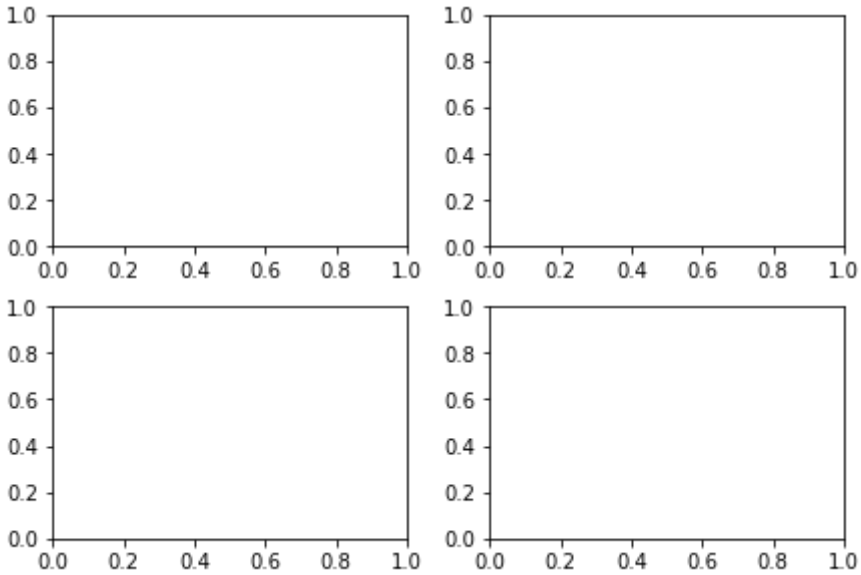
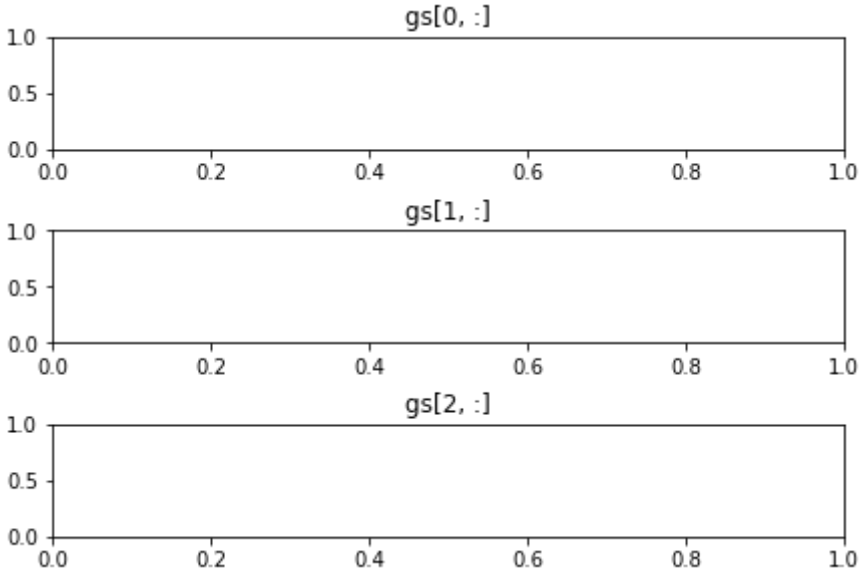
In [2]:

```
# Layouts
# You can also use gridspec to create subplots as follows:

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
fig = plt.figure(constrained_layout=True)
specs = gridspec.GridSpec(ncols=2, nrows=2, figure=fig)
ax1 = fig.add_subplot(specs[0, 0])
ax2 = fig.add_subplot(specs[0, 1])
ax3 = fig.add_subplot(specs[1, 0])
ax4 = fig.add_subplot(specs[1, 1])
plt.show()

# The above code will create a subplot. You have to write a lot of code
# for the output that can be obtained in just a couple of lines of code.
```



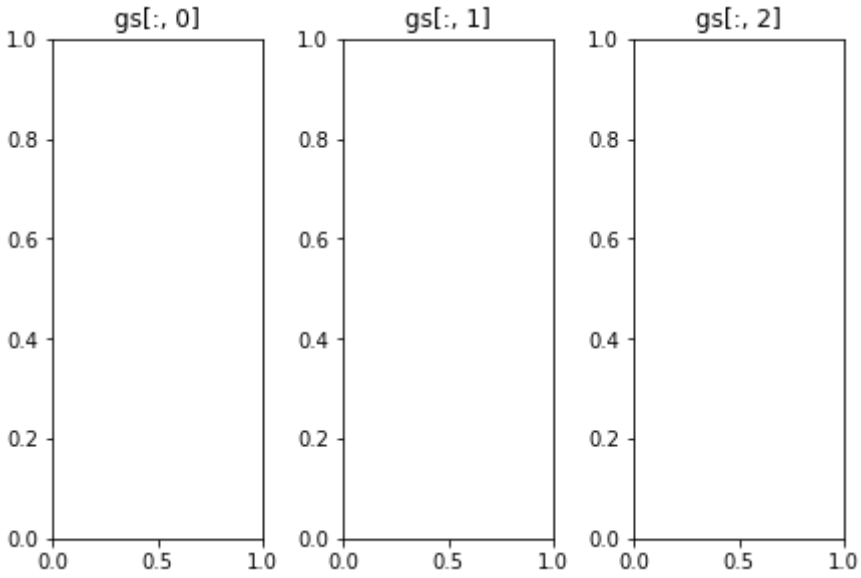
In [3]:

```
# you can use this method to create more complex visualizations.
# Let's create a 3×3 visualization such that an entire row is
# occupied by a plot.

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
fig = plt.figure(constrained_layout=True)
gs = fig.add_gridspec(3, 3)
ax1 = fig.add_subplot(gs[0, :])
ax1.set_title('gs[0, :]')
ax2 = fig.add_subplot(gs[1, :])
ax2.set_title('gs[1, :]')
ax3 = fig.add_subplot(gs[2, :])
ax3.set_title('gs[2, :]')
plt.show()
```



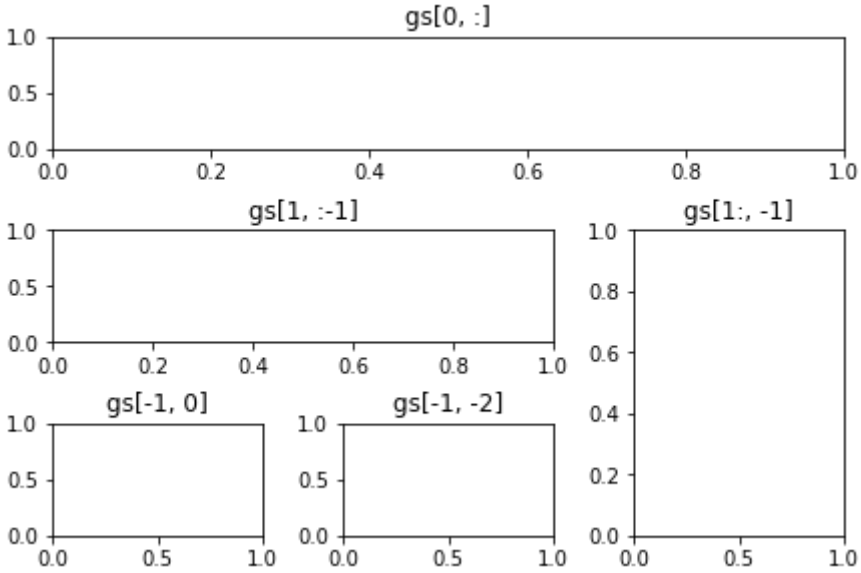
In [4]:

```
# You can also have vertical plots as follows:

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
fig = plt.figure(constrained_layout=True)
gs = fig.add_gridspec(3, 3)
ax1 = fig.add_subplot(gs[:, 0])
ax1.set_title('gs[:, 0]')
ax2 = fig.add_subplot(gs[:, 1])
ax2.set_title('gs[:, 1]')
ax3 = fig.add_subplot(gs[:, 2])
ax3.set_title('gs[:, 2]')
plt.show()
```



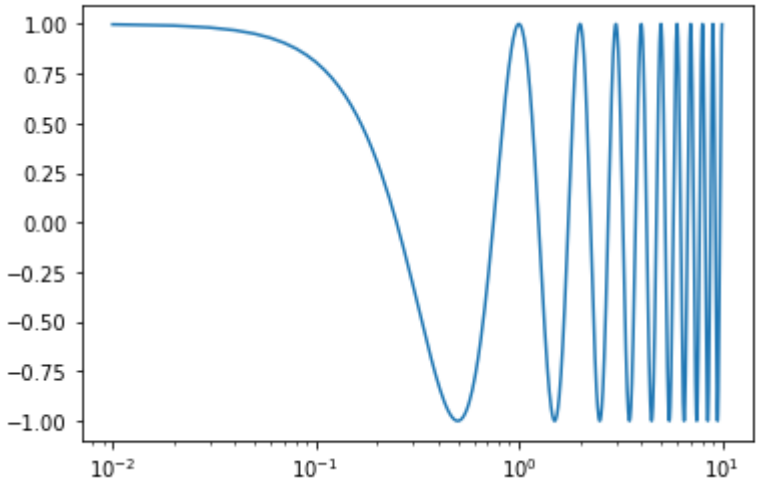
In [5]:

```
# Let's see a more complex example.

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
fig = plt.figure(constrained_layout=True)
gs = fig.add_gridspec(3, 3)
ax1 = fig.add_subplot(gs[0, :])
ax1.set_title('gs[0, :]')
ax2 = fig.add_subplot(gs[1, :-1])
ax2.set_title('gs[1, :-1]')
ax3 = fig.add_subplot(gs[1:, -1])
ax3.set_title('gs[1:, -1]')
ax4 = fig.add_subplot(gs[-1, 0])
ax4.set_title('gs[-1, 0]')
ax5 = fig.add_subplot(gs[-1, -2])
ax5.set_title('gs[-1, -2]')
plt.show()

# This is how you can customize subplots.
```



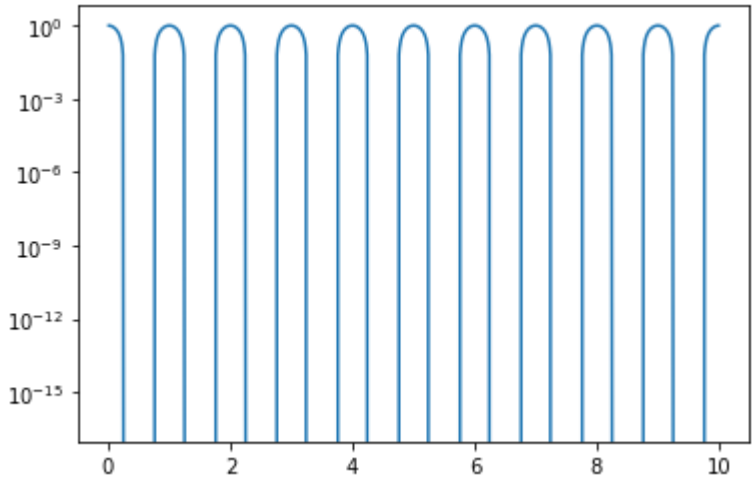
In [6]:

```
# Let's create a graph such that the x-axis is logarithmic and the y-axis is normal, as
# shown here:

import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0.01, 10, 0.01)
plt.semilogx(t, np.cos(2 * np.pi * t))
plt.show()
```



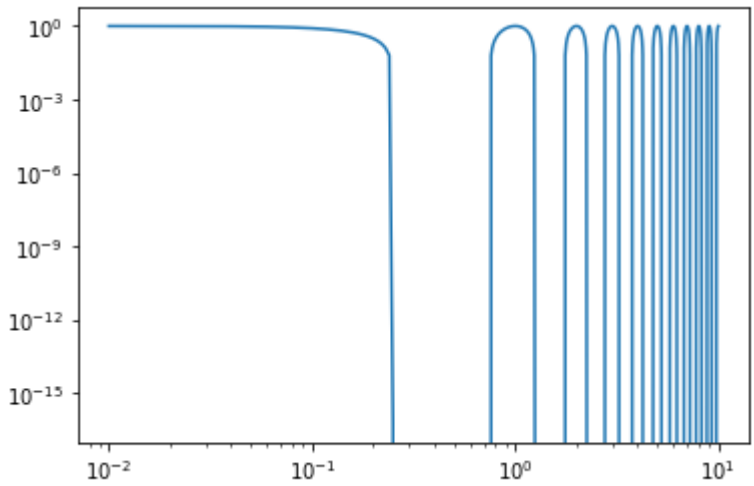
In [7]:

```
# Similarly, you can create a logarithmic y-axis and a normal x-axis as follows:

import numpy as np
import matplotlib.pyplot as plt
plt.semilogy(t, np.cos(2 * np.pi * t))
plt.show()
```



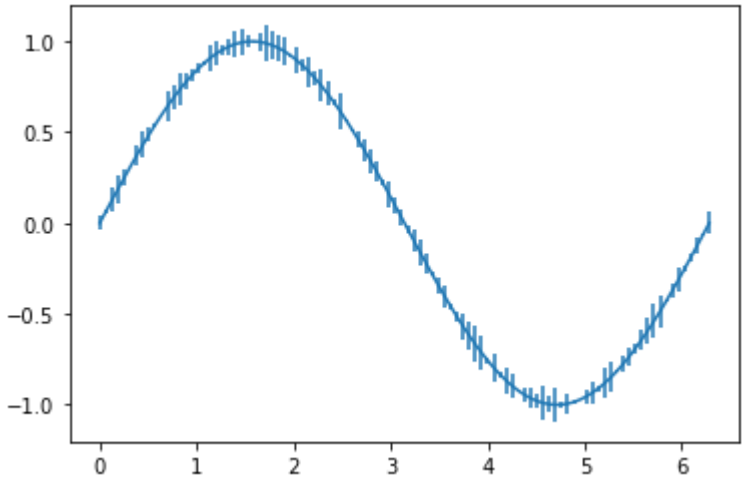
In [8]:

```
# You can have both axes be logarithmic, as shown here:

import numpy as np
import matplotlib.pyplot as plt
plt.loglog(t, np.cos(2 * np.pi * t))
plt.show()
```



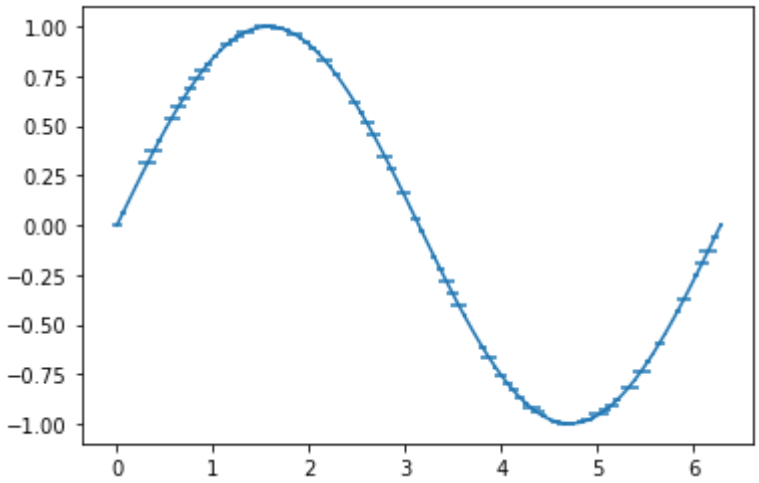
In [10]:

```
# You can also use visualizations to show error in data.
# When there is a possibility of errors in the observed data,
# you usually want to mention it in the observation.
# You would say something like “there’s a 96 percent confidence interval.”
# This means that there is a possibility of 4 percent error in the given data.
# This gives people a general idea about the precision of the quantity.
# When you want to represent this confidence (or lack thereof), you can use error bars.
# You have to use the function errorbar() for this. You can create an Ndarray or list to
# store the error data.
# We can either have real-life data or simulate it as follows:

import numpy as np
import matplotlib.pyplot as plt
x = np.linspace (0, 2 * np.pi, 100)
y = np.sin(x)
ye = np.random.rand(len(x))/10
plt.errorbar(x, y, yerr = ye)
plt.show()
```



In [11]:

```
# Similarly, you can show the error data on the x-axis.

import numpy as np
import matplotlib.pyplot as plt
x = np.linspace (0, 2 * np.pi, 100)
y = np.sin(x)
xe = np.random.rand(len(x))/10
plt.errorbar(x, y, xerr = xe)
plt.show()
```



In [12]:

```
# You can show errors on both axes as follows:

import numpy as np
import matplotlib.pyplot as plt
x = np.linspace (0, 2 * np.pi, 100)
y = np.sin(x)
xe = np.random.rand(len(x))/10
ye = np.random.rand(len(x))/10
plt.errorbar(x, y, xerr = xe, yerr = ye)
plt.show()
```