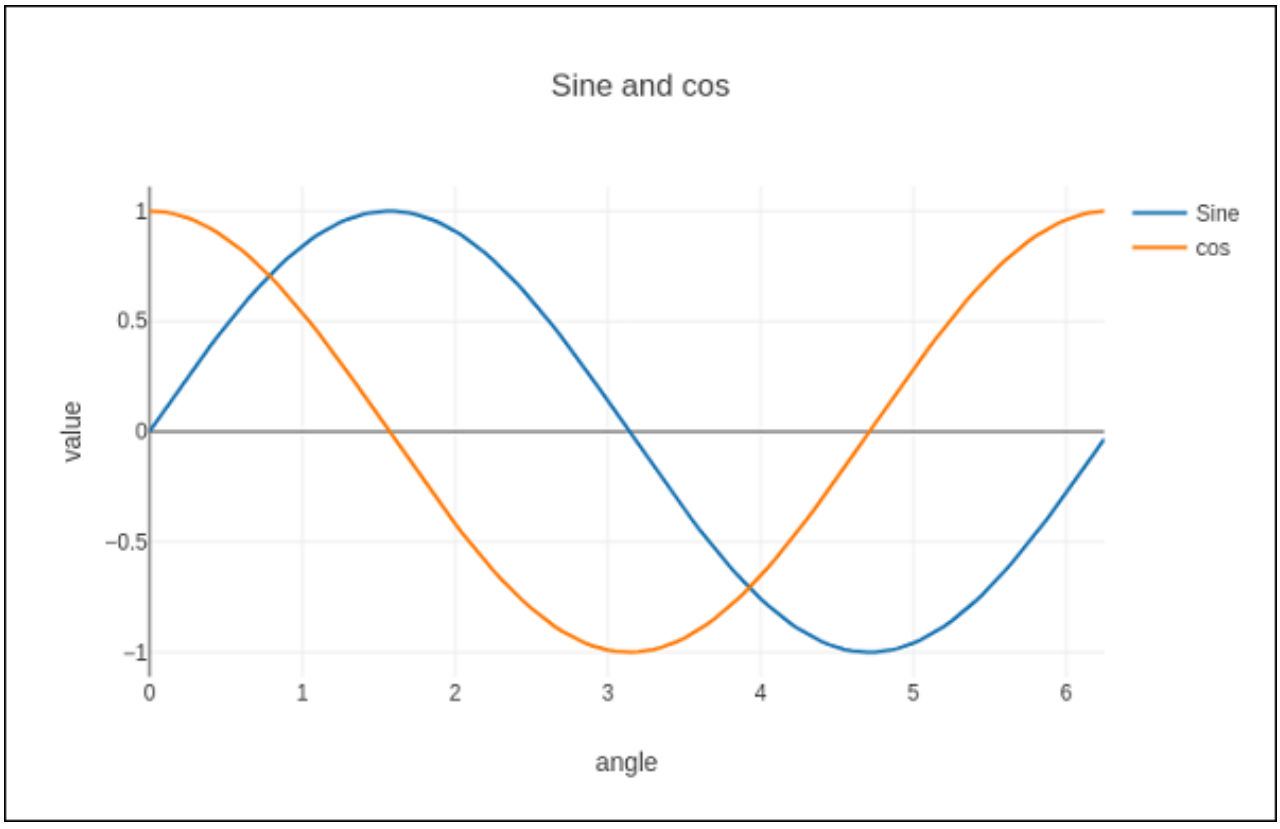
图例的默认标签是跟踪对象名称。要设置图例标签，请显式设置跟踪的名称属性。

在以下示例中，绘制了两个具有 name 属性的散点轨迹。

```
import numpy as np
import math #needed for definition of pi

xpoints = np.arange(0, math.pi*2, 0.05)
y1 = np.sin(xpoints)
y2 = np.cos(xpoints)
trace0 = go.Scatter(
   x = xpoints,
   y = y1,
   name='Sine'
)
trace1 = go.Scatter(
   x = xpoints,
   y = y2,
   name = 'cos'
)
data = [trace0, trace1]
layout = go.Layout(title = "Sine and cos", xaxis = {'title':'angle'}, yaxis =
{'title':'value'})
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

该图如下所示 –



Plotly – 格式化轴和刻度

您可以通过指定线宽和颜色来配置每个轴的外观。也可以定义网格宽度和网格颜色。让我们在本章中详细了解相同的内容。

使用轴和刻度线绘图

在 Layout 对象的属性中，将**showticklabels**设置为 true 将启用刻度。tickfont 属性是一个 dict 对象，指定字体名称、大小、颜色等。tickmode 属性可以有两个可能的值——线性和数组。如果它是线性的，则起始刻度的位置由**tick0**决定，**刻度**之间的步进由**dtick**属性决定。

如果**tickmode**设置为数组，则必须提供值和标签列表作为**tickval**和**ticktext**属性。

Layout 对象还具有设置为**“e”**的**Exponentformat**属性，将导致刻度值以科学计数法显示。您还需要将**showexponent**属性设置为**‘all’**。

我们现在通过指定线条、网格和标题字体属性以及刻度模式、值和字体来格式化上面示例中的 Layout 对象以配置 x 和**y 轴**。

```
layout = go.Layout(
   title = "Sine and cos",
   xaxis = dict(
      title = 'angle',
      showgrid = True,
      zeroline = True,
      showline = True,
      showticklabels = True,
      gridwidth = 1
   ),
   yaxis = dict(
      showgrid = True,
      zeroline = True,
      showline = True,
      gridcolor = '#bdbdbd',
      gridwidth = 2,
      zerolinecolor = '#969696',
      zerolinewidth = 2,
      linecolor = '#636363',
      linewidth = 2,
      title = 'VALUE',
      titlefont = dict(
         family = 'Arial, sans-serif',
         size = 18,
         color = 'lightgrey'
      ),
      showticklabels = True,
      tickangle = 45,
      tickfont = dict(
      family = 'Old Standard TT, serif',
      size = 14,
      color = 'black'
      ),
```

```
      tickmode = 'linear',
      tick0 = 0.0,
      dtick = 0.25
   )
)
```

多轴绘图

有时在图形中具有双 x**或 y 轴**很有用；例如，一起绘制具有不同单位的曲线时。Matplotlib支持这一与**twinx**和**twiny**功能。在以下示例中，该图具有**双 y 轴**，一个显示**exp(x)**，另一个显示**log(x)**

```
x = np.arange(1,11)
y1 = np.exp(x)
y2 = np.log(x)
trace1 = go.Scatter(
   x = x,
   y = y1,
   name = 'exp'
)
trace2 = go.Scatter(
   x = x,
   y = y2,
   name = 'log',
   yaxis = 'y2'
)
data = [trace1, trace2]
layout = go.Layout(
   title = 'Double Y Axis Example',
```

```
   yaxis = dict(
      title = 'exp',zeroline=True,
      showline = True
   ),
   yaxis2 = dict(
      title = 'log',
      zeroline = True,
      showline = True,
      overlaying = 'y',
      side = 'right'
   )
)
fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

此处，附加 y 轴配置为**yaxis2**并显示在右侧，标题为**"log"**。结果图如下 –

Plotly – 子图和插入图

在这里，我们将了解 Plotly 中的 subplots 和 inset plots 的概念。

制作子图

有时，并排比较不同的数据视图会很有帮助。这支持子图的概念。它在**plotly.tools 模块中**提供**make_subplots()**函数。该函数返回一个 Figure 对象。

以下语句在一行中创建两个子图。

```
fig = tools.make_subplots(rows = 1, cols = 2)
```

我们现在可以向图中添加两个不同的跟踪（上面示例中的 exp 和 log 跟踪）。

```
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
```

通过使用**update()**方法指定**标题**、**宽度**、**高度**等进一步配置图形的布局。

```
fig['layout'].update(height = 600, width = 800s, title = 'subplots')
```

这是完整的脚本 –

```
from plotly import tools
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode(connected = True)
import numpy as np
```

```
x = np.arange(1,11)
y1 = np.exp(x)
y2 = np.log(x)
trace1 = go.Scatter(
   x = x,
   y = y1,
   name = 'exp'
)
trace2 = go.Scatter(
   x = x,
   y = y2,
   name = 'log'
)
fig = tools.make_subplots(rows = 1, cols = 2)
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig['layout'].update(height = 600, width = 800, title = 'subplot')
iplot(fig)
```

这是绘图网格的格式：[(1,1) x1,y1] [(1,2) x2,y2]

插图

要将子图显示为插图，我们需要配置其跟踪对象。首先，插图的**xaxis**和**yaxis**属性分别跟踪到**'x2'**和**'y2'**。以下语句将**"日志"**跟踪放入插图中。

```
trace2 = go.Scatter(
   x = x,
   y = y2,
   xaxis = 'x2',
   yaxis = 'y2',
   name = 'log'
)
```

其次，配置 Layout 对象，其中插入的 x 和 y 轴的位置由指定相对于长轴的位置的**域**属性定义。

```
xaxis2=dict(
   domain = [0.1, 0.5],
   anchor = 'y2'
),
yaxis2 = dict(
   domain = [0.5, 0.9],
   anchor = 'x2'
)
```

下面给出了在插图中显示日志跟踪和主轴上的 exp 跟踪的完整脚本 –

```
trace1 = go.Scatter(
   x = x,
   y = y1,
   name = 'exp'
)
trace2 = go.Scatter(
   x = x,
   y = y2,
   xaxis = 'x2',
   yaxis = 'y2',
   name = 'log'
)
data = [trace1, trace2]
layout = go.Layout(
   yaxis = dict(showline = True),
   xaxis2 = dict(
      domain = [0.1, 0.5],
      anchor = 'y2'
   ),
   yaxis2 = dict(
      showline = True,
      domain = [0.5, 0.9],
      anchor = 'x2'
   )
)
fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

下面提到了输出 -

Plotly – 条形图和饼图

在本章中，我们将学习如何在 Plotly 的帮助下制作条形图和饼图。让我们从了解条形图开始。

条形图

条形图使用矩形条显示分类数据，矩形条的高度或长度与其所代表的值成正比。条可以垂直或水平显示。它有助于显示离散类别之间的比较。图表的一个轴显示正在比较的特定类别，另一个轴表示测量值。

以下示例绘制了一个关于注册不同课程的学生人数的简单**条形图**。该**go.Bar（）**函数返回以x酒吧跟踪坐标集为主题列表，y坐标为学生人数。

```
import plotly.graph_objs as go
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
data = [go.Bar(
   x = langs,
   y = students
)]
fig = go.Figure(data=data)
iplot(fig)
```

输出将如下所示 –

要显示分组条形图，必须将 Layout 对象的**barmode**属性设置为**group**。在以下代码中，代表每年学生的多条轨迹针对科目绘制并显示为分组条形图。

```
branches = ['CSE', 'Mech', 'Electronics']
fy = [23,17,35]
sy = [20, 23, 30]
ty = [30,20,15]
trace1 = go.Bar(
   x = branches,
   y = fy,
   name = 'FY'
)
trace2 = go.Bar(
   x = branches,
   y = sy,
   name = 'SY'
)
trace3 = go.Bar(
   x = branches,
   y = ty,
   name = 'TY'
)
data = [trace1, trace2, trace3]
layout = go.Layout(barmode = 'group')
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

相同的输出如下 –

所述**barmode**属性确定如何在相同的位置坐标条被显示在图表上。定义的值是“堆叠”（条形堆叠在另一个顶部），“相对”，（条形堆叠在另一个顶部，轴下方为负值，轴上方为正值），“**组**”（条形图旁边绘制另一个）。

通过将 barmode 属性更改为“**堆栈**”，绘制的图形如下所示 –

饼形图

饼图仅显示一系列数据。**饼图**显示一个数据系列中项目（称为**楔形**）的大小，与项目的总和成正比。数据点显示为整个饼图的百分比。

graph_objs模块中的**pie()**函数– **go.Pie()**，返回 Pie 跟踪。两个必需的参数是**标签**和**值**。让我们在此处给出的示例中绘制一个简单的语言课程与学生人数的饼图。

```
import plotly
plotly.tools.set_credentials_file(
   username = 'lathkar', api_key = 'U7vgRe1hqmRp4ZNf4PTN'
)
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode(connected = True)
import plotly.graph_objs as go
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
trace = go.Pie(labels = langs, values = students)
data = [trace]
fig = go.Figure(data = data)
iplot(fig)
```

以下输出显示在 Jupyter 笔记本中 –

甜甜圈图是一个饼图，中间有一个圆孔，使它看起来像一个甜甜圈。在以下示例中，两个圆环图以 1X2 网格布局显示。虽然两个饼图的“**标签**”布局相同，但每个子图的行和列目标由域属性决定。

为此，我们使用了 2019 年议会选举中党派席位和投票份额的数据。在 Jupyter 笔记本单元格中输入以下代码 –

```
parties = ['BJP', 'CONGRESS', 'DMK', 'TMC', 'YSRC', 'SS', 'JDU','BJD', 'BSP','OTH']
seats = [303,52,23,22,22,18,16,12,10, 65]
percent = [37.36, 19.49, 2.26, 4.07, 2.53, 2.10, 1.46, 1.66, 3.63, 25.44]
import plotly.graph_objs as go
data1 = {
   "values": seats,
   "labels": parties,
   "domain": {"column": 0},
   "name": "seats",
   "hoverinfo":"label+percent+name",
   "hole": .4,
   "type": "pie"
}
data2 = {
   "values": percent,
   "labels": parties,
   "domain": {"column": 1},
   "name": "vote share",
   "hoverinfo":"label+percent+name",
   "hole": .4,
   "type": "pie"
}
```

```
data = [data1,data2]
layout = go.Layout(
   {
      "title":"Parliamentary Election 2019",
      "grid": {"rows": 1, "columns": 2},
      "annotations": [
         {
            "font": {
               "size": 20
            },
            "showarrow": False,
            "text": "seats",
            "x": 0.20,
            "y": 0.5
         },
         {
            "font": {
               "size": 20
            },
            "showarrow": False,
            "text": "votes",
            "x": 0.8,
            "y": 0.5
         }
      ]
   }
)
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

下面给出了相同的输出 -

散点图、散点图和气泡图

本章重点介绍散点图、散点图和气泡图的详细信息。首先，让我们研究散点图。

散点图

散点图用于在水平轴和垂直轴上**绘制数据**点，以显示一个变量如何影响另一个变量。数据表中的每一行都由一个标记表示，其位置取决于其在**X**轴和**Y**轴上设置的列中的值。

graph_objs 模块**(go.Scatter)**的**scatter()**方法产生散点轨迹。在这里，**模式**属性决定了数据点的外观。mode 的默认值是 lines，它显示连接数据点的连续线。如果设置为**marker**，则仅显示由小实心圆圈表示的数据点。当模式指定为“线+标记”时，圆和线都会显示。

在以下示例中，绘制笛卡尔坐标系中三组随机生成的点的散点轨迹。下面解释了以不同模式属性显示的每个轨迹。

```
import numpy as np
N = 100
```

```
x_vals = np.linspace(0, 1, N)
y1 = np.random.randn(N) + 5
y2 = np.random.randn(N)
y3 = np.random.randn(N) - 5
trace0 = go.Scatter(
   x = x_vals,
   y = y1,
   mode = 'markers',
   name = 'markers'
)
trace1 = go.Scatter(
   x = x_vals,
   y = y2,
   mode = 'lines+markers',
   name = 'line+markers'
)
trace2 = go.Scatter(
   x = x_vals,
   y = y3,
   mode = 'lines',
   name = 'line'
)
data = [trace0, trace1, trace2]
fig = go.Figure(data = data)
iplot(fig)
```

Jupyter notebook 单元的输出如下所示 –

散点图

WebGL（Web 图形库）是一种 JavaScript API，用于在任何兼容的 Web 浏览器中渲染交互式**2D**和**3D 图形**，而无需使用插件。WebGL 与其他 Web 标准完全集成，允许图形处理单元 (GPU) 加速使用图像处理。

Plotly 您可以使用**Scattergl()**代替 Scatter()来实现 WebGL，以提高速度、改进交互性以及绘制更多数据的能力。该**go.scattergl（）**函数，它给出了当涉及大量的数据点的更好的性能。

```
import numpy as np
N = 100000
x = np.random.randn(N)
y = np.random.randn(N)
   trace0 = go.Scattergl(
   x = x, y = y, mode = 'markers'
)
data = [trace0]
layout = go.Layout(title = "scattergl plot ")
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

下面提到了输出 –

气泡图

气泡图显示数据的三个维度。每个实体及其关联数据的三个维度被绘制为一个**圆盘**（气泡），通过圆盘的**xy 位置**表示其中两个维度，通过其大小表示第三个维度。气泡的大小由第三个数据系列中的值决定。

气泡图是散点图的一种变体，其中数据点被气泡替换。如果您的数据具有如下所示的三个维度，那么创建气泡图将是一个不错的选择。

Company	产品	销售	分享
A	13	2354	23
B	6	5423	47
C	23	2451	30

气泡图是使用**go.Scatter()**跟踪生成的。上述数据系列中的两个作为 x 和 y 属性给出。第三维由标记显示，其大小代表第三个数据系列。在上述情况下，我们使用**产品**和**销售**作为**x**和**y**属性，**市场份额**作为**标记大小**。

在 Jupyter Notebook 中输入以下代码。

```
company = ['A','B','C']
products = [13,6,23]
sale = [2354,5423,4251]
share = [23,47,30]
fig = go.Figure(data = [go.Scatter(
   x = products, y = sale,
   text = [
      'company:'+c+' share:'+str(s)+'%'
      for c in company for s in share if company.index(c)==share.index(s)
   ],
   mode = 'markers',
   marker_size = share, marker_color = ['blue','red','yellow'])
])
iplot(fig)
```

输出如下所示 –

Plotly – 点图和表格

在这里，我们将学习 Plotly 中的点图和表格函数。首先，让我们从点图开始。

点图

点图以非常简单的比例显示点。它仅适用于少量数据，因为大量点会使其看起来非常混乱。点图也称为**克利夫兰点图**。它们显示两个（或更多）时间点之间或两个（或更多）条件之间的变化。

点图类似于水平条形图。但是，它们可以不那么混乱，并且可以更轻松地比较条件。该图绘制了一条散点轨迹，模式属性设置为标记。

以下示例显示了印度独立后每次人口普查中记录的男性和女性识字率的比较。图中的两条轨迹代表了 1951 年至 2011 年每次人口普查中男性和女性的识字率。

```
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode(connected = True)
census = [1951,1961,1971,1981,1991,2001, 2011]
x1 = [8.86, 15.35, 21.97, 29.76, 39.29, 53.67, 64.63]
x2 = [27.15, 40.40, 45.96, 56.38,64.13, 75.26, 80.88]
traceA = go.Scatter(
   x = x1,
   y = census,
   marker = dict(color = "crimson", size = 12),
   mode = "markers",
   name = "Women"
)
traceB = go.Scatter(
x = x2,
y = census,
marker = dict(color = "gold", size = 12),
mode = "markers",
name = "Men")
data = [traceA, traceB]
layout = go.Layout(
   title = "Trend in Literacy rate in Post independent India",
   xaxis_title = "percentage",
   yaxis_title = "census"
)
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

输出如下所示 –

Plotly 中的表格

Plotly 的 Table 对象由**go.Table()**函数返回。表跟踪是一个图形对象，可用于在行和列的网格中查看详细数据。表使用列主序，即网格被表示为列向量的向量。

的两个重要参数**go.Table（）**函数是**首部**，其是表和的第一行**的细胞**，其形成行的其余部分。这两个参数都是字典对象。headers 的 values 属性是一个列标题列表和一个列表列表，每个列表对应一行。

进一步的样式定制由 linecolor、fill_color、font 和其他属性完成。

以下代码显示了最近结束的 2019 年板球世界杯循环赛阶段的积分表。

```
trace = go.Table(
   header = dict(
```

```
      values = ['Teams','Mat','Won','Lost','Tied','NR','Pts','NRR'],
      line_color = 'gray',
      fill_color = 'lightskyblue',
      align = 'left'
   ),
   cells = dict(
      values =
      [
         [
            'India',
            'Australia',
            'England',
            'New Zealand',
            'Pakistan',
            'Sri Lanka',
            'South Africa',
            'Bangladesh',
            'West Indies',
            'Afghanistan'
         ],
         [9,9,9,9,9,9,9,9,9,9],
         [7,7,6,5,5,3,3,3,2,0],
         [1,2,3,3,3,4,5,5,6,9],
         [0,0,0,0,0,0,0,0,0,0],
         [1,0,0,1,1,2,1,1,1,0],
         [15,14,12,11,11,8,7,7,5,0],
         [0.809,0.868,1.152,0.175,-0.43,-0.919,-0.03,-0.41,-0.225,-1.322]
      ],
      line_color='gray',
      fill_color='lightcyan',
      align='left'
   )
)
data = [trace]
fig = go.Figure(data = data)
iplot(fig)
```

输出如下所述 –

表数据也可以从 Pandas 数据框填充。让我们创建一个逗号分隔的文件（**points-table.csv**），如下所示 –

Teams	垫	韩元	丢失的	绑	NR	分	NRR
India	9	7	1	0	1	15	0.809
Australia	9	7	2	0	0	14	0.868
England	9	6	3	0	0	14	1.152
New Zealand	9	5	3	0	1	11	0.175
Pakistan	9	5	3	0	1	11	-0.43
Sri Lanka	9	3	4	0	2	8	-0.919
South Africa	9	3	5	0	1	7	-0.03
Bangladesh	9	3	5	0	1	7	-0.41

```
Teams,Matches,Won,Lost,Tie,NR,Points,NRR
India,9,7,1,0,1,15,0.809
Australia,9,7,2,0,0,14,0.868
England,9,6,3,0,0,12,1.152
New Zealand,9,5,3,0,1,11,0.175
Pakistan,9,5,3,0,1,11,-0.43
Sri Lanka,9,3,4,0,2,8,-0.919
South Africa,9,3,5,0,1,7,-0.03
Bangladesh,9,3,5,0,1,7,-0.41
West Indies,9,2,6,0,1,5,-0.225
Afghanistan,9,0,9,0,0,0,-1.322
```

我们现在从这个 csv 文件构建一个数据帧对象，并使用它来构建表跟踪，如下所示 –

```
import pandas as pd
df = pd.read_csv('point-table.csv')
trace = go.Table(
   header = dict(values = list(df.columns)),
   cells = dict(
      values = [
         df.Teams,
         df.Matches,
         df.Won,
         df.Lost,
         df.Tie,
         df.NR,
         df.Points,
         df.NRR
      ]
   )
)
data = [trace]
```

```
fig = go.Figure(data = data)
iplot(fig)
```

情节 – 直方图

由 Karl Pearson 引入，直方图是数值数据分布的准确表示，它是对连续变量 (CORAL) 概率分布的估计。它看起来类似于条形图，但是，条形图涉及两个变量，而直方图只涉及一个变量。

直方图需要**bin**（或**bucket**）将整个值范围划分为一系列区间——然后计算每个区间有多少值。bin 通常指定为变量的连续、非重叠区间。垃圾箱必须相邻，并且通常大小相同。在箱子上方竖立一个矩形，其高度与频率（每个箱子中的病例数）成正比。

直方图跟踪对象由**go.Histogram()**函数返回。它的定制是通过各种参数或属性完成的。一个基本参数是 x 或 y 设置为列表、**numpy 数组**或**Pandas 数据帧对象**，这些**对象**将分布在 bin 中。

默认情况下，Plotly 将数据点分布在自动调整大小的 bin 中。但是，您可以定义自定义 bin 大小。为此，您需要将 autobins 设置为 false，指定**nbins**（bin 数量）、其开始和结束值以及大小。

以下代码生成一个简单的直方图，显示学生在班级内的分数分布（自动调整大小） –

```
import numpy as np
x1 = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
data = [go.Histogram(x = x1)]
fig = go.Figure(data)
iplot(fig)
```

输出如下所示 –

所述**go.Histogram（）**函数接受**histnorm**，其指定用于该直方图跟踪正常化的类型。默认为“”，每个条形的跨度对应于出现的次数（即位于 bin 内的数据点数）。如果分配了**“percent” / “probability”**，则每个条形的跨度对应于相对于样本点总数的百分比/发生率。如果它等于“**密度**”，则每个条形的跨度对应于 bin 中出现的次数除以 bin 间隔的大小。

还有一个**histfunc**参数，其默认值为**count**。因此，bin 上矩形的高度对应于数据点的计数。它可以设置为 sum、avg、min 或 max。

的**直方图（）**函数可以被设置为显示在连续的二进制位值的累积分布。为此，您需要将**累积属性**设置为启用。结果如下所示 –

```
data=[go.Histogram(x = x1, cumulative_enabled = True)]
fig = go.Figure(data)
iplot(fig)
```

输出如下所述 –

Plotly – 箱线图小提琴图和轮廓图

本章重点详细了解各种绘图，包括箱形图、小提琴图、等高线图和箭袋图。最初，我们将从箱线图开始。

箱形图

箱线图显示一组数据的摘要，其中包含最小值、**第一四分位数**、**中位数**、**第三四分位数**和**最大值**。在箱线图中，我们从第一个四分位数到第三个四分位数绘制一个方框。一条垂直线穿过中间的盒子。从表示上下四分位数之外的可变性的框垂直延伸的线称为须线。因此，箱线图也称为箱线**图**。胡须从每个四分位数到最小值或最大值。

要绘制箱形图，我们必须使用**go.Box()**函数。数据系列可以分配给 x 或 y 参数。因此，箱线图将水平或垂直绘制。在下面的示例中，某公司在其各个分支机构的销售数据被转换为水平箱线图。它显示了最小值和最大值的中位数。

```
trace1 = go.Box(y = [1140,1460,489,594,502,508,370,200])
data = [trace1]
fig = go.Figure(data)
iplot(fig)
```

相同的输出如下 –

所述**go.Box**（）函数可以给出各种其它参数来控制箱线图的外观和行为。其中之一是 boxmean 参数。

该**boxmean**参数默认设置为true。因此，框的基础分布的平均值被绘制为框内的虚线。如果设置为sd，也会绘制分布的标准差。

该**boxpoints**参数的缺省设置为“**异常值**”。仅显示位于晶须外部的样本点。如果是“疑似异常值”，则显示异常值点，并且突出显示小于 4"Q1-3"Q3 或大于 4"Q3-3"Q1 的点。如果为“False”，则仅显示框而没有样本点。

在以下示例中，**框线**是用标准差和异常点绘制的。

```
trc = go.Box(
   y = [
      0.75, 5.25, 5.5, 6, 6.2, 6.6, 6.80, 7.0, 7.2, 7.5, 7.5, 7.75, 8.15,
      8.15, 8.65, 8.93, 9.2, 9.5, 10, 10.25, 11.5, 12, 16, 20.90, 22.3, 23.25
   ],
   boxpoints = 'suspectedoutliers', boxmean = 'sd'
)
data = [trc]
fig = go.Figure(data)
iplot(fig)
```

相同的输出如下所述 –

小提琴情节

小提琴图类似于箱线图，不同之处在于它们还显示了数据在不同值下的概率密度。小提琴图将包括数据中位数的标记和指示四分位距的框，如标准箱线图。叠加在此箱线图上的是核密度估计。与箱线图一样，小提琴图用于表示不同“类别”之间的变量分布（或样本分布）的比较。

小提琴图比普通箱线图提供更多信息。事实上，箱线图仅显示汇总统计数据，例如均值/中位数和四分位距，而小提琴图则显示**了数据**的**完整分布**。

小提琴跟踪对象由**graph_objects**模块中的**go.Violin()**函数返回。为了显示底层箱线图，**boxplot_visible**属性设置为 True。同样，通过将**meanline_visible**属性设置为 true，小提琴内会显示一条与样本均值相对应的线。

以下示例演示了如何使用 plotly 的功能显示小提琴图。

```
import numpy as np
np.random.seed(10)
c1 = np.random.normal(100, 10, 200)
c2 = np.random.normal(80, 30, 200)
trace1 = go.Violin(y = c1, meanline_visible = True)
trace2 = go.Violin(y = c2, box_visible = True)
data = [trace1, trace2]
fig = go.Figure(data = data)
iplot(fig)
```

输出如下 –

等高线图

二维等高线图显示了一个二维数字阵列z的轮廓线，即，内插的行**isovalues** z与。两个变量的函数的等高线是一条曲线，沿着该曲线该函数具有恒定值，因此该曲线连接等值的点。

如果您想查看某个值 Z 如何作为两个输入**X**和**Y**的函数而变化，使得**Z = f(X,Y)** ，则使用等高线图是合适的。两个变量的函数的等高线或等值线是函数沿其具有恒定值的曲线。

自变量 x 和 y 通常被限制在称为 meshgrid 的规则网格中。numpy.meshgrid 从 x 值数组和 y 值数组中创建一个矩形网格。

让我们首先使用Numpy 库中的**linspace()**函数为 x、y 和 z 创建数据值。我们从 x 和 y 值创建一个网**格**，并获得由**x2+y2 的**平方根组成的 z 数组

我们在**graph_objects**模块中有**go.Contour()**函数，它接受 x、**y**和**z**属性。以下代码片段显示了如上计算的 x、**y**和**z**值的等高线图。

```
import numpy as np
xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
trace = go.Contour(x = xlist, y = ylist, z = Z)
data = [trace]
fig = go.Figure(data)
iplot(fig)
```

输出如下 –

可以通过以下一个或多个参数自定义等高线图 –

- **Transpose (boolean)** – 转置 z 数据。

如果**xtype**（或**ytype**）等于“array”，则 x/y 坐标由“x”/“y”给出。如果“缩放”，x 坐标由“x0”和“ **dx** ”给出。

- 所述**connectgaps**参数确定z数据是否间隙填充。
- **ncontours**参数的默认值为15。实际轮廓数将自动选择小于或等于 `ncontours` 的值。仅当 `autocontour` 为“True”时有效。

等高线类型默认为：“ **levels** ”，因此数据表示为显示多个级别的等高线图。如果为**constrain**，则数据表示为约束，其中无效区域按**操作**和**值**参数指定的阴影显示。

showlines – 确定是否绘制轮廓线。

zauto默认为**True**并确定是否根据输入数据（此处为“ **z** ”）或在“ **zmin** ”和“ **zmax** ”中设置的边界计算颜色域。当“ **zmin** ”和“ **zmin** ”时，默认为“ **False** ” `zmax` 由用户设置。

箭袋图

箭袋图也称为**速度图**。它将速度矢量显示为在点 (x,y) 处具有分量 (**u,v**) 的箭头。为了绘制 Quiver 图，我们将使用**Plotly**中的**figure_factory**模块中定义的**create_quiver()**函数。

Plotly 的 Python API 包含一个图形工厂模块，该模块包含许多包装函数，这些函数可以创建尚未包含在**Plotly**的开源图形库 plotly.js 中的独特图表类型。

create_quiver() 函数接受以下参数 –

- **x** – 箭头位置的 x 坐标
- **y** – y 箭头位置的坐标
- **u** – x 箭头向量的分量
- **v** – y 箭头向量的分量
- **scale** – 缩放箭头的大小
- **arrow_scale** – 箭头的长度。
- **角度**– 箭头的角度。

以下代码在 Jupyter 笔记本中呈现一个简单的箭袋图 –

```
import plotly.figure_factory as ff
import numpy as np
x,y = np.meshgrid(np.arange(-2, 2, .2), np.arange(-2, 2, .25))
z = x*np.exp(-x**2 - y**2)
v, u = np.gradient(z, .2, .2)

# Create quiver figure
fig = ff.create_quiver(x, y, u, v,
scale = .25, arrow_scale = .4,
name = 'quiver', line = dict(width = 1))
iplot(fig)
```

代码的输出如下 –

Plotly – Distplots 密度图和误差条图

在本章中，我们将详细了解 distplots、密度图和误差条图。让我们从学习 distplots 开始。

分布图

distplot 图工厂显示数值数据的统计表示的组合，例如直方图、核密度估计或正态曲线和地毯图。

distplot 可以由以下 3 个组件的全部或任意组合组成 –

- 直方图
- 曲线：（a）核密度估计或（b）正态曲线，以及
- 地毯图

所述**figure_factory**模块具有**create_distplot（）**函数，其需要称为hist_data强制参数。

以下代码创建了一个基本的 distplot，由直方图、kde 图和 rug 图组成。

```
x = np.random.randn(1000)
hist_data = [x]
group_labels = ['distplot']
fig = ff.create_distplot(hist_data, group_labels)
iplot(fig)
```

上述代码的输出如下 –

密度图

密度图是根据数据估计的直方图的平滑连续版本。最常见的估计形式称为**核密度估计 (KDE)**。在这种方法中，在每个单独的数据点绘制一条连续曲线（内核），然后将所有这些曲线加在一起以进行单个平滑的密度估计。

模块**plotly.figure_factory._2d_density 中**的**create_2d_density()**函数返回一个用于 2D 密度图的图形对象。

以下代码用于在直方图数据上生成二维密度图。

```
t = np.linspace(-1, 1.2, 2000)
x = (t**3) + (0.3 * np.random.randn(2000))
y = (t**6) + (0.3 * np.random.randn(2000))
fig = ff.create_2d_density( x, y)
iplot(fig)
```

下面提到的是上面给出的代码的输出。

误差条图

误差棒是数据中误差或不确定性的图形表示，它们有助于正确解释。出于科学目的，错误报告对于理解给定数据至关重要。

误差线对问题解决者很有用，因为误差线显示了一组测量值或计算值的置信度或精度。

大多数误差线代表数据集的范围和标准偏差。它们可以帮助可视化数据如何围绕平均值分布。可以在各种图上生成误差线，例如条形图、折线图、散点图等。

所述**go.Scatter（）**函数具有**error_x**和**error_y**性质控制如何误差棒被生成。

- **可见（布尔值）** – 确定这组误差线是否可见。

类型属性有可能的值**“百分比”**| **“常数”** | **” sqrt ”** | **“ data ”**。它设置用于生成误差线的规则。如果是“百分比”，则柱线长度对应于基础数据的百分比。在 `value` 中设置此百分比。如果是“sqrt”，则柱线长度对应于基础数据的平方。如果是“data”，条形长度用数据集 `array` 设置。

- **对称**属性可以为真或为假。因此，误差线在两个方向上的长度或不相同（垂直条的顶部/底部，水平条的左/右。
- **数组**– 设置与每个误差条长度相对应的数据。值是相对于基础数据绘制的。
- **arrayminus** – 设置与垂直（水平）条的底部（左）方向上每个误差条的长度相对应的数据相对于基础数据绘制的值。

以下代码在散点图上显示对称误差条 –

```
trace = go.Scatter(
   x = [0, 1, 2], y = [6, 10, 2],
   error_y = dict(
   type = 'data', # value of error bar given in data coordinates
   array = [1, 2, 3], visible = True)
)
data = [trace]
layout = go.Layout(title = 'Symmetric Error Bar')
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

下面给出的是上述代码的输出。

通过以下脚本呈现非对称误差图 –

```
trace = go.Scatter(
   x = [1, 2, 3, 4],
   y =[ 2, 1, 3, 4],
   error_y = dict(
      type = 'data',
      symmetric = False,
      array = [0.1, 0.2, 0.1, 0.1],
      arrayminus = [0.2, 0.4, 1, 0.2]
   )
)
data = [trace]
layout = go.Layout(title = 'Asymmetric Error Bar')
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

相同的输出如下所示 –

情节 – 热图

热图（或热图）是数据的图形表示，其中包含在矩阵中的各个值以颜色表示。热图的主要目的是更好地可视化数据集中的位置/事件数量，并帮助将查看者引导至数据可视化中最重要的区域。

由于它们依赖于颜色来传达值，因此热图可能最常用于显示更通用的数值视图。热图在引起对趋势的关注方面非常通用且有效，因此它们在分析社区中变得越来越流行。

热图本质上是不言自明的。颜色越深，数量越大（数值越高，分散越紧等）。Plotly 的 graph_objects 模块包含 Heatmap **()**函数。它需要 x、**y**和**z**属性。它们的值可以是列表、numpy 数组或 Pandas 数据框。

在以下示例中，我们有一个 2D 列表或数组，用于将数据（不同农民的收获量，以吨/年为单位）定义为颜色代码。然后我们还需要两个农民和他们种植的蔬菜的名称列表。

```
vegetables = [
   "cucumber",
   "tomato",
   "lettuce",
   "asparagus",
   "potato",
   "wheat",
   "barley"
]
farmers = [
   "Farmer Joe",
   "Upland Bros.",
   "Smith Gardening",
   "Agrifun",
   "Organiculture",
   "BioGoods Ltd.",
   "Cornylee Corp."
]
harvest = np.array(
   [
      [0.8, 2.4, 2.5, 3.9, 0.0, 4.0, 0.0],
      [2.4, 0.0, 4.0, 1.0, 2.7, 0.0, 0.0],
      [1.1, 2.4, 0.8, 4.3, 1.9, 4.4, 0.0],
      [0.6, 0.0, 0.3, 0.0, 3.1, 0.0, 0.0],
      [0.7, 1.7, 0.6, 2.6, 2.2, 6.2, 0.0],
      [1.3, 1.2, 0.0, 0.0, 0.0, 3.2, 5.1],
      [0.1, 2.0, 0.0, 1.4, 0.0, 1.9, 6.3]
   ]
)
trace = go.Heatmap(
   x = vegetables,
   y = farmers,
   z = harvest,
   type = 'heatmap',
   colorscale = 'Viridis'
```

```
)
data = [trace]
fig = go.Figure(data = data)
iplot(fig)
```

上述代码的输出如下 –

Plotly – 极坐标图和雷达图

在本章中，我们将学习如何在 Plotly 的帮助下制作极地图和雷达图。

首先，让我们研究一下极坐标图。

极地图

极坐标图是圆形图的常见变体。当数据点之间的关系可以根据半径和角度最容易地可视化时，它很有用。

在极坐标图中，一系列由连接极坐标系中的点的闭合曲线表示。每个数据点由到极点的距离（径向坐标）和到固定方向的角度（角坐标）确定。

极坐标图表示沿径向和角轴的数据。径向和角坐标由**go.Scatterpolar()**函数的**r**和**theta**参数**给出**。theta 数据可以是分类数据，但数值数据也是可能的，并且是最常用的。

以下代码生成一个基本的极坐标图。除了 r 和 theta 参数，我们将 mode 设置为**lines**（它可以很好地设置为标记，在这种情况下只会显示数据点）。

```
import numpy as np
r1 = [0,6,12,18,24,30,36,42,48,54,60]
t1 = [1,0.995,0.978,0.951,0.914,0.866,0.809,0.743,0.669,0.588,0.5]
trace = go.Scatterpolar(
   r = [0.5,1,2,2.5,3,4],
   theta = [35,70,120,155,205,240],
   mode = 'lines',
)
data = [trace]
fig = go.Figure(data = data)
iplot(fig)
```

输出如下 –

在以下示例中，来自**逗号分隔值 (CSV) 文件的**数据用于生成极坐标图。**polar.csv 的**前几行如下 –

```
y,x1,x2,x3,x4,x5,
0,1,1,1,1,1,
6,0.995,0.997,0.996,0.998,0.997,
12,0.978,0.989,0.984,0.993,0.986,
18,0.951,0.976,0.963,0.985,0.969,
24,0.914,0.957,0.935,0.974,0.946,
30,0.866,0.933,0.9,0.96,0.916,
36,0.809,0.905,0.857,0.943,0.88,
42,0.743,0.872,0.807,0.923,0.838,
48,0.669,0.835,0.752,0.901,0.792,
54,0.588,0.794,0.691,0.876,0.74,
60,0.5,0.75,0.625,0.85,0.685,
```

在笔记本的输入单元格中输入以下脚本以生成如下极坐标图 –

```
import pandas as pd
df = pd.read_csv("polar.csv")
t1 = go.Scatterpolar(
   r = df['x1'], theta = df['y'], mode = 'lines', name = 't1'
)
t2 = go.Scatterpolar(
   r = df['x2'], theta = df['y'], mode = 'lines', name = 't2'
)
t3 = go.Scatterpolar(
   r = df['x3'], theta = df['y'], mode = 'lines', name = 't3'
)
data = [t1,t2,t3]
fig = go.Figure(data = data)
iplot(fig)
```

下面给出的是上述代码的输出 –

雷达图

雷达图（也称为**蜘蛛图**或**星图**）以定量变量的二维图表的形式显示多变量数据，该图表在源自中心的轴上表示。轴的相对位置和角度通常不提供信息。

对于雷达图，在一般情况下，在**go.Scatterpolar()**函数中使用带有分类角度变量的极坐标图。

以下代码使用**Scatterpolar() 函数**渲染基本雷达图–

```
radar = go.Scatterpolar(
   r = [1, 5, 2, 2, 3],
   theta = [
      'processing cost',
      'mechanical properties',
      'chemical stability',
      'thermal stability',
```

```
      'device integration'
   ],
   fill = 'toself'
)
data = [radar]
fig = go.Figure(data = data)
iplot(fig)
```

下面提到的输出是上面给出的代码的结果 –

OHLC 图、瀑布图和漏斗图

本章重点介绍可以在 Plotly 的帮助下制作的其他三种类型的图表，包括 OHLC、瀑布图和漏斗图。

OHLC 图表

一个**开-高-低-关闭**图表（也OHLC）是一种类型的**条形图**通常用来说明在金融工具的价格运动，如股。OHLC 图表很有用，因为它们显示了一段时间内的四个主要数据点。图表类型很有用，因为它可以显示增加或减少的动量。高低数据点可用于评估波动性。

图表上的每条垂直线都显示了一个时间单位（例如天或小时）内的价格范围（最高和最低价格）。刻度线从线的每一侧突出，左侧表示开盘价（例如，对于日线图，这将是当天的起始价格），右侧表示该时间段的收盘价。

用于演示 OHLC 图表的示例数据如下所示。它具有与对应日期字符串上的最高、最低、开盘和收盘值对应的列表对象。使用来自 datetime 模块的**strtp()**函数将字符串的日期表示形式转换为日期对象。

```
open_data = [33.0, 33.3, 33.5, 33.0, 34.1]
high_data = [33.1, 33.3, 33.6, 33.2, 34.8]
low_data = [32.7, 32.7, 32.8, 32.6, 32.8]
close_data = [33.0, 32.9, 33.3, 33.1, 33.1]
date_data = ['10-10-2013', '11-10-2013', '12-10-2013','01-10-2014','02-10-2014']
import datetime
dates = [
   datetime.datetime.strptime(date_str, '%m-%d-%Y').date()
   for date_str in date_data
]
```

我们必须使用上面的日期对象作为 x 参数和其他参数作为返回 OHLC 跟踪的**go.Ohlc()**函数所需的开盘、高、低和收盘参数。

```
trace = go.Ohlc(
   x = dates,
   open = open_data,
   high = high_data,
   low = low_data,
   close = close_data
)
data = [trace]
fig = go.Figure(data = data)
iplot(fig)
```

代码的输出如下 –

烛台图

的**烛图**类似于OHLC图。它就像**line-chart**和**bar-chart 的组合**。方框代表开盘价和收盘价之间的价差，线条代表低价和高价之间的价差。收盘价高于（低于）开盘价的样本点称为增加（减少）。

烛线轨迹由**go.Candlestick() 函数**返回。我们使用相同的数据（如 OHLC 图表）来渲染烛台图表，如下所示 –

```
trace = go.Candlestick(
   x = dates,
   open = open_data,
   high = high_data,
   low = low_data,
   close = close_data
)
```

上面给出的代码的输出如下所述 –

瀑布图

瀑布图（也称为**飞砖图或马里奥图**）有助于理解顺序引入的正值或负值的累积效应，这些值可以是基于时间的，也可以是基于类别的。

初始值和最终值显示为列，单独的负调整和正调整表示为浮动步骤。一些瀑布图将列之间的线连接起来，使图表看起来像一座桥梁。

go.Waterfall()函数返回瀑布跟踪。该对象可以通过各种命名参数或属性进行自定义。这里，x 和 y 属性为图形的 x 和 y 坐标设置数据。两者都可以是 Python 列表、numpy 数组或 Pandas 系列或字符串或日期时间对象。

另一个属性是**量度**，它是一个包含值类型的数组。默认情况下，这些值被视为**relative**。将其设置为 'total' 以计算总和。如果它等于**绝对值**，它会重置计算的总数或在需要时声明一个初始值。'base' 属性设置绘制条形基座的位置（以位置轴为单位）。

以下代码呈现瀑布图 –

```
s1=[
   "Sales",
```

```
   "Consulting",
   "Net revenue",
   "Purchases",
   "Other expenses",
   "Profit before tax"
]
s2 = [60, 80, 0, -40, -20, 0]
trace = go.Waterfall(
   x = s1,
   y = s2,
   base = 200,
   measure = [
      "relative",
      "relative",
      "total",
      "relative",
      "relative",
      "total"
   ]
)
data = [trace]
fig = go.Figure(data = data)
iplot(fig)
```

下面提到的输出是上面给出的代码的结果。

漏斗图

漏斗图表示业务流程不同阶段的数据。它是商业智能中识别流程潜在问题区域的重要机制。漏斗图用于可视化数据在从一个阶段传递到另一个阶段时如何逐渐减少。每个阶段中的数据表示为 100%（整体）的不同部分。

与饼图一样，漏斗图也不使用任何轴。它也可以被视为类似于**堆积百分比条形图**。任何漏斗都由称为头部（或底部）的较高部分和称为颈部的较低部分组成。漏斗图最常见的用途是可视化销售转化数据。

Plotly 的**go.Funnel()**函数产生漏斗轨迹。要提供给此函数的基本属性是 x 和**y**。它们中的每一个都被分配了一个 Python 项目列表或一个数组。

```
from plotly import graph_objects as go
fig = go.Figure(
   go.Funnel(
      y = [
         "Website visit",
         "Downloads",
         "Potential customers",
         "Requested price",
         "invoice sent"
      ],
      x = [39, 27.4, 20.6, 11, 2]
   )
```

```
)
fig.show()
```

输出如下 –

Plotly – 3D 散点图和曲面图

本章将提供有关 3D (3D) Scatter Plot 和 3D Surface Plot 以及如何在 Plotly 的帮助下制作它们的信息。

3D 散点图

三维 (3D) 散点图类似于散点图，但具有三个变量 – **x、y 和 z 或 f(x, y)**是实数。该图可以表示为三维笛卡尔坐标系中的点。它通常使用透视方法（等距或透视）绘制在二维页面或屏幕上，因此其中一个维度似乎是从页面中出来的。

3D 散点图用于在三个轴上绘制数据点，以尝试显示三个变量之间的关系。数据表中的每一行都由一个标记表示，其位置取决于其在**X、Y 和 Z 轴上**设置的列中的值。

第四变量可以设置为对应于**颜色**或**大小**的的**标记物**，从而，增加另一个维度的情节。不同变量之间的关系称为**相关性**。

甲**Scatter3D迹**是由go.Scatter3D（）函数返回一个图形对象。此函数的必需参数是 x、**y 和 z**，它们中的每一个都是**列表或数组对象**。

例如 –

```
import plotly.graph_objs as go
import numpy as np
z = np.linspace(0, 10, 50)
x = np.cos(z)
y = np.sin(z)
trace = go.Scatter3d(
   x = x, y = y, z = z,mode = 'markers', marker = dict(
      size = 12,
      color = z, # set color to an array/list of desired values
      colorscale = 'Viridis'
      )
   )
layout = go.Layout(title = '3D Scatter plot')
fig = go.Figure(data = [trace], layout = layout)
iplot(fig)
```

代码的输出如下 –

3D 曲面图

曲面图是三维数据的图表。在曲面图中，每个点由 3 个点定义：**纬度**、**经度**和**高度**（X、Y 和 Z）。表面图不是显示单个数据点，而是显示指定**因变量 (Y)**和两个自变量（X 和 Z）之间的函数关系。该图是等高线图的配套图。

这是一个 Python 脚本，用于渲染简单的曲面图，其中**y 数组**是 x 的转置，z 计算为 cos(x2+y2)

```
import numpy as np
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T # transpose
z = np.cos(x ** 2 + y ** 2)
trace = go.Surface(x = x, y = y, z =z )
data = [trace]
layout = go.Layout(title = '3D Surface plot')
fig = go.Figure(data = data)
iplot(fig)
```

下面提到的是上面解释的代码的输出 –

Plotly – 添加按钮下拉菜单

Plotly 通过在绘图区域上使用不同的控件（例如按钮、下拉菜单和滑块等）来提供高度的交互性。这些控件与绘图布局的**updatemenu**属性相结合。您可以通过指定要调用的方法来**添加按钮**及其行为。

有四种可能的方法可以与按钮相关联，如下所示 –

- **restyle** – 修改数据或数据属性
- **重新布局**– 修改布局属性
- **更新**– 修改数据和布局属性
- **animate** – 开始或暂停动画

所述**restyle**时，应使用的方法，**修改所述数据和数据属性**图表的。在以下示例中，**Updatemenu()**方法使用**restyle**方法将两个按钮添加到布局中。

```
go.layout.Updatemenu(
type = "buttons",
direction = "left",
buttons = list([
   dict(args = ["type", "box"], label = "Box", method = "restyle"),
   dict(args = ["type", "violin"], label = "Violin", method = "restyle" )]
))
```

价值**型**属性**按钮**默认。要呈现按钮的下拉列表，请将 type 更改为**dropdown**。在更新其布局之前添加到 Figure 对象的 Box 跟踪如上。根据单击的按钮呈现**箱线图**和**小提琴图**的完整代码如下 –

```
import plotly.graph_objs as go
fig = go.Figure()
fig.add_trace(go.Box(y = [1140,1460,489,594,502,508,370,200]))
fig.layout.update(
   updatemenus = [
      go.layout.Updatemenu(
         type = "buttons", direction = "left", buttons=list(
            [
```

```
                dict(args = ["type", "box"], label = "Box", method = "restyle"),
                dict(args = ["type", "violin"], label = "Violin", method = "restyle")
            ]
         ),
         pad = {"r": 2, "t": 2},
         showactive = True,
         x = 0.11,
         xanchor = "left",
         y = 1.1,
         yanchor = "top"
      ),
   ]
)
iplot(fig)
```

代码的输出如下 –

单击**Violin**按钮可显示相应的**Violin 图**。

如上所述，**Updatemenu()**方法中的**key类型的**值被分配给**下拉列表**以显示按钮的下拉列表。该图如下所示 –

所述**更新**方法应该修改所述图形的数据和布局部分时使用。以下示例演示了如何在更新布局属性（例如图表标题）的同时更新以及显示哪些跟踪。对应于**正弦和余弦波的**两条 Scatter 轨迹被添加到**Figure 对象**。可见**属性**为**True的**迹线将显示在图上，其他迹线将被隐藏。

```
import numpy as np
import math #needed for definition of pi

xpoints = np.arange(0, math.pi*2, 0.05)
y1 = np.sin(xpoints)
y2 = np.cos(xpoints)
fig = go.Figure()
# Add Traces
fig.add_trace(
   go.Scatter(
      x = xpoints, y = y1, name = 'Sine'
   )
)
fig.add_trace(
   go.Scatter(
      x = xpoints, y = y2, name = 'cos'
   )
)
fig.layout.update(
   updatemenus = [
      go.layout.Updatemenu(
         type = "buttons", direction = "right", active = 0, x = 0.1, y = 1.2,
         buttons = list(
```

```
            [
               dict(
                  label = "first", method = "update",
                  args = [{"visible": [True, False]},{"title": "Sine"} ]
               ),
               dict(
                  label = "second", method = "update",
                  args = [{"visible": [False, True]},{"title": Cos"}]
               )
            ]
         )
      )
   ]
)
iplot(fig)
```

最初，将显示**正弦曲线**。如果单击第二个按钮，则会出现**cos 跟踪**。

请注意，**图表标题**也会相应更新。

为了使用**animate**方法，我们需要向**Figure**对象添加一个或多个**Frames**。与数据和布局一起，框架可以添加为图形对象中的键。帧关键点指向一个图形列表，每个图形将在触发动画时循环显示。

您可以通过向布局添加更新**菜单数组**来添加、播放和暂停按钮以在图表中引入动画。

```
"updatemenus": [{
   "type": "buttons", "buttons": [{
      "label": "Your Label", "method": "animate", "args": [frames]
   }]
}]
```

在以下示例中，首先绘制**散点曲线**轨迹。然后添加**框架**，这是一个包含**50 个框架对象**的列表，每个**对象**代表曲线上的一个**红色标记**。请注意，按钮的**args**属性设置为 None，因此所有帧都是动画的。

```
import numpy as np
t = np.linspace(-1, 1, 100)
x = t + t ** 2
y = t - t ** 2
xm = np.min(x) - 1.5
xM = np.max(x) + 1.5
ym = np.min(y) - 1.5
yM = np.max(y) + 1.5
N = 50
s = np.linspace(-1, 1, N)
#s = np.arange(0, math.pi*2, 0.1)
xx = s + s ** 2
yy = s - s ** 2
fig = go.Figure(
   data = [
```

```
      go.Scatter(x = x, y = y, mode = "lines", line = dict(width = 2, color = "blue")),
      go.Scatter(x = x, y = y, mode = "lines", line = dict(width = 2, color = "blue"))
   ],
   layout = go.Layout(
      xaxis=dict(range=[xm, xM], autorange=False, zeroline=False),
      yaxis=dict(range=[ym, yM], autorange=False, zeroline=False),
      title_text="Moving marker on curve",
      updatemenus=[
         dict(type="buttons", buttons=[dict(label="Play", method="animate", args=
[None])])]
      )
   ),
   frames = [go.Frame(
      data = [
            go.Scatter(
            x = [xx[k]], y = [yy[k]], mode = "markers", marker = dict(
               color = "red", size = 10
            )
         )
      ]
   )
   for k in range(N)]
)
iplot(fig)
```

代码的输出如下 –

单击**播放**按钮后，红色标记将开始沿曲线移动。

Plotly – 滑块控件

Plotly 有一个方便的**Slider**，可用于通过滑动位于渲染图底部的控件上的旋钮来更改图的**数据/样式**视图。

滑块控件由不同的属性组成，如下所示 –

- 定义旋钮在控件上的滑动位置需要**steps 属性**。
- **方法属性**有可能的值作为**restyle | 重新布局| 动画 | 更新 | 跳过**，默认为**restyle**。
- **args 属性**设置要传递给幻灯片方法中设置的 Plotly 方法的参数值。

我们现在在散点图上部署一个简单的滑块控件，当旋钮沿着控件滑动时，它将改变**正弦波**的频率。滑块配置为具有 50 个步骤。首先添加 50 条具有递增频率的正弦波曲线，除第 10 条以外的所有迹线设置为可见。

然后，我们使用**restyle**方法配置每个步骤。对于每个步骤，所有其他步骤对象的可见性设置为**false**。最后，通过初始化滑块属性更新 Figure 对象的布局。

```
# Add traces, one for each slider step
for step in np.arange(0, 5, 0.1):
fig.add_trace(
   go.Scatter(
      visible = False,
```

```
      line = dict(color = "blue", width = 2),
      name = "ð•œ = " + str(step),
      x = np.arange(0, 10, 0.01),
      y = np.sin(step * np.arange(0, 10, 0.01))
   )
)
fig.data[10].visible=True

# Create and add slider
steps = []
for i in range(len(fig.data)):
step = dict(
   method = "restyle",
   args = ["visible", [False] * len(fig.data)],
)
step["args"][1][i] = True # Toggle i'th trace to "visible"
steps.append(step)
sliders = [dict(active = 10, steps = steps)]
fig.layout.update(sliders=sliders)
iplot(fig)
```

首先，**10th正弦波**轨迹将可见。尝试在底部的水平控件上滑动旋钮。您将看到如下所示的频率变化。

Plotly – FigureWidget 类

Plotly 3.0.0 引入了一个新的 Jupyter 小部件类：**plotly.graph_objs.FigureWidget**。它与我们现有的 Figure 具有相同的调用签名，并且是专门为**Jupyter Notebook**和**JupyterLab 环境制作的**。

所述**go.FigureWiget（）函数**返回与默认x和一个空FigureWidget对象**ÿ**轴。

```
f = go.FigureWidget()
iplot(f)
```

下面给出的是代码的输出 –

FigureWidget 最重要的特性是生成的 Plotly 图形，它可以随着我们继续向其添加数据和其他布局属性而动态更新。

例如，将以下图形轨迹一一添加，查看动态更新的原始空图。这意味着我们不必一次又一次地调用 iplot() 函数，因为绘图会自动刷新。FigureWidget 的最终外观如下所示 –

```
f.add_scatter(y = [2, 1, 4, 3]);
f.add_bar(y = [1, 4, 3, 2]);
f.layout.title = 'Hello FigureWidget'
```

这个小部件能够作为悬停、点击、选择点和放大区域的事件监听器。

在以下示例中，FigureWidget 被编程为响应绘图区域上的单击事件。小部件本身包含一个带有标记的简单散点图。鼠标点击位置用不同的颜色和大小标记。

```
x = np.random.rand(100)
y = np.random.rand(100)
f = go.FigureWidget([go.Scatter(x=x, y=y, mode='markers')])

scatter = f.data[0]
colors = ['#a3a7e4'] * 100

scatter.marker.color = colors
scatter.marker.size = [10] * 100
f.layout.hovermode = 'closest'
def update_point(trace, points, selector):

c = list(scatter.marker.color)
s = list(scatter.marker.size)
for i in points.point_inds:

c[i] = 'red'
s[i] = 20

scatter.marker.color = c
scatter.marker.size = s
scatter.on_click(update_point)
f
```

在 Jupyter notebook 中运行上面的代码。显示散点图。单击该区域中标记为红色的位置。

Plotly 的 FigureWidget 对象也可以使用**Ipython**自己的小部件。在这里，我们使用**ipwidgets**模块中定义的交互控制。我们首先构建一个**FigureWidget**并添加一个**空的散点图**。

```
from ipywidgets import interact
fig = go.FigureWidget()
scatt = fig.add_scatter()
fig
```

我们现在定义一个**更新函数**，它输入频率和相位并设置上面定义的**散射轨迹**的 x 和**y**属性。来自 ipywidgets 模块的**@interact 装饰器**用于创建一组简单的小部件来控制绘图的参数。更新函数用来自**ipywidgets 包的@interact 装饰器装饰**。装饰器参数用于指定我们要扫描的参数范围。

```
xs = np.linspace(0, 6, 100)
@interact(a = (1.0, 4.0, 0.01), b = (0, 10.0, 0.01), color = ['red', 'green', 'blue'])
def update(a = 3.6, b = 4.3, color = 'blue'):
with fig.batch_update():
scatt.x = xs
scatt.y = np.sin(a*xs-b)
scatt.line.color = color
```

空 FigureWidget 现在以蓝色填充，**正弦曲线**a 和**b**分别为 3.6 和 4.3。在当前笔记本单元格下方，您将获得一组滑块，用于选择**a**和**b 的**值。还有一个下拉菜单可以选择跟踪颜色。这些参数在**@interact 装饰器**中定义。

Pandas 和袖扣的阴谋

Pandas 是 Python 中非常流行的用于数据分析的库。它也有自己的绘图功能支持。但是，Pandas 图不提供可视化中的交互性。值得庆幸的是，可以使用**Pandas 数据框**对象构建 plotly 的交互式和动态图。

我们首先从简单的列表对象构建一个 Dataframe。

```
data = [['Ravi',21,67],['Kiran',24,61],['Anita',18,46],['Smita',20,78],['Sunil',17,90]]
df = pd.DataFrame(data,columns = ['name','age','marks'],dtype = float)
```

数据框列用作图形对象轨迹的**x**和**y**属性的数据值。在这里，我们将使用**名称**和**标记**列生成条形跟踪。

```
trace = go.Bar(x = df.name, y = df.marks)
fig = go.Figure(data = [trace])
iplot(fig)
```

一个简单的条形图将显示在 Jupyter 笔记本中，如下所示 –

Plotly 建立在**d3.js**之上，**特别**是一个图表库，可以使用另一个名为**Cufflinks 的**库直接与**Pandas 数据帧一起**使用。

如果尚不可用，请使用您最喜欢的包管理器（如**pip**）安装 cufflinks 包，如下所示 –

```
pip install cufflinks
or
conda install -c conda-forge cufflinks-py
```

首先，将袖扣与其他库（如**Pandas**和**numpy**）一起导入，这些库可以将其配置为离线使用。

```
import cufflinks as cf
cf.go_offline()
```

现在，您可以直接使用**Pandas 数据框**来显示各种绘图，而无需像我们之前所做的那样使用来自**graph_objs 模块的**跟踪和图形对象。

```
df.iplot(kind = 'bar', x = 'name', y = 'marks')
```

条形图，与之前的非常相似，如下所示 –

来自数据库的 Pandas 数据框

它可以由不同类型数据库中的数据填充，而不是使用 Python 列表来构建数据框。例如，可以将 CSV 文件、SQLite 数据库表或 mysql 数据库表中的数据提取到 Pandas 数据帧中，最终使用**Figure 对象**或**Cufflinks 接口**绘制图形。

要从**CSV 文件中**获取数据，我们可以使用Pandas 库中的**read_csv()**函数。

```
import pandas as pd
df = pd.read_csv('sample-data.csv')
```

如果数据在**SQLite 数据库表中**可用，则可以使用**SQLAlchemy 库**进行检索，如下所示 –

```
import pandas as pd
from sqlalchemy import create_engine
disk_engine = create_engine('sqlite:///mydb.db')
df = pd.read_sql_query('SELECT name,age,marks', disk_engine)
```

另一方面，来自**MySQL 数据库的**数据在 Pandas 数据框中检索如下 –

```
import pymysql
import pandas as pd
conn = pymysql.connect(host = "localhost", user = "root", passwd = "xxxx", db = "mydb")
cursor = conn.cursor()
cursor.execute('select name,age,marks')
rows = cursor.fetchall()
df = pd.DataFrame( [[ij for ij in i] for i in rows] )
df.rename(columns = {0: 'Name', 1: 'age', 2: 'marks'}, inplace = True)
```

使用 Matplotlib 和 Chart Studio 进行绘图

本章涉及名为 Matplotlib 的数据可视化库和名为 Chart Studio 的在线绘图制作器。

Matplotlib

Matplotlib 是一个流行的 Python 数据可视化库，能够生成生产就绪但静态的绘图。您可以借助**plotly.tools**模块中的**mpl_to_plotly()**函数将静态**matplotlib 图形**转换为交互式绘图。

以下脚本使用**Matplotlib 的 PyPlot API**生成**正弦波线图**。

```
from matplotlib import pyplot as plt
import numpy as np
import math
#needed for definition of pi
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
plt.plot(x,y)
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
plt.show()
```

现在我们将它转换成一个情节图如下 –

```
fig = plt.gcf()
plotly_fig = tls.mpl_to_plotly(fig)
py.iplot(plotly_fig)
```

代码的输出如下所示 –

图表工作室

Chart Studio 是 Plotly 提供的在线绘图制作工具。它提供了一个图形用户界面，用于将数据导入和分析到网格中并使用统计工具。图形可以嵌入或下载。它主要用于更快、更有效地创建图形。

登录 plotly 帐户后，通过访问链接https://plot.ly/create启动图表工作室应用程序。该网页在绘图区域下方提供了一个空白工作表。Chart Studio 允许您通过按 +**跟踪按钮**来添加绘图**跟踪**。

菜单中提供了各种绘图结构元素，例如注释、样式等，以及保存、导出和共享绘图的工具。

让我们在工作表中添加数据并从跟踪类型中添加**选择条形图**跟踪。

单击类型文本框并选择条形图。

然后，为**x**和**y**轴提供数据列并输入绘图标题。