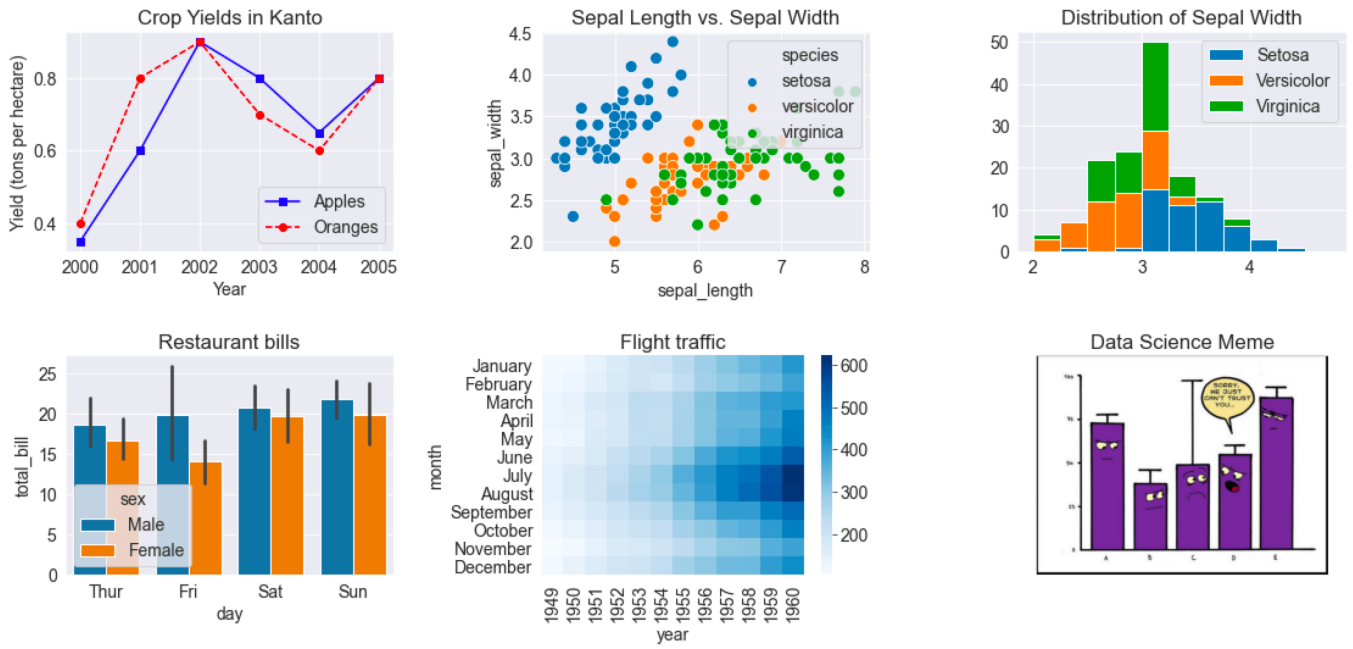
Data Visualization using Python, Matplotlib and Seaborn



This tutorial series is a beginner-friendly introduction to programming and data analysis using the Python programming language. These tutorials take a practical and coding-focused approach. The best way to learn the material is to execute the code and experiment with it yourself.

This tutorial covers the following topics:

- Creating and customizing line charts using Matplotlib
- Visualizing relationships between two or more variables using scatter plots
- Studying distributions of variables using histograms & bar charts to
- Visualizing two-dimensional data using heatmaps
- Displaying images using Matplotlib's `plt.imshow`
- Plotting multiple Matplotlib and Seaborn charts in a grid

How to run the code

This tutorial is an executable Jupyter notebook hosted on Jovian. You can *run* this tutorial and experiment with the code examples in a couple of ways: *using free online resources* (recommended) or *on your computer*.

Option 1: Running using free online resources (1-click, recommended)

The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Binder**. You can also select "Run on Colab" or "Run on Kaggle", but you'll need to create an account on Google Colab or Kaggle to use these platforms.

Option 2: Running on your computer locally

To run the code on your computer locally, you'll need to set up Python, download the notebook and install the required libraries. We recommend using the Conda distribution of Python. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

> **Jupyter Notebooks**: This tutorial is a Jupyter notebook - a document made of *cells*. Each cell can contain code written in Python or explanations in plain English. You can execute code cells and view the results, e.g., numbers, messages, graphs, tables, files, etc., instantly within the notebook. Jupyter is a powerful platform for experimentation and analysis. Don't be afraid to mess around with the code & break things - you'll learn a lot by encountering and fixing errors. You can use the "Kernel > Restart & Clear Output" menu option to clear all outputs and start again from the top.

Introduction

Data visualization is the graphic representation of data. It involves producing images that communicate relationships among the represented data to viewers. Visualizing data is an essential part of data analysis and machine learning. We'll use Python libraries Matplotlib and Seaborn to learn and apply some popular data visualization techniques. We'll use the words *chart*, *plot*, and *graph* interchangeably in this tutorial.

To begin, let's install and import the libraries. We'll use the `matplotlib.pyplot` module for basic plots like line & bar charts. It is often imported with the alias `plt`. We'll use the `seaborn` module for more advanced plots. It is commonly imported with the alias `sns`.

```
!pip install matplotlib seaborn --upgrade --quiet
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Notice this we also include the special command `%matplotlib inline` to ensure that our plots are shown and embedded within the Jupyter notebook itself. Without this command, sometimes plots may show up in pop-up windows.

Line Chart

The line chart is one of the simplest and most widely used data visualization techniques. A line chart displays information as a series of data points or markers connected by straight lines. You can customize the shape, size, color, and other aesthetic elements of the lines and markers for better visual clarity.

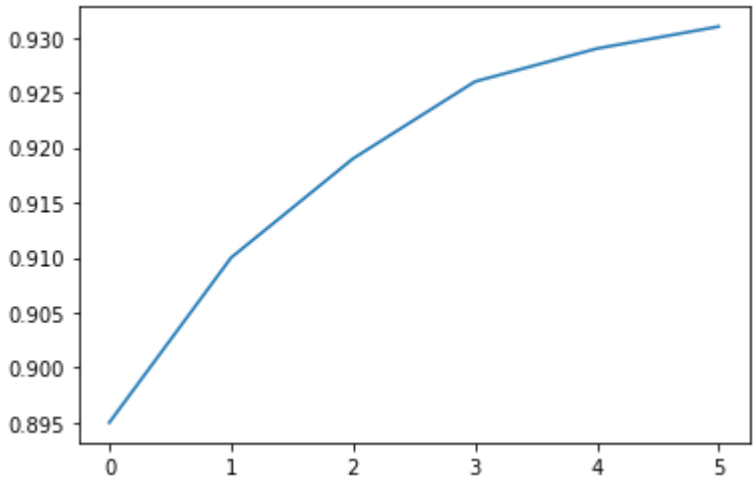
Here's a Python list showing the yield of apples (tons per hectare) over six years in an imaginary country called Kanto.

```
yield_apples = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931]
```

We can visualize how the yield of apples changes over time using a line chart. To draw a line chart, we can use the `plt.plot` function.

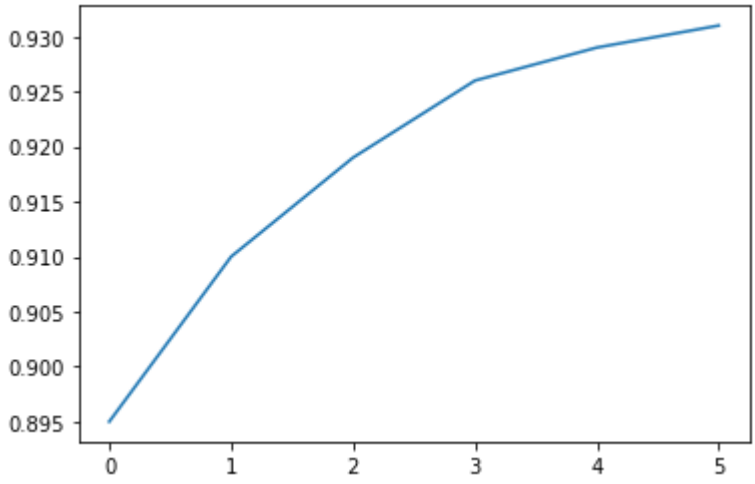
```
plt.plot(yield_apples)
```

```
[<matplotlib.lines.Line2D at 0x7f90047a8520>]
```



Calling the `plt.plot` function draws the line chart as expected. It also returns a list of plots drawn `[<matplotlib.lines.Line2D at 0x7ff70aa20760>]`, shown within the output. We can include a semicolon (`;`) at the end of the last statement in the cell to avoiding showing the output and display just the graph.

```
plt.plot(yield_apples);
```



Let's enhance this plot step-by-step to make it more informative and beautiful.

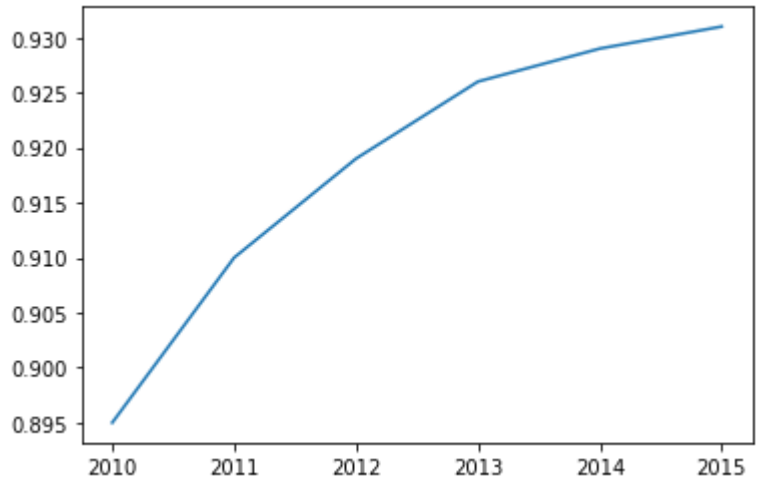
Customizing the X-axis

The X-axis of the plot currently shows list element indexes 0 to 5. The plot would be more informative if we could display the year for which we're plotting the data. We can do this by two arguments `plt.plot`.

```
years = [2010, 2011, 2012, 2013, 2014, 2015]
yield_apples = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931]
```

```
plt.plot(years, yield_apples)
```

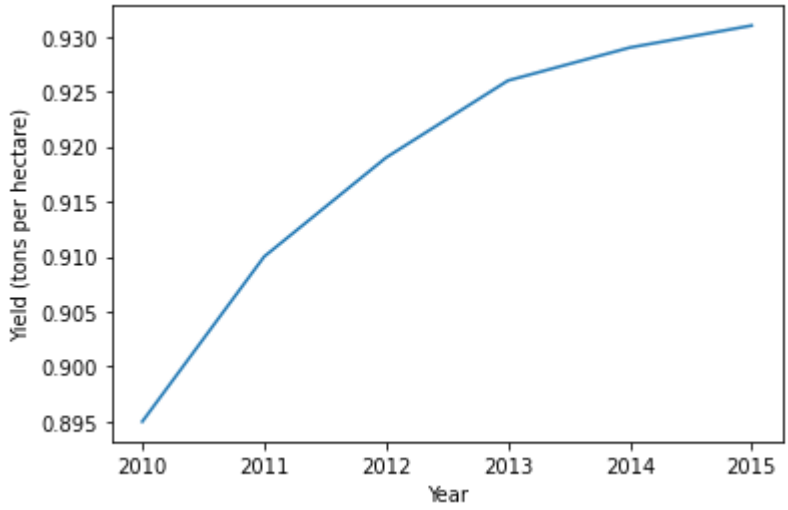
```
[<matplotlib.lines.Line2D at 0x7f90051fcfa0>]
```



Axis Labels

We can add labels to the axes to show what each axis represents using the `plt.xlabel` and `plt.ylabel` methods.

```
plt.plot(years, yield_apples)
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)');
```



Plotting Multiple Lines

You can invoke the `plt.plot` function once for each line to plot multiple lines in the same graph. Let's compare the yields of apples vs. oranges in Kanto.

```
years = range(2000, 2012)
apples = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931, 0.934, 0.936, 0.937, 0.9375, 0.9372,
oranges = [0.962, 0.941, 0.930, 0.923, 0.918, 0.908, 0.907, 0.904, 0.901, 0.898, 0.9, 0
```

```
plt.plot(years, apples)
plt.plot(years, oranges)
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)');
```

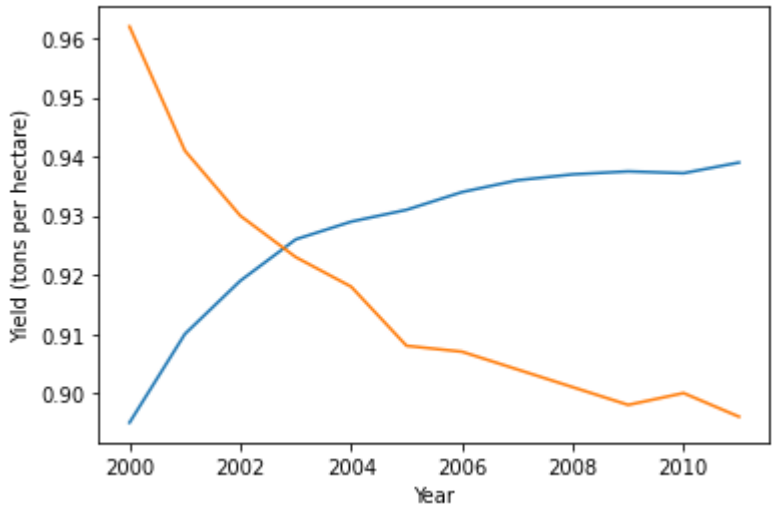
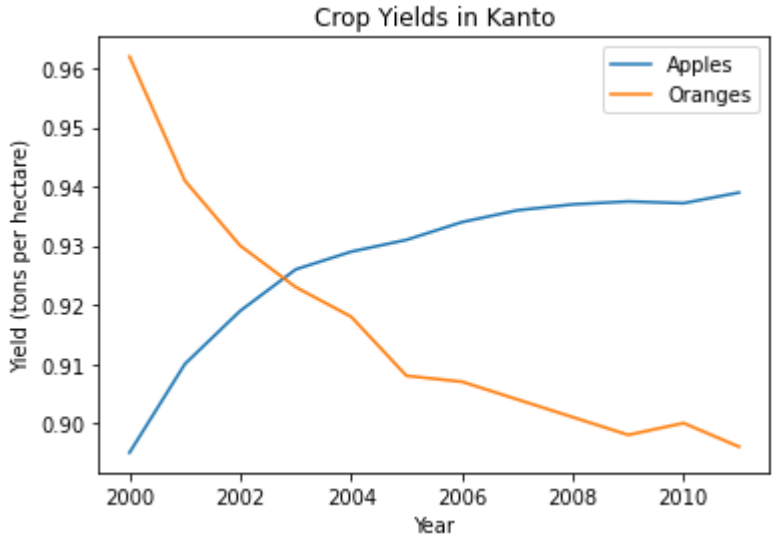


Chart Title and Legend

To differentiate between multiple lines, we can include a legend within the graph using the `plt.legend` function. We can also set a title for the chart using the `plt.title` function.

```
plt.plot(years, apples)
plt.plot(years, oranges)

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges']);
```

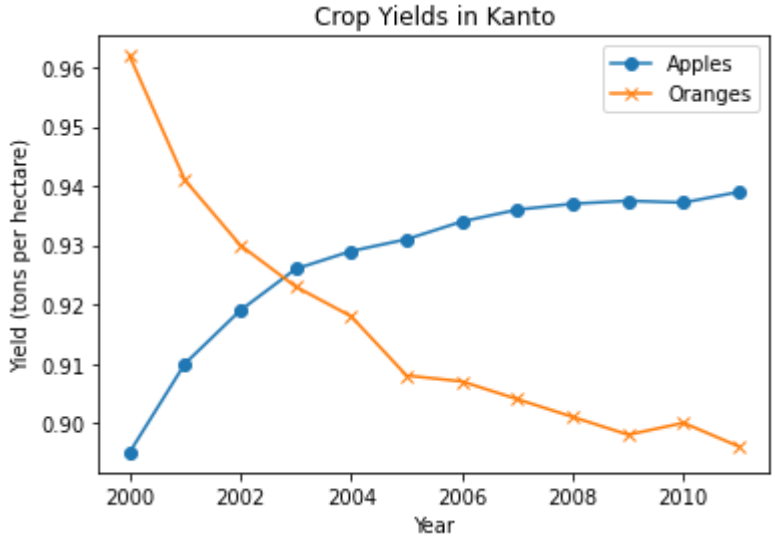


Line Markers

We can also show markers for the data points on each line using the `marker` argument of `plt.plot` . Matplotlib provides many different markers, like a circle, cross, square, diamond, etc. You can find the full list of marker types here: https://matplotlib.org/3.1.1/api/markers_api.html .

```
plt.plot(years, apples, marker='o')
plt.plot(years, oranges, marker='x')
```

```
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges']);
```



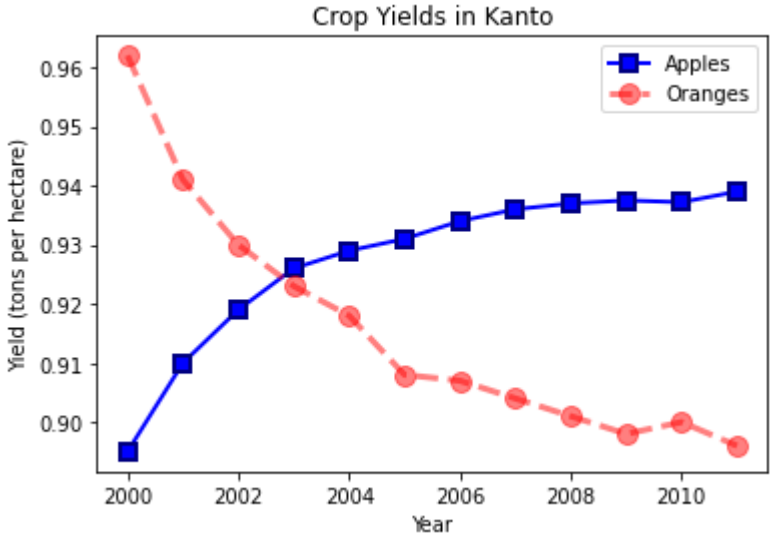
Styling Lines and Markers

The `plt.plot` function supports many arguments for styling lines and markers:

- `color` or `c`: Set the color of the line (supported colors)
- `linestyle` or `ls`: Choose between a solid or dashed line
- `linewidth` or `lw`: Set the width of a line
- `markersize` or `ms`: Set the size of markers
- `markeredgecolor` or `mec`: Set the edge color for markers
- `markeredgewidth` or `mew`: Set the edge width for markers
- `markerfacecolor` or `mfc`: Set the fill color for markers
- `alpha`: Opacity of the plot

Check out the documentation for `plt.plot` to learn more: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot .

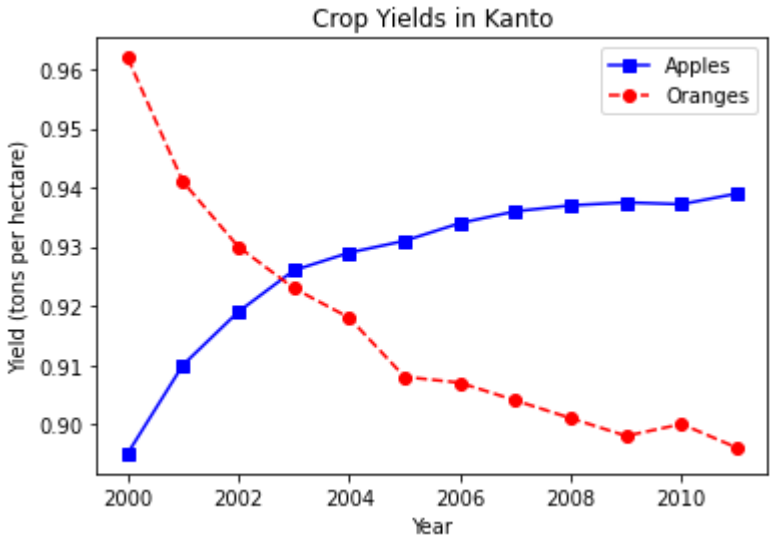
```
plt.plot(years, apples, marker='s', c='b', ls='-', lw=2, ms=8, mew=2, mec='navy')
plt.plot(years, oranges, marker='o', c='r', ls='--', lw=3, ms=10, alpha=.5)

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges']);
```



The `fmt` argument provides a shorthand for specifying the marker shape, line style, and line color. It can be provided as the third argument to `plt.plot`.

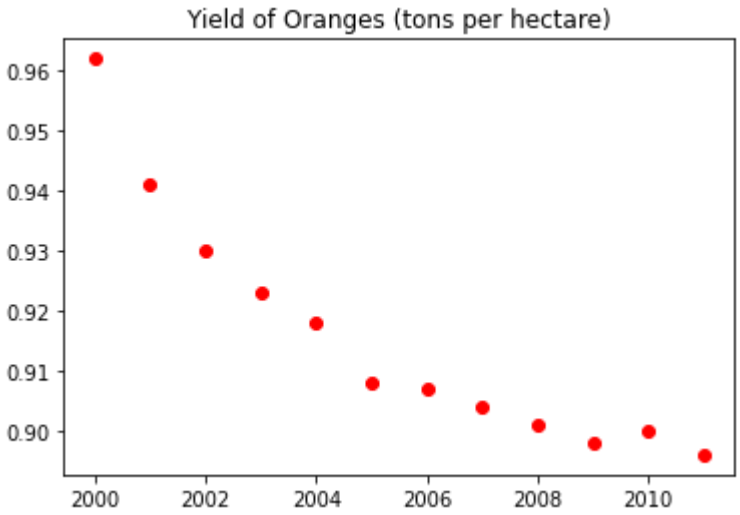
```
fmt = '[marker][line][color]'
```

```
plt.plot(years, apples, 's-b')
plt.plot(years, oranges, 'o--r')

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges']);
```



If you don't specify a line style in `fmt`, only markers are drawn.

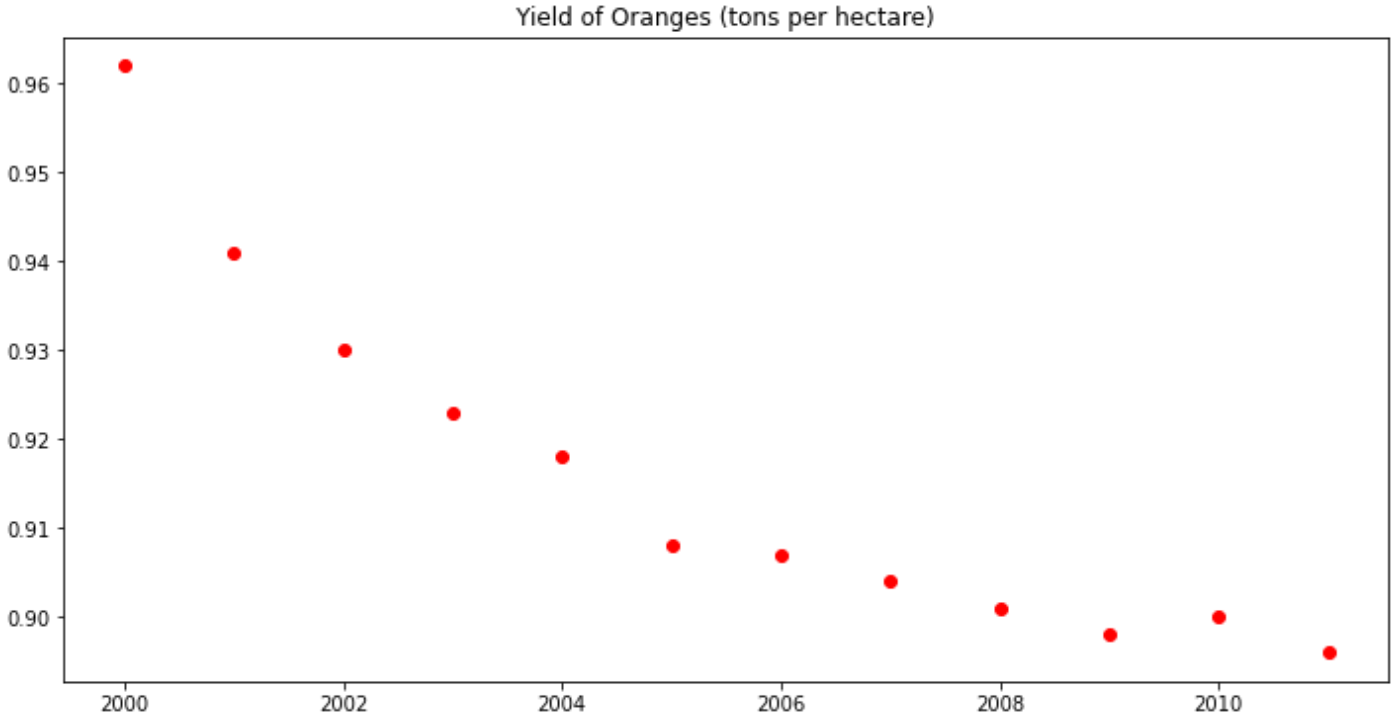
```
plt.plot(years, oranges, 'or')
plt.title("Yield of Oranges (tons per hectare)");
```



Changing the Figure Size

You can use the `plt.figure` function to change the size of the figure.

```
plt.figure(figsize=(12, 6))

plt.plot(years, oranges, 'or')
plt.title("Yield of Oranges (tons per hectare)");
```

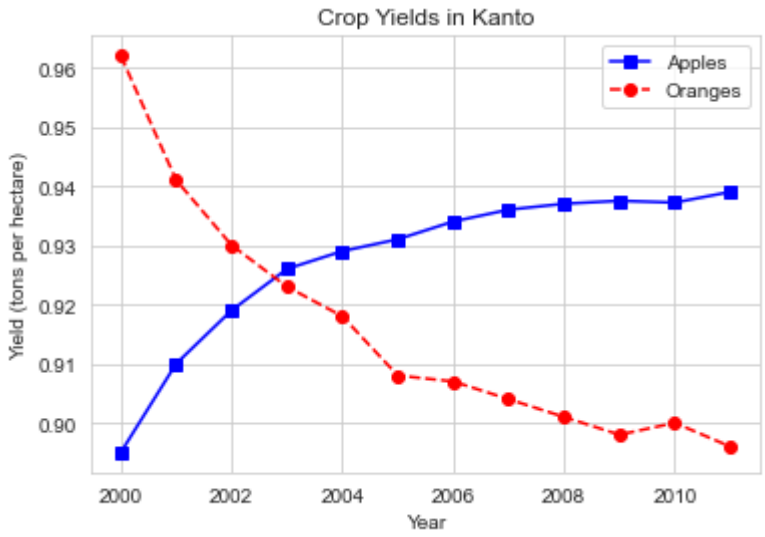


Improving Default Styles using Seaborn

An easy way to make your charts look beautiful is to use some default styles from the Seaborn library. These can be applied globally using the `sns.set_style` function. You can see a full list of predefined styles here: https://seaborn.pydata.org/generated/seaborn.set_style.html .

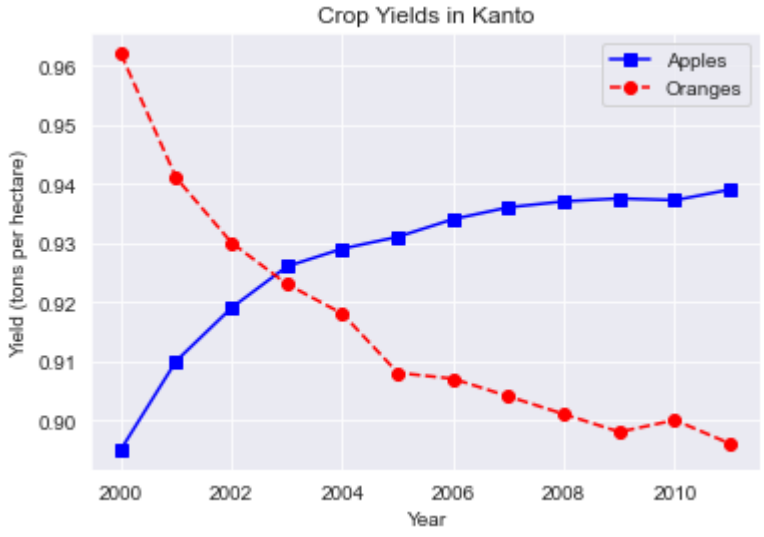
```
sns.set_style("whitegrid")
```

```
plt.plot(years, apples, 's-b')
plt.plot(years, oranges, 'o--r')

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges']);
```

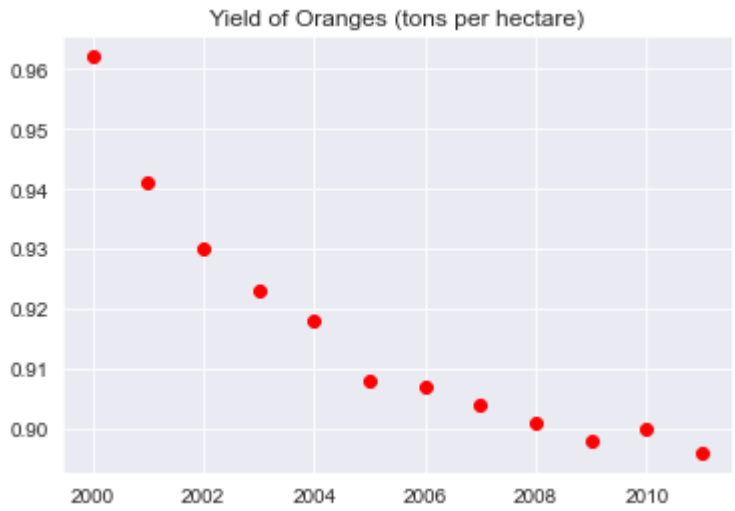



```
sns.set_style("darkgrid")
```

```
plt.plot(years, apples, 's-b')
plt.plot(years, oranges, 'o--r')

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges']);
```




```
plt.plot(years, oranges, 'or')
plt.title("Yield of Oranges (tons per hectare)");
```



You can also edit default styles directly by modifying the `matplotlib.rcParams` dictionary. Learn more: https://matplotlib.org/3.2.1/tutorials/introductory/customizing.html#matplotlib-rcparams .

```
import matplotlib
```

```
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (9, 5)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

Save and upload your notebook

Whether you're running this Jupyter notebook online or on your computer, it's essential to save your work from time to time. You can continue working on a saved notebook later or share it with friends and colleagues to let them execute your code. Jovian offers an easy way of saving and sharing your Jupyter notebooks online.

```
# Install the library
!pip install jovian --upgrade --quiet
```

```
import jovian
```

```
jovian.commit(project='python-matplotlib-data-visualization')
```

```
[jovian] Attempting to save notebook..
[jovian] Updating notebook "aakashns/python-matplotlib-data-visualization" on https://jovian.ai/
[jovian] Uploading notebook..
[jovian] Capturing environment..
[jovian] Committed successfully! https://jovian.ai/aakashns/python-matplotlib-data-visualization
```

```
'https://jovian.ai/aakashns/python-matplotlib-data-visualization'
```

The first time you run `jovian.commit`, you'll be asked to provide an API Key to securely upload the notebook to your Jovian account. You can get the API key from your Jovian profile page after logging in / signing up.

`jovian.commit` uploads the notebook to your Jovian account, captures the Python environment, and creates a shareable link for your notebook, as shown above. You can use this link to share your work and let anyone (including you) run your notebooks and reproduce your work.

Scatter Plot

In a scatter plot, the values of 2 variables are plotted as points on a 2-dimensional grid. Additionally, you can also use a third variable to determine the size or color of the points. Let's try out an example.

The Iris flower dataset provides sample measurements of sepals and petals for three species of flowers. The Iris dataset is included with the Seaborn library and can be loaded as a Pandas data frame.

```
# Load data into a Pandas dataframe
flowers_df = sns.load_dataset("iris")
```

```
flowers_df
```

	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

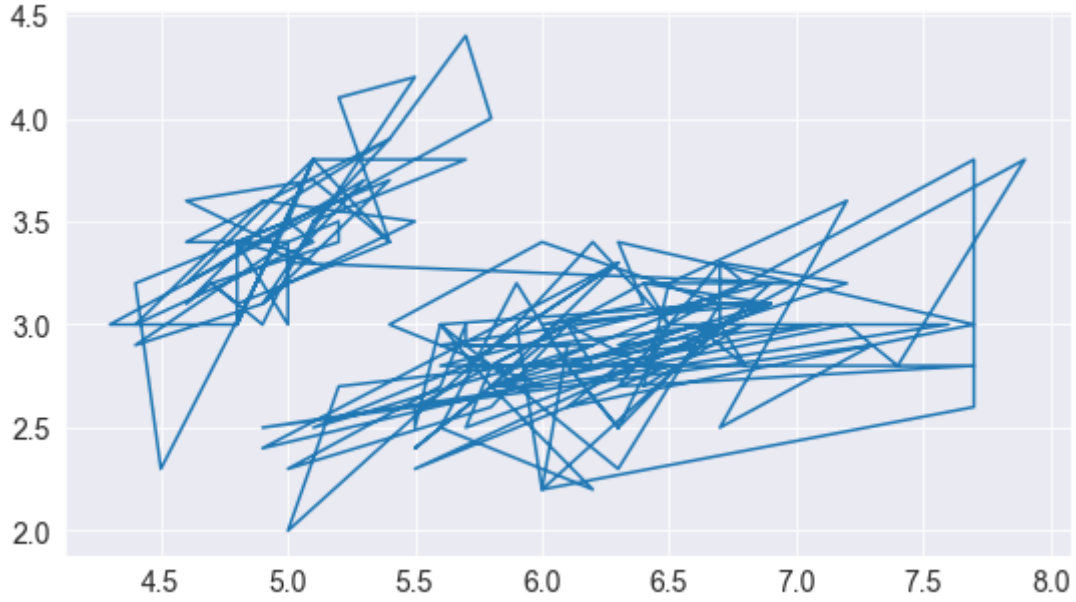
150 rows × 5 columns

```
flowers_df.species.unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

Let's try to visualize the relationship between sepal length and sepal width. Our first instinct might be to create a line chart using `plt.plot` .

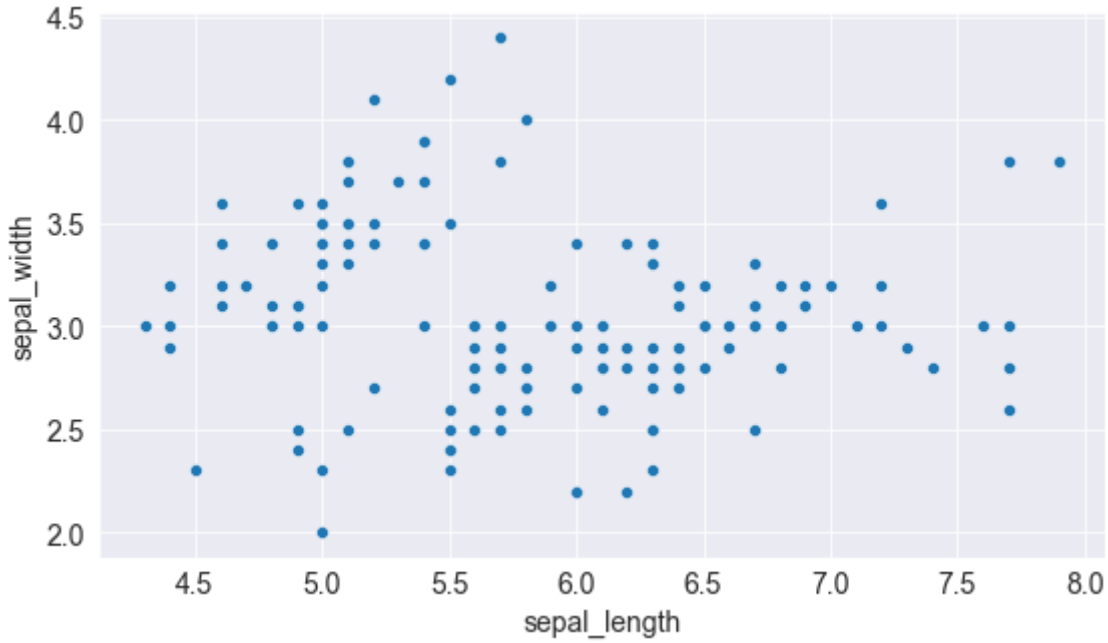
```
plt.plot(flowers_df.sepal_length, flowers_df.sepal_width);
```



The output is not very informative as there are too many combinations of the two properties within the dataset. There doesn't seem to be simple relationship between them.

We can use a scatter plot to visualize how sepal length & sepal width vary using the `scatterplot` function from the `seaborn` module (imported as `sns`).

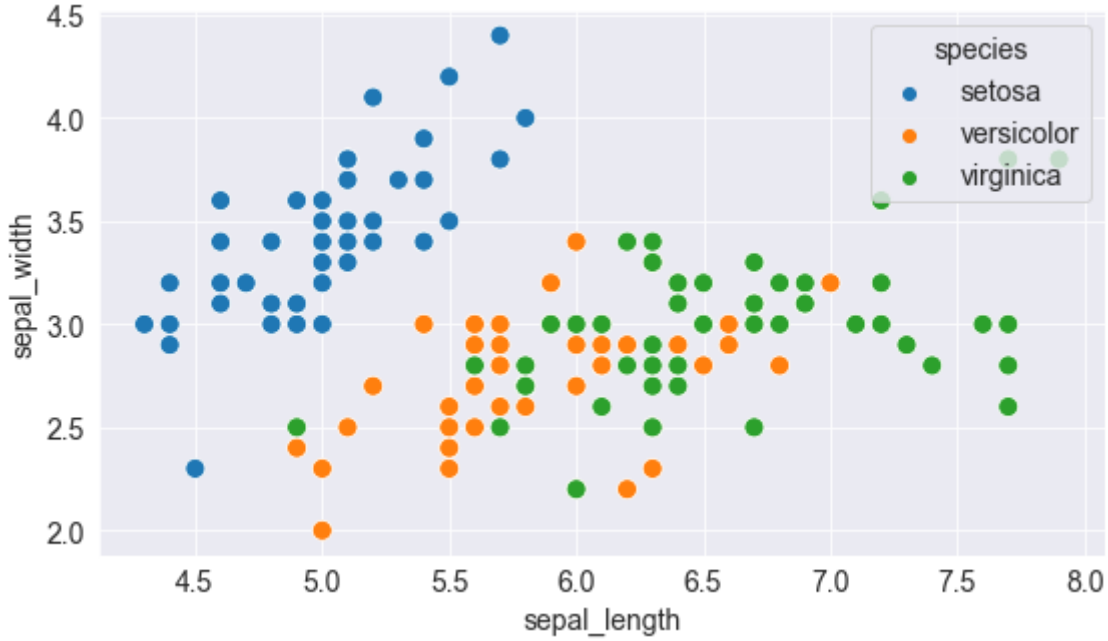
```
sns.scatterplot(x=flowers_df.sepal_length, y=flowers_df.sepal_width);
```



Adding Hues

Notice how the points in the above plot seem to form distinct clusters with some outliers. We can color the dots using the flower species as a `hue`. We can also make the points larger using the `s` argument.

```
sns.scatterplot(x=flowers_df.sepal_length, y=flowers_df.sepal_width, hue=flowers_df.spe
```

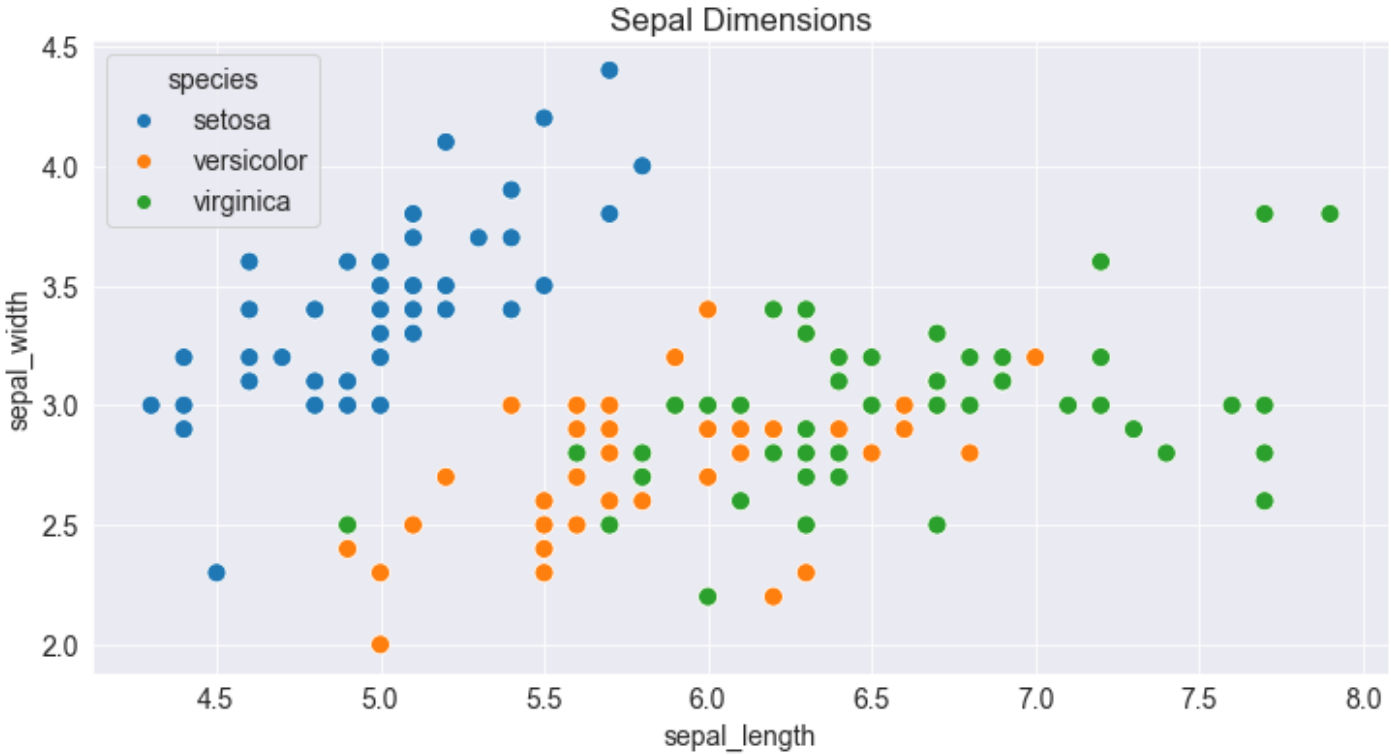


Adding hues makes the plot more informative. We can immediately tell that Setosa flowers have a smaller sepal length but higher sepal widths. In contrast, the opposite is true for Virginica flowers.

Customizing Seaborn Figures

Since Seaborn uses Matplotlib's plotting functions internally, we can use functions like `plt.figure` and `plt.title` to modify the figure.

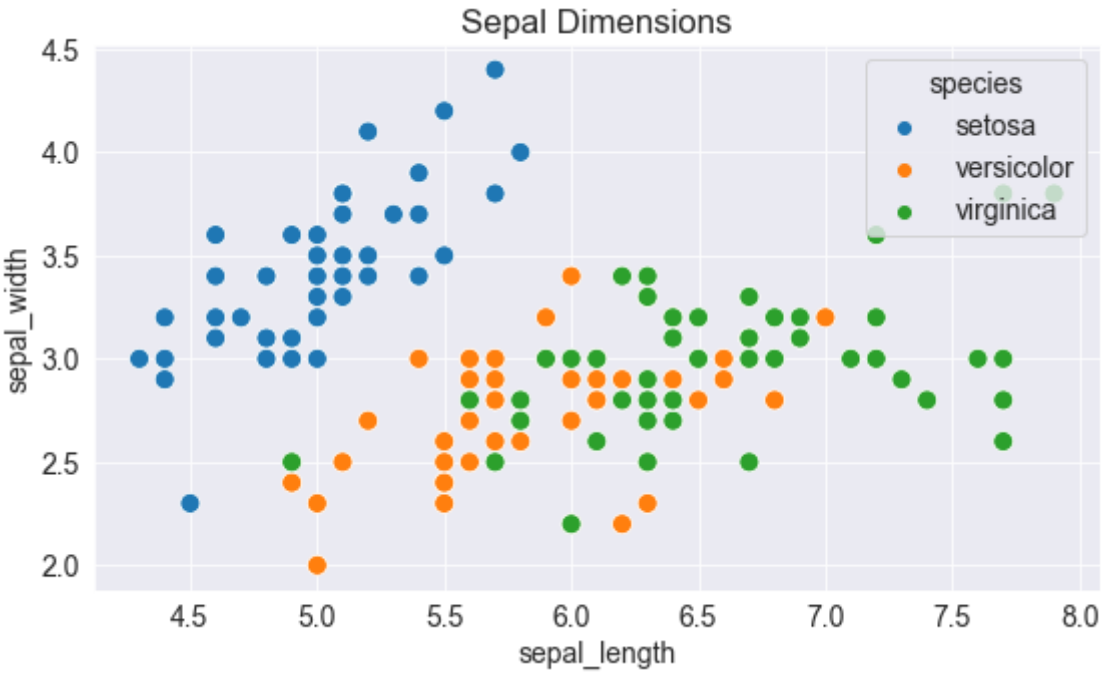
```
plt.figure(figsize=(12, 6))
plt.title('Sepal Dimensions')

sns.scatterplot(x=flowers_df.sepal_length,
                y=flowers_df.sepal_width,
                hue=flowers_df.species,
                s=100);
```



Plotting using Pandas Data Frames

Seaborn has in-built support for Pandas data frames. Instead of passing each column as a series, you can provide column names and use the `data` argument to specify a data frame.

```
plt.title('Sepal Dimensions')
sns.scatterplot(x='sepal_length',
                y='sepal_width',
                hue='species',
                s=100,
                data=flowers_df);
```



Let's save and upload our work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Attempting to save notebook..
[jovian] Updating notebook "aakashns/python-matplotlib-data-visualization" on https://jovian.ai/
[jovian] Uploading notebook..
[jovian] Capturing environment..
[jovian] Committed successfully! https://jovian.ai/aakashns/python-matplotlib-data-visualization
```

```
'https://jovian.ai/aakashns/python-matplotlib-data-visualization'
```

Histogram

A histogram represents the distribution of a variable by creating bins (interval) along the range of values and showing vertical bars to indicate the number of observations in each bin.

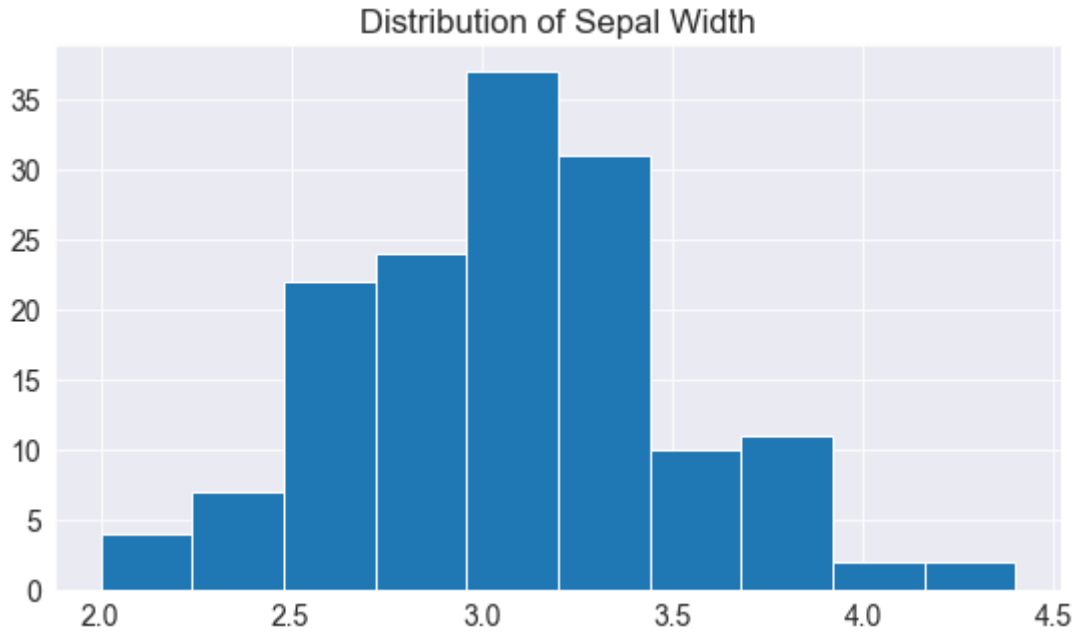
For example, let's visualize the distribution of values of sepal width in the flowers dataset. We can use the `plt.hist` function to create a histogram.

```
# Load data into a Pandas dataframe
flowers_df = sns.load_dataset("iris")
```

```
flowers_df.sepal_width
```

```
0      3.5
1      3.0
2      3.2
3      3.1
4      3.6
      ...
145    3.0
146    2.5
147    3.0
148    3.4
149    3.0
Name: sepal_width, Length: 150, dtype: float64
```

```
plt.title("Distribution of Sepal Width")
plt.hist(flowers_df.sepal_width);
```

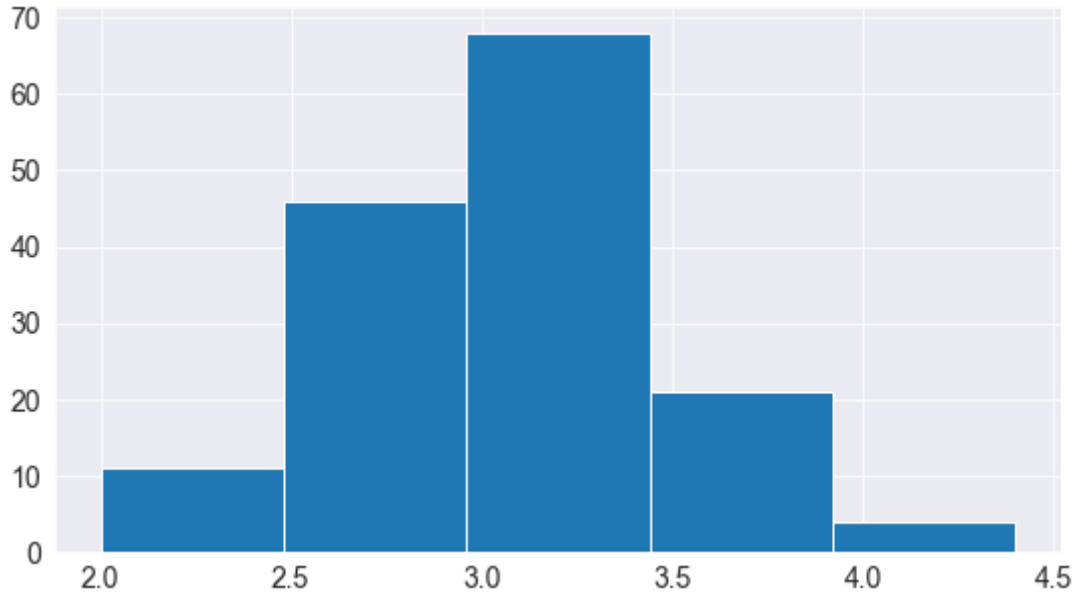


We can immediately see that the sepal widths lie in the range 2.0 - 4.5, and around 35 values are in the range 2.9 - 3.1, which seems to be the most populous bin.

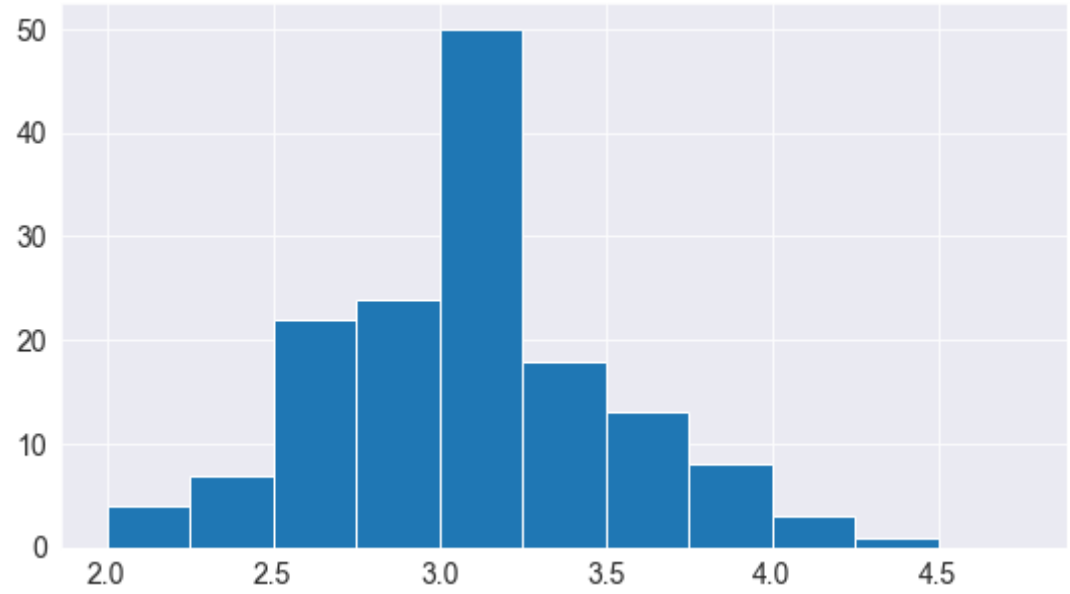
Controlling the size and number of bins

We can control the number of bins or the size of each one using the bins argument.

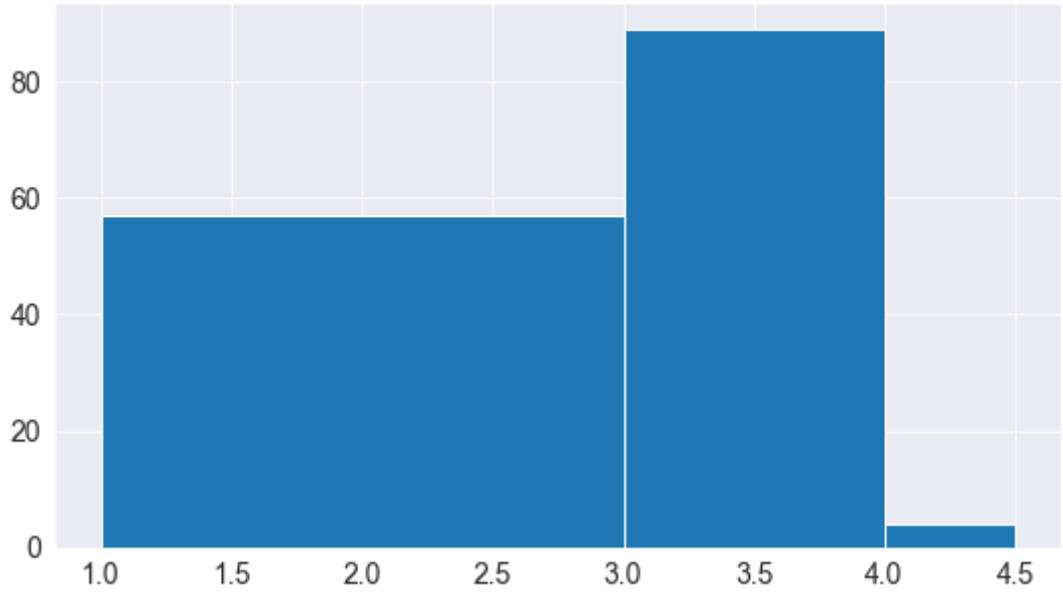
```
# Specifying the number of bins
plt.hist(flowers_df.sepal_width, bins=5);
```




```
import numpy as np

# Specifying the boundaries of each bin
plt.hist(flowers_df.sepal_width, bins=np.arange(2, 5, 0.25));
```




```
# Bins of unequal sizes
plt.hist(flowers_df.sepal_width, bins=[1, 3, 4, 4.5]);
```



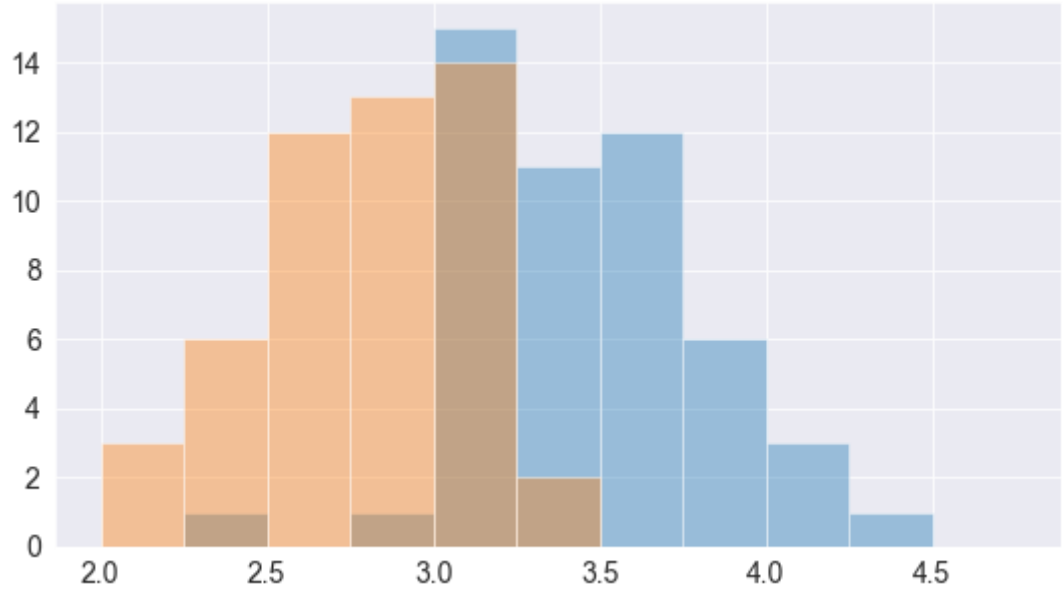
Multiple Histograms

Similar to line charts, we can draw multiple histograms in a single chart. We can reduce each histogram's opacity so that one histogram's bars don't hide the others'.

Let's draw separate histograms for each species of flowers.

```
setosa_df = flowers_df[flowers_df.species == 'setosa']
versicolor_df = flowers_df[flowers_df.species == 'versicolor']
virginica_df = flowers_df[flowers_df.species == 'virginica']
```

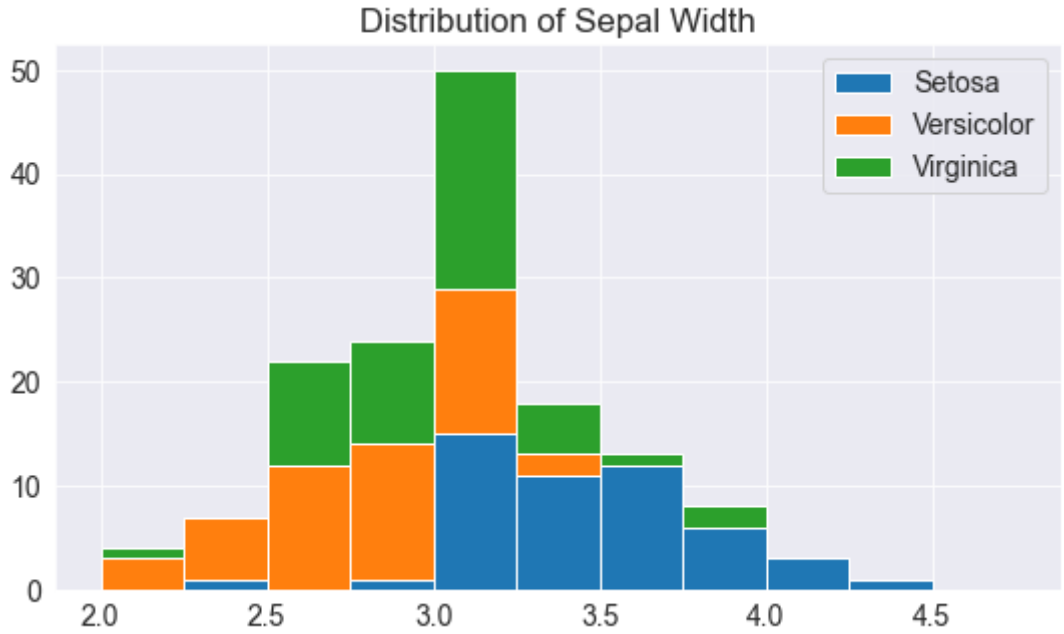
```
plt.hist(setosa_df.sepal_width, alpha=0.4, bins=np.arange(2, 5, 0.25));
plt.hist(versicolor_df.sepal_width, alpha=0.4, bins=np.arange(2, 5, 0.25));
```



We can also stack multiple histograms on top of one another.

```
plt.title('Distribution of Sepal Width')

plt.hist([setosa_df.sepal_width, versicolor_df.sepal_width, virginica_df.sepal_width],
```

```
        bins=np.arange(2, 5, 0.25),
        stacked=True);

plt.legend(['Setosa', 'Versicolor', 'Virginica']);
```



Let's save and commit our work before continuing

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Attempting to save notebook..
[jovian] Updating notebook "aakashns/python-matplotlib-data-visualization" on https://jovian.ai/
[jovian] Uploading notebook..
[jovian] Capturing environment..
[jovian] Committed successfully! https://jovian.ai/aakashns/python-matplotlib-data-visualization
```

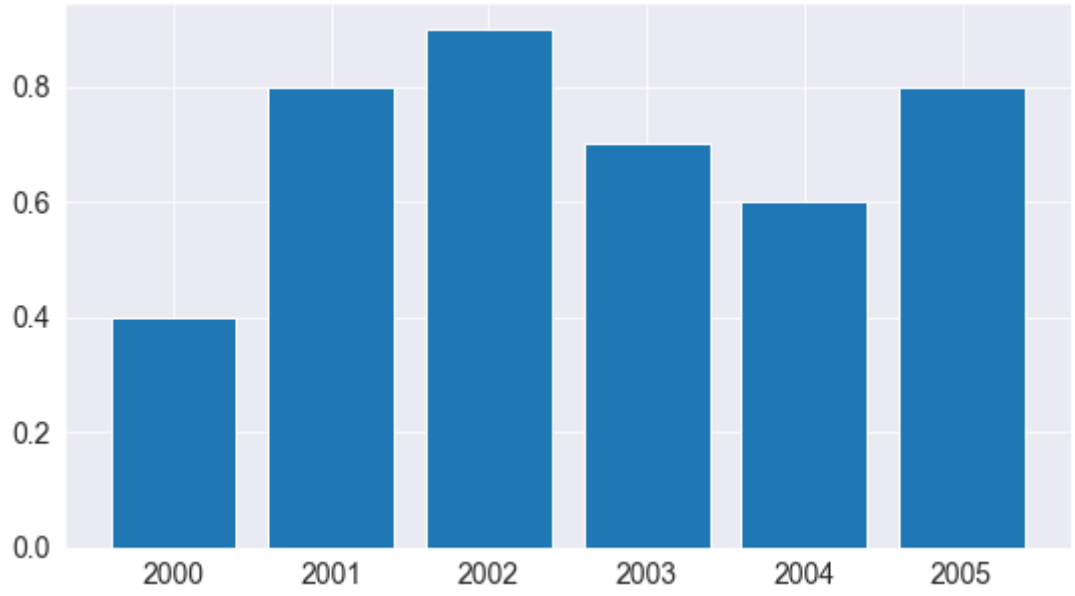
```
'https://jovian.ai/aakashns/python-matplotlib-data-visualization'
```

Bar Chart

Bar charts are quite similar to line charts, i.e., they show a sequence of values. However, a bar is shown for each value, rather than points connected by lines. We can use the `plt.bar` function to draw a bar chart.

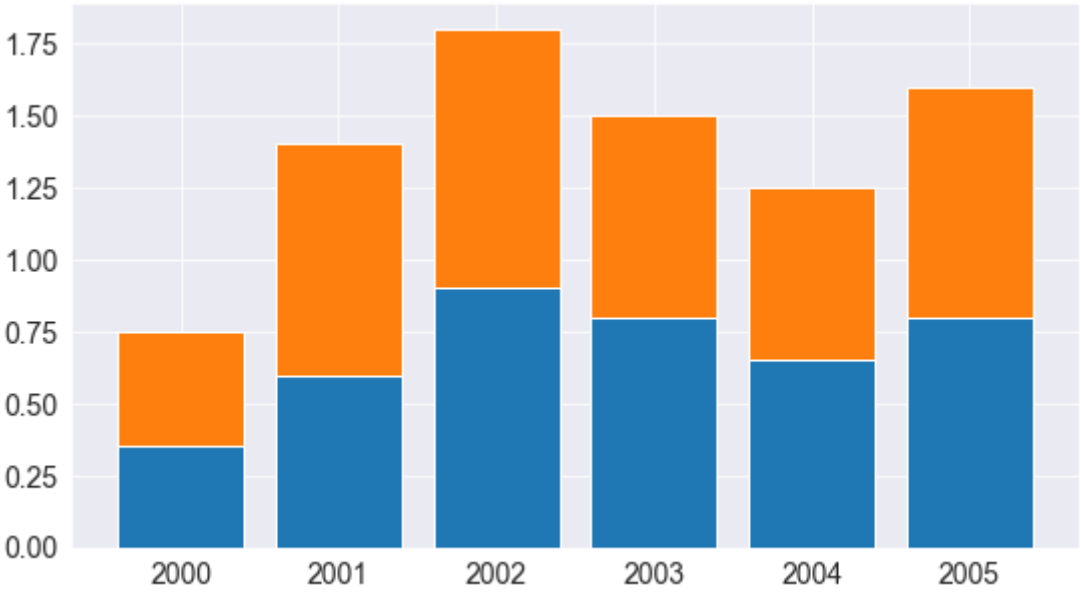
```
years = range(2000, 2006)
apples = [0.35, 0.6, 0.9, 0.8, 0.65, 0.8]
oranges = [0.4, 0.8, 0.9, 0.7, 0.6, 0.8]
```

```
plt.bar(years, oranges);
```



Like histograms, we can stack bars on top of one another. We use the `bottom` argument of `plt.bar` to achieve this.

```
plt.bar(years, apples)
plt.bar(years, oranges, bottom=apples);
```



Bar Plots with Averages

Let's look at another sample dataset included with Seaborn, called `tips` . The dataset contains information about the sex, time of day, total bill, and tip amount for customers visiting a restaurant over a week.

```
tips_df = sns.load_dataset("tips");
```

```
tips_df
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

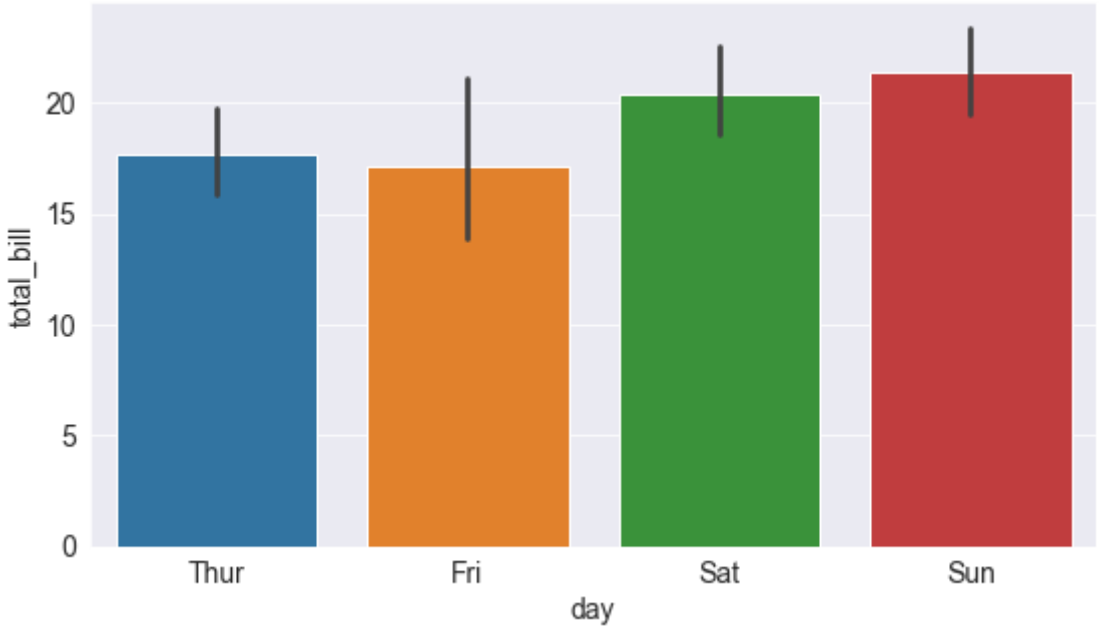
	total_bill	tip	sex	smoker	day	time	size
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

We might want to draw a bar chart to visualize how the average bill amount varies across different days of the week. One way to do this would be to compute the day-wise averages and then use `plt.bar` (try it as an exercise).

However, since this is a very common use case, the Seaborn library provides a `barplot` function which can automatically compute averages.

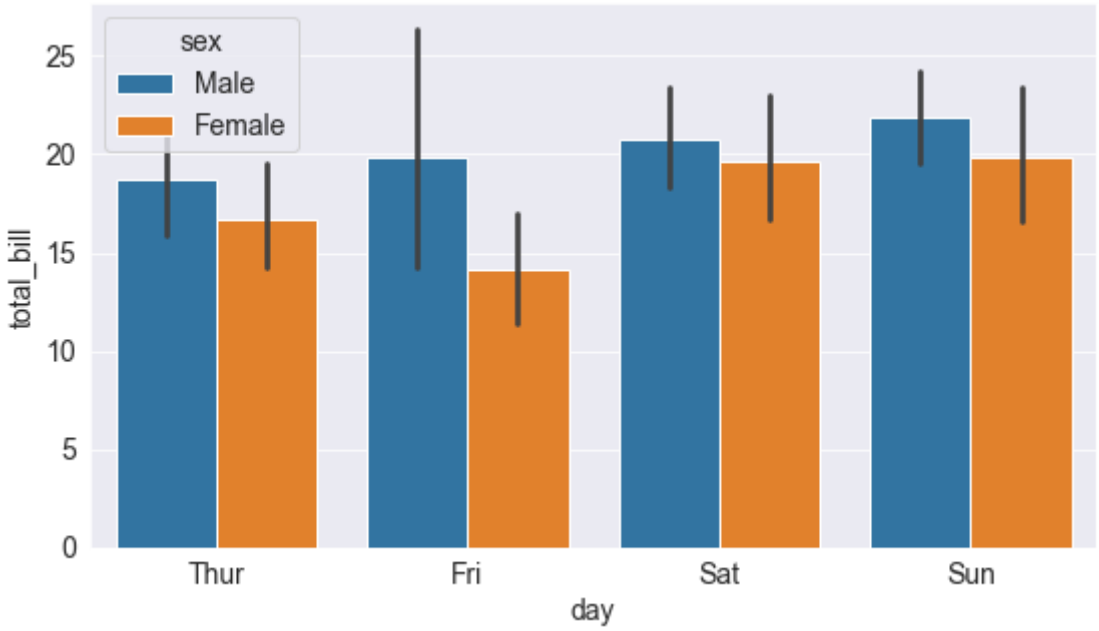
```
sns.barplot(x='day', y='total_bill', data=tips_df);
```



The lines cutting each bar represent the amount of variation in the values. For instance, it seems like the variation in the total bill is relatively high on Fridays and low on Saturday.

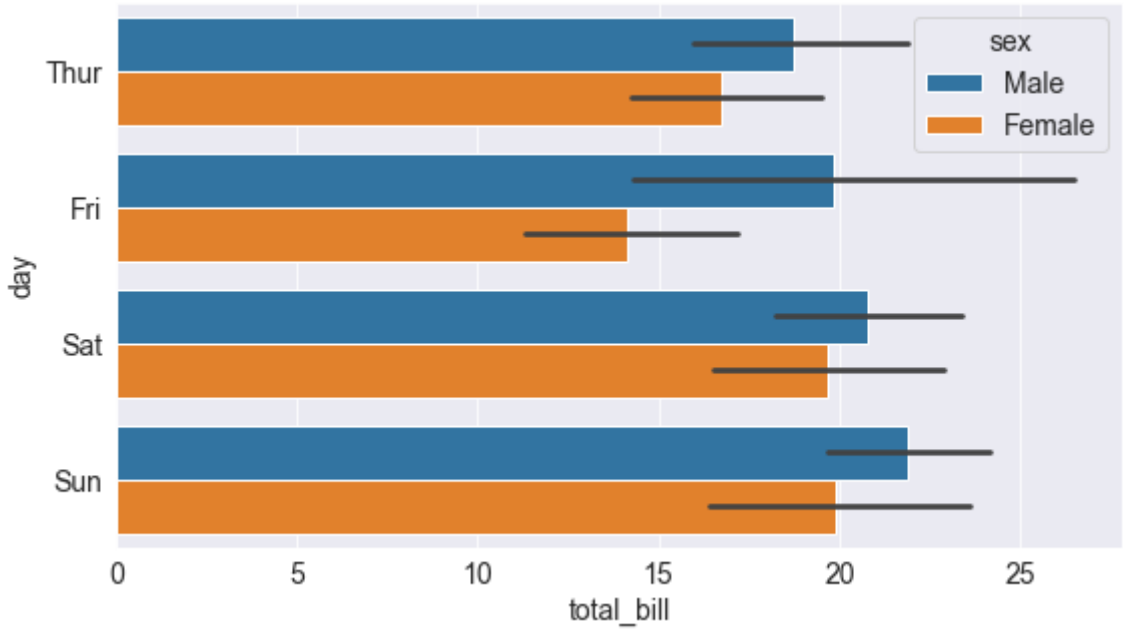
We can also specify a `hue` argument to compare bar plots side-by-side based on a third feature, e.g., sex.

```
sns.barplot(x='day', y='total_bill', hue='sex', data=tips_df);
```



You can make the bars horizontal simply by switching the axes.

```
sns.barplot(x='total_bill', y='day', hue='sex', data=tips_df);
```



Let's save and commit our work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Attempting to save notebook..
[jovian] Updating notebook "aakashns/python-matplotlib-data-visualization" on
https://jovian.ai/
[jovian] Uploading notebook..
[jovian] Capturing environment..
```

```
[jovian] Committed successfully! https://jovian.ai/aakashns/python-matplotlib-data-visualization
```

```
'https://jovian.ai/aakashns/python-matplotlib-data-visualization'
```

Heatmap

A heatmap is used to visualize 2-dimensional data like a matrix or a table using colors. The best way to understand it is by looking at an example. We'll use another sample dataset from Seaborn, called `flights`, to visualize monthly passenger footfall at an airport over 12 years.

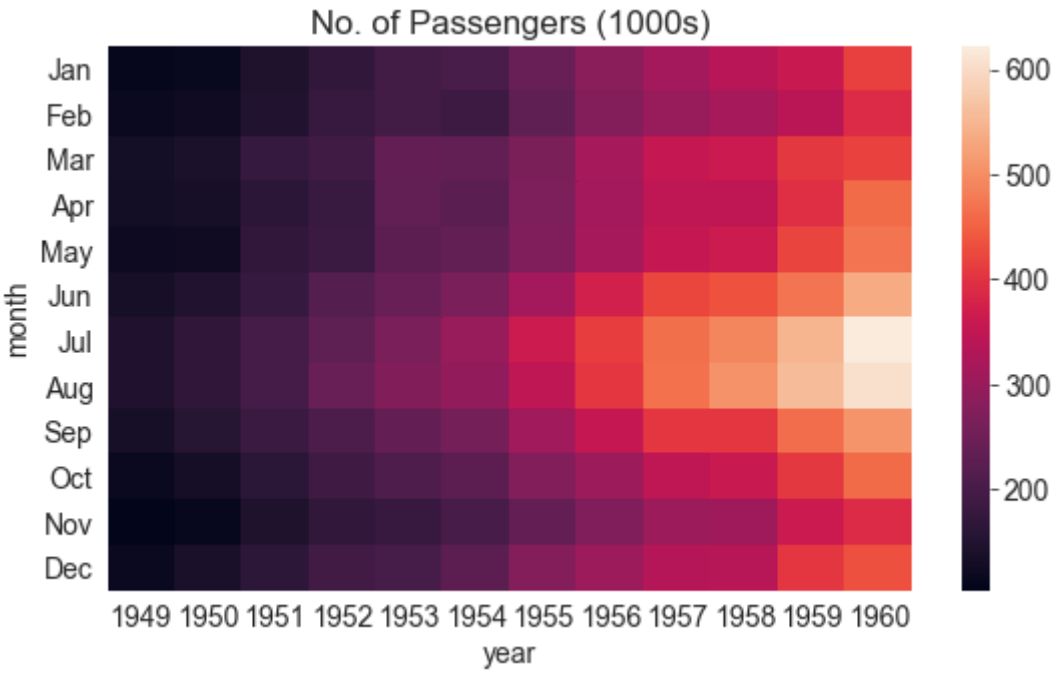
```
flights_df = sns.load_dataset("flights").pivot("month", "year", "passengers")
```

```
flights_df
```

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
Jan	112	115	145	171	196	204	242	284	315	340	360	417
Feb	118	126	150	180	196	188	233	277	301	318	342	391
Mar	132	141	178	193	236	235	267	317	356	362	406	419
Apr	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472
Jun	135	149	178	218	243	264	315	374	422	435	472	535
Jul	148	170	199	230	264	302	364	413	465	491	548	622
Aug	148	170	199	242	272	293	347	405	467	505	559	606
Sep	136	158	184	209	237	259	312	355	404	404	463	508
Oct	119	133	162	191	211	229	274	306	347	359	407	461
Nov	104	114	146	172	180	203	237	271	305	310	362	390
Dec	118	140	166	194	201	229	278	306	336	337	405	432

`flights_df` is a matrix with one row for each month and one column for each year. The values show the number of passengers (in thousands) that visited the airport in a specific month of a year. We can use the `sns.heatmap` function to visualize the footfall at the airport.

```
plt.title("No. of Passengers (1000s)")
sns.heatmap(flights_df);
```

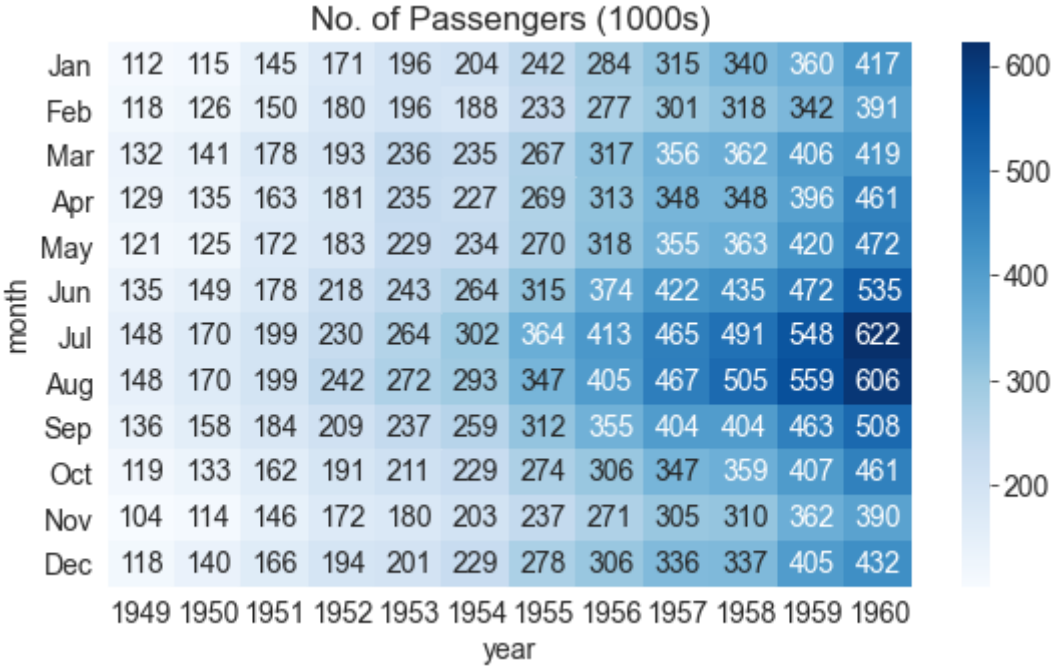


The brighter colors indicate a higher footfall at the airport. By looking at the graph, we can infer two things:

- The footfall at the airport in any given year tends to be the highest around July & August.
- The footfall at the airport in any given month tends to grow year by year.

We can also display the actual values in each block by specifying `annot=True` and using the `cmap` argument to change the color palette.

```
plt.title("No. of Passengers (1000s)")
sns.heatmap(flights_df, fmt="d", annot=True, cmap='Blues');
```



Let's save and upload our work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Attempting to save notebook..
[jovian] Updating notebook "aakashns/python-matplotlib-data-visualization" on https://jovian.ai/
[jovian] Uploading notebook..
[jovian] Capturing environment..
[jovian] Committed successfully! https://jovian.ai/aakashns/python-matplotlib-data-visualization
```

```
'https://jovian.ai/aakashns/python-matplotlib-data-visualization'
```

Images

We can also use Matplotlib to display images. Let's download an image from the internet.

```
from urllib.request import urlretrieve
```

```
urlretrieve('https://i.imgur.com/SkPbq.jpg', 'chart.jpg');
```

Before displaying an image, it has to be read into memory using the `PIL` module.

```
from PIL import Image
```

```
img = Image.open('chart.jpg')
```

An image loaded using PIL is simply a 3-dimensional numpy array containing pixel intensities for the red, green & blue (RGB) channels of the image. We can convert the image into an array using `np.array`.

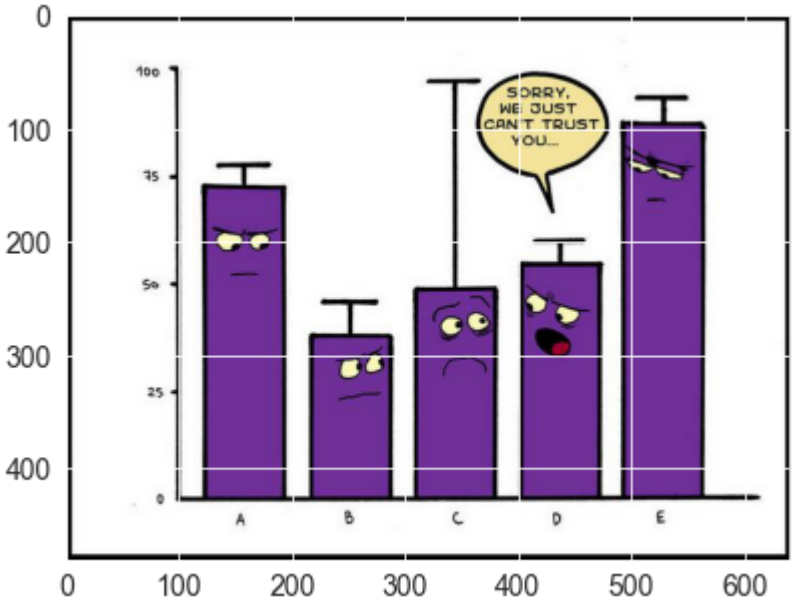
```
img_array = np.array(img)
```

```
img_array.shape
```

```
(481, 640, 3)
```

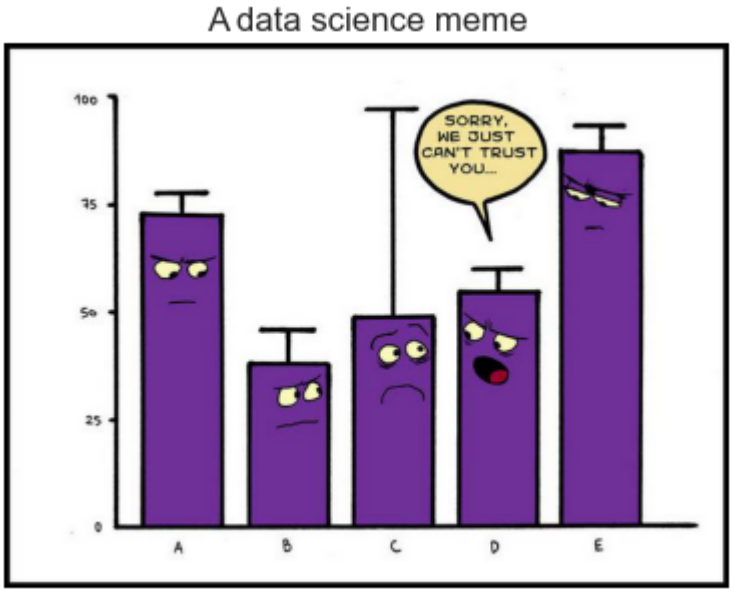
We can display the PIL image using `plt.imshow`.

```
plt.imshow(img);
```



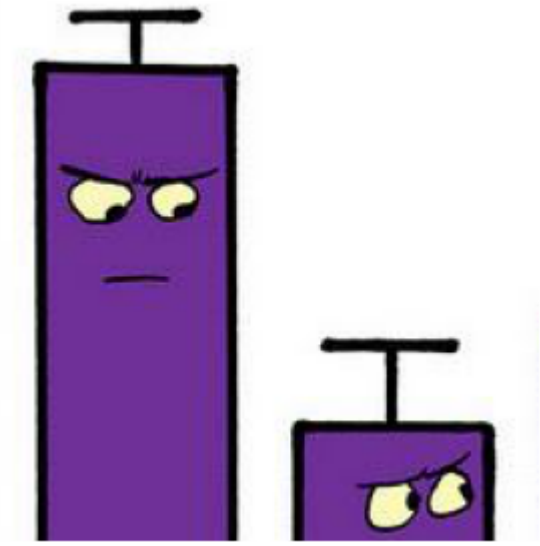
We can turn off the axes & grid lines and show a title using the relevant functions.

```
plt.grid(False)
plt.title('A data science meme')
plt.axis('off')
plt.imshow(img);
```



To display a part of the image, we can simply select a slice from the numpy array.

```
plt.grid(False)
plt.axis('off')
plt.imshow(img_array[125:325,105:305]);
```



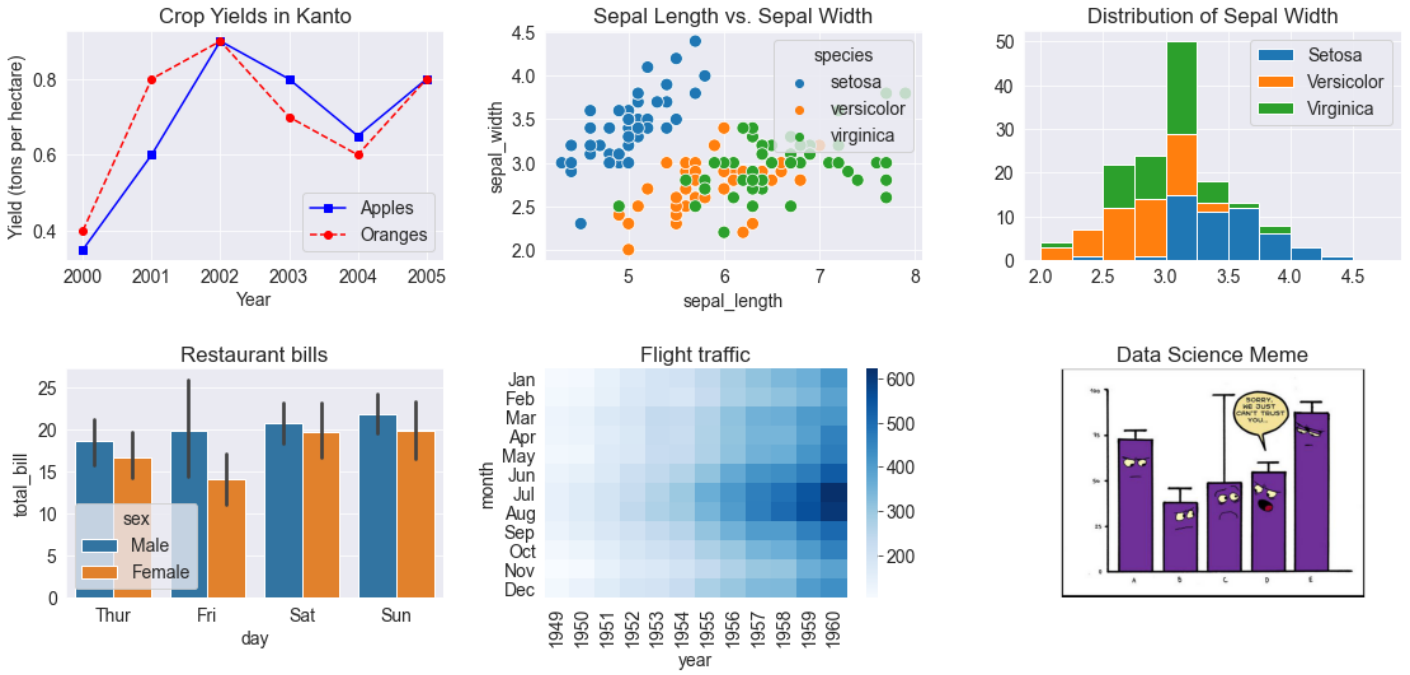
Plotting multiple charts in a grid

Matplotlib and Seaborn also support plotting multiple charts in a grid, using `plt.subplots`, which returns a set of axes for plotting.

Here's a single grid showing the different types of charts we've covered in this tutorial.

```
fig, axes = plt.subplots(2, 3, figsize=(16, 8))

# Use the axes for plotting
axes[0,0].plot(years, apples, 's-b')
axes[0,0].plot(years, oranges, 'o--r')
axes[0,0].set_xlabel('Year')
axes[0,0].set_ylabel('Yield (tons per hectare)')
axes[0,0].legend(['Apples', 'Oranges']);
axes[0,0].set_title('Crop Yields in Kanto')


# Pass the axes into seaborn
axes[0,1].set_title('Sepal Length vs. Sepal Width')
sns.scatterplot(x=flowers_df.sepal_length,
                y=flowers_df.sepal_width,
                hue=flowers_df.species,
                s=100,
                ax=axes[0,1]);

# Use the axes for plotting
axes[0,2].set_title('Distribution of Sepal Width')
axes[0,2].hist([setosa_df.sepal_width, versicolor_df.sepal_width, virginica_df.sepal_wi
         bins=np.arange(2, 5, 0.25),
         stacked=True);

axes[0,2].legend(['Setosa', 'Versicolor', 'Virginica']);

# Pass the axes into seaborn
axes[1,0].set_title('Restaurant bills')
```

```
sns.barplot(x='day', y='total_bill', hue='sex', data=tips_df, ax=axes[1,0]);

# Pass the axes into seaborn
axes[1,1].set_title('Flight traffic')
sns.heatmap(flights_df, cmap='Blues', ax=axes[1,1]);

# Plot an image using the axes
axes[1,2].set_title('Data Science Meme')
axes[1,2].imshow(img)
axes[1,2].grid(False)
axes[1,2].set_xticks([])
axes[1,2].set_yticks([])

plt.tight_layout(pad=2);
```

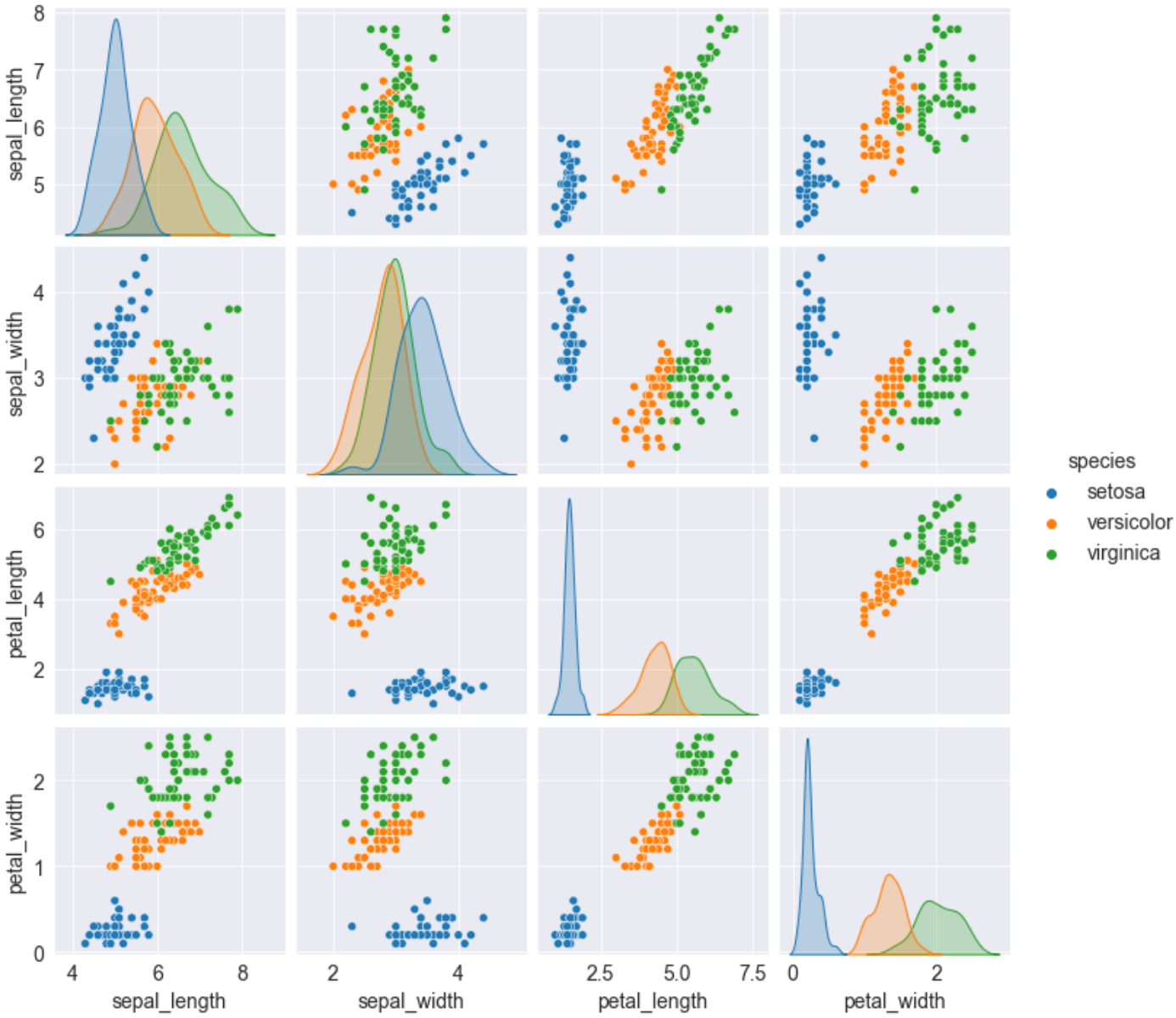


See this page for a full list of supported functions: https://matplotlib.org/3.3.1/api/axes_api.html#the-axes-class .

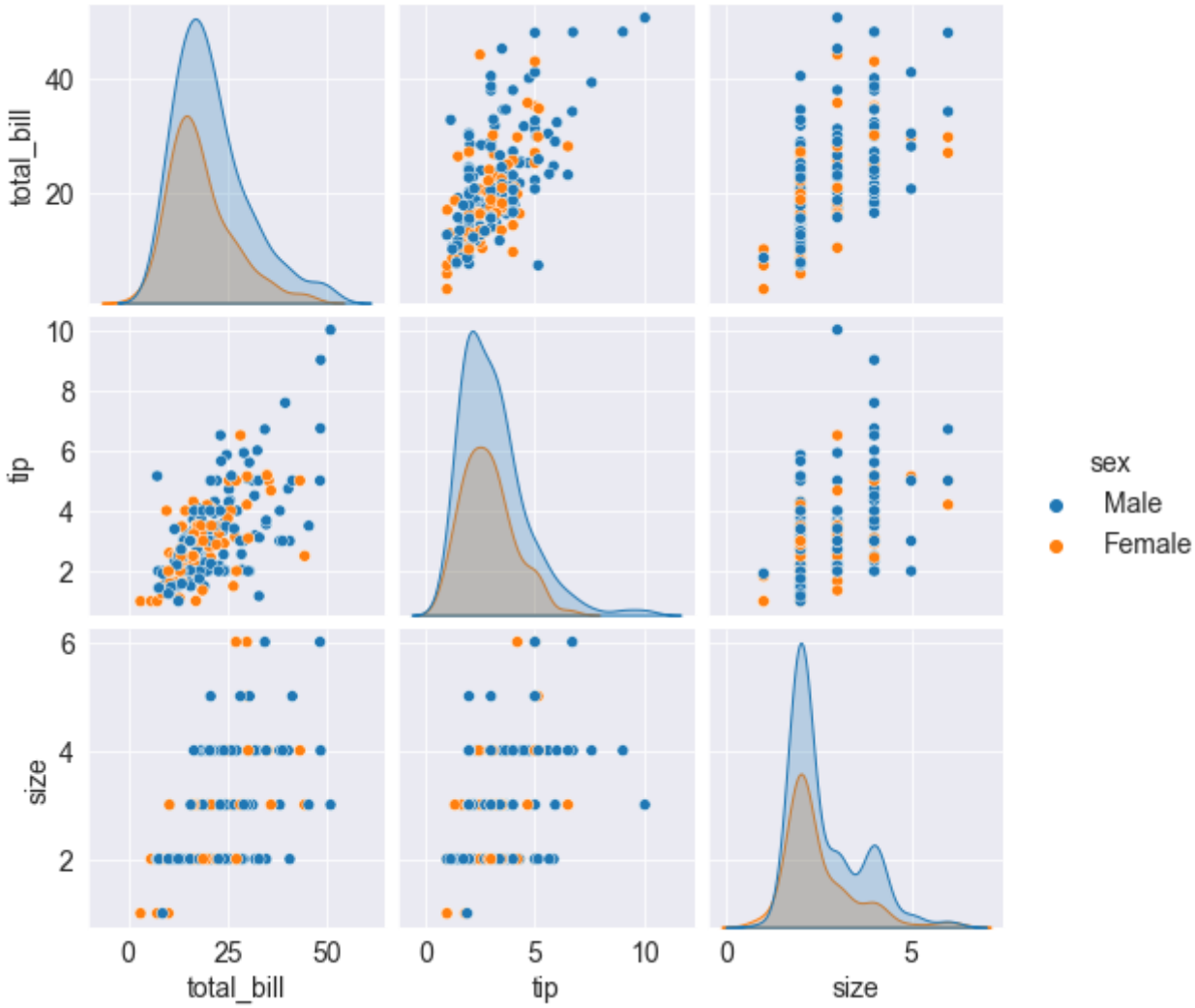
Pair plots with Seaborn

Seaborn also provides a helper function `sns.pairplot` to automatically plot several different charts for pairs of features within a dataframe.

```
sns.pairplot(flowers_df, hue='species');
```




```
sns.pairplot(tips_df, hue='sex');
```



Let's save and upload our work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Attempting to save notebook..
```

Summary and Further Reading

We have covered the following topics in this tutorial:

- Creating and customizing line charts using Matplotlib
- Visualizing relationships between two or more variables using scatter plots
- Studying distributions of variables using histograms & bar charts to
- Visualizing two-dimensional data using heatmaps
- Displaying images using Matplotlib's `plt.imshow`
- Plotting multiple Matplotlib and Seaborn charts in a grid

In this tutorial we've covered some of the fundamental concepts and popular techniques for data visualization using Matplotlib and Seaborn. Data visualization is a vast field and we've barely scratched the surface here. Check out these references to learn and discover more:

- Data Visualization cheat sheet: https://jovian.ml/aakashns/dataviz-cheatsheet
- Seaborn gallery: https://seaborn.pydata.org/examples/index.html
- Matplotlib gallery: https://matplotlib.org/3.1.1/gallery/index.html
- Matplotlib tutorial: https://github.com/rougier/matplotlib-tutorial

You are now ready to move on to the next tutorial: Exploratory Data Analysis - A Case Study

Questions for Revision

Try answering the following questions to test your understanding of the topics covered in this notebook:

1. What is data visualization?
2. What is Matplotlib?
3. What is Seaborn?
4. How do you install Matplotlib and Seaborn?
5. How you import Matplotlib and Seaborn? What are the common aliases used while importing these modules?
6. What is the purpose of the magic command `%matplotlib inline`?
7. What is a line chart?
8. How do you plot a line chart in Python? Illustrate with an example.
9. How do you specify values for the X-axis of a line chart?
10. How do you specify labels for the axes of a chart?
11. How do you plot multiple line charts on the same axes?
12. How do you show a legend for a line chart with multiple lines?
13. How you set a title for a chart?
14. How do you show markers on a line chart?
15. What are the different options for styling lines & markers in line charts? Illustrate with examples?
16. What is the purpose of the `fmt` argument to `plt.plot`?
17. How do you markers without a line using `plt.plot`?
18. Where can you see a list of all the arguments accepted by `plt.plot`?
19. How do you change the size of the figure using Matplotlib?
20. How do you apply the default styles from Seaborn globally for all charts?
21. What are the predefined styles available in Seaborn? Illustrate with examples.
22. What is a scatter plot?
23. How is a scatter plot different from a line chart?
24. How do you draw a scatter plot using Seaborn? Illustrate with an example.
25. How do you decide when to use a scatter plot v.s. a line chart?
26. How do you specify the colors for dots on a scatter plot using a categorical variable?
27. How do you customize the title, figure size, legend, etc., for Seaborn plots?
28. How do you use a Pandas dataframe with `sns.scatterplot`?
29. What is a histogram?

30. When should you use a histogram v.s. a line chart?
31. How do you draw a histogram using Matplotlib? Illustrate with an example.
32. What are "bins" in a histogram?
33. How do you change the sizes of bins in a histogram?
34. How do you change the number of bins in a histogram?
35. How do you show multiple histograms on the same axes?
36. How do you stack multiple histograms on top of one another?
37. What is a bar chart?
38. How do you draw a bar chart using Matplotlib? Illustrate with an example.
39. What is the difference between a bar chart and a histogram?
40. What is the difference between a bar chart and a line chart?
41. How do you stack bars on top of one another?
42. What is the difference between `plt.bar` and `sns.barplot`?
43. What do the lines cutting the bars in a Seaborn bar plot represent?
44. How do you show bar plots side-by-side?
45. How do you draw a horizontal bar plot?
46. What is a heat map?
47. What type of data is best visualized with a heat map?
48. What does the `pivot` method of a Pandas dataframe do?
49. How do you draw a heat map using Seaborn? Illustrate with an example.
50. How do you change the color scheme of a heat map?
51. How do you show the original values from the dataset on a heat map?
52. How do you download images from a URL in Python?
53. How do you open an image for processing in Python?
54. What is the purpose of the `PIL` module in Python?
55. How do you convert an image loaded using PIL into a Numpy array?
56. How many dimensions does a Numpy array for an image have? What does each dimension represent?
57. What are "color channels" in an image?
58. What is RGB?
59. How do you display an image using Matplotlib?
60. How do you turn off the axes and gridlines in a chart?
61. How do you display a portion of an image using Matplotlib?
62. How do you plot multiple charts in a grid using Matplotlib and Seaborn? Illustrate with examples.
63. What is the purpose of the `plt.subplots` function?
64. What are pair plots in Seaborn? Illustrate with an example.
65. How do you export a plot into a PNG image file using Matplotlib?
66. Where can you learn about the different types of charts you can create using Matplotlib and Seaborn?