Week07-01

October 6, 2024

1 Python Visualization

1.1 Learning objectives

1. Introdunction to `matplotlib`
   - https://matplotlib.org/stable/tutorials/index.html
2. Different types of plots
3. Customizing plots
4. Creating subplots
5. Text and annotations
6. 3D plots
7. Seaborn

[ ]:

```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

2 Review from last lecture

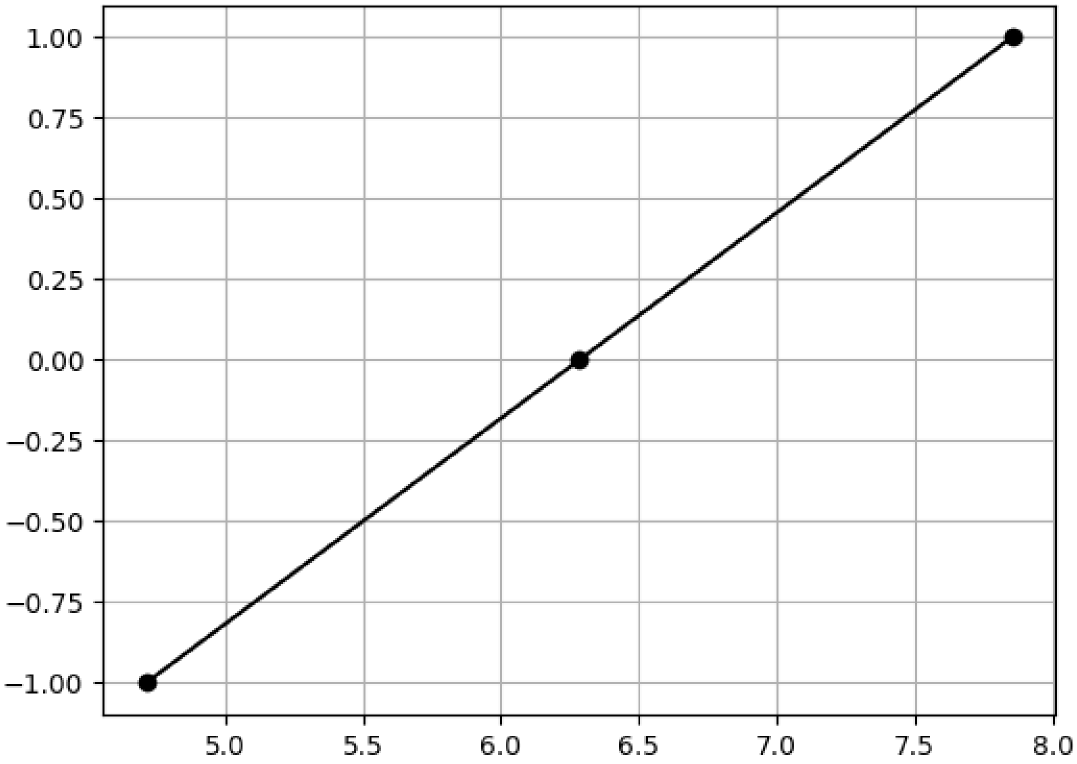
[ ]:

```python
x1 = np.array([ 1.5, 2, 2.5]) * np.pi
```

In `-ok`: The '-' specifies that we want a solid line style. The 'o' specifies the marker style. The 'k' specifies the color of the line and the markers, which is black.

[ ]:

```python
plt.plot(x1, np.sin(x1), '-ok');
```

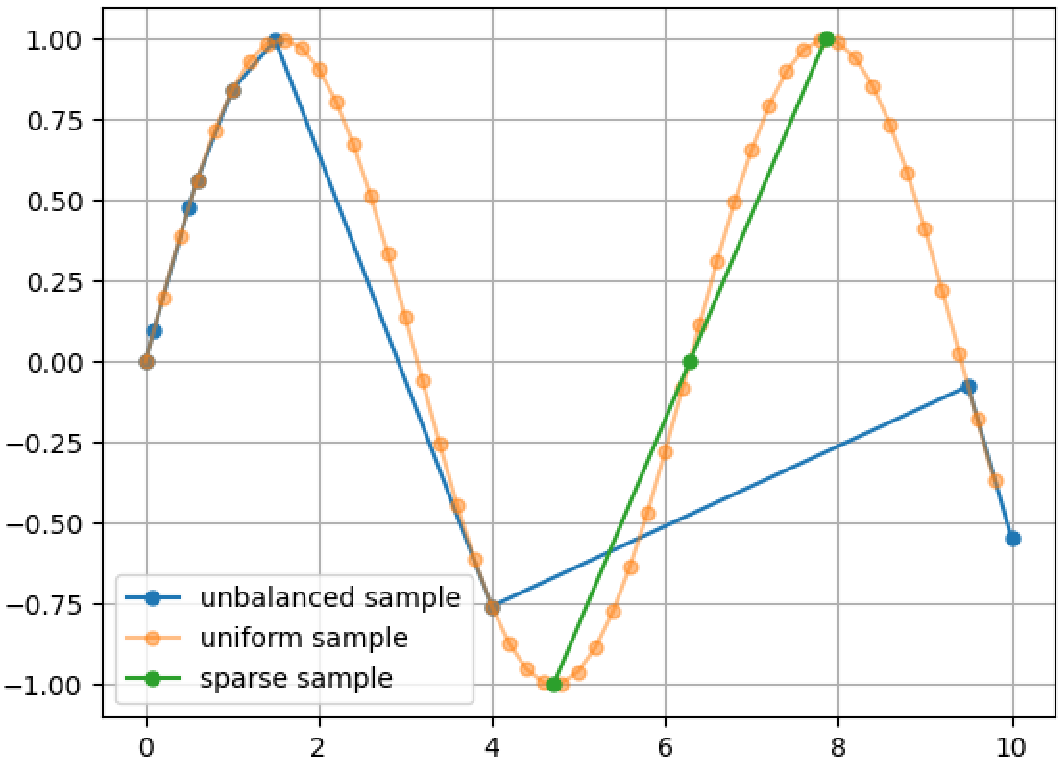


2.0.1 Question: Why sin(x) is a straight line in the above plot?

- The three data points form a straight line.

```
[ ]:
```

```
[ ]: x1 = np.array([ 1.5, 2, 2.5]) * np.pi
x2 = [0, 0.1, 0.5,0.6,1,1.5, 4, 9.5, 10]
x3 = np.arange(0,10,0.2)
plt.plot(x2, np.sin(x2), label = 'unbalanced sample', marker = '.', markersize␣
  ↪= 10)
plt.plot(x3, np.sin(x3), label = 'uniform sample', marker = '.', markersize =␣
  ↪10, alpha = 0.5)
plt.plot(x1, np.sin(x1), label = 'sparse sample', marker = '.', markersize = 10)
plt.legend();
```



[ ]:

[ ]:

[ ]:

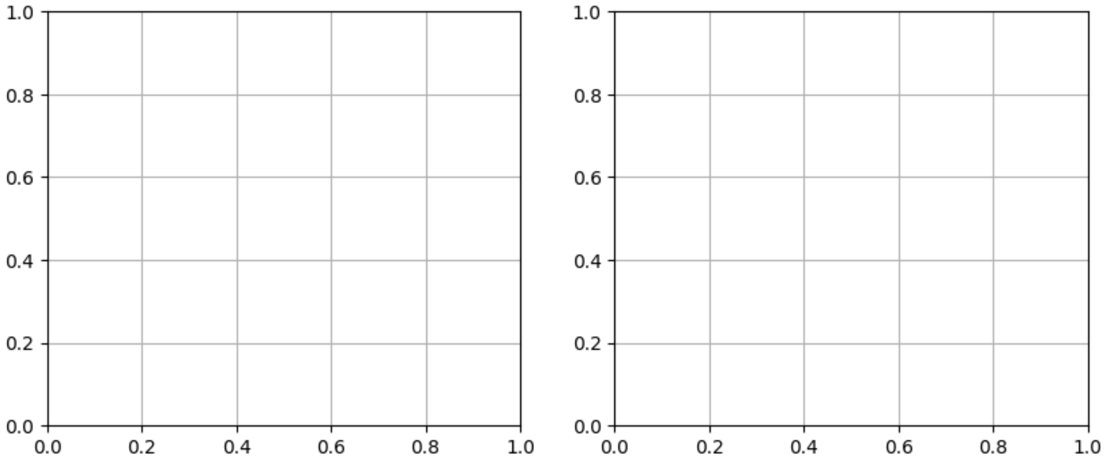
[ ]:

```
plt.rcParams['axes.grid'] = True # display the grid of the plot by default
```

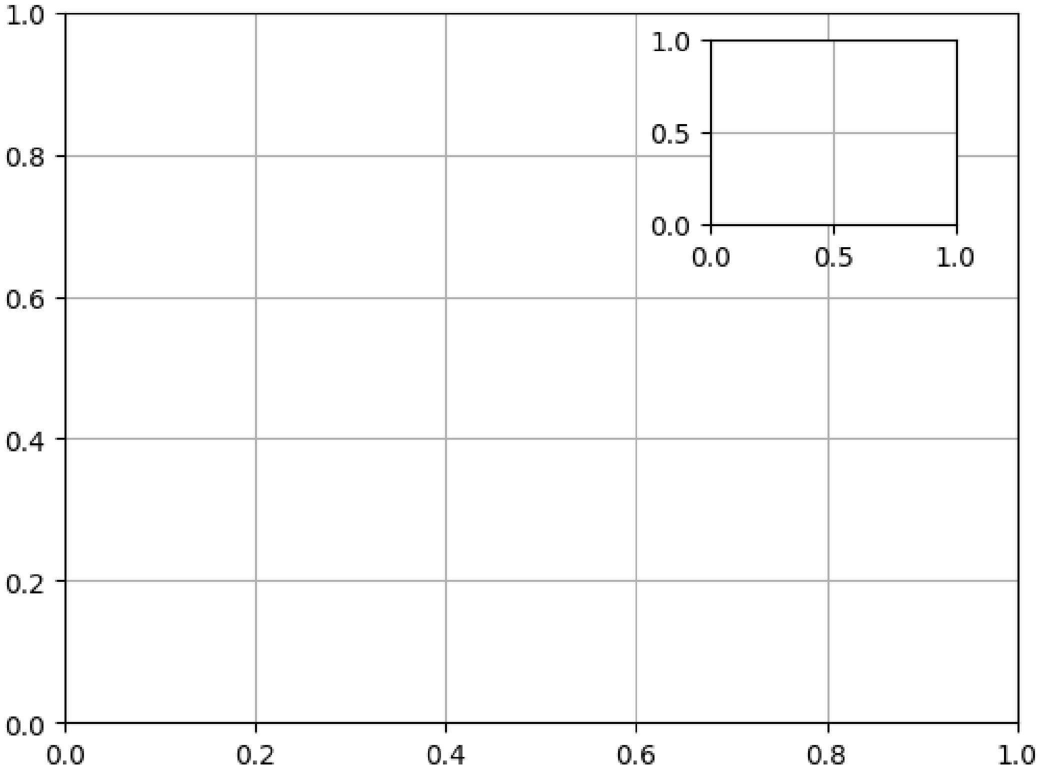
2.1 Multiple Subplots

[ ]:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
```

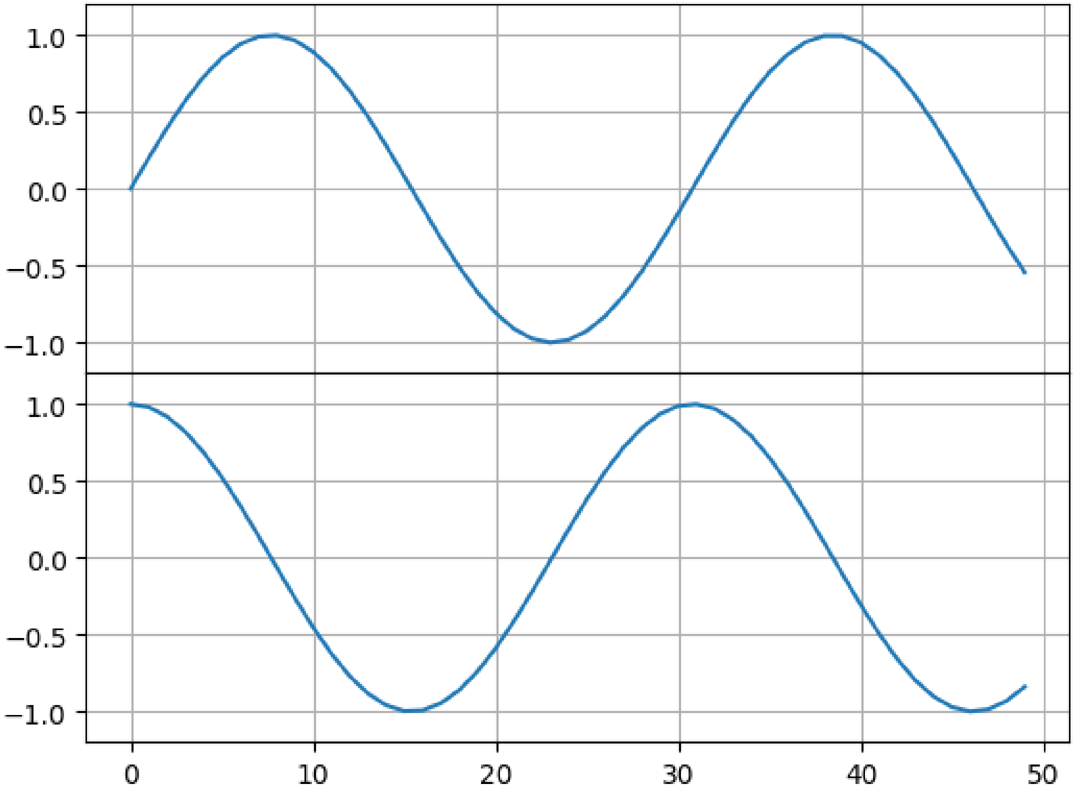


```
[ ]: ax1 = plt.axes() # standard axes
     ax2 = plt.axes([0.65, 0.65, # WHERE THE BOTTOM LEFT CORNER OF THE PLOT IS
                     0.2, 0.2]) # LENGTH AND WIDTH OF THE SMALLER PLOT
```



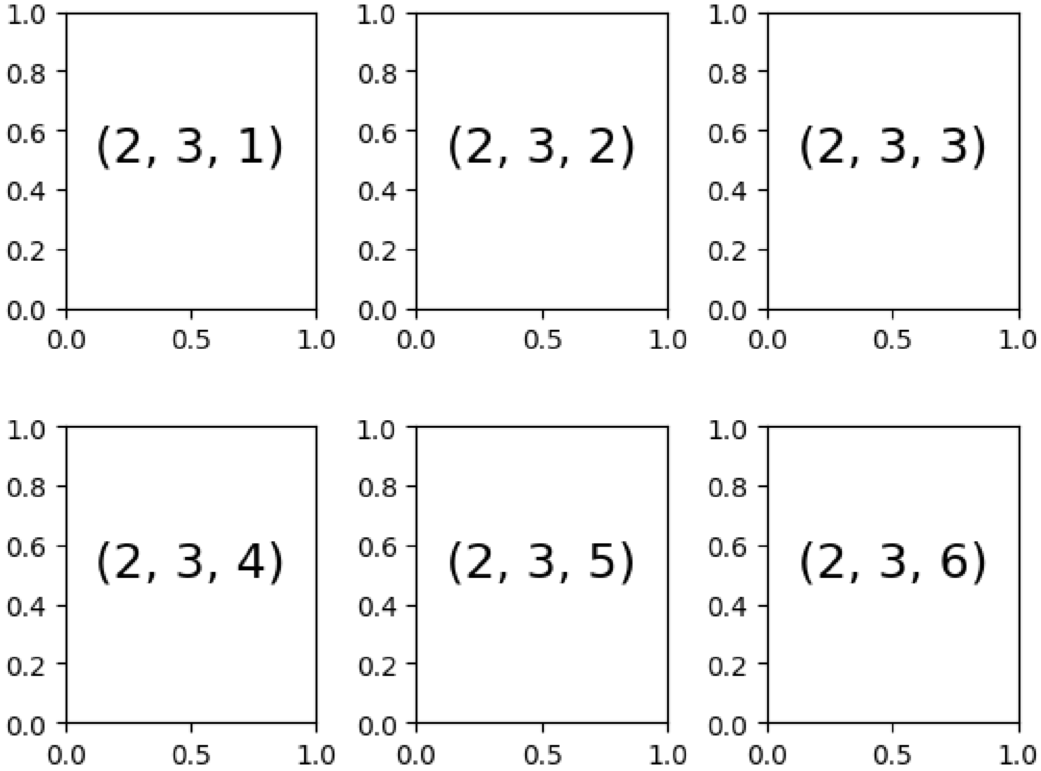
[ ]:

```
fig = plt.figure()
ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4],
                   xticklabels=[], ylim=(-1.2, 1.2))
ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4],
                    ylim=(-1.2, 1.2))
x = np.linspace(0, 10)
ax1.plot(np.sin(x))
ax2.plot(np.cos(x));
```



[ ]:

```
fig = plt.figure()
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i in range(1, 7):
    ax = plt.subplot(2, 3, i)
    ax.text(0.5, 0.5, str((2, 3, i)), fontsize=18, ha='center')
    ax.grid()
```



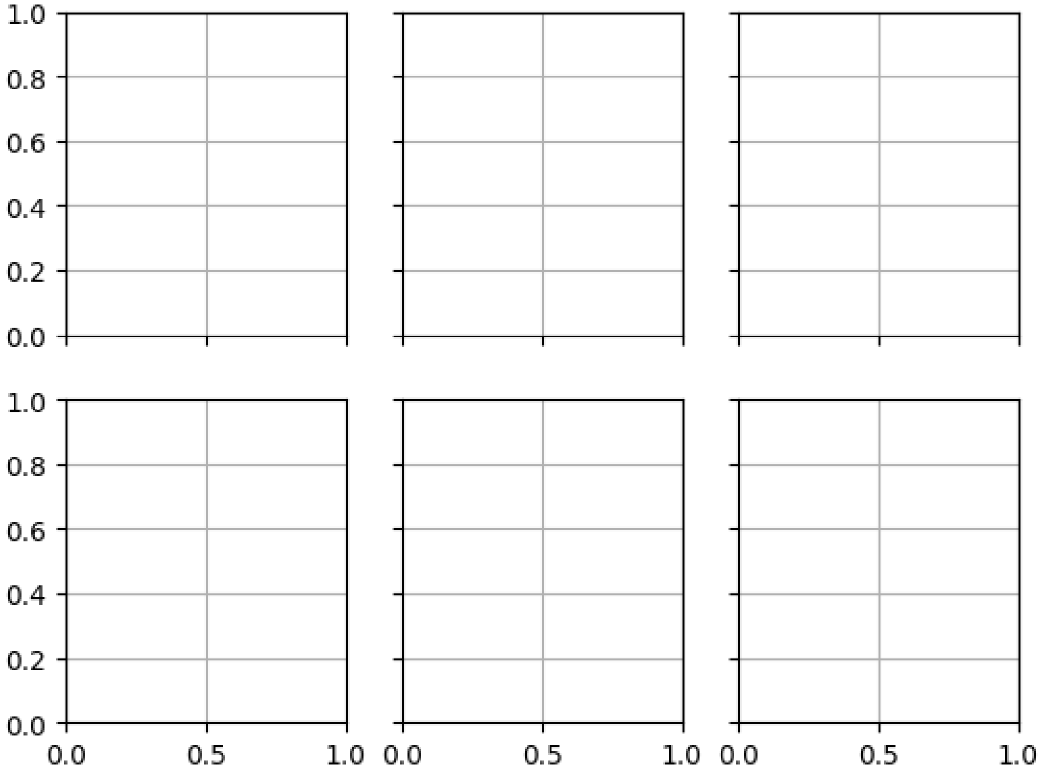
Question: Right now the code below creates an empty 2 x 3 grid of subplots. Add a line plot of y=x to the top right plot. Add a scatter plot of x vs. sin(x) to the bottom left plot. Make sure to label the axes for clarity.

Hint: - Use `ax[row, col].plot()` for the line plot. - Use `ax[row, col].scatter()` for the scatter plot.

[ ]:

```
fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')

# Generate data
x = np.linspace(0, 10, 100)
```

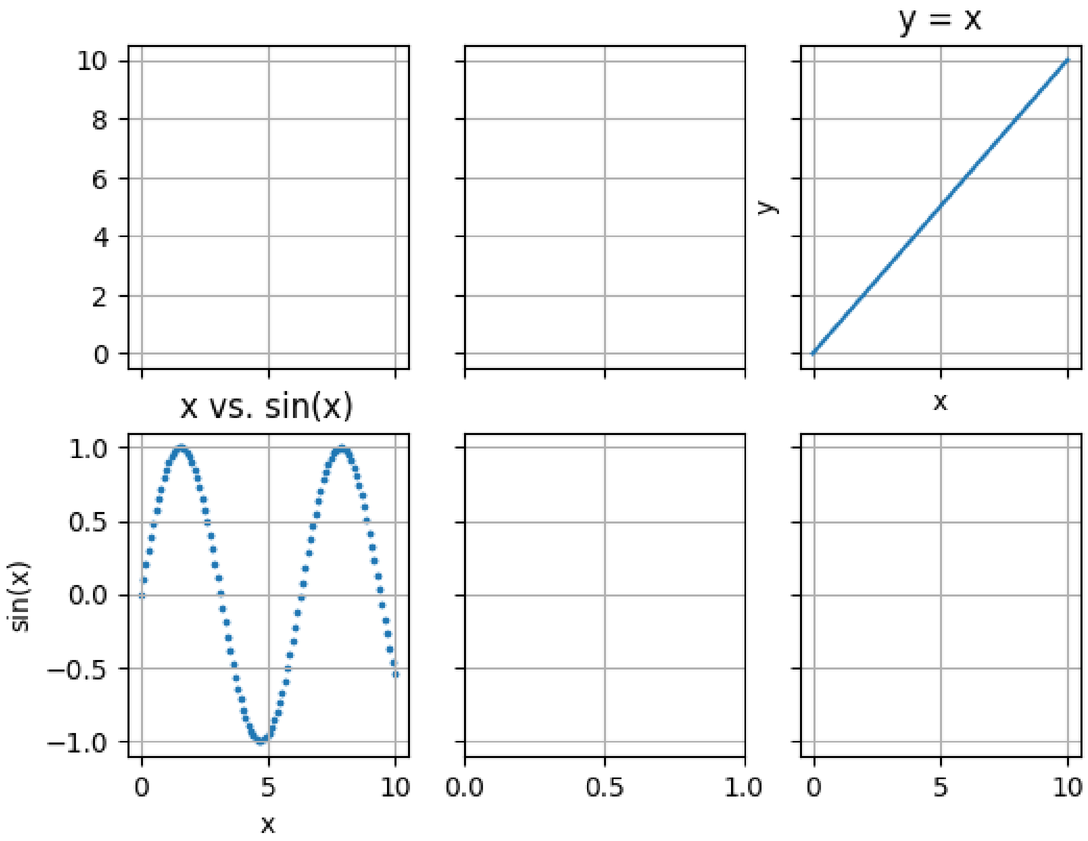


[ ]:
```
fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')

# Generate data
x = np.linspace(0, 10, 100)

# Add a line plot of y = x to the top right plot
ax[0, 2].plot(x, x)
ax[0, 2].set_title('y = x')
ax[0, 2].set_xlabel('x')
ax[0, 2].set_ylabel('y')

# Add a scatter plot of x vs. sin(x) to the bottom left plot
ax[1, 0].scatter(x, np.sin(x), s = 3)
ax[1, 0].set_title('x vs. sin(x)')
ax[1, 0].set_xlabel('x')
ax[1, 0].set_ylabel('sin(x)');
```

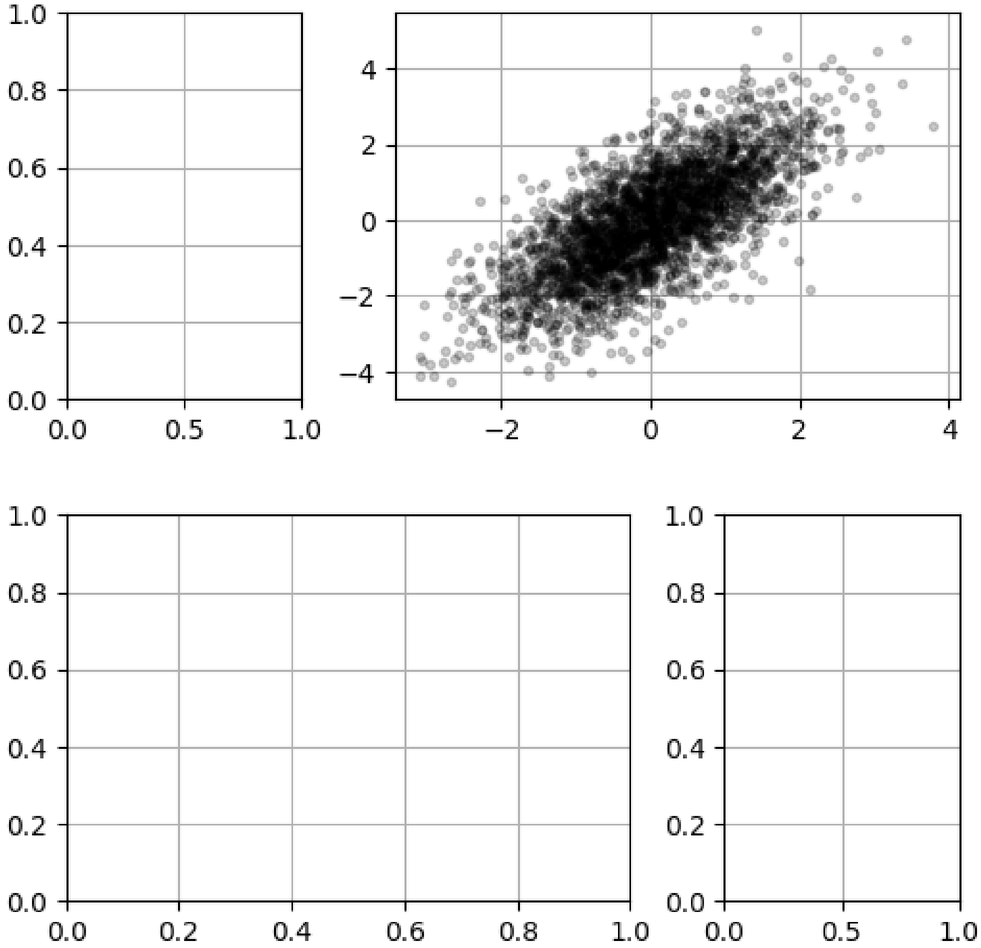


[ ]:
```
fig = plt.figure(figsize=(6, 6))

grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
plt.subplot(grid[0, 0])
main = plt.subplot(grid[0, 1:])
plt.subplot(grid[1, :2])
plt.subplot(grid[1, 2]);

mean = [0, 0]
cov = [[1, 1], [1, 2]]
rng = np.random.default_rng(1701)
x, y = rng.multivariate_normal(mean, cov, 3000).T

# Scatter points on the main axes
main.plot(x, y, 'ok', markersize=3, alpha=0.2);
```



Question: Rewrite the code above in the box below to move the scatterplot from the top right grid space to the bottom left grid space.

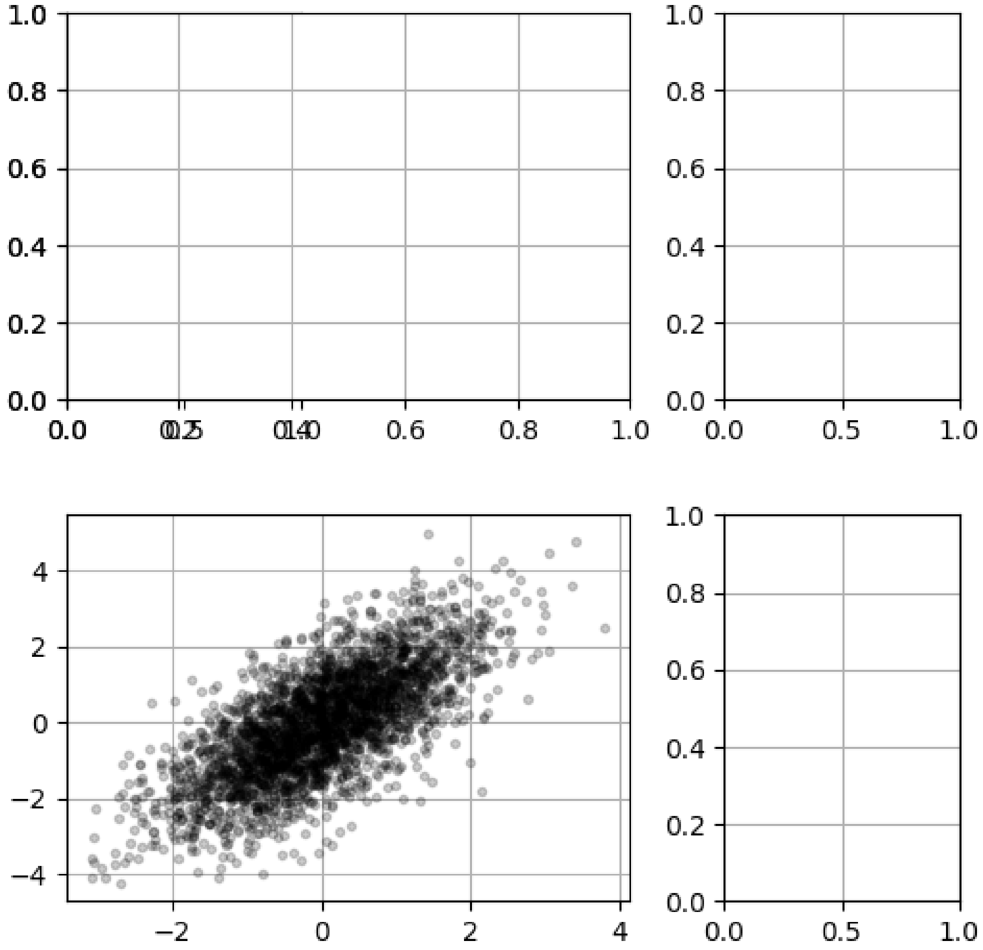
[ ]:

```
fig = plt.figure(figsize=(6, 6))

grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
plt.subplot(grid[0, 0])
main = plt.subplot(grid[1, :2])
plt.subplot(grid[0, :2])
plt.subplot(grid[1, 2:])
plt.subplot(grid[0, 2:]);

mean = [0, 0]
cov = [[1, 1], [1, 2]]
rng = np.random.default_rng(1701)
x, y = rng.multivariate_normal(mean, cov, 3000).T
```

```
# Scatter points on the main axes
main.plot(x, y, 'ok', markersize=3, alpha=0.2);
```



3 Text and annotations

Text and annotations make plots more informative by adding context to key points and making the plot easy to understand.

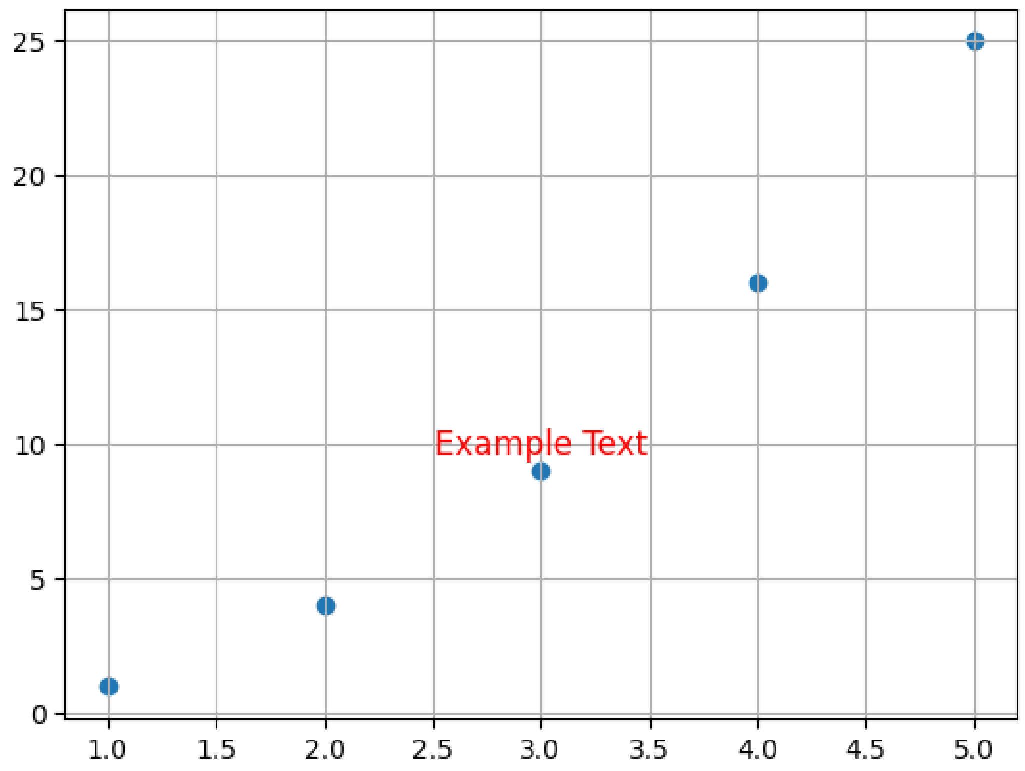
Adding text to specific locations with `plt.text(x, y, 'text')`

[ ]:

```
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

plt.scatter(x, y)

plt.text(3, 10, 'Example Text', fontsize=12,
         color='red', ha='center', va='center');
```



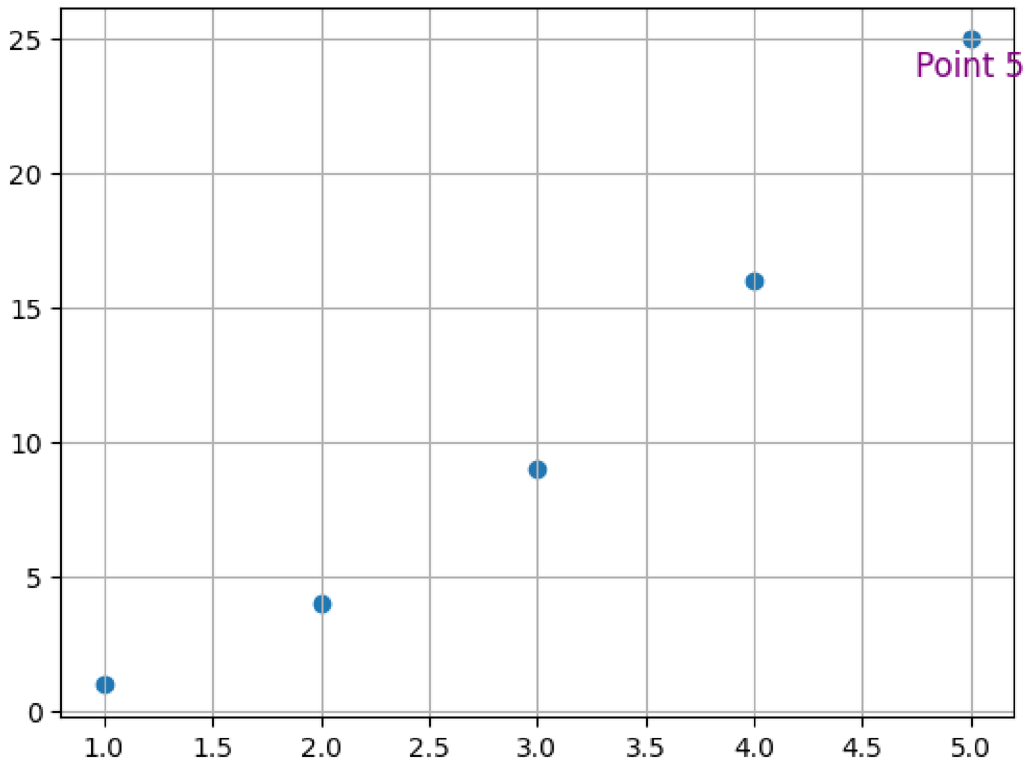
3.1 In-class activity 1: Recreate the plot above in the code chunk below moving the "Example Text" to be below the point (5, 25) and to state "Point 5" in the color purple instead of "Example Text".

[ ]:

```python
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

plt.scatter(x, y)

plt.text(5, 24, 'Point 5', fontsize=12,
         color='purple', ha='center', va='center');
```

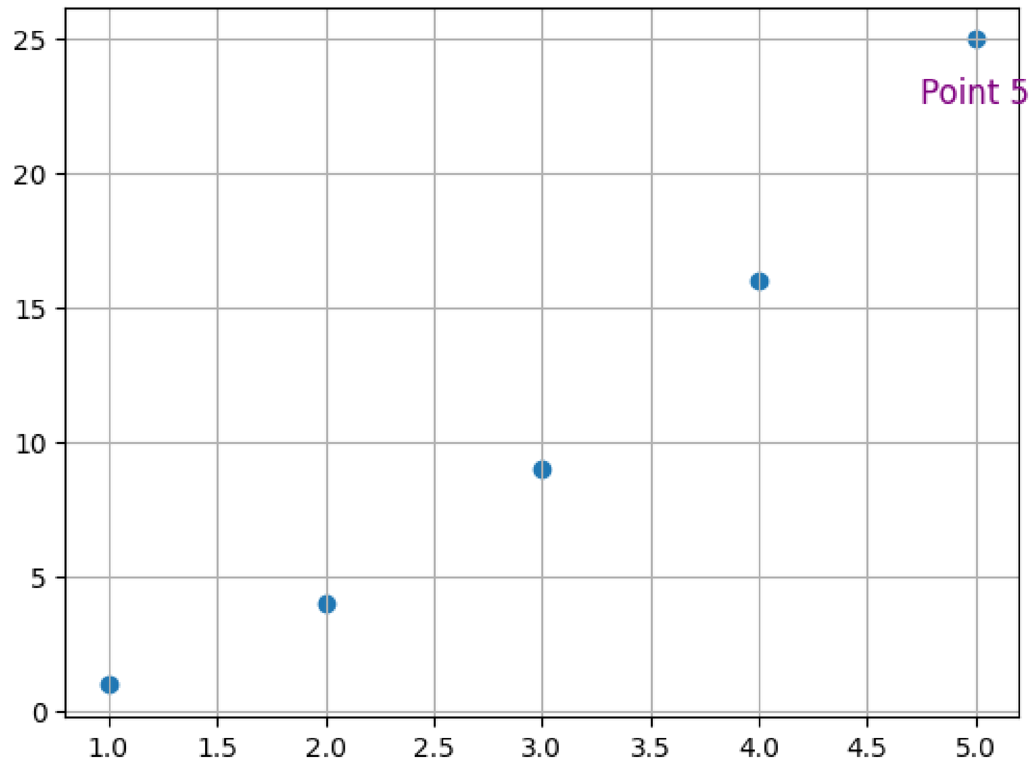


[ ]:

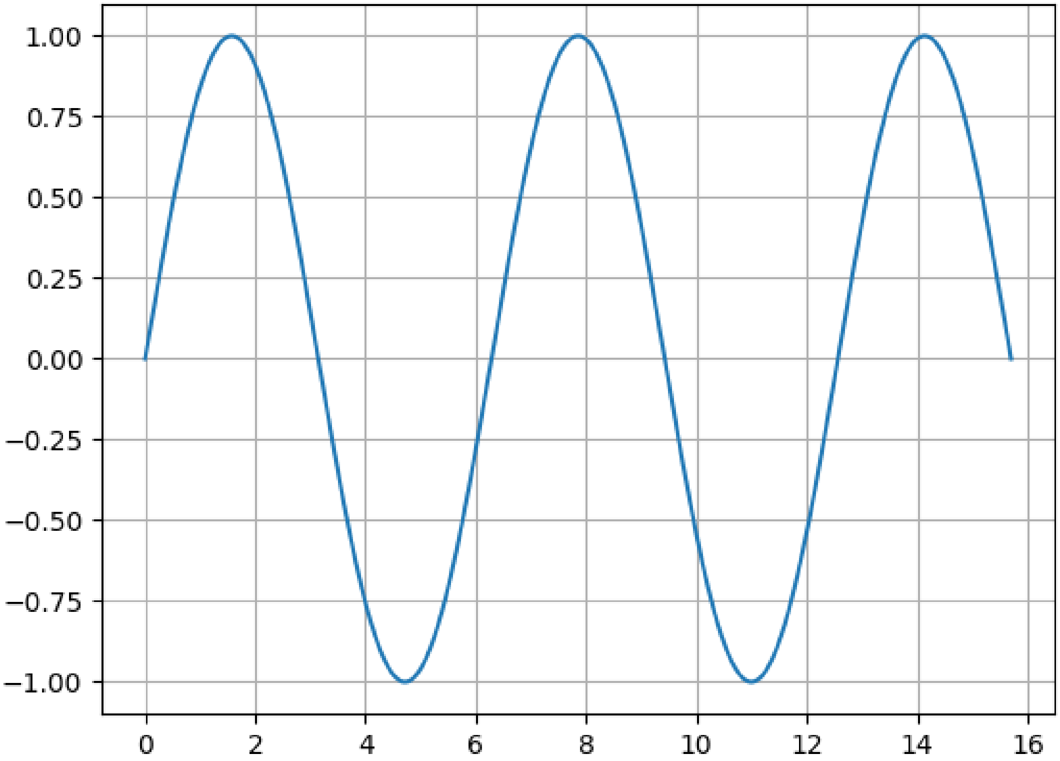
```
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

plt.scatter(x, y)

plt.text(5, 23, 'Point 5', fontsize=12,
         color='purple', ha='center', va='center');
```

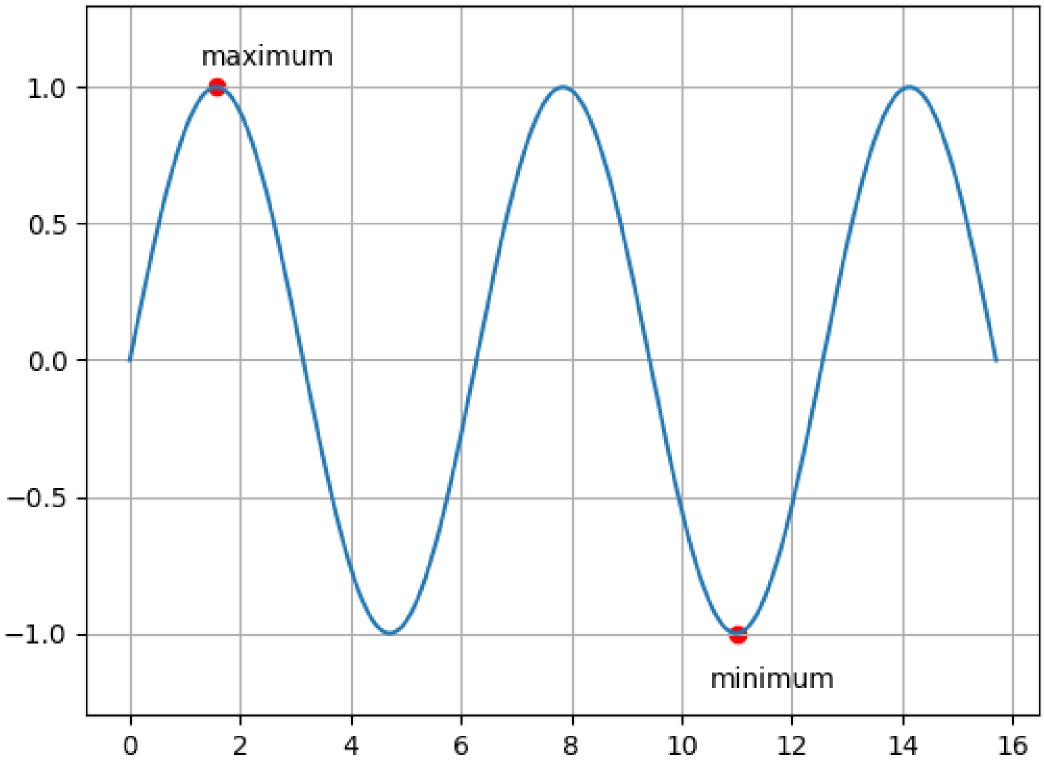


[ ]:
```
x = np.linspace(0, 5*np.pi, 1000)
plt.plot(x, np.sin(x));
```



[ ]:

```
plt.plot(x, np.sin(x))
plt.text(np.pi/2-0.3, 1+0.08, "maximum")
plt.scatter(np.pi/2, 1, color="red")
plt.scatter(11, -1, color="red")
plt.text(10.5, -1.2, "minimum")
plt.ylim(-1.3, 1.3);
```



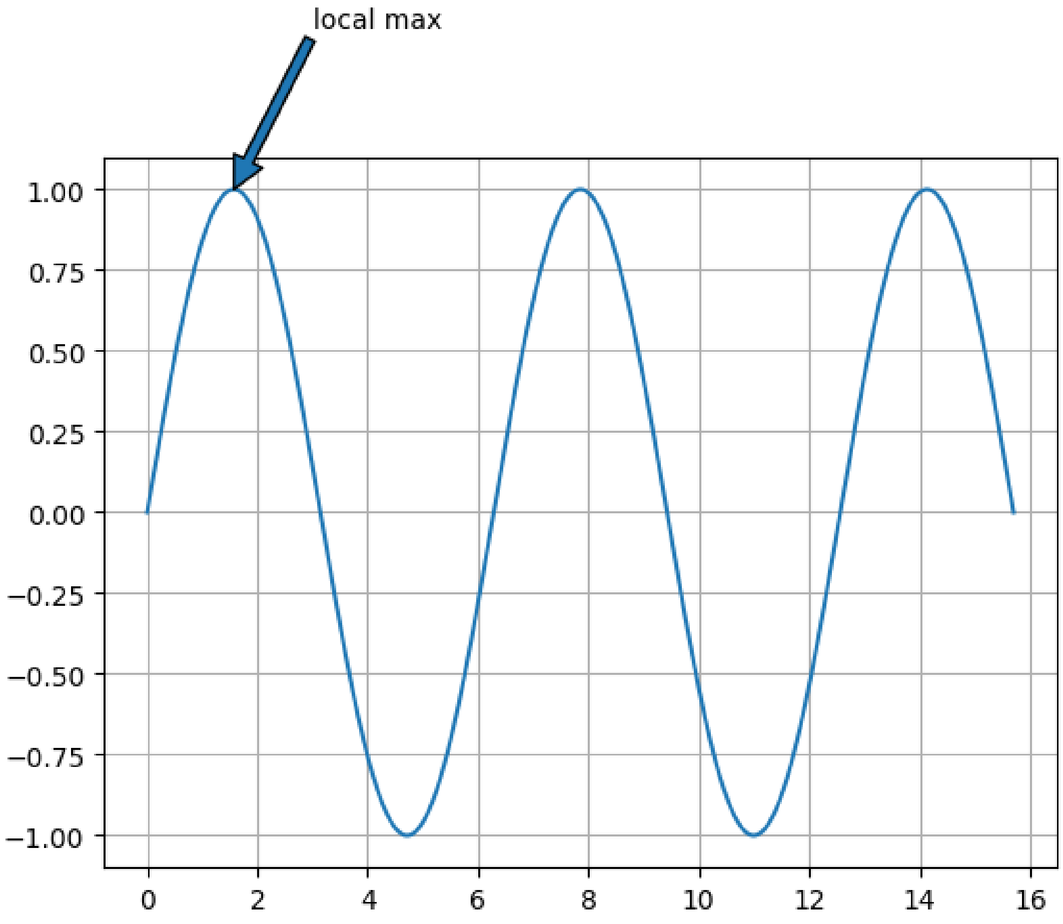
3.2 Adding annotations with `plt.annotate()`

The `plt.annotate()` function allows you to add text and connect it to a point using an arrow.

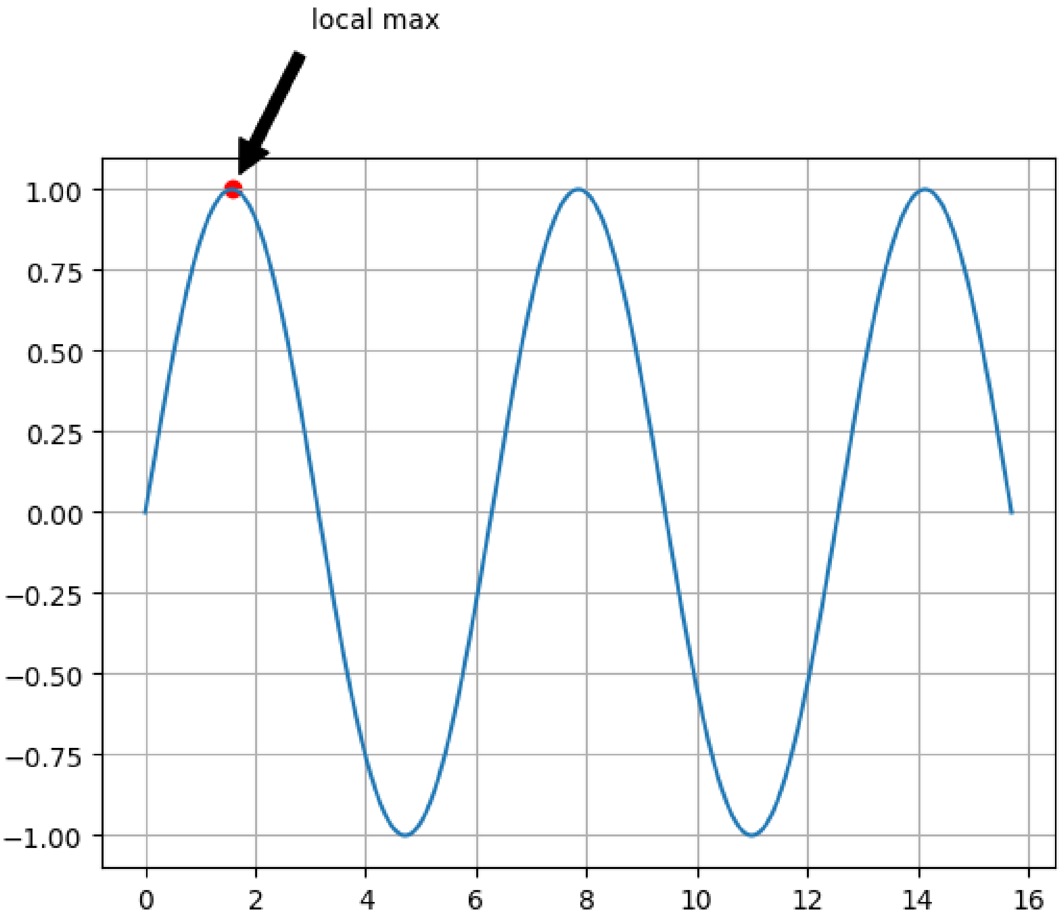
`plt.annotate('text', xy=(x, y), xytext=(x_text, y_text), arrowprops=dict()...)`

[ ]:

```
plt.plot(x, np.sin(x))
plt.annotate("local max", xy=(np.pi/2, 1), xytext=(3, 1.5), arrowprops=dict());
```



[ ]:
```
plt.plot(x, np.sin(x))
plt.scatter(np.pi/2, 1, color="red")
plt.annotate("local max", xy=(np.pi/2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor="black", shrink=0.1));
```



[ ]:

```

```

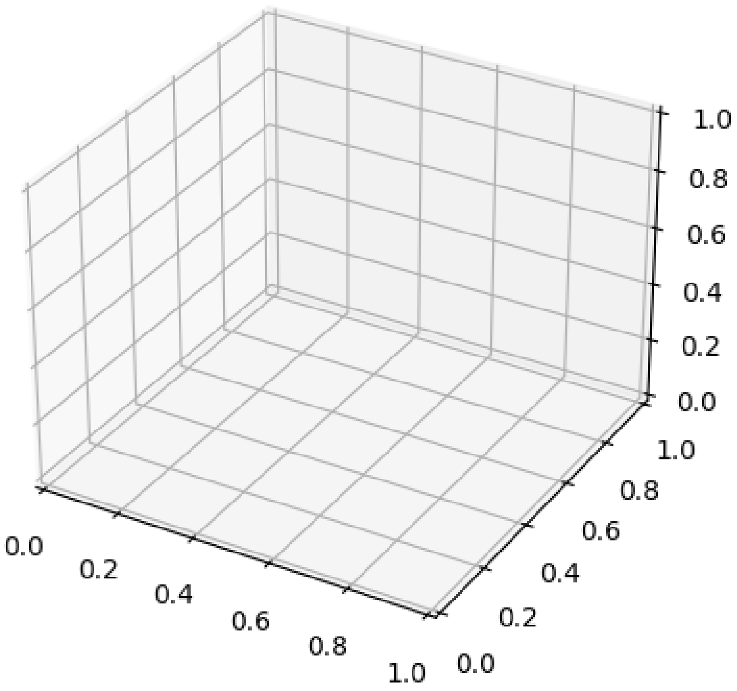
[ ]:

```
# Just run this cell
from mpl_toolkits import mplot3d
```

3.3 Three-Dimensional Plotting (8 Points)

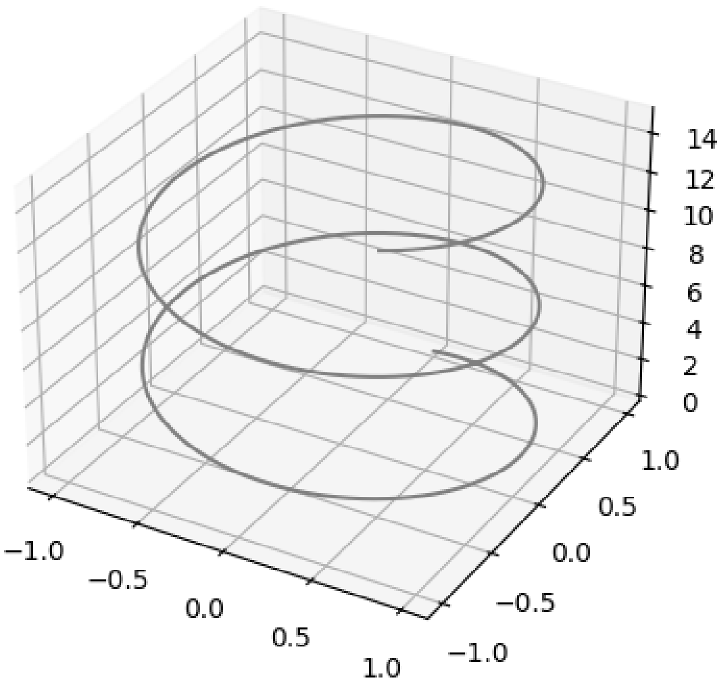
[ ]:

```
fig = plt.figure()
ax = plt.axes(projection='3d');
```



`ax.plot3D()` and `ax.scatter3D`

```
[ ]: fig = plt.figure()
     ax = plt.axes(projection='3d')
     zline = np.linspace(0, 15, 1000)
     xline = np.sin(zline)
     yline = np.cos(zline)
     ax.plot3D(xline, yline, zline, 'gray');
```

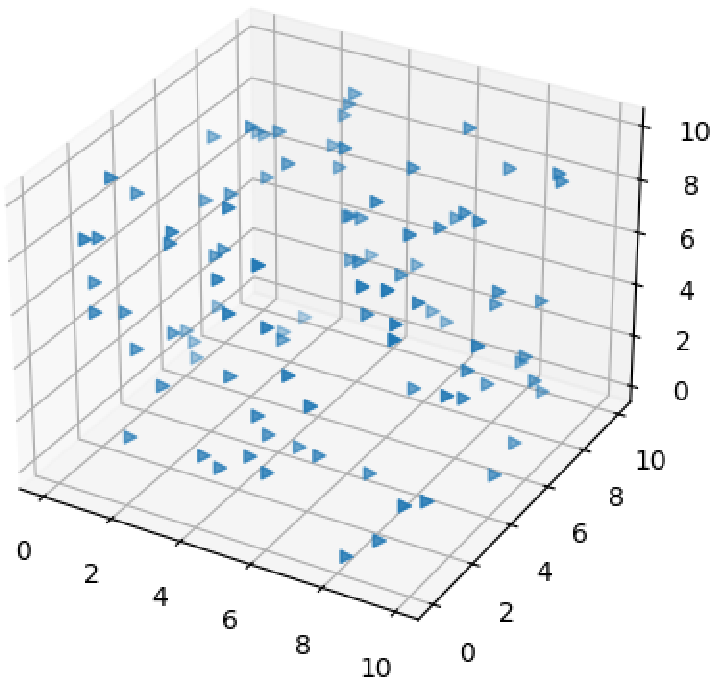


[ ]:

```
x = np.random.rand(100) * 10
y = np.random.rand(100) * 10
z = np.random.rand(100) * 10
```

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(x, y, z, marker=">");
```



Note: When you create a 3D axis using `projection='3d'`, `ax.scatter()` also creates a 3D plot.

[ ]:
```
%matplotlib notebook
```

3.3.1 Three-Dimensional Points and Lines

[ ]:
```
ax = plt.axes(projection='3d')

# Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'gray')

# Data for three-dimensional scattered points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.randn(100)
ydata = np.cos(zdata) + 0.1 * np.random.randn(100)
ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='Greens');
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Example: Generate 100 random points for x, y, and z using a normal distribution. Then, plot a

3-D scatterplot of x, y, and z. - Use different colors for each point based on their z-value. - Add a color bar to indicate the color mapping. - Label the axes and plot

```
[ ]: fig = plt.figure()
     ax = plt.axes(projection='3d')
     x = np.random.normal(size=100)
     y = np.random.normal(size=100)
     z = np.random.normal(size=100)
     scatter = ax.scatter3D(x, y, z, c=z, cmap='viridis');
     ax.set_title('3D Scatter Plot')
     ax.set_xlabel('X Axis')
     ax.set_ylabel('Y Axis')
     ax.set_zlabel('Z Axis')
     fig.colorbar(scatter, ax=ax, shrink=0.5, aspect=5, pad=0.1);
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

3.3.2 Three-Dimensional Contour Plots

```
[ ]: def f(x, y):
         return np.sin(np.sqrt(x ** 2 + y ** 2))

     x = np.linspace(-6, 6, 30)
     y = np.linspace(-6, 6, 30)
     X, Y = np.meshgrid(x, y)
     Z = f(X, Y)

     fig = plt.figure()
     ax = plt.axes(projection='3d')
     ax.contour3D(X, Y, Z, 40, cmap='binary')
     ax.set_xlabel('x')
     ax.set_ylabel('y')
     ax.set_zlabel('z');
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

```
[ ]: ax.view_init(60, 35)
     fig
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

3.3.3 Wireframes and Surface Plots

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, Z);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none');
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Example: Create a grid of x and y values from -5 to 5 with a step of 0.25. Calculate the z values using the function

$$z = sin(x^2 + y^2)$$

Then, plot a 3-D surface plot of x, y, and z. (4 points) - Use a color map to visualize the height (z-values). - Add a color bar to show the color mapping. - Label the axes and plot

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection='3d')

x = np.arange(-5, 5, 0.25)
y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(x, y)

Z = np.sin(np.sqrt(X**2 + Y**2))

surface = ax.plot_surface(X, Y, Z, cmap='coolwarm')
ax.set_title('3D Surface Plot')
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_zlabel('Z Axis')
fig.colorbar(surface, ax=ax, shrink=0.5, aspect=5, pad=0.1);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

3.3.4 Surface Triangulations

[ ]:

```
theta = 2 * np.pi * np.random.random(1000)
r = 6 * np.random.random(1000)
x = np.ravel(r * np.sin(theta))
y = np.ravel(r * np.cos(theta))
```

```
z = f(x, y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(x, y, z, c=z, cmap='viridis', linewidth=0.5);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_trisurf(x, y, z, cmap='viridis',
                edgecolor='none');
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

3.4 3D Contour plots using `ax.contour3D()`

`contour3D` generates contour lines (level curves) in 3D space that represent constant values of the function. The contours are like boundaries between different regions of Z-values at specific "levels."

[ ]:

```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
```

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection="3d")
ax.contour3D(X, Y, Z, 40);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

```

```

3.5 3D surface plot using `ax.plot_surface()`

`plot_surface` generates a 3D surface plot, which visualizes the entire surface defined by the function over the 2D grid. This provides a more "solid" view of the surface, where the entire surface is rendered with polygons, and colors can represent variations in height or slope across the surface.

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection="3d")
ax.plot_surface(X, Y, Z);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

```
```

3.6 Wireframe plot `ax.plot_wireframe()`

A wireframe plot is a way of representing data in three dimensions using lines to represent the shape of an object. A wireframe plot connects the data points of an object to create a mesh-like structure to show the shape of the object.

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection="3d")
ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

```
```

4 In-class activity 2: Generate a surface plot and a wireframe plot of the function $Z = X^2 + Y^2$.

[ ]:

```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
Z = X**2 + Y**2;

# generate a surface plot

fig = plt.figure()
ax = plt.axes(projection="3d")
ax.plot_surface(X, Y, Z);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

```
fig = plt.figure()
ax = plt.axes(projection="3d")
ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

5 Seaborn

Seaborn is a Python library built on top of Matplotlib, offering a simplified and aesthetically pleasing interface. It's tightly integrated with pandas DataFrames, making it ideal for quick data exploration and visualization. Seaborn makes it easier to create complex visualizations like statistical plots and pair plots with minimal code.

Use `%pip install --upgrade seaborn` to update the seaborn

```
[ ]: %matplotlib notebook
     fig = plt.figure()
     ax = plt.axes(projection="3d")
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

```
[ ]: %matplotlib inline
     import seaborn as sns
     import matplotlib.pyplot as plt
```

```
[ ]: df = sns.load_dataset("tips")
     df.head(5)
```

```
---------------------------------------------------------------------------
SSLCertVerificationError                  Traceback (most recent call last)
File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
  ↪request.py:1344, in AbstractHTTPHandler.do_open(self, http_class, req,␣
  ↪**http_conn_args)
   1343 try:
-> 1344     h.request(req.get_method(), req.selector, req.data, headers,
   1345               encode_chunked=req.has_header('Transfer-encoding'))
   1346 except OSError as err: # timeout error

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
  ↪client.py:1327, in HTTPConnection.request(self, method, url, body, headers,␣
  ↪encode_chunked)
   1326 """Send a complete request to the server."""
-> 1327 self._send_request(method, url, body, headers, encode_chunked)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
  ↪client.py:1373, in HTTPConnection._send_request(self, method, url, body,␣
  ↪headers, encode_chunked)
   1372     body = _encode(body, 'body')
-> 1373 self.endheaders(body, encode_chunked=encode_chunked)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
  ↪client.py:1322, in HTTPConnection.endheaders(self, message_body,␣
  ↪encode_chunked)
   1321     raise CannotSendHeader()
-> 1322 self._send_output(message_body, encode_chunked=encode_chunked)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
  ↪client.py:1081, in HTTPConnection._send_output(self, message_body,␣
  ↪encode_chunked)
   1080 del self._buffer[:]
-> 1081 self.send(msg)
```

```
   1083 if message_body is not None:
   1084
   1085     # create a consistent interface to message_body

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
  ↪client.py:1025, in HTTPConnection.send(self, data)
   1024 if self.auto_open:
-> 1025     self.connect()
   1026 else:

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/http/
  ↪client.py:1468, in HTTPSConnection.connect(self)
   1466     server_hostname = self.host
-> 1468 self.sock = self._context.wrap_socket(self.sock,
   1469                                       server_hostname=server_hostname)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/ssl.py:
  ↪455, in SSLContext.wrap_socket(self, sock, server_side,␣
  ↪do_handshake_on_connect, suppress_ragged_eofs, server_hostname, session)
    449 def wrap_socket(self, sock, server_side=False,
    450                 do_handshake_on_connect=True,
    451                 suppress_ragged_eofs=True,
    452                 server_hostname=None, session=None):
    453     # SSLSocket class handles server_hostname encoding before it calls
    454     # ctx._wrap_socket()
--> 455     return self.sslsocket_class._create(
    456         sock=sock,
    457         server_side=server_side,
    458         do_handshake_on_connect=do_handshake_on_connect,
    459         suppress_ragged_eofs=suppress_ragged_eofs,
    460         server_hostname=server_hostname,
    461         context=self,
    462         session=session
    463     )

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/ssl.py:
  ↪1046, in SSLSocket._create(cls, sock, server_side, do_handshake_on_connect,␣
  ↪suppress_ragged_eofs, server_hostname, context, session)
   1045             raise ValueError("do_handshake_on_connect should not be␣
  ↪specified for non-blocking sockets")
-> 1046         self.do_handshake()
   1047 except (OSError, ValueError):

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/ssl.py:
  ↪1321, in SSLSocket.do_handshake(self, block)
   1320         self.settimeout(None)
-> 1321     self._sslobj.do_handshake()
   1322 finally:
```

```
SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify␣
 ↪failed: unable to get local issuer certificate (_ssl.c:1000)

During handling of the above exception, another exception occurred:

URLError                                  Traceback (most recent call last)
Cell In[98], line 1
----> 1 df = sns.load_dataset("tips")
      2 df.head(5)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
 ↪site-packages/seaborn/utils.py:572, in load_dataset(name, cache, data_home,␣
 ↪**kws)
    570 cache_path = os.path.join(get_data_home(data_home), os.path.
 ↪basename(url))
    571 if not os.path.exists(cache_path):
--> 572     if name not in get_dataset_names():
    573         raise ValueError(f"'{name}' is not one of the example datasets. )
    574     urlretrieve(url, cache_path)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
 ↪site-packages/seaborn/utils.py:499, in get_dataset_names()
    493 def get_dataset_names():
    494     """Report available example datasets, useful for reporting issues.
    495
    496     Requires an internet connection.
    497
    498     """
--> 499     with urlopen(DATASET_NAMES_URL) as resp:
    500         txt = resp.read()
    502     dataset_names = [name.strip() for name in txt.decode().split("\n")]

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:215, in urlopen(url, data, timeout, cafile, capath, cadefault,␣
 ↪context)
    213 else:
    214     opener = _opener
--> 215 return opener.open(url, data, timeout)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:515, in OpenerDirector.open(self, fullurl, data, timeout)
    512     req = meth(req)
    514 sys.audit('urllib.Request', req.full_url, req.data, req.headers, req.
 ↪get_method())
--> 515 response = self._open(req, data)
    517 # post-process response
    518 meth_name = protocol+"_response"
```

```
File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:532, in OpenerDirector._open(self, req, data)
    529     return result
    531 protocol = req.type
--> 532 result = self._call_chain(self.handle_open, protocol, protocol +
    533                           '_open', req)
    534 if result:
    535     return result

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:492, in OpenerDirector._call_chain(self, chain, kind, meth_name,␣
 ↪*args)
    490 for handler in handlers:
    491     func = getattr(handler, meth_name)
--> 492     result = func(*args)
    493     if result is not None:
    494         return result

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:1392, in HTTPSHandler.https_open(self, req)
   1391 def https_open(self, req):
-> 1392     return self.do_open(http.client.HTTPSConnection, req,
   1393                         context=self._context)

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/
 ↪request.py:1347, in AbstractHTTPHandler.do_open(self, http_class, req,␣
 ↪**http_conn_args)
   1344         h.request(req.get_method(), req.selector, req.data, headers,
   1345                   encode_chunked=req.has_header('Transfer-encoding'))
   1346     except OSError as err: # timeout error
-> 1347         raise URLError(err)
   1348     r = h.getresponse()
   1349 except:

URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify␣
 ↪failed: unable to get local issuer certificate (_ssl.c:1000)>
```

[ ]:

```python
plt.figure(); # Reset the environment
```

```
<Figure size 640x480 with 0 Axes>
```

5.1 Create scatterplot using sns.scatterplot()

[ ]:

```python
sns.scatterplot(data=df, x="total_bill", y="tip")
plt.title("total_bill vs tip");
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[54], line 1
----> 1 sns.scatterplot(data=df, x="total_bill", y="tip")
      2 plt.title("total_bill vs tip");

NameError: name 'df' is not defined
```

```
[ ]: sns.scatterplot(data=df, x="total_bill", y="tip", hue="time");
```

```
[ ]: sns.jointplot(data=df, x="total_bill", y="tip", hue="time",marker="x", s=50,);
```

5.2 create lineplot using sns.lineplot()

```
[ ]: sns.lineplot(data=df, x="size", y="tip", errorbar='sd')

     plt.title("Tip amount vs Party size");
```

```
[ ]: sns.set_style("whitegrid")
     sns.lineplot(data=df, x="size", y="tip", errorbar='sd')
     plt.title("Tip amount vs Party size");
```

```
[ ]: sns.barplot(data=df, x="size", y="tip");
```

A quick linear regression model

```
[ ]: sns.lmplot(data=df, x="total_bill", y="tip", hue="sex", ci = 95,␣
      ↪scatter_kws={'alpha':0.5},)

     # Add a title
     plt.title("Line Plot of Total Bill vs Tip");
```

5.3 create barplot using sns.barplot()

```
[ ]: sns.set_style("white")
```

```
[ ]: sns.barplot(data=df, x="day", y="total_bill", hue="time");
```

5.4 Create violin plot using sns.violinplot()

Visualizing the distribution of data across categories, combining aspects of a box plot and a density plot.

```
[ ]: sns.violinplot(data=df, x="day", y="total_bill", hue="time",␣
      ↪split=True,inner="quartile", linewidth=1);
```

[ ]:

```
sns.violinplot(data=df, x="day", y="total_bill", hue="time",inner="quartile",␣
 ↪linewidth=1);
```

[ ]:

```

```

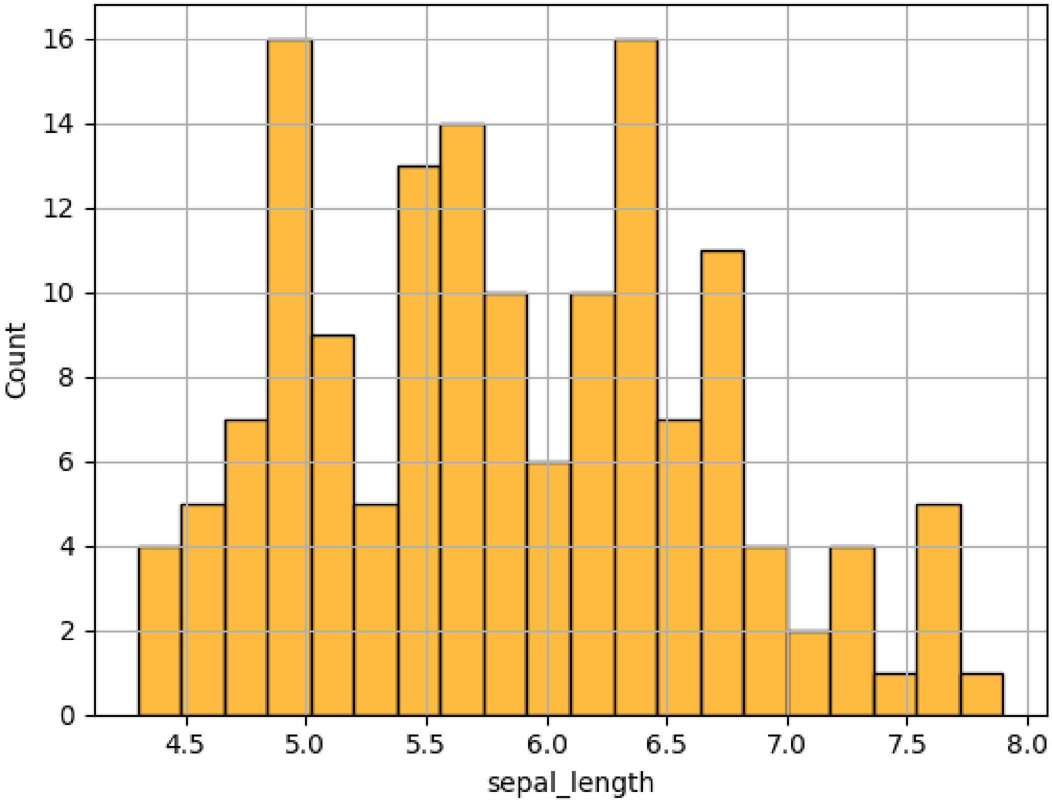
5.5 create histogram using sns.histplot()

[ ]:

```
iris = pd.read_csv("iris.csv")
```

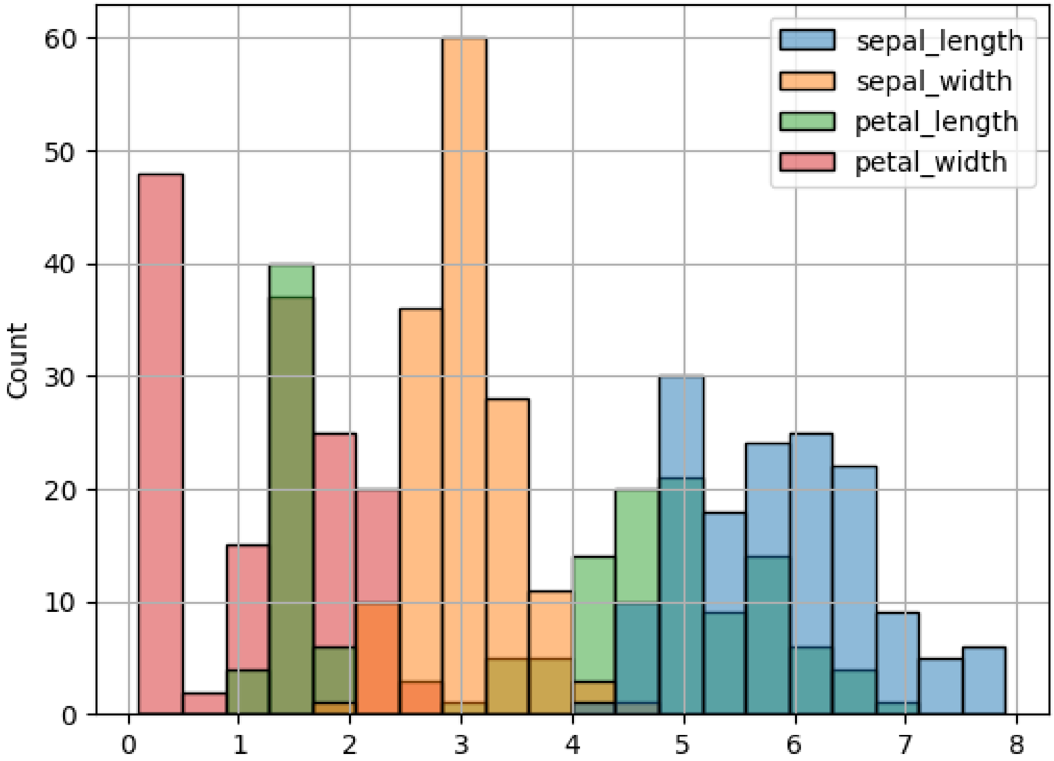
[ ]:

```
sns.histplot(iris["sepal_length"], bins=20, color="orange")
plt.show()
```

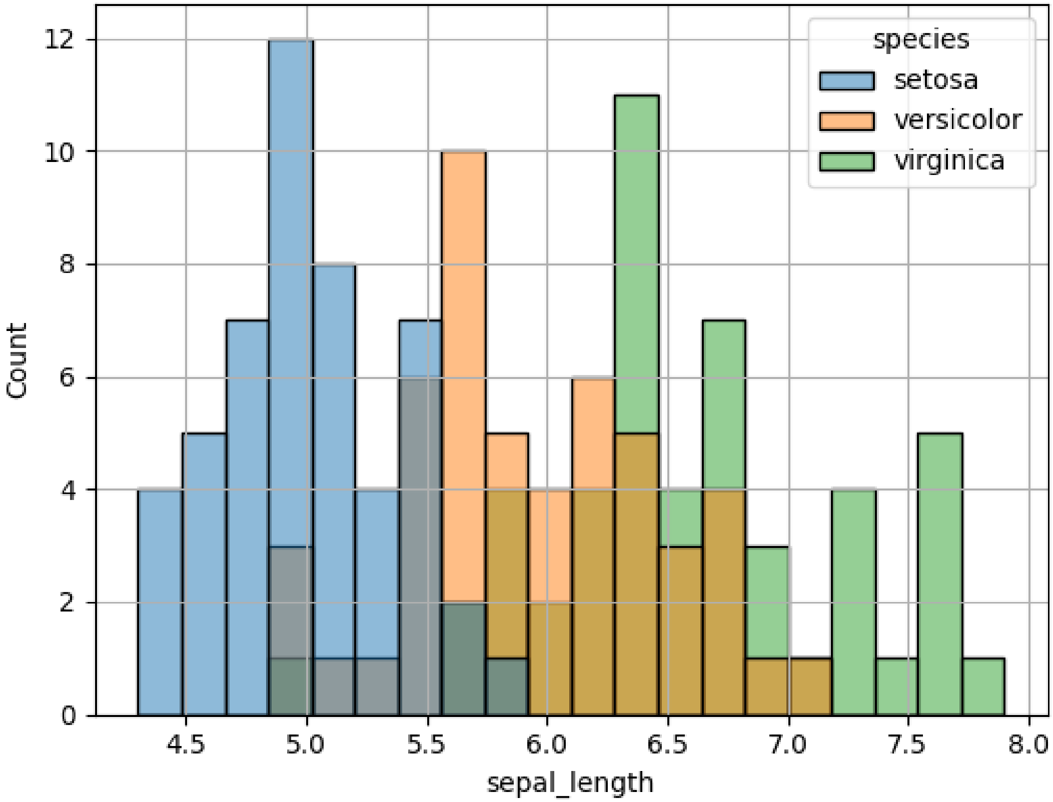


[ ]:

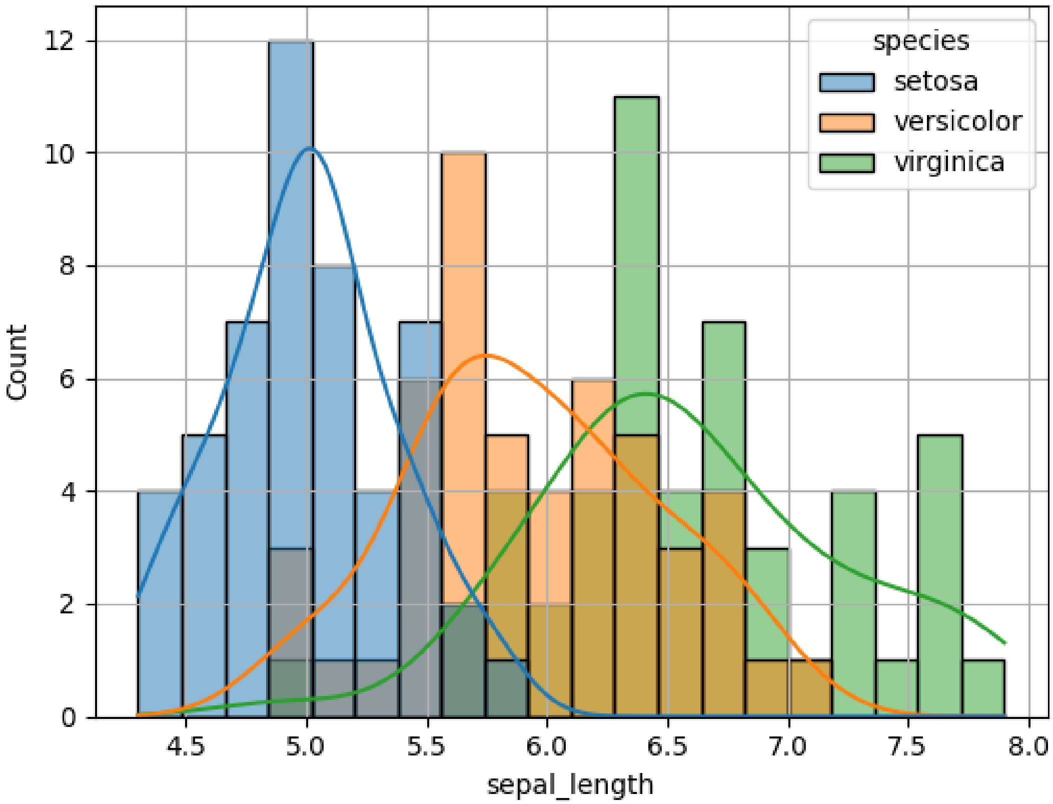
```
sns.histplot(iris, bins=20);
```



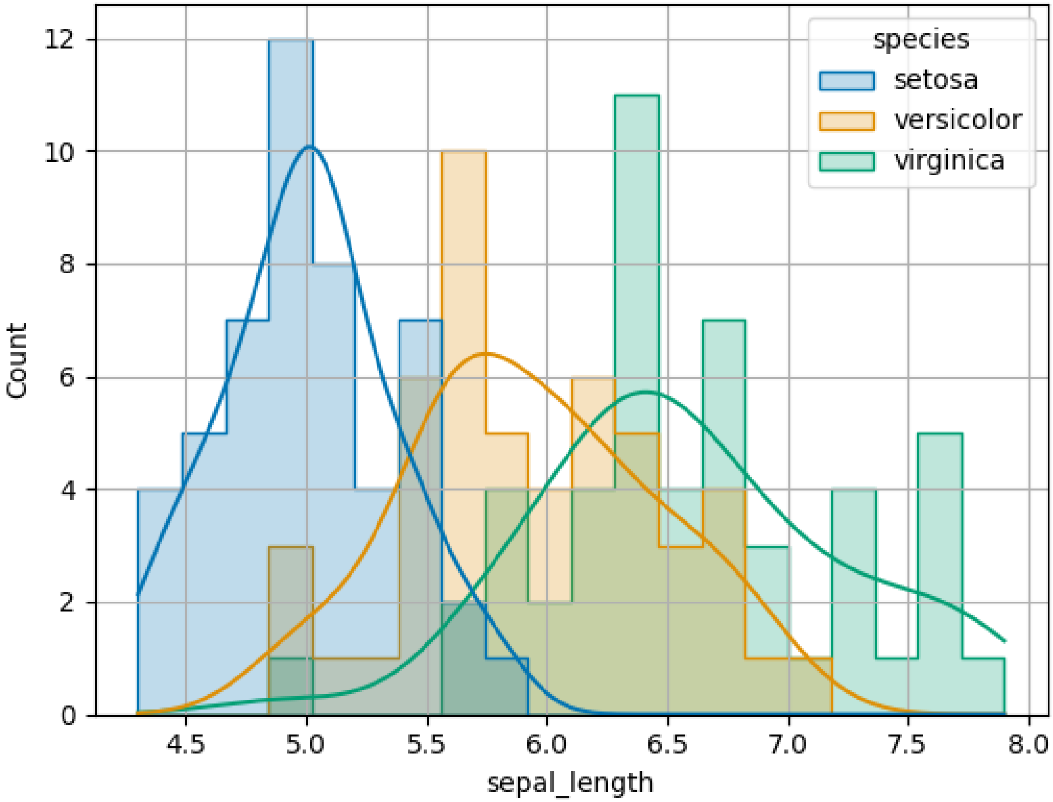
```
[ ]: sns.histplot(iris, x="sepal_length", bins=20,hue="species");
```



```
[ ]: sns.histplot(iris, x="sepal_length", bins=20,hue="species", kde = True);
```

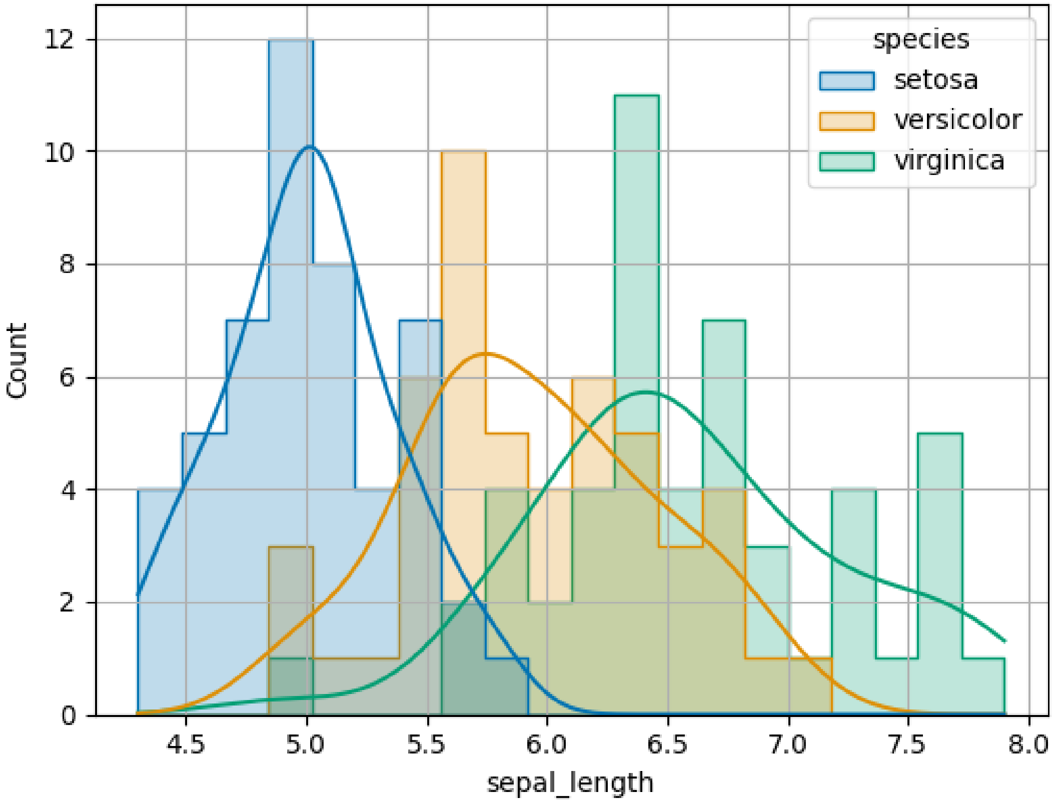


```
[ ]: sns.histplot(iris, x="sepal_length", bins=20,hue="species", kde = True,␣
      ↪element="step");
```



[ ]:

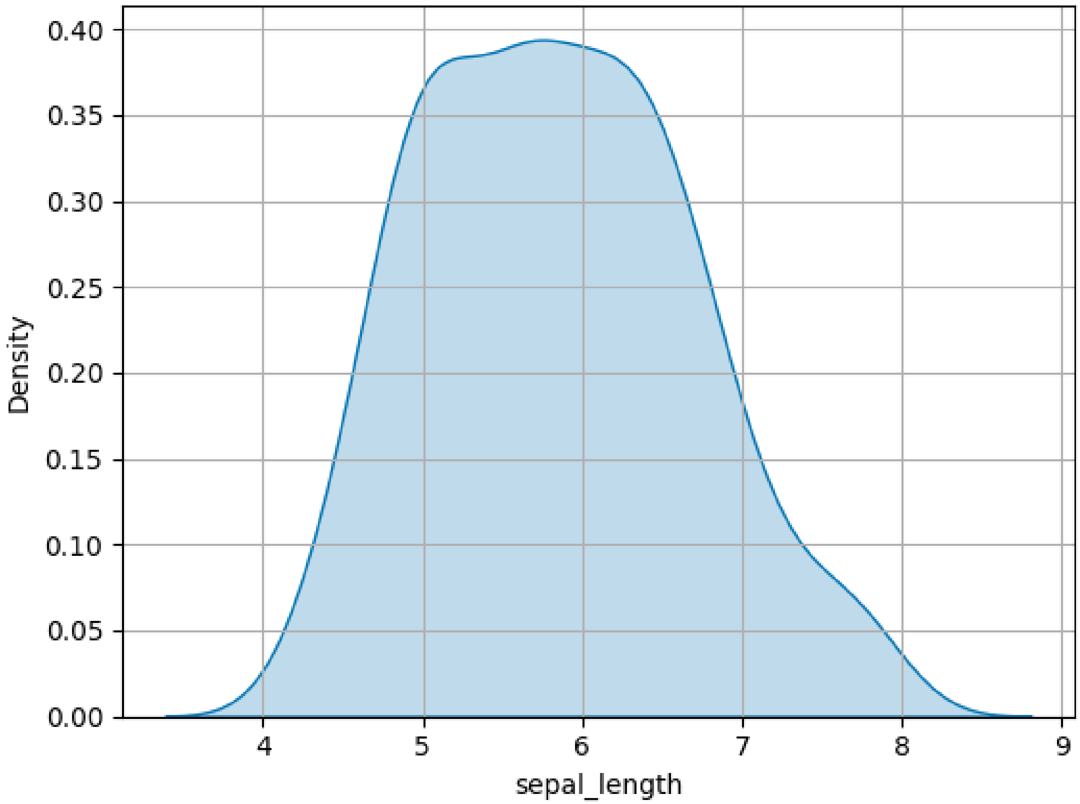
```
sns.set_palette("colorblind")
sns.histplot(iris, x="sepal_length", bins=20,hue="species", kde = True,␣
  ↪element="step");
```



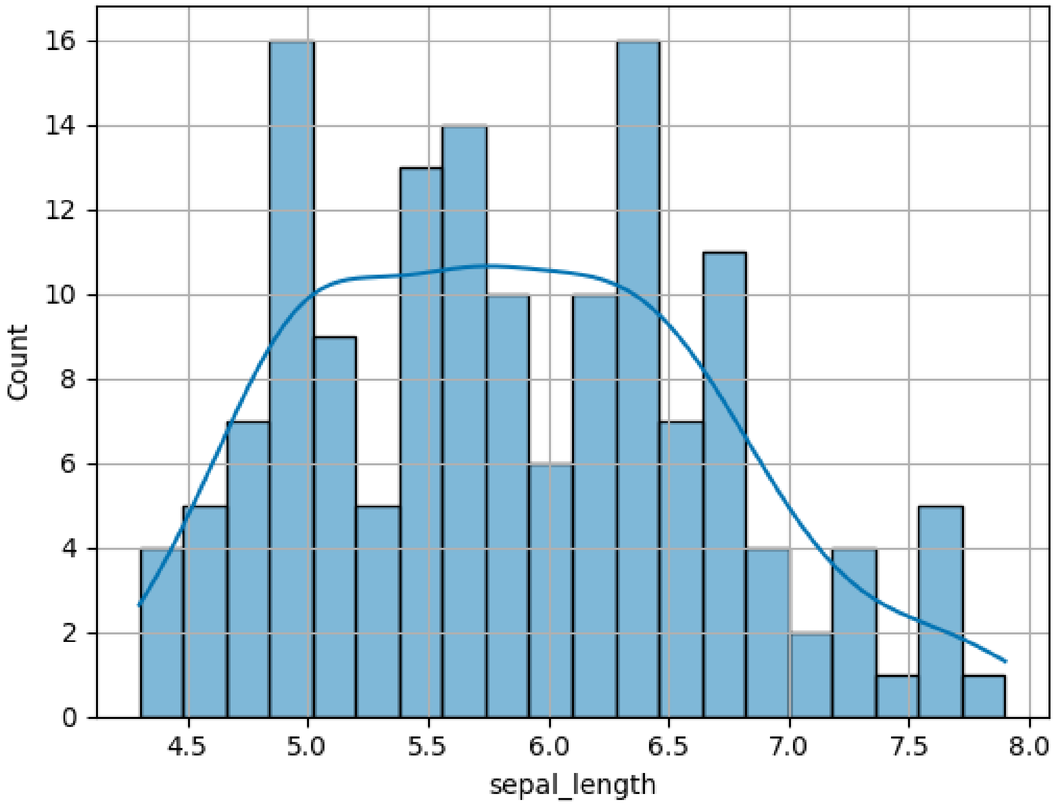
5.6 create kernel density estimate (KDE) plot using `sns.kdeplot()`

a smoothed continuous curve represents the probability density of a dataset.

```
[ ]: sns.kdeplot(iris['sepal_length'], fill=True);
```



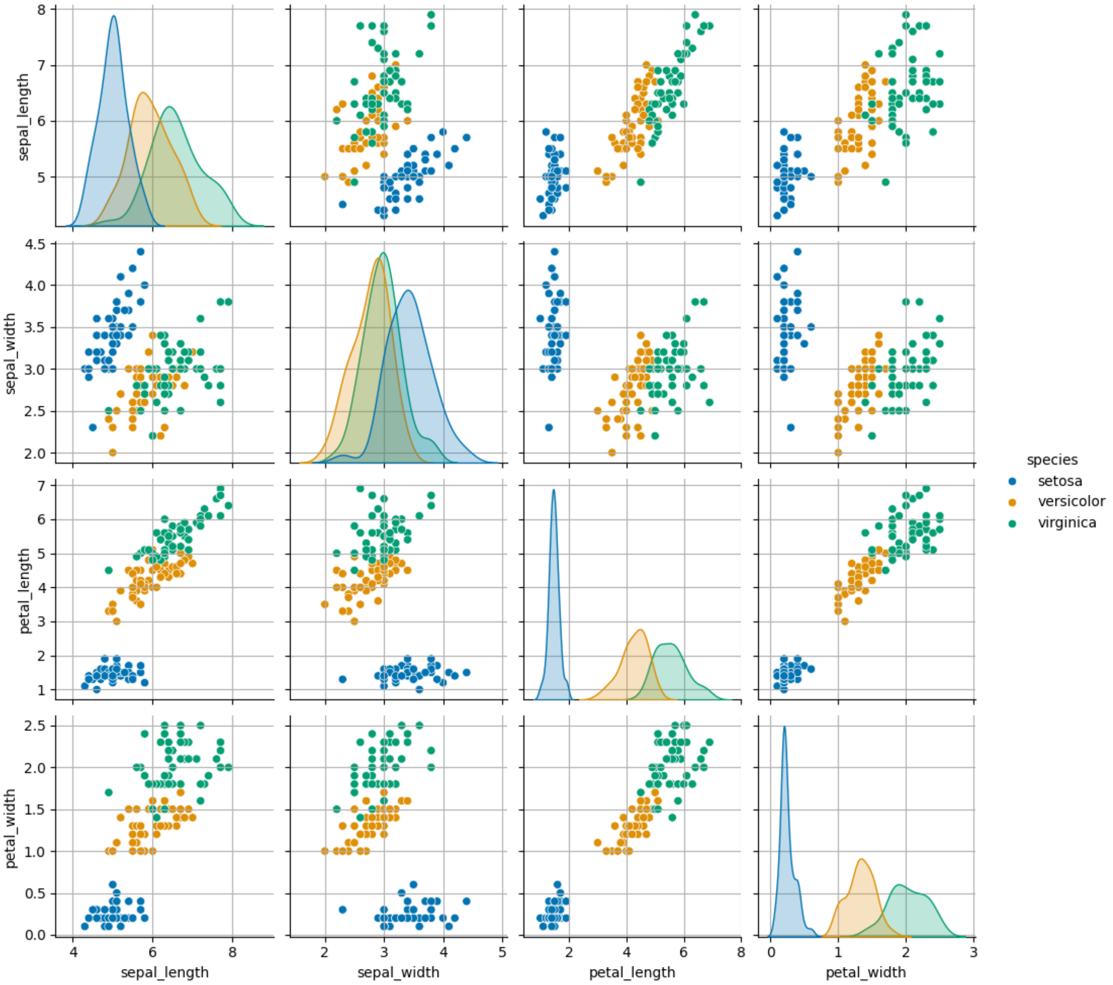
```
[ ]: sns.histplot(iris['sepal_length'], bins=20, kde=True);
```



5.7 create pair plot using `sns.pairplot()`

Visualizing relationships between multiple variables in a DataFrame.

```
[ ]: sns.pairplot(iris, hue='species');
```



5.8 sns.catplot()

Use `sns.catplot()` for more flexibility, especially if you want to further customize the layout (e.g., faceting, multi-dimensional plotting)

```
[ ]: sns.catplot(data=df, x='day', y='total_bill', hue='time');
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[109], line 1
----> 1 sns.catplot(data=df, x='day', y='total_bill', hue='time');

NameError: name 'df' is not defined
```

[ ]:

```
sns.catplot(data=df, x="day", y="total_bill", hue="time", kind="box",␣
 ↪palette="pastel");
```

[ ]:

```
sns.catplot(data=df, x="day", y="total_bill", hue="smoker", kind="box");
```

[ ]:

```
sns.catplot(
    data=df, x="day", y="total_bill", hue="sex",
    kind="violin", inner="stick", split=True, palette="pastel");
```

[ ]:

```
```

[ ]:

```
```