BINF 5007

Lecture 12: Plotting with Seaborn and Pandas

Learning Objectives

1.Simple Plotting with Matplotlib and Seaborn

Create and customize basic plots using Matplotlib and Seaborn.

2.Histograms: **kdeplot, histplot, distplot**

Create histograms with different plot types and interpret their results.

3.Draw Plots of Two Variables

Visualize relationships between two variables using scatter and line plots.

4.Pair Plots

Generate pair plots to explore relationships in multi-variable datasets.

5.Plotting Subplots

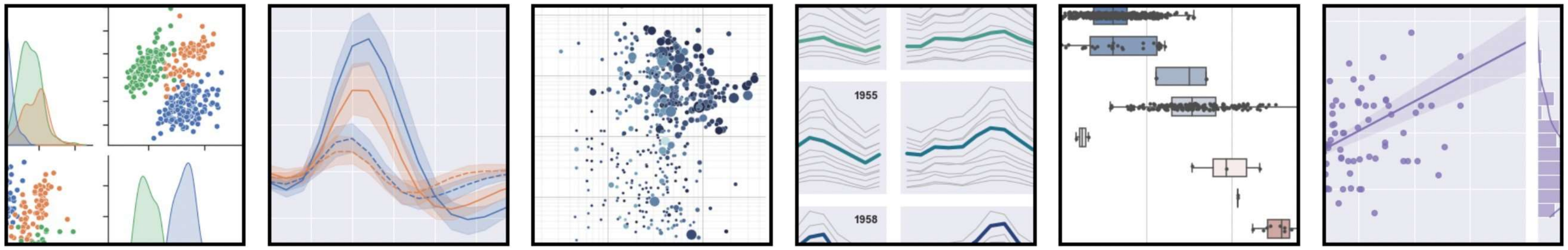
Display multiple plots in one figure using subplots and customize their layout.

Matplotlib

- Matplotlib has proven to be an incredibly useful and popular visualization tool, but there are several valid complaints about Matplotlib that often come up:
- Prior to version 2.0, Matplotlib's defaults are not exactly the best choices. It was based off of MATLAB circa 1999, and this often shows
- Matplotlib's API is relatively low level. Doing sophisticated statistical visualization is possible, but often requires a lot of boilerplate code
- The answer to these problems is Seaborn

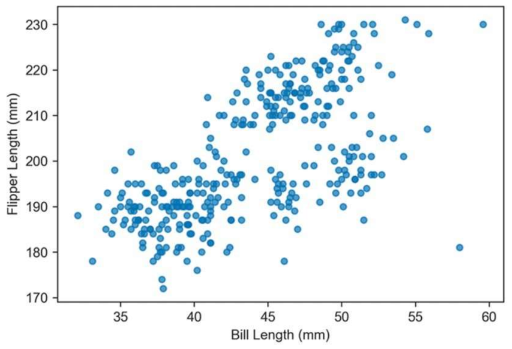
Seaborn

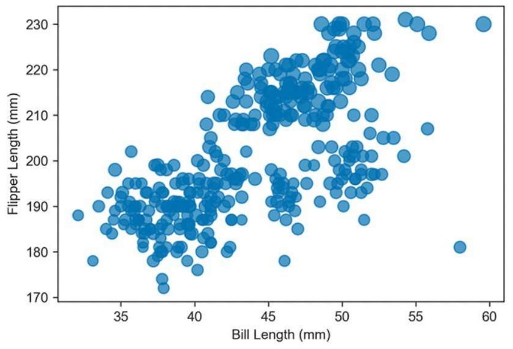
- Seaborn provides an API on top of Matplotlib that offers **sane choices for plot style and color defaults**, defines simple high-level functions for common statistical plot types, and integrates with the functionality provided by Pandas DataFrames
- The main idea of Seaborn is that it provides high-level commands to create a variety of plot types useful for statistical data exploration, and even some statistical model fitting

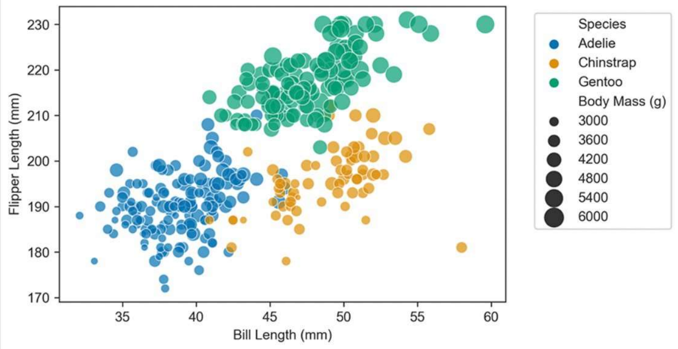


https://seaborn.pydata.org/

A Comparison







Pandas

```
penguins.plot.scatter(x = "bill_length_mm",
                      y= "flipper_length_mm",
                      alpha = 0.7,
                      color = "C0")
```

MatPlotLib

```
plt.scatter(penguins["bill_length_mm"],
            penguins["flipper_length_mm"],
            s = penguins["body_mass_g"]/50,
            alpha=0.7)
```

Seaborn

```
sns.scatterplot(data=penguins,
                x="bill_length_mm",
                y="flipper_length_mm",
                size = "body_mass_g",
                hue="species",
                sizes=(20, 200),
                alpha=0.7)
```

A comparison of the code and the resultant simple scatter plots produced by three popular packages for data visualization, using an example dataset from the Seaborn package

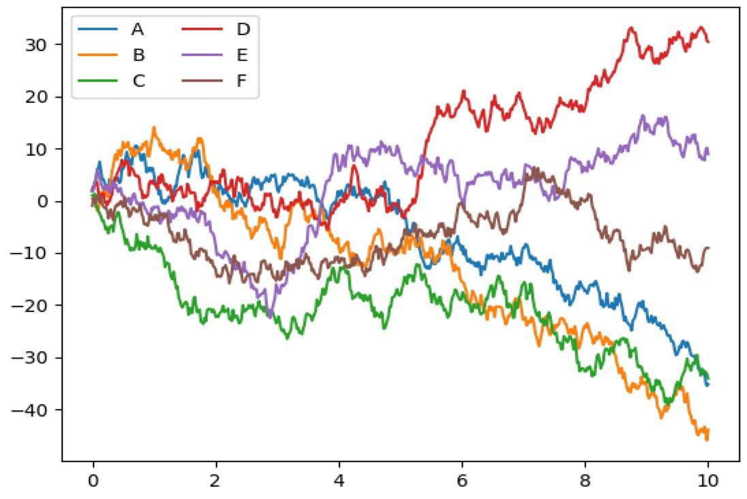
```
penguins = sns.load_dataset("penguins")
```

Classic Matplotlib

- Simple random-walk plot in Matplotlib, using its classic plot formatting and colors. We start with the typical imports:

```
import numpy as np
import matplotlib.pyplot as plt

# Create some data
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
print(x, y)
# Plot the data with Matplotlib defaults
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
plt.savefig("test1.png")
```

Chart contains all the information we'd like it to convey, but aesthetically not pleasing, and even looks a bit old-fashioned in the context of 21st-century data visualization

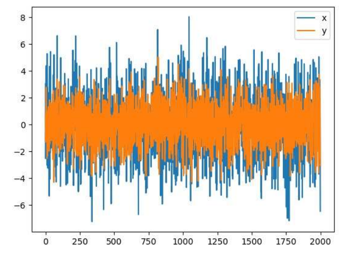


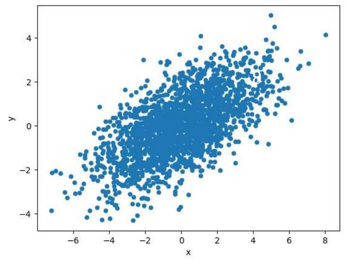
Simple DataFrame Plotting

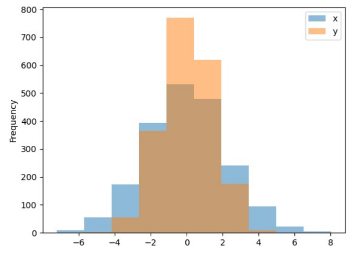
- Sometimes during your development, you want to quickly see some relationships and pandas can do that!

```
import numpy as np
import pandas as pd

data = np.random.multivariate_normal(
    [0, 0], [[5, 2], [2, 2]], size=2000)

data = pd.DataFrame(data, columns=['x', 'y'])

# do a simple line plot of all variables
ax = data.plot()
ax.figure.savefig('test2a.png’)

# do a simple scatter plot of x and y
ax = data.plot scatter(x='x', y='y')
ax.figure.savefig('test2b.png’)

# do a simple histogram plot of x and y
ax = data.plot hist(alpha=0.5)
ax.figure.savefig('test2c.png')
```

These charts are once again not the most aesthetically pleasing, but they do help you during development to get quick views!







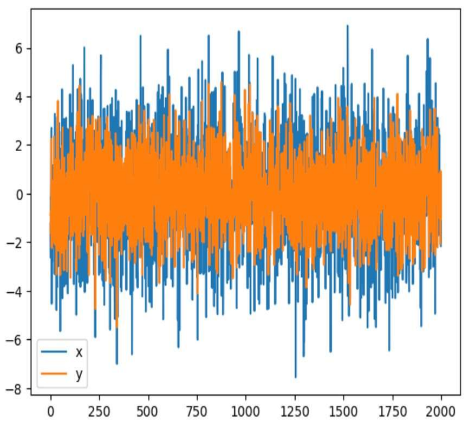
Simple DataFrame Plotting

```
In [1]: import numpy as np
        import pandas as pd

        data = np.random.multivariate_normal(
            [0, 0], [[5, 2], [2, 2]], size=2000)

        data = pd.DataFrame(data, columns=['x', 'y'])
```

```
In [4]: data.head()
Out[4]:
```

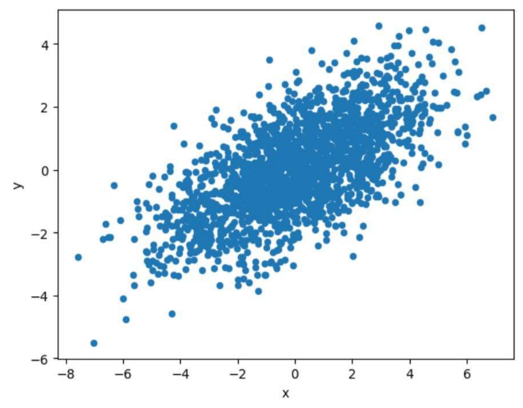
	x	y
0	-1.225056	-0.865099
1	-2.630913	-0.769030
2	-0.158060	-0.720066
3	-0.159920	-1.112421
4	0.444219	-1.670484

```
In [5]: data.plot()
Out[5]: <Axes: >
```

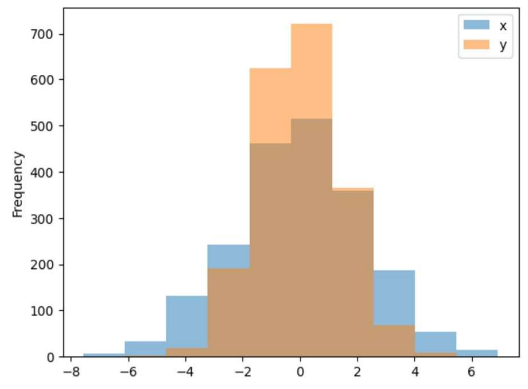


Plotting works nicely in Jupyter Notebooks!

```
In [6]: data.plot.scatter(x='x', y='y')
Out[6]: <Axes: xlabel='x', ylabel='y'>
```



```
In [7]: data.plot.hist(alpha=0.5)
Out[7]: <Axes: ylabel='Frequency'>
```



Using Jupyter Notebooks is out of the scope of this course, but a good skill to learn
https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/

Why Use Jupyter Notebooks?

- Jupyter notebooks are interactive web-based environments that allow users to create and share documents that contain live code, equations, visualizations and explanatory text. They are widely used for data analysis, scientific computing, machine learning and education

- Notebooks enable users to execute code in various languages (such as Python, R, Julia and Scala) using kernels that run in the background and communicate with the notebook interface. Users can also switch between different kernels within the same notebook

- Notebooks support rich output formats that can display images, videos, audio, HTML, LaTeX and more. Users can also embed interactive widgets that can interact with the code and the data

- Notebooks facilitate reproducible research and collaboration by allowing users to document their workflows, results and insights in a single document that can be easily shared and reproduced by others. Users can also export their notebooks to various formats such as HTML, PDF, slideshows and scripts

- Notebooks offer a flexible way of exploring data and developing code. Users can write code in small chunks called cells that can be executed individually or in sequence. Users can also edit, delete or rearrange cells as they wish. Users can also use markdown cells to add headings, lists, links and other formatting options to their notebooks

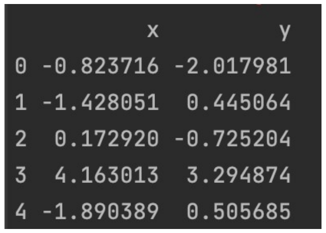
- Notebooks integrate with many popular tools and libraries for data science such as pandas, numpy, matplotlib, seaborn, scikit-learn and tensorflow. Users can also install additional packages using pip or conda commands within the notebook

- Notebooks run locally on a user's machine or remotely on a server or cloud platform. Users can also access hosted services such as Google Colab or Microsoft Azure Notebooks that provide free online environments for running jupyter notebooks

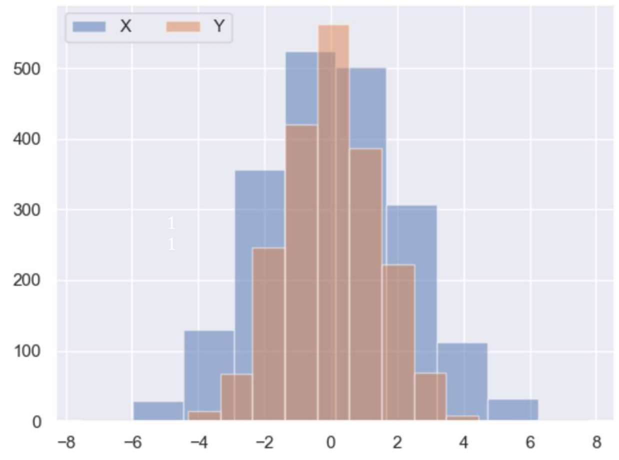
Histograms

- Often in statistical data visualization, all you want is to plot histograms and joint distributions of variables
- We have seen that this is relatively straightforward in Matplotlib and pandas. Here we'll use Seaborn:

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
sns.set()  #  We can set the style by calling Seaborn's set() method

data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]],
                                     size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])
print(data.head())
# loop over x and y
for col in 'xy':
    plt.hist(data[col], alpha=0.5)

# plot a legend and save
plt.legend('XY', ncol=2, loc='upper left');
plt.savefig("test2.png")
```



```
          x         y
0 -0.823716 -2.017981
1 -1.428051  0.445064
2  0.172920 -0.725204
3  4.163013  3.294874
4 -1.890389  0.505685
```

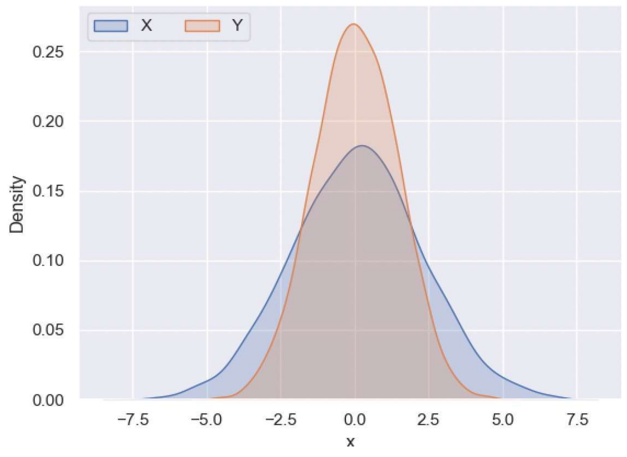


Python Data Science Handbook

Histograms: Smooth Estimate of the Distribution

- Rather than a histogram, we can get a smooth estimate of the distribution using a kernel density estimation, which Seaborn does with `sns.kdeplot`:
- Plot univariate or bivariate distributions using kernel density estimation

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
sns.set()  #  We can set the style by calling Seaborn's set() method

data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]],
                                      size=2000)
data = pd.DataFrame(data, columns=['x', 'y’])

# loop over x and y
for col in 'xy':
    sns.kdeplot(data[col], fill=True)

# plot a legend and save
plt.legend('XY', ncol=2, loc='upper left');
```

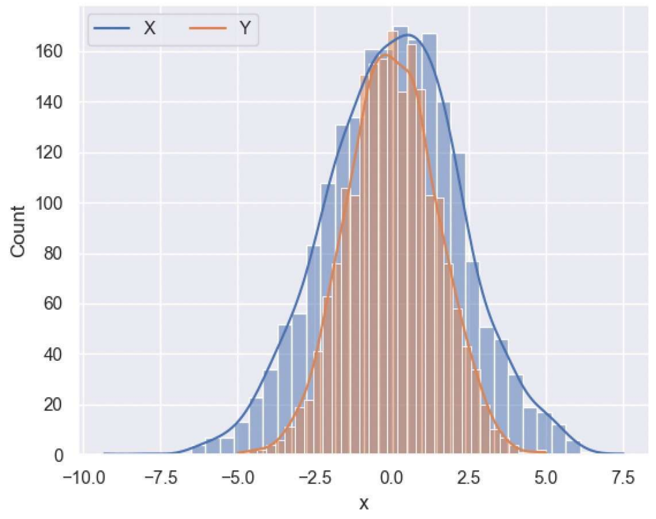


Python Data Science Handbook

Histograms: With Both

- Histograms and KDE can be combined using `histplot`:

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
sns.set()  # We can set the style by calling Seaborn's set() method

data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]],
                                      size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])

# loop over x and y
for col in 'xy':
    sns.histplot(data[col], fill=True, kde=True)

# plot a legend and save
plt.legend('XY', ncol=2, loc='upper left')
plt.savefig("test4.png")
```

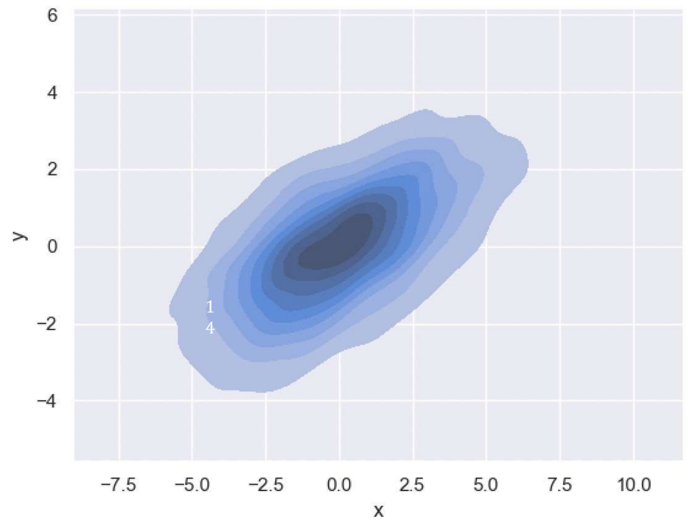


Python Data Science Handbook

Plot a Bivariate Distribution w/ Contours

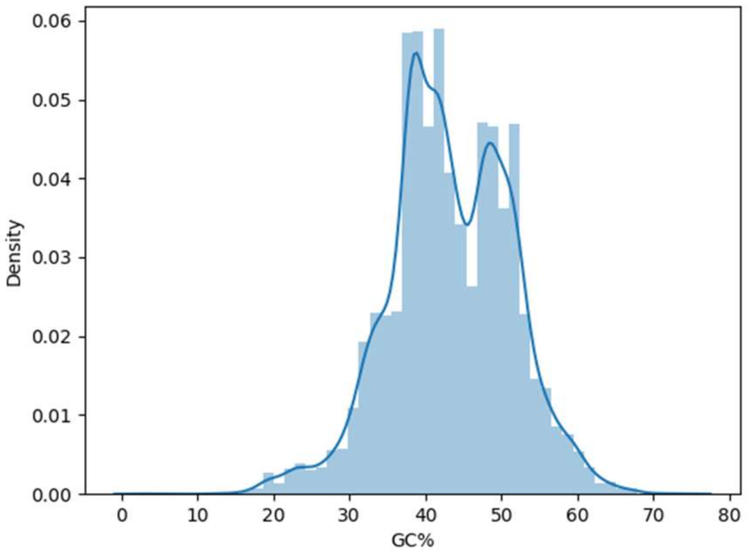
- Plot a bivariate distribution using `kdeplot`:

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
sns.set()  # We can set the style by calling Seaborn's set() method

data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]],
                                      size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])

# Plot a bivariate distribution, with filled contours:
sns.kdeplot(data=data, x="x", y="y", fill=True)

#  save
plt.savefig("test5.png")
```



Distribution plot

- The distplot function (short for *distribution plot*) takes a list of values, which can be in the form of a pandas series. Unlike the rest of the seaborn functions, distplot can't deal with missing data.



```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

my_types = {
    'Species' : 'string',
    'Kingdom' : 'string',
    'Class'   : 'string',
    'Assembly status' : 'string',
    'Number of genes' : 'Int64',
    'Number of proteins' : 'Int64'
  }

euk = pd.read_csv(
  "eukaryotes.tsv",
  sep="\t",
  dtype = my_types,
  na_values=['-']
)

euk_float = euk[euk["Size (Mb)"] < 4_000].dropna()
euk_float["Number of genes"] = euk_float["Number of genes"].astype(float)
euk_float["Number of proteins"] = euk_float["Number of proteins"].astype(float)
```

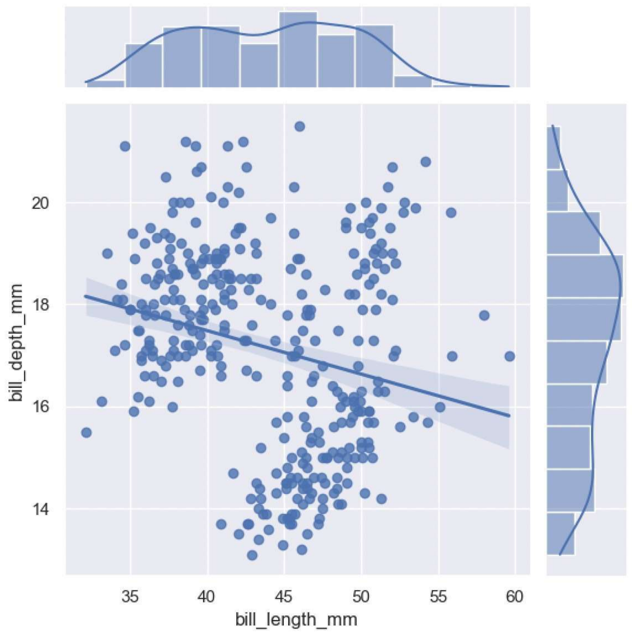
Your turn

- Using a dataset “eukaryotes.tsv", containing GC content percentages of DNA sequences, write one line code to create a distplot. Use the seaborn library to visualize the data, labeling the x-axis as "GC%" and the y-axis as "Density." Save the plot as an image file and display it in your script output.

Draw a plot of two variables with bivariate and univariate graphs

- This function provides a convenient interface to the `JointGrid` class, with several canned plot kinds using `jointplot`:

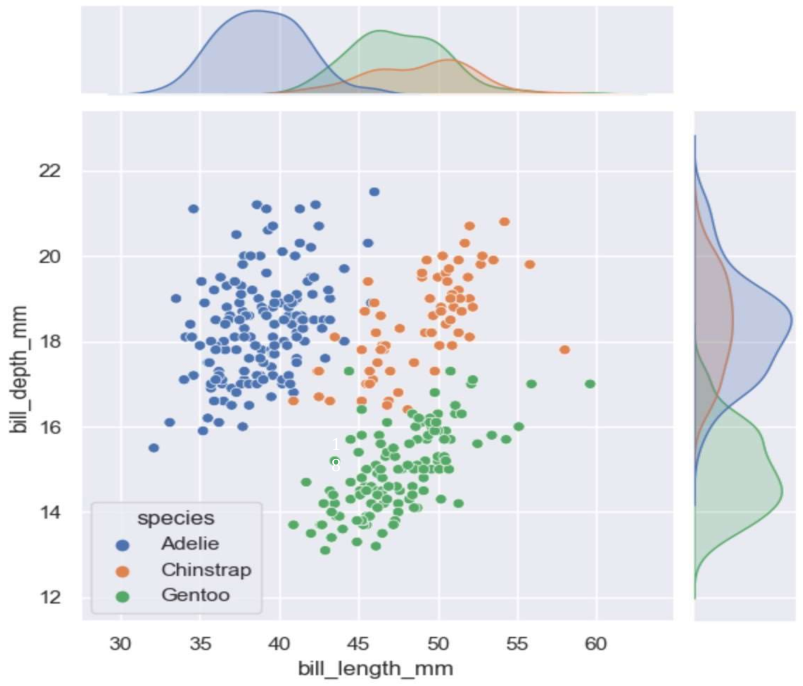
```
import seaborn as sns

import matplotlib.pyplot as plt

sns.set()        # We can set the style by calling Seaborn's set() method


penguins = sns.load_dataset("penguins")


# Set kind="reg" to add a linear regression fit (using regplot())

# and univariate KDE curves:
sns.jointplot(data=penguins, x="bill_length_mm",
                  y="bill_depth_mm", kind="reg")


#    save
plt.savefig("test6.png")
```



Draw a plot of two variables with bivariate and univariate graphs

- Assigning a hue variable will add conditional colors to the scatterplot and draw separate density curves (using `kdeplot()`) on the marginal axes:

```
import seaborn as sns

import matplotlib.pyplot as plt

sns.set()        # We can set the style by calling Seaborn's set() method


penguins = sns.load_dataset("penguins")


# Assigning a hue variable will add conditional colors to the

# scatterplot and draw separate density curves (using kdeplot()) #
on the marginal axes:

sns.jointplot(data=penguins,  x="bill_length_mm",

                  y="bill_depth_mm", hue="species")


plt.savefig("test7.png")
```



Pair Plots

- When you generalize joint plots to datasets of larger dimensions, you end up with **pair plots**
- This is very useful for exploring correlations between multidimensional data, when you'd like to plot all pairs of values against each other
- We’ll use the Iris dataset, which lists measurements of petals and sepals of three iris species

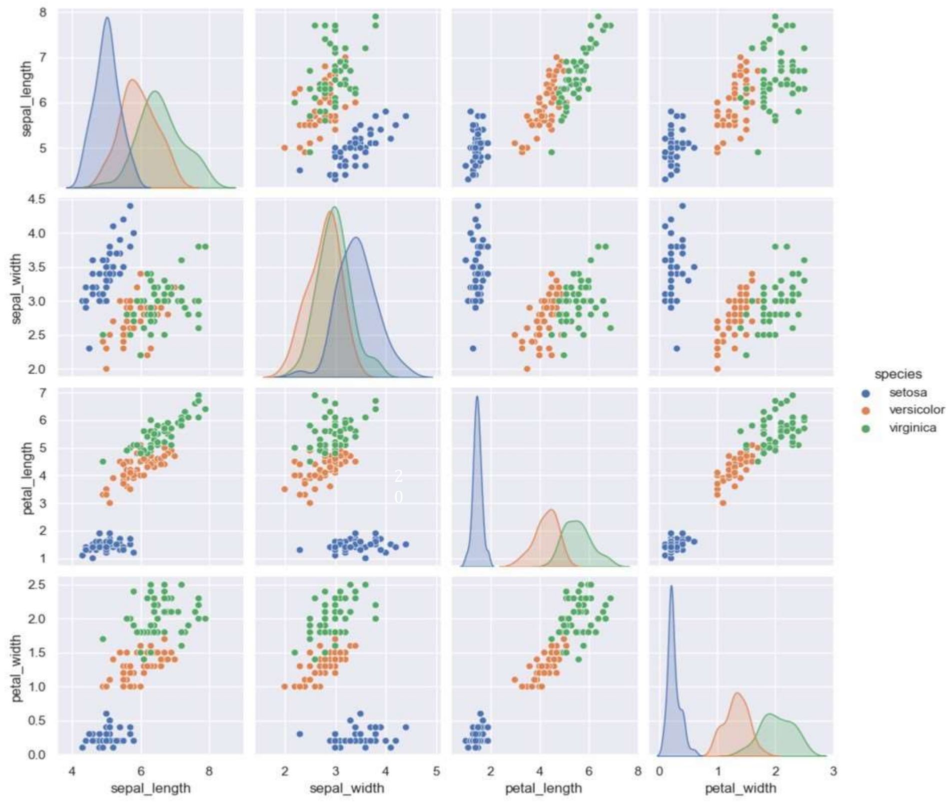
```
>>> import seaborn as sns
>>> iris = sns.load_dataset("iris")
>>> iris
       sepal_length   sepal_width   petal_length   petal_width     species
 0              5.1           3.5            1.4           0.2      setosa
 1              4.9           3.0            1.4           0.2      setosa
 2              4.7           3.2            1.3           0.2      setosa
 3              4.6           3.1            1.5           0.2      setosa
 4              5.0           3.6            1.4           0.2      setosa
 ..             ...           ...            ...           ...         ...
 145            6.7           3.0            5.2           2.3   virginica
 146            6.3           2.5            5.0           1.9   virginica
 147            6.5           3.0            5.2           2.0   virginica
 148            6.2           3.4            5.4           2.3   virginica
 149            5.9           3.0            5.1           1.8   virginica
```

Python Data Science Handbook

Draw a plot of two variables with bivariate and univariate graphs

- Visualizing the multidimensional relationships among the samples is as easy as calling `pairplot`:

```
import seaborn as sns

import matplotlib.pyplot as plt

sns.set()          # We can set the style by calling Seaborn's set() method

iris = sns.load_dataset("iris")


 sns.pairplot(iris, hue='species', height=2.5)

plt.savefig("test9.png")
```



Plotting Subplots

- You can use the following basic syntax to create subplots in the seaborn data visualization library in Python
- `pyplot.subplots` creates a figure and a grid of subplots with a single call, while providing reasonable control over how the individual plots are created
- One row, or one column, and you get back a list of Axes. Here one row

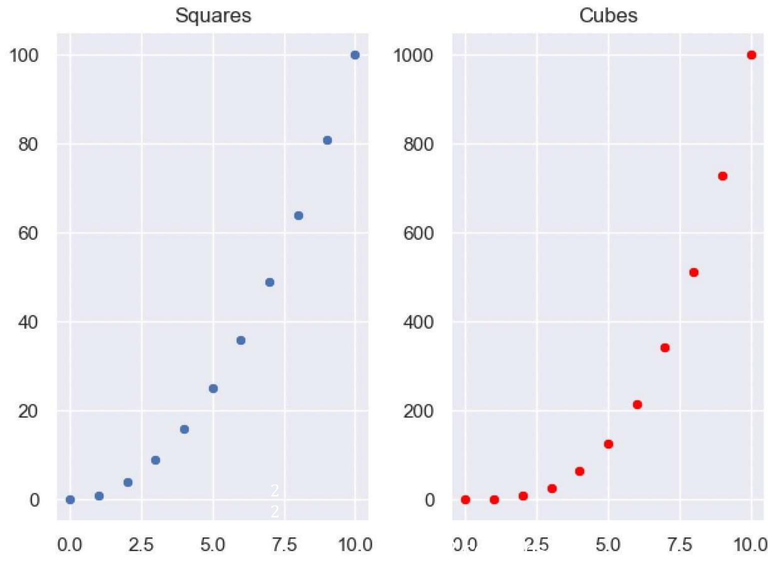
```
fig, ax = plt.subplots(1, 2). # row x column

# create chart in each subplot
sns.boxplot(data=df, x='team', y='points', ax=ax[0])
sns.boxplot(data=df, x='team', y='assists', ax=ax[1])
```

- two rows, you get back a 2-d dimensional list of Axes

```
fig, ax = plt.subplots(2, 2). # row x column

# create boxplot in each subplot
sns.boxplot(data=df, x='team', y='points', ax=axes[0, 0]). # same as ax=axes[0][0]
sns.boxplot(data=df, x='team', y='assists', ax=axes[0, 1])
sns.boxplot(data=df, x='team', y='rebounds', ax=axes[1, 0])
sns.boxplot(data=df, x='team', y='blocks', ax=axes[1, 1])
```

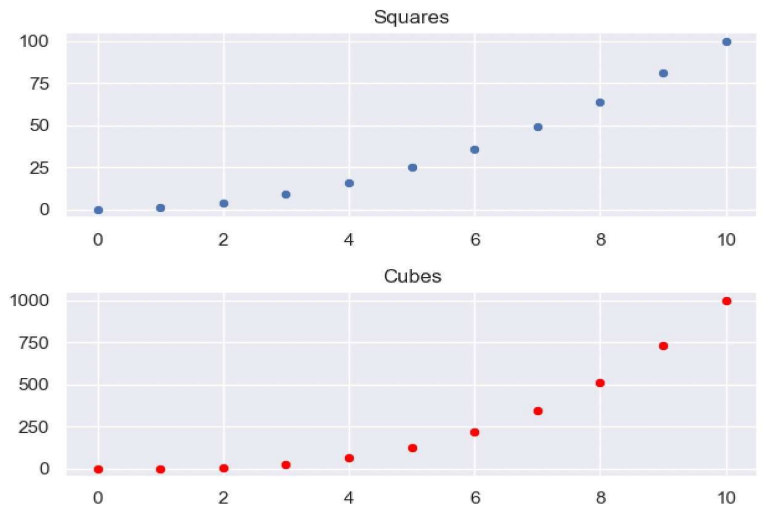
Plotting Subplots

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()   # We can set the style by calling Seaborn's set() method

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]

# number row x columns
# here ax is a list of Axes
fig, ax = plt.subplots(1, 2)

sns.scatterplot(x=x_values, y=squares, ax=ax[0])
ax[0].set_title('Squares')
sns.scatterplot(x=x_values, y=cubes, ax=ax[1], color='red')
ax[1].set_title('Cubes')

# automatically adjusts subplot params so that the subplot(s) fits in to the figure area.
fig.tight_layout()
#  save
plt.savefig("test10.png")
```



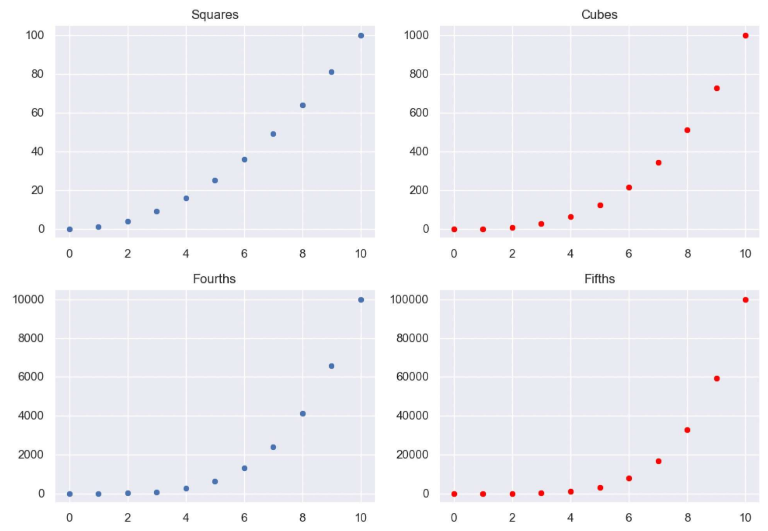
Plotting Subplots

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()   # We can set the style by calling Seaborn's set() method

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]

# number row x columns
# here ax is a list of Axes
fig, ax = plt.subplots(2, 1)

sns.scatterplot(x=x_values, y=squares, ax=ax[0])
ax[0].set_title('Squares')
sns.scatterplot(x=x_values, y=cubes, ax=ax[1], color='red')
ax[1].set_title('Cubes')

# automatically adjusts subplot params so that the subplot(s) fits in to the figure area.
fig.tight_layout()
#  save
plt.savefig("test10.png")
```



Plotting Subplots

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()   # We can set the style by calling Seaborn's set() method

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]
fourths = [x**4 for x in x_values]
fifths = [x**5 for x in x_values]

# number row x columns, and update the size of the figure
# here ax is a two-dimensional list of Axes
fig, ax = plt.subplots(2, 2, figsize=(10, 7))

sns.scatterplot(x=x_values, y=squares, ax=ax[0, 0])
ax[0, 0].set_title('Squares')
sns.scatterplot(x=x_values, y=cubes, ax=ax[0, 1], color='red')
ax[0, 1].set_title('Cubes')
sns.scatterplot(x=x_values, y=fourths, ax=ax[1, 0])
ax[1, 0].set_title('Fourths')
sns.scatterplot(x=x_values, y=fifths, ax=ax[1, 1], color='red')
ax[1, 1].set_title('Fifths')

# automatically adjusts subplot params so that the subplot(s) fits in to the figure area.
fig.tight_layout()
#  save
fig.savefig("test11.png")
```


Squares
Cubes
Fourths
Fifths