**Python Seaborn Tutorial For Beginners**

This Seaborn tutorial introduces you to the basics of statistical data visualization

## Seaborn: Python's Statistical Data Visualization Library

One of the best but also more challenging ways to get your insights across is to visualize them: that way, you can more easily identify patterns, grasp difficult concepts or draw the attention to key elements. When you’re using Python for data science, you’ll most probably will have already used [Matplotlib](https://matplotlib.org/), a 2D plotting library that allows you to create publication-quality figures. Another complimentary package that is based on this data visualization library is [Seaborn](http://seaborn.pydata.org/), which provides a high-level interface to draw statistical graphics.

Today’s post will cover some of the most frequently asked questions users had while they started out working with the Seaborn library. How many of the following questions can you answer correctly?

1. [Seaborn vs Matplotlib?](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#sm)
2. [How To Load Data To Construct Seaborn Plots](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#load)
   * Loading A Built-in Data Set
   * Loading Your Pandas DataFrame
3. [How To Show Seaborn Plots](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#show)
4. [How To Use Seaborn With Matplotlib Defaults](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#defaults)
5. [How To Use Seaborn’s Colors As A colormap in Matplotlib?](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#colormap)
6. [How To Scale Seaborn Plots For Other Context](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#scale)
7. [How To Temporarily Set The Plot Style](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#style)
8. [How To Set The Figure Size in Seaborn](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#figsize)
9. [How To Rotate Label Text](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#rotate)
10. [How To Set xlim or ylim in Seaborn](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#xlim)
11. [How To Set Log Scale](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#log)
12. [How To Add A Title](https://www.datacamp.com/community/tutorials/seaborn-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=278443377095&utm_targetid=aud-299261629574:dsa-473406580275&utm_loc_interest_ms=&utm_loc_physical_ms=1007740&gclid=EAIaIQobChMIj_jSk7Sr5gIVTA4rCh2ZYQv_EAAYASAAEgKBTPD_BwE#title)

[Introduction to Data Visualization with Python](https://www.datacamp.com/courses/introduction-to-data-visualization-with-python).

## Seaborn vs Matplotlib

As you have just read, Seaborn is complimentary to Matplotlib and it specifically targets statistical data visualization. But it goes even further than that: Seaborn extends Matplotlib and that’s why it can address the two biggest frustrations of working with Matplotlib. Or, as Michael Waskom says in the “[introduction to Seaborn](http://seaborn.pydata.org/introduction.html)”: “If matplotlib “tries to make easy things easy and hard things possible”, seaborn tries to make a well-defined set of hard things easy too.”

One of these hard things or frustrations had to do with the default Matplotlib parameters. Seaborn works with different parameters, which undoubtedly speaks to those users that don’t use the default looks of the Matplotlib plots.

Compare the following plots:

Script.py

# Import the necessary libraries

import matplotlib.pyplot as plt

import pandas as pd

# Initialize Figure and Axes object

fig, ax = plt.subplots()

# Load in data

tips = pd.read\_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/master/tips.csv")

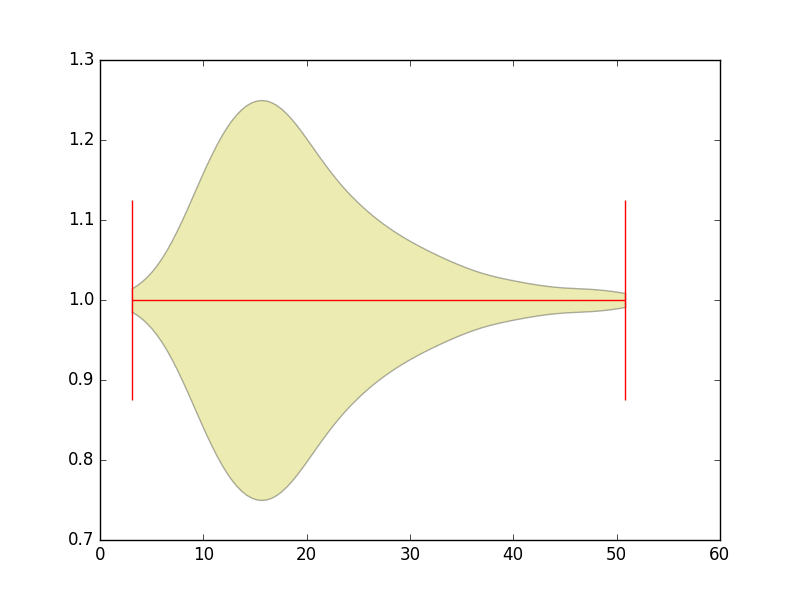
# Create violinplot

ax.violinplot(tips["total\_bill"], vert=False)

# Show the plot

plt.show()

RUN



Script.py

# Import the necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

# Load the data

tips = sns.load\_dataset("tips")

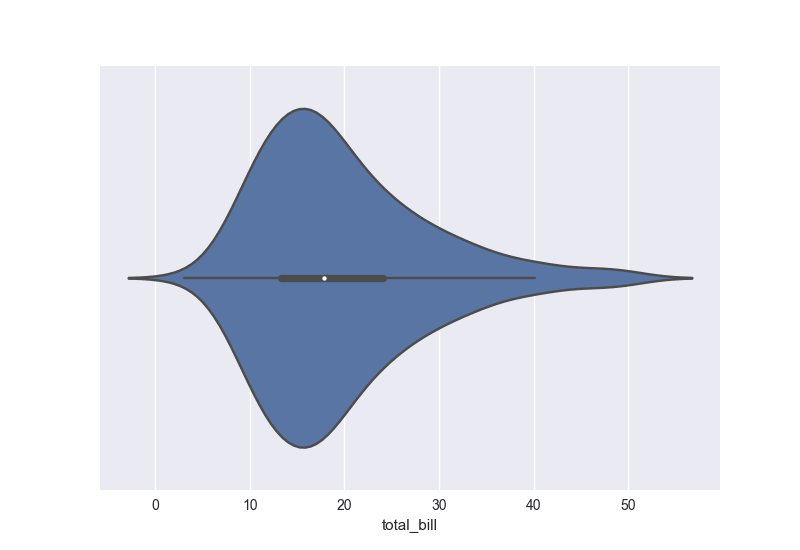
# Create violinplot

sns.violinplot(x = "total\_bill", data=tips)

# Show the plot

plt.show()

RUN



The Matplotlib defaults that usually don’t speak to users are the colors, the tick marks on the upper and right axes, the style,…

The examples above also makes another frustration of users more apparent: the fact that working with DataFrames doesn’t go quite as smoothly with Matplotlib, which can be annoying if you’re doing exploratory analysis with Pandas. And that’s exactly what Seaborn addresses: the plotting functions operate on DataFrames and arrays that contain a whole dataset.

As Seaborn compliments and extends Matplotlib, the learning curve is quite gradual: if you know Matplotlib, you’ll already have most of Seaborn down.

If you feel your matplotlib skills are rusty, check out the following resources:

* [Intermediate Python For Data Science](https://www.datacamp.com/courses/intermediate-python-for-data-science) for an introduction to Matplotlib,
* [Matplotlib Tutorial](https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python),
* [Matplotlib cheat sheet](https://www.datacamp.com/community/blog/python-matplotlib-cheat-sheet), and
* [Viewing Volumetric 3D Data with Matplotlib](https://www.datacamp.com/community/tutorials/matplotlib-3d-volumetric-data) tutorial on matplotlib’s event handler API.

## How To Load Data To Construct Seaborn Plots

When you’re working with Seaborn, you can either use one of the built-in data sets that the library itself has to offer or you can load a Pandas DataFrame. In this section, you’ll see how to do both.

### Loading A Built-in Seaborn Data Set

To start working with a built-in Seaborn data set, you can make use of the load\_dataset() function. To get an overview or inspect all data sets that this function opens up to you, go [here](https://github.com/mwaskom/seaborn-data). Check out the following example to see how the load\_dataset() function works:

Script.py

# Import necessary libraries

import seaborn as sns

import matplotlib.pyplot as plt

# Load iris data

iris = sns.load\_dataset("iris")

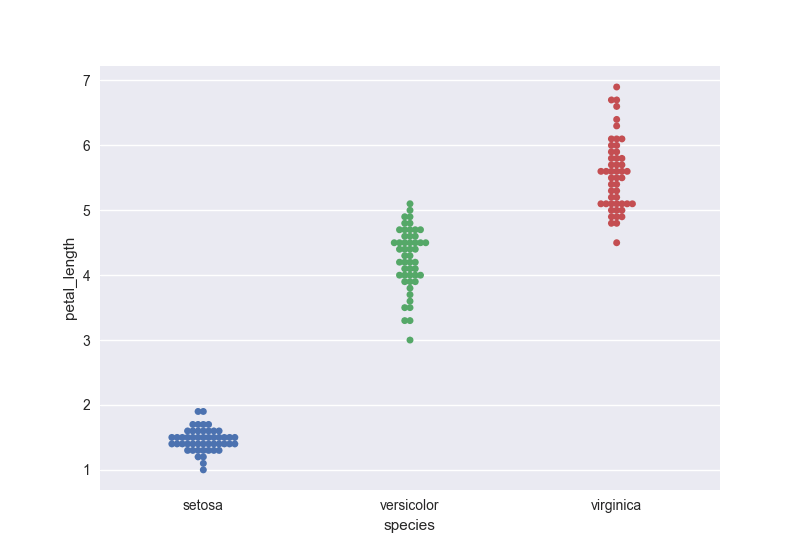
# Construct iris plot

sns.swarmplot(x="species", y="petal\_length", data=iris)

# Show plot

plt.show()

RUN



As an anecdote, it might be interesting for you to know that the import convention sns comes from the fictional character Samuel Norman “Sam” Seaborn on the television serial drama The West Wing. It’s an inside joke by the core developer of Seaborn, namely, Michael Waskom.

### Loading Your Pandas DataFrame Getting Your Data

Of course, most of the fun in visualizing data lies in the fact that you would be working with your own data and not the built-in data sets of the Seaborn library. Seaborn works best with Pandas DataFrames and arrays that contain a whole data set.

**Remember** that DataFrames are a way to store data in rectangular grids that can easily be overviewed. Each row of these grids corresponds to measurements or values of an instance, while each column is a vector containing data for a specific variable. This means that a DataFrame’s rows do not need to contain, but can contain, the same type of values: they can be numeric, character, logical, etc. Specifically for Python, DataFrames come with the Pandas library, and they are defined as a two-dimensional labeled data structures with columns of potentially different types.

The reason why Seaborn is so great with DataFrames is, for example, because labels from DataFrames are automatically propagated to plots or other data structures, as you saw in the first example of this tutorial, where you plotted a violinplot with Seaborn. There, you saw that the x-axis had a legend total\_bill, while this was not the case with the Matplotlib plot. This already takes a lot of work away from you.

But that doesn’t mean that all the work is done -quite the opposite. In many cases, you’ll need to still manipulate your Pandas DataFrame so that the plot will render correctly. If you want to know more, check out [Pandas Tutorial on DataFrames in Python](https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python) or the [Pandas Foundations](https://www.datacamp.com/courses/pandas-foundations) course.

## How To Show Seaborn Plots

Matplotlib still underlies Seaborn, which means that the anatomy of the plot is still the same and that you’ll need to use plt.show() to make the image appear to you. You might have already seen this from the previous example in this tutorial. In any case, here’s another example where the show() function is used to show the plot:

Script.py

# Import necessarily libraries

import matplotlib.pyplot as plt

import seaborn as sns

# Load data

titanic = sns.load\_dataset("titanic")

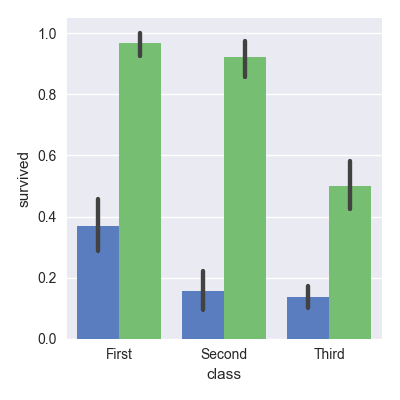
# Set up a factorplot

g = sns.factorplot("class", "survived", "sex", data=titanic, kind="bar", palette="muted", legend=False)

# Show plot

plt.show()

Run



**Note** that in the code chunk above you work with a built-in Seaborn data set and you create a factorplot with it. A factorplot is a categorical plot, which in this case is a bar plot. That’s because you have set the kind argument to "bar". Also, you set which colors should be displayed with the palette argument and that you set the legend to False.

## How To Use Seaborn With Matplotlib Defaults

As you read in the introduction, the Matplotlib defaults are something that users might not find as pleasing than the Seaborn defaults. However, there are also many questions in the opposite direction, namely, those use Seaborn and that want to plot with Matplotlib defaults.

Before, you could solve this question by importing the apionly module from the Seaborn package. This is now deprecated (since July 2017). The default style is no longer applied when Seaborn is imported, so you’ll need to explicitly call set() or one or more of set\_style(), set\_context(), and set\_palette() to get either Seaborn or Matplotlib defaults for plotting.

Script.py

# Import Matplotlib

import matplotlib.pyplot as plt

# Check the available styles

plt.style.available

# Use Matplotlib defaults

plt.style.use("classic")

Run

## How To Use Seaborn’s Colors As A colormap in Matplotlib?

Besides using Seaborn with Matplotlib defaults, there’s also questions on how to bring in Seaborn colors into Matplotlib plots. You can make use of color\_palette() to define a color map that you want to be using and the number of colors with the argument n\_colors. In this case, the example will assume that there are 5 labels assigned to the data points that are defined in data1 and data2, so that’s why you pass 5 to this argument and you also make a list with length equal to N where 5 integers vary in the variable colors.

Script.py

# Import the necessary libraries

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.colors import ListedColormap

# Define a variable N

N = 500

# Construct the colormap

current\_palette = sns.color\_palette("muted", n\_colors=5)

cmap = ListedColormap(sns.color\_palette(current\_palette).as\_hex())

# Initialize the data

data1 = np.random.randn(N)

data2 = np.random.randn(N)

# Assume that there are 5 possible labels

colors = np.random.randint(0,5,N)

# Create a scatter plot

plt.scatter(data1, data2, c=colors, cmap=cmap)

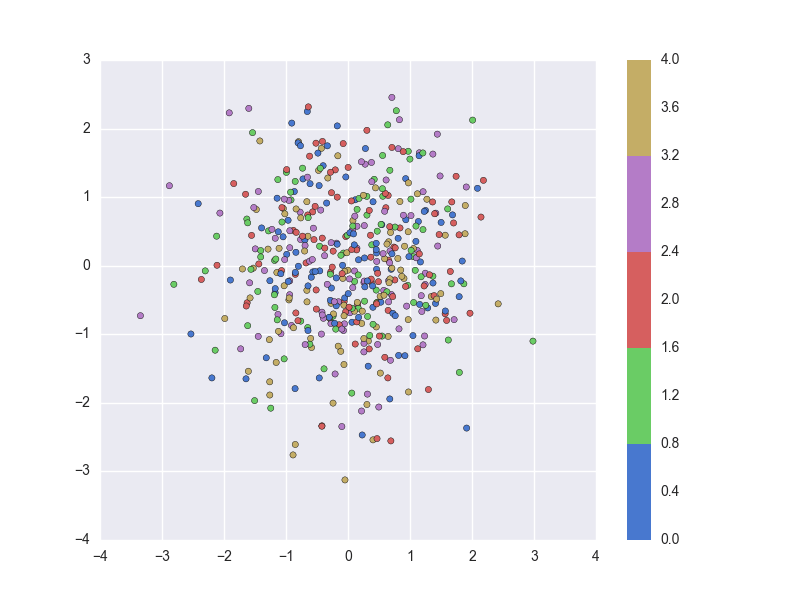
# Add a color bar

plt.colorbar()

# Show the plot

plt.show()

Run



**Tip**: do you need to revise NumPy? Consider this [NumPy Tutorial](https://www.datacamp.com/community/tutorials/python-numpy-tutorial) or the [NumPy cheat sheet](https://www.datacamp.com/community/blog/python-numpy-cheat-sheet).

## How To Scale Seaborn Plots For Other Contexts

If you need your plots for talks, posters, on paper or in notebooks, you might want to have larger or smaller plots. Seaborn has got you covered on this. You can make use of set\_context() to control the plot elements:

Script.py

# Import necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

# Reset default params

sns.set()

# Set context to `"paper"`

sns.set\_context("paper")

# Load iris data

iris = sns.load\_dataset("iris")

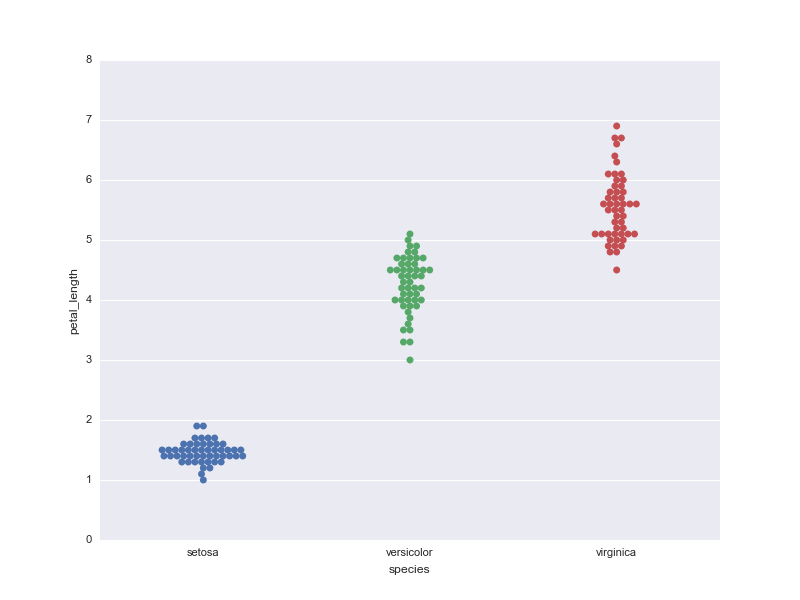
# Construct iris plot

sns.swarmplot(x="species", y="petal\_length", data=iris)

# Show plot

plt.show()

Run



The four predefined contexts are "paper", "notebook", "talk" and "poster". **Tip**: try changing the context in the DataCamp Light chunk above to another context to study the effect of the contexts on the plot.

You can also pass more arguments to set\_context() to scale more plot elements, such as font\_scale or more parameter mappings that can override the values that are preset in the Seaborn context dictionaries. In the following code chunk, you overwrite the values that are set for the parameters font.size and axes.labelsize:

Script.py

# Import necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

# Set context to `"paper"`

sns.set\_context("paper", font\_scale=3, rc={"font.size":8,"axes.labelsize":5})

# Load iris data

iris = sns.load\_dataset("iris")

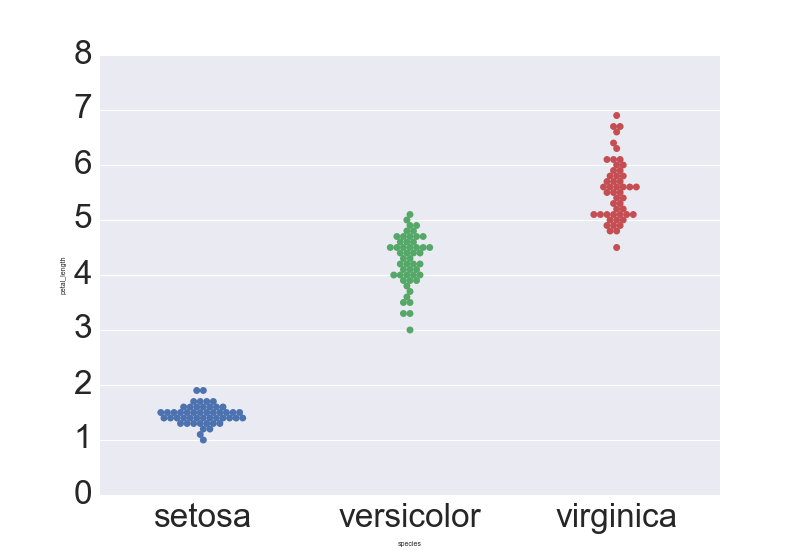
# Construct iris plot

sns.swarmplot(x="species", y="petal\_length", data=iris)

# Show plot

plt.show()

Run



**Note** that in the first code chunk, you have first done a reset to get the default Seaborn parameters back. You did this by calling set(). This is extremely handy if you have experimented with setting other parameters before, such as the plot style.

Additionally, it’s good to keep in mind that you can use the higher-level set() function instead of set\_context() to adjust other plot elements:

Script.py

# Import necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

# Reset default params

sns.set(rc={"font.size":8,"axes.labelsize":5})

# Load iris data

iris = sns.load\_dataset("iris")

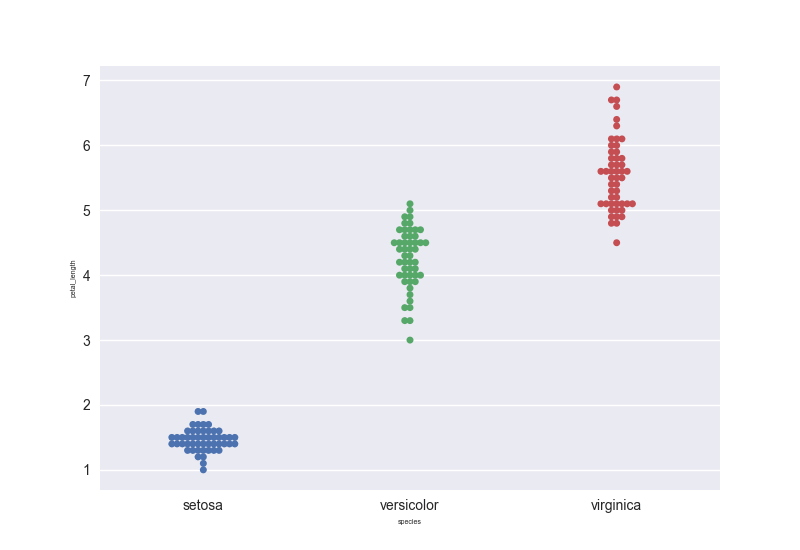
# Construct iris plot

sns.swarmplot(x="species", y="petal\_length", data=iris)

# Show plot

plt.show()

Run



One of the hardest things about data visualizations is customizing the graphs further until they meet your expectations and this stays the same when you’re working with Seaborn. That’s why it’s good to keep in mind the anatomy of the Matplotlib plot and also what this means for the Seaborn library.

As for Seaborn, you have two types of functions: axes-level functions and figure-level functions. The ones that operate on the Axes level are, for example, regplot(), boxplot(), kdeplot(), …, while the functions that operate on the Figure level are lmplot(), factorplot(), jointplot() and a couple others.

This means that the first group is identified by taking an explicit ax argument and returning an Axes object, while the second group of functions create plots that potentially include Axes which are always organized in a “meaningful” way. The Figure-level functions will therefore need to have total control over the figure so you won’t be able to plot an lmplot onto one that already exists. When you call the Figure-level functions, you always initialize a figure and set it up for the specific plot it’s drawing.

You can easily see this when you make a boxplot and an lmplot, for example:

>>> sns.boxplot(x="total\_bill", data=tips)

<matplotlib.axes.\_subplots.AxesSubplot object at 0x117e8da20>

>>> sns.lmplot('x', 'y', data, size=7, truncate=True, scatter\_kws={"s": 100})

<seaborn.axisgrid.FacetGrid object at 0x11fa03438>

However, you see that, once you’ve called lmplot(), it returns an object of the type FacetGrid. This object has some methods for operating on the resulting plot that know a bit about the structure of the plot. It also exposes the underlying figure and array of axes at the FacetGrid.fig and FacetGrid.axes arguments.

When you’re customizing your plots, this means that you will prefer to make customizations to your regression plot that you constructed with regplot() on Axes level, while you will make customizations for lmplot() on Figure level.

Let’s see how this works in practice by covering some of the following, most frequently asked questions:

## How To Temporarily Set The Plot Style

You can use axes\_style() in a with statement to temporarily set the plot style. This, in addition to the use of plt.subplot(), will allow you to make figures that have differently-styled axes, like in the example below:

Script.py

# Import necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

# Load data

iris = sns.load\_dataset("iris")

tips = sns.load\_dataset("tips")

# Set axes style to white for first subplot

with sns.axes\_style("white"):

plt.subplot(211)

sns.swarmplot(x="species", y="petal\_length", data=iris)

# Initialize second subplot

plt.subplot(212)

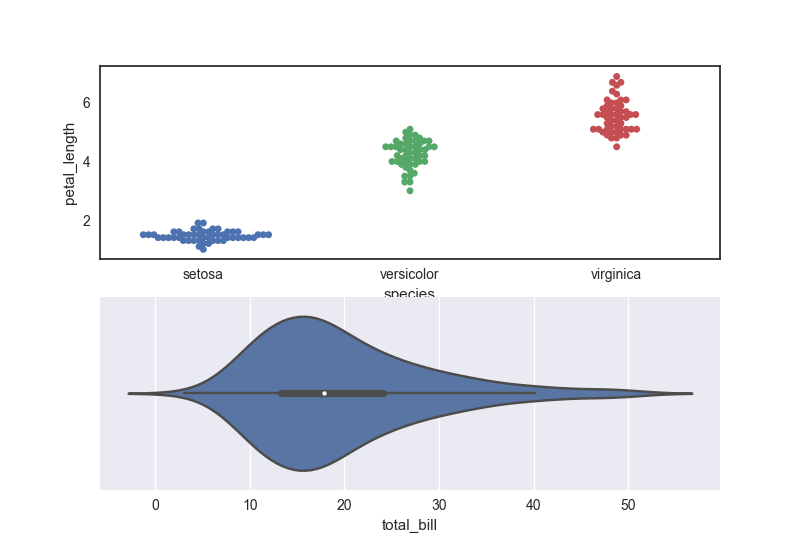
# Plot violinplot

sns.violinplot(x = "total\_bill", data=tips)

# Show the plot

plt.show()

Run



## How To Set The Figure Size in Seaborn

For axes level functions, you can make use of the plt.subplots() function to which you pass the figsize argument.

Script.py

# Import necessary libraries

import seaborn as sns

import matplotlib.pyplot as plt

# Initialize Figure and Axes object

fig, ax = plt.subplots(figsize=(10,4))

# Load in the data

iris = sns.load\_dataset("iris")

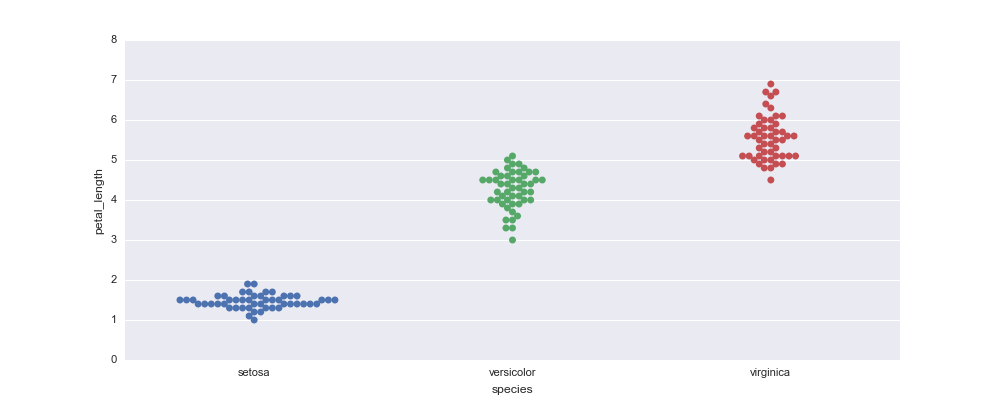
# Create swarmplot

sns.swarmplot(x="species", y="petal\_length", data=iris, ax=ax)

# Show plot

plt.show()

Run



For Figure-level functions, you rely on two parameters to set the Figure size, namely, size and aspect:

Script.py

# Import the libraries

import matplotlib.pyplot as plt

import seaborn as sns

# Load data

titanic = sns.load\_dataset("titanic")

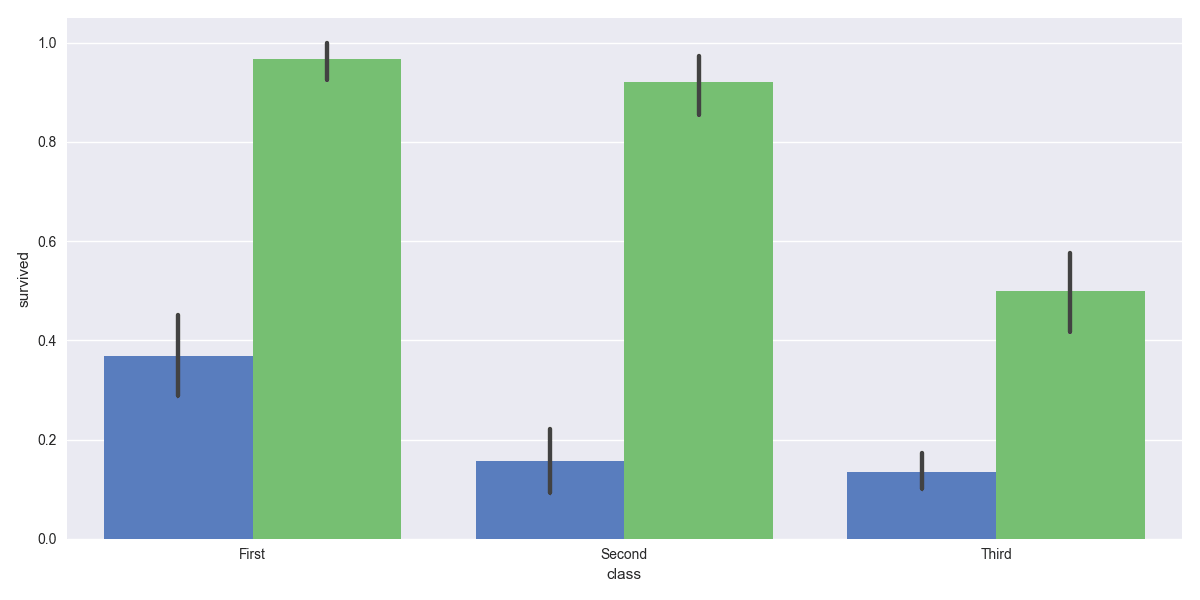
# Set up a factorplot

g = sns.factorplot("class", "survived", "sex", data=titanic, kind="bar", size=6, aspect=2, palette="muted", legend=False)

# Show plot

plt.show()

Run



## How To Rotate Label Text in Seaborn

To rotate the label text in a Seaborn plot, you will need to work on the Figure level. Note that in the code chunk below, you make use of one of the FacetGrid methods, namely, set\_xticklabels, to rotate the text label:

Script.py

# Import the necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

# Initialize the data

x = 10 \*\* np.arange(1, 10)

y = x \* 2

data = pd.DataFrame(data={'x': x, 'y': y})

# Create an lmplot

grid = sns.lmplot('x', 'y', data, size=7, truncate=True, scatter\_kws={"s": 100})

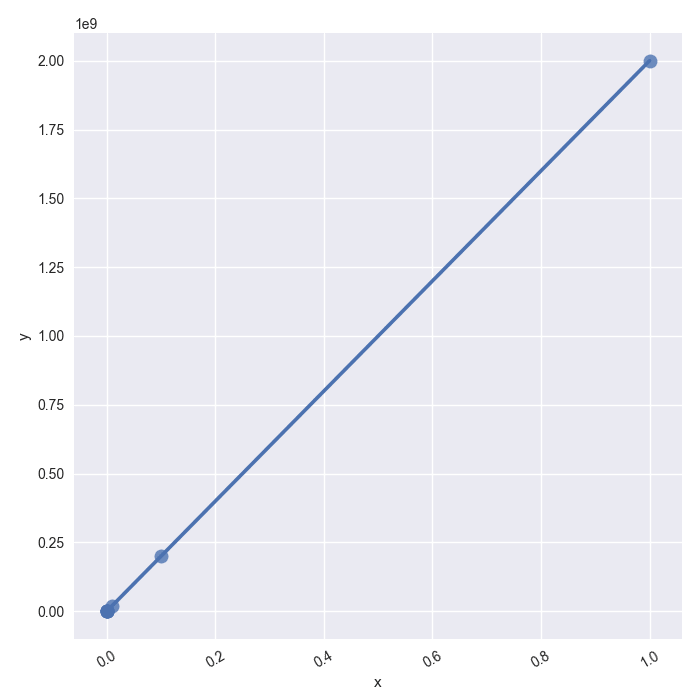
# Rotate the labels on x-axis

grid.set\_xticklabels(rotation=30)

# Show the plot

plt.show()

Run



## How To Set xlim or ylim in Seaborn

For a boxplot, which works at the Axes level, you’ll need to make sure to assign your boxplot to a variable ax, which will be a matplotlib.axes.\_subplots.AxesSubplot object, as you saw above. With the object at Axes level, you can make use of the set() function to set xlim, ylim,… Just like in the following example:

Script.py

# Import necessary libraries

import seaborn as sns

import matplotlib.pyplot as plt

# Load the data

tips = sns.load\_dataset("tips")

# Create the boxplot

ax = sns.boxplot(x="total\_bill", data=tips)

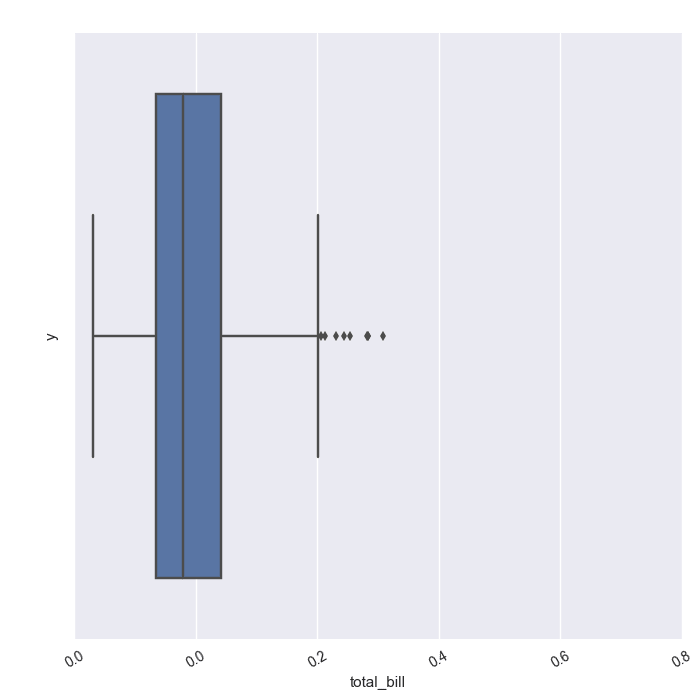
# Set the `xlim`

ax.set(xlim=(0, 100))

# Show the plot

plt.show()

Run



**Note** that alternatively, you could have also used ax.set\_xlim(10,100) to limit the x-axis.

Now, for functions at Figure-level, you can access the Axes object with the help of the axes argument. Let’s see how you can use the ax argument to your advantage to set the xlim and ylim properties:

Script.py

# Import the necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

# Initialize the data

x = 10 \*\* np.arange(1, 10)

y = x \* 2

data = pd.DataFrame(data={'x': x, 'y': y})

# Create lmplot

lm = sns.lmplot('x', 'y', data, size=7, truncate=True, scatter\_kws={"s": 100})

# Get hold of the `Axes` objects

axes = lm.ax

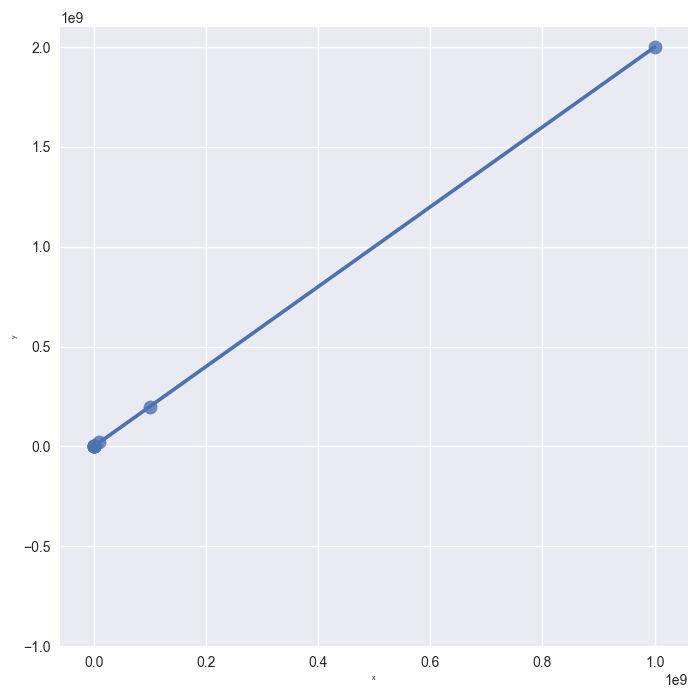
# Tweak the `Axes` properties

axes.set\_ylim(-1000000000,)

# Show the plot

plt.show()

Run



Likewise, FacetGrid exposes the underlying figure with the help of the fig argument.

Script.py

# Import the necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

# Initialize the data

x = 10 \*\* np.arange(1, 10)

y = x \* 2

data = pd.DataFrame(data={'x': x, 'y': y})

# Create lmplot

lm = sns.lmplot('x', 'y', data, size=7, truncate=True, scatter\_kws={"s": 100})

# Access the Figure

fig = lm.fig

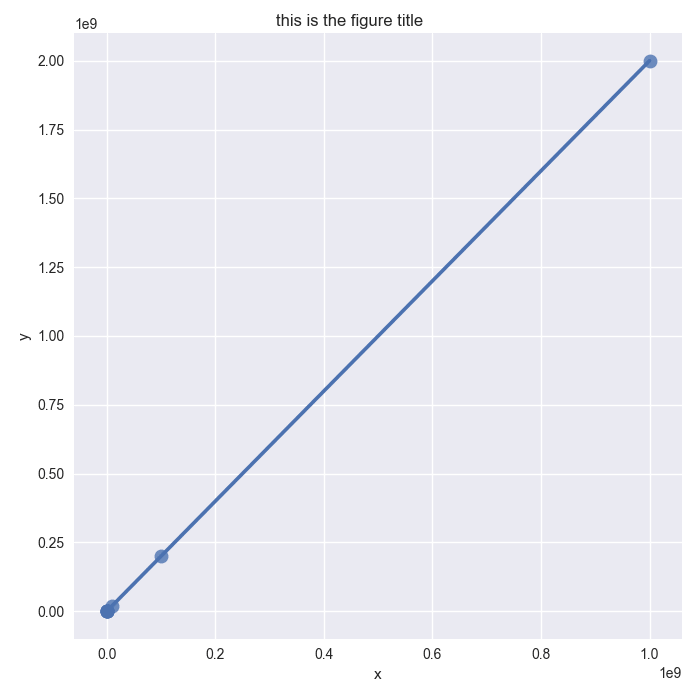
# Add a title to the Figure

fig.suptitle('this is the figure title', fontsize=12)

# Show the plot

plt.show()

Run



## How To Set Log Scale

You can modify the scale of your axes to better show trends. That’s why it might be useful in some cases to use the logarithmic scale on one or both axes. For a simple regression with regplot(), you can set the scale with the help of the Axes object.

Script.py

# Import the necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

# Create the data

x = 10 \*\* np.arange(1, 10)

y = x \* 2

data = pd.DataFrame(data={'x': x, 'y': y})

# Initialize figure and ax

fig, ax = plt.subplots()

# Set the scale of the x-and y-axes

ax.set(xscale="log", yscale="log")

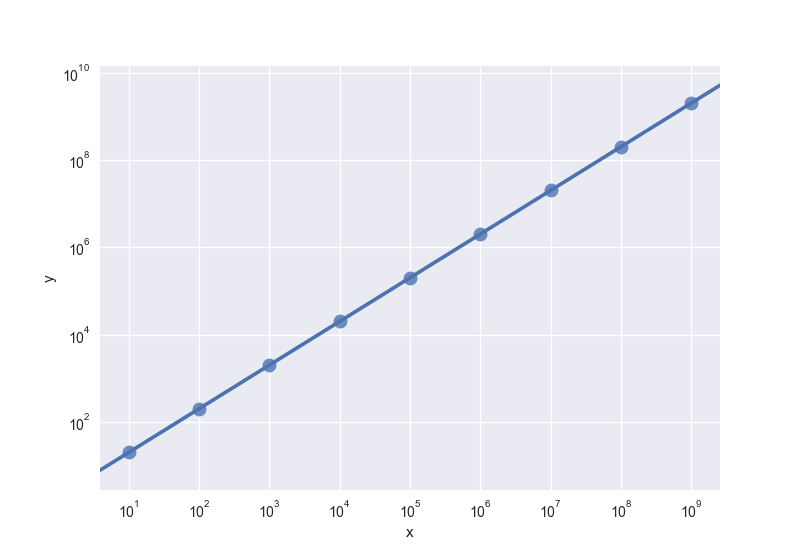
# Create a regplot

sns.regplot("x", "y", data, ax=ax, scatter\_kws={"s": 100})

# Show plot

plt.show()

Run



When you’re working with Figure level functions, you can set the xscale and yscale properties with the help of the set() method of the FacetGrid object:

Script.py

# Import the libraries

import matplotlib.pyplot as plt

import seaborn as sns

# Load data

titanic = sns.load\_dataset("titanic")

# Set up a factorplot

g = sns.factorplot("class", "survived", "sex", data=titanic, kind="bar", size=6, palette="muted", legend=False)

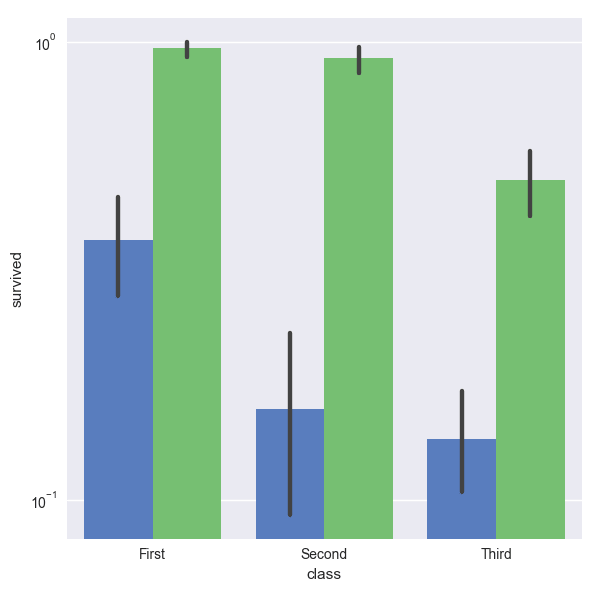
# Set the `yscale`

g.set(yscale="log")

# Show plot

plt.show()

Run



## How To Add A Title

To add titles to your Seaborn plots, you basically follow the same procedure as you have done in the previous sections. For Axes-level functions, you’ll adjust the title on the Axes level itself with the help of set\_title(). Just pass in the title that you want to see appear:

Script.py

# Import the libraries

import matplotlib.pyplot as plt

import seaborn as sns

tips = sns.load\_dataset("tips")

# Create the boxplot

ax = sns.boxplot(x="total\_bill", data=tips)

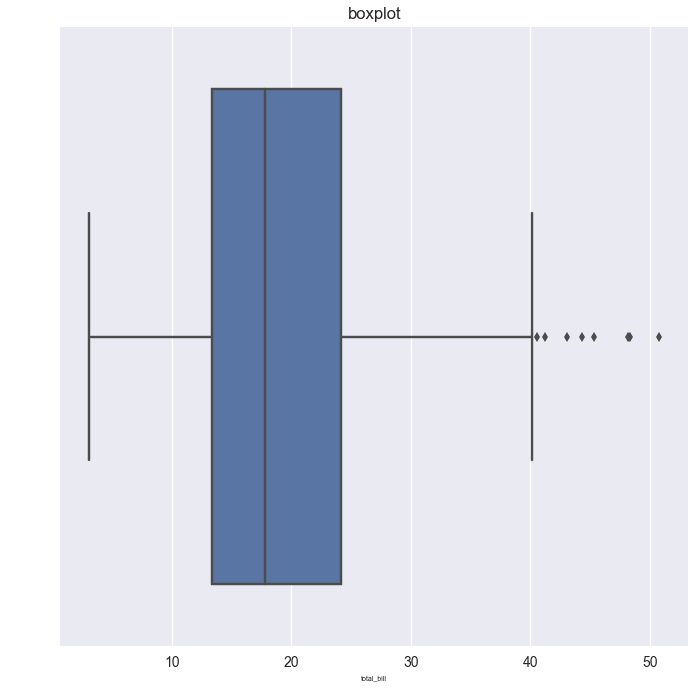
# Set title

ax.set\_title("boxplot")

# Show the plot

plt.show()

Run



For Figure-level functions, you can go via fig, just like in the factorplot that you have made in one of the previous sections, or you can also work via the Axes:

Script.py

# Import the necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

# Load the data

tips = sns.load\_dataset("tips")

# Create scatter plots

g = sns.FacetGrid(tips, col="sex", row="smoker", margin\_titles=True)

g.map(sns.plt.scatter, "total\_bill", "tip")

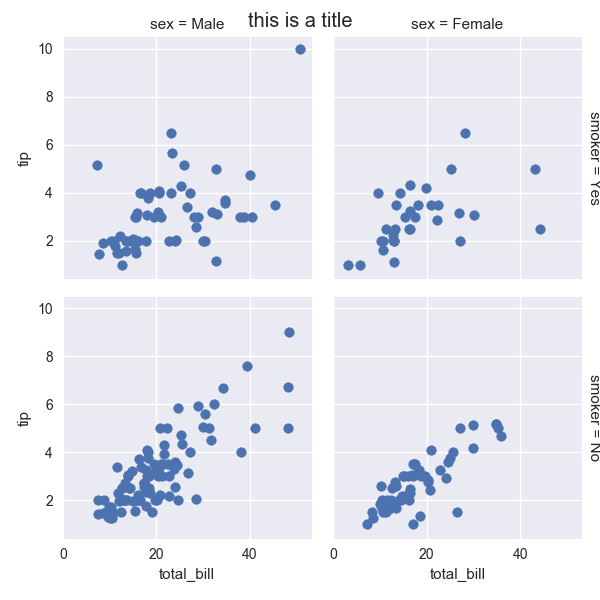
# Add a title to the figure

g.fig.suptitle("this is a title")

# Show the plot

plt.show()

Run



## Data Visualization in Python

Congratulations! You have finished this Seaborn tutorial for beginners.

If you are interested in interactive visualizations, check out, In this course, you’ll learn how to create diverse, rich, data-driven, and interactive visualizations with Bryan Van de Ven, developer of Bokeh and software engineer at Continuum Analytics. While you’re at it, also don’t miss out on DataCamp’s  [Bokeh cheat sheet](https://www.datacamp.com/community/blog/bokeh-cheat-sheet-python).

# SciPy Tutorial: What is Python SciPy and How to use it?

Mathematics deals with a huge number of concepts that are very important but at the same time, complex and time-consuming. However, [Python](https://www.edureka.co/blog/python-programming-language) provides the full-fledged SciPy library that resolves this issue for us. In this SciPy tutorial, you will be learning how to make use of this library along with a few functions and their examples.

Before moving on, take a look at all the topics discussed in this article:

* [What is SciPy?](https://www.edureka.co/blog/scipy-tutorial/#whatisscipy)
* [NumPy vs SciPy](https://www.edureka.co/blog/scipy-tutorial/#numpyvsscipy)
* [Subpackages in SciPy](https://www.edureka.co/blog/scipy-tutorial/#subpackages)
* [Basic Functions](https://www.edureka.co/blog/scipy-tutorial/#basic)
* [Special Functions](https://www.edureka.co/blog/scipy-tutorial/#special)
* [Integration Functions](https://www.edureka.co/blog/scipy-tutorial/#integration)
* [Optimization Functions](https://www.edureka.co/blog/scipy-tutorial/#optimization)
* [Fourier Transform Functions](https://www.edureka.co/blog/scipy-tutorial/#fourier)
* [Signal Processing Functions](https://www.edureka.co/blog/scipy-tutorial/#signal)
* [Linear Algebra](https://www.edureka.co/blog/scipy-tutorial/#linear)
* [Sparse Eigenvalues](https://www.edureka.co/blog/scipy-tutorial/#sparse)
* [Spatial Data Structures and Algorithms](https://www.edureka.co/blog/scipy-tutorial/#spatial)
* [Multidimensional Image Processing Functions](https://www.edureka.co/blog/scipy-tutorial/#imageprocessing)
* [File IO](https://www.edureka.co/blog/scipy-tutorial/#fileio)

So let’s get started. :)

## ****What is SciPy?****

SciPy is an open-source Python library which is used to solve scientific and mathematical problems. It is built on the [NumPy](https://www.edureka.co/blog/python-numpy-tutorial/" \t "_blank) extension and allows the user to manipulate and visualize data with a wide range of high-level commands. As mentioned earlier, SciPy builds on NumPy and therefore if you import SciPy, there is no need to import NumPy.

## ****NumPy vs SciPy****

Both NumPy and SciPy are [Python libraries](https://www.edureka.co/blog/python-libraries/) used for used mathematical and numerical analysis. NumPy contains array data and basic operations such as sorting, indexing, etc whereas, SciPy consists of all the numerical code. Though NumPy provides a number of [functions](https://www.edureka.co/blog/python-functions) that can help resolve linear algebra, Fourier transforms, etc, SciPy is the library that actually contains fully-featured versions of these functions along with many others. However, if you are doing scientific analysis using Python, you will need to install both NumPy and SciPy since SciPy builds on NumPy.

## ****Subpackages in SciPy:****

SciPy has a number of subpackages for various scientific computations which are shown in the following table:

|  |  |
| --- | --- |
| Name | Description |
| cluster | Clustering algorithms |
| constants | Physical and mathematical constants |
| fftpack | Fast Fourier Transform routines |
| integrate | Integration and ordinary differential equation solvers |
| interpolate | Interpolation and smoothing splines |
| io | Input and Output |
| linalg | Linear algebra |
| ndimage | N-dimensional image processing |
| odr | Orthogonal distance regression |
| optimize | Optimization and root-finding routines |
| signal | Signal processing |
| sparse | Sparse matrices and associated routines |
| spatial | Spatial data structures and algorithms |
| special | Special functions |
| stats | Statistical distributions and functions |

However, for a detailed description, you can follow the [official documentation](https://docs.scipy.org/doc/scipy/reference/tutorial/).

These packages need to be imported exclusively prior to using them. For example:

|  |  |
| --- | --- |
| 1 | from scipy import cluster |

Before looking at each of these functions in detail, let’s first take a look at the functions that are common both in NumPy and SciPy.

## ****Basic Functions:****

### ****Interaction with NumPy:****

SciPy builds on NumPy and therefore you can make use of NumPy functions itself to handle arrays. To know in-depth about these functions, you can simply make use of help(), info() or source() functions.

#### ****help():****

To get information about any function, you can make use of the ***help()***function. There are two ways in which this function can be used:

* without any parameters
* using parameters

Here is an example that shows both of the above methods:

|  |  |
| --- | --- |
| 1  2  3 | from scipy import cluster  help(cluster)               #with parameter  help()                       #without parameter |

When you execute the above code, the first help() returns the information about the cluster submodule. The second help() asks the user to enter the name of any module, keyword, etc for which the user desires to seek information. To stop the execution of this function, simply type ‘quit’ and hit enter.

#### ****info():****

This function returns information about the desired [functions](https://www.edureka.co/blog/python-functions), modules, etc.

|  |  |
| --- | --- |
| 1 | scipy.info(cluster) |

#### ****source():****

The source code is returned only for objects written in [Python](https://www.edureka.co/blog/python-programming-language). This function does not return useful information in case the methods or objects are written in any other language such as C. However in case you want to make use of this function, you can do it as follows:

|  |  |
| --- | --- |
| 1 | scipy.source(cluster) |

**Special Functions:**

SciPy provides a number of special functions that are used in mathematical physics such as elliptic, convenience functions, gamma, beta, etc. To look for all the functions, you can make use of help() function as described earlier.

### ****Exponential and Trigonometric Functions:****

SciPy’s Special Function package provides a number of functions through which you can find exponents and solve trigonometric problems.

Consider the following example:

**EXAMPLE:**

from scipy import special

a=special.exp10(3)

print(a)

b=special.exp(3)

print(b)

c=special.sindg(90)

print(c)

d=special.cosdg(45)

print(d)

**OUTPUT:**

1000.0  
8.0  
1.0  
0.7071067811865475

​There are many other functions present in the special functions package of SciPy that you can try for yourself.

## ****Integration Functions:****

SciPy provides a number of functions to solve integrals. Ranging from ordinary differential integrator to using trapezoidal rules to compute integrals, SciPy is a storehouse of functions to solve all types of integrals problems.

### ****General Integration:****

SiPy provides a function named ***quad***to calculate the integral of a function which has one variable. The limits can be ±∞ (± inf) to indicate infinite limits. The syntax of the quad() function is as follows:

**SYNTAX:**

quad(func, a, b, args=(), full\_output=0, epsabs=1.49e-08, epsrel=1.49e-08, limit=50, points=None, weight=None, wvar=None, wopts=None, maxp1=50, limlst=50)

Here, the function will be integrated between the limits a and b (can also be infinite).

**EXAMPLE:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | from scipy import special  from scipy import integrate  a= lambda x:special.exp10(x)  b = scipy.integrate.quad(a, 0, 1)  print(b) |

In the above example, the function ‘a’ is evaluated between the limits 0, 1. When this code is executed, you will see the following output.

**OUTPUT:**

(3.9086503371292665, 4.3394735994897923e-14)

### ****Double Integral Function:****

SciPy provides ***dblquad***that can be used to calculate double integrals. A double integral, as many of us know, consists of two real variables. The dblquad() function will take the function to be integrated as its parameter along with 4 other variables which define the limits and the functions dy and dx.

**EXAMPLE:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | from scipy import integrate  a = lambda y, x: x\*y\*\*2  b = lambda x: 1  c = lambda x: -1  integrate.dblquad(a, 0, 2, b, c) |

**OUTPUT:**

-1.3333333333333335, 1.4802973661668755e-14)

SciPy provides various other functions to evaluate triple integrals, n integrals, Romberg Integrals, etc that you can explore further in detail. To find all the details about the required functions, use the help function.

### ****Optimization Functions:****

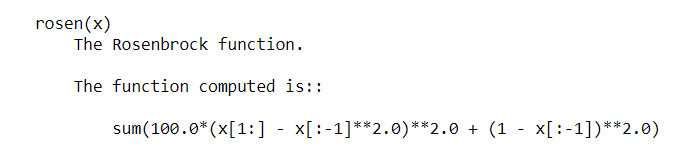
The scipy.optimize provides a number of commonly used optimization algorithms which can be seen using the help function.

It basically consists of the following:

* Unconstrained and constrained minimization of multivariate scalar functions i.e minimize (eg. BFGS, Newton Conjugate Gradient, Nelder\_mead simplex, etc)
* Global optimization routines (eg. differential\_evolution, dual\_annealing, etc)
* Least-squares minimization and curve fitting (eg. least\_squares, curve\_fit, etc)
* Scalar univariate functions minimizers and root finders (eg. minimize\_scalar and root\_scalar)
* Multivariate equation system solvers using algorithms such as hybrid Powell, Levenberg-Marquardt.

### ****Rosenbrook Function:****

Rosenbrook function (rosen) is a test problem used for gradient-based optimization algorithms. It is defined as follows in SciPy:

**EXAMPLE:**

import numpy as np

from scipy.optimize import rosen

a= 1.2 \* np.arange(5)

rosen(a)

**OUTPUT:**7371.0399999999945

### ****Nelder-Mead:****

The Nelder–Mead method is a numerical method often used to find the min/ max of a function in a multidimensional space. In the following example, the minimize method is used along with the Nelder-Mead algorithm.

**EXAMPLE:**

|  |  |
| --- | --- |
| 1  2  3  4 | from scipy import optimize  a = [2.4, 1.7, 3.1, 2.9, 0.2]  b = optimize.minimize(optimize.rosen, a, method='Nelder-Mead')  b.x |

**OUTPUT:**array([0.96570182, 0.93255069, 0.86939478, 0.75497872, 0.56793357])

## ****Interpolation Functions:****

In the field of numerical analysis, interpolation refers to constructing new data points within a set of known data points. The SciPy library consists of a subpackage named scipy.interpolate that consists of spline functions and classes, one-dimensional and multi-dimensional (univariate and multivariate) interpolation classes, etc.

### ****Univariate Interpolation:****

Univariate interpolation is basically an area of curve-fitting which finds the curve that provides an exact fit to a series of two-dimensional data points. SciPy provides interp1d function that can be utilized to produce univariate interpolation.

**EXAMPLE:**

import matplotlib.pyplot as plt

from scipy import interpolate

x= np.arange(5,20)

y= np.exp(x/3.0)

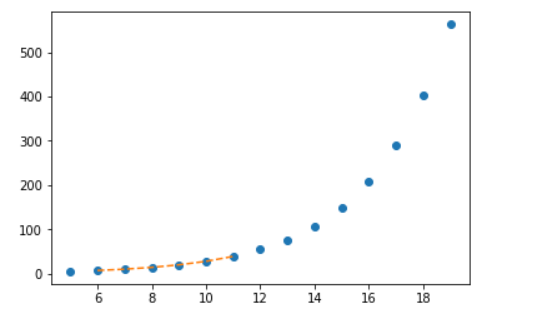
f=interpolate.interp1d(x,y)x1=np.arange(6,12)

y1=f(x1) # use interpolation function returned by ‘interp1d’

plt.plot(x,y,’o’,x1,y1,’- -‘)

plt.show( )

**OUTPUT:**



### ****Multivariate Interpolation:****

Multivariate interpolation (spatial interpolation ) is a kind interpolation on functions that consist of more than one variables. The following example demonstrates an example of the nterp2d function. Interpolating over a 2-D grid using the interp2d(x, y, z) function basically will use x, y, z arrays to approximate some function f: “z = f(x, y)“ and returns a function whose call method uses spline interpolation to find the value of new points.

**EXAMPLE:**

from scipy import interpolate

import matplotlib.pyplot as plt

x=np. Arrange(0,10)

y=np.arange (10,25)

x1,y1=np.meshgrid(x,y)

z=np.tan(xx+yy)

f=Interpolate.interp2d(x,y,z, kind=’cubic’)

x2=np.arange(2,8)

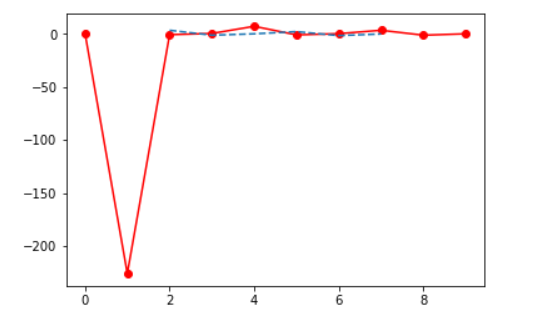
y2=np.arange(15,20)

z2=f(xnew, ynew)

plt.plot(x,z[0,:], ‘ro-, x2,z2[0,:],’- -‘)

plt.show( )

**OUTPUT:**



## ****Fourier Transform Functions:****

Fourier analysis is a method that deals with expressing a function as a sum of periodic components and recovering the signal from those components. The fft functions can be used to return the discrete Fourier transform of a real or complex sequence.

**EXAMPLE:**

from scipy.fftpack import fft, ifft

x = np.array([0,1,2,3])

y = fft(x)

print(y)

**OUTPUT:**[ 6.+0.j -2.+2.j -2.+0.j -2.-2.j ]

Similarly, you can find the inverse of this by using the ifft function as follows:

**EXAMPLE:**

**rom scipy. fftpack import fft, ifft**

**x = np.array([ 0,1,2,3])**

**y= ifft(x)**

**print(y)**

**OUTPUT:** [ 1.5+0.j -0.5-0.5j -0.5+0.j -0.5+0.5j]

## ****Signal Processing Functions:****

Signal processing deals with analyzing, modifying and synthesizing signals such as sound, images, etc. SciPy provides some functions using which you can design, filter and interpolate one-dimensional and two-dimensional data.

### ****Filtering:****

By filtering a signal, you basically remove unwanted components from it. To perform ordered filtering, you can make use of the order\_filter  function. This function basically performs ordered filtering on an array. The syntax of this function is as follows:

**SYNTAX:**  
order\_filter(a, domain, rank)

a = N-dimensional input array

domain = mask array having the same number of dimensions as `a`

rank = Non-negative number that selects elements from the list after it has been sorted (0 is the smallest followed by 1…)

**EXAMPLE:**

**from scipy import signal**

**x =np.arange(35).reshape(7,5)**

**domain= np.identity(3)**

**print(x,end=’nn’)**

**print(signal.order\_filter(x,domain,1))**

**OUTPUT:**

[[ 0 1 2 3 4]  
[ 5 6 7 8 9]  
[10 11 12 13 14]  
[15 16 17 18 19]  
[20 21 22 23 24]  
[25 26 27 28 29]  
[30 31 32 33 34]]

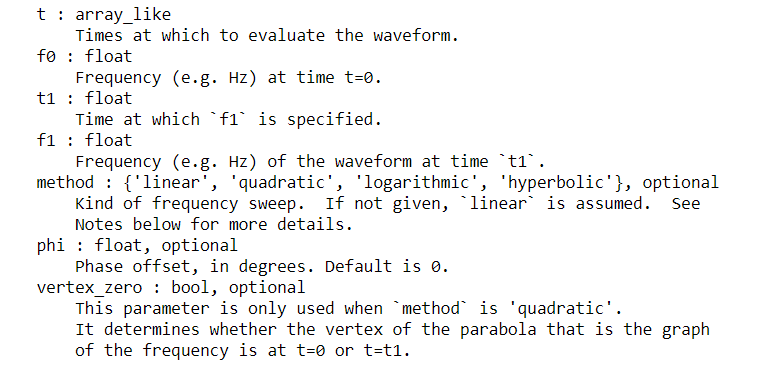
[[ 0. 1. 2. 3. 0.]  
[ 5. 6. 7. 8. 3.]  
[10. 11. 12. 13. 8.]  
[15. 16. 17. 18. 13.]  
[20. 21. 22. 23. 18.]  
[25. 26. 27. 28. 23.]  
[ 0. 25. 26. 27. 28.]]

### ****Waveforms:****

The scipy.signal subpackage also consists of various functions that can be used to generate waveforms. One such function is chirp. This function is a frequency-swept cosine generator and the syntax is as follows:

**SYNTAX:**  
chirp(t, f0, t1, f1, method=’linear’, phi=0, vertex\_zero=True)

where,



**EXAMPLE:**

**from scipy.signal import chirp, spectrogram**

**import matplotlib.pyplot as plt**

**t=np.linspace (6,10, 500)**

**w= chirp(t,f0=4,f1=2,t1=5,method=’linear’)**

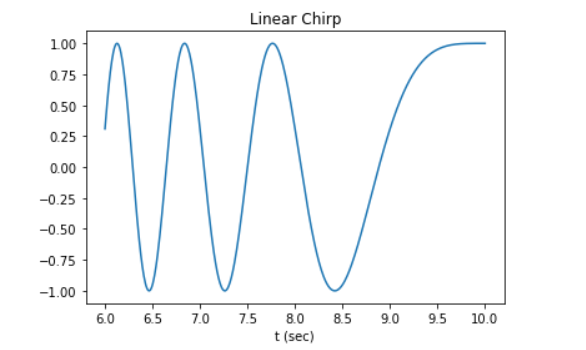
**plt.plot(t,w)**

**plt.title(“linear chirp”)**

**plt.xlabel(‘time in sec)’)**

**plt.show( )**

**OUTPUT:**



## ****Linear Algebra:****

Linear algebra deals with linear equations and their representations using vector spaces and matrices. SciPy is built on  ATLAS LAPACK and BLAS libraries and is extremely fast in solving problems related to linear algebra. In addition to all the functions from numpy.linalg, scipy.linalg also provides a number of other advanced functions. Also, if numpy.linalg is not used along with ATLAS LAPACK and BLAS support, scipy.linalg is faster than numpy.linalg.

### ****Finding the Inverse of a Matrix:****

Mathematically, the inverse of a matrix A is the matrix B such that AB=I where I is the identity matrix consisting of ones down the main diagonal denoted as B=A-1. In SciPy, this inverse can be obtained using the linalg.inv method.

**EXAMPLE:**

**import numpy as np**

**from scipy import linalg**

**A= np.array([[1,2],[4,3]])**

**B= linalg.inv(A)**

**print(B)**

**OUTPUT:**

[[-0.6 0.4]  
[ 0.8 -0.2]]

### ****Finding the Determinants:****

The value derived arithmetically from the coefficients of the matrix is known as the determinant of a square matrix. In SciPy, this can be done using a function det which has the following syntax:

**SYNTAX:**  
det(a, overwrite\_a=False, check\_finite=True)where,

a : (M, M) Is a square matrix

overwrite\_a( bool, optional) : Allow overwriting data in a

check\_finite ( bool, optional): To check whether input matrix consist only of finite numbers

**EXAMPLE:**

**import numpy as np**

**from scipy import linalg**

**A= np.array([1,2],[4,3]])**

**B=linalg.det(A)**

**print(B)**

**OUTPUT:**-5.0

## ****Sparse Eigenvalues:****

Eigenvalues are a specific set of scalars linked with linear equations. The ARPACK provides that allow you to find eigenvalues ( eigenvectors ) quite fast. The complete functionality of ARPACK is packed within two high-level interfaces which are scipy.sparse.linalg.eigs and scipy.sparse.linalg.eigsh. eigs. The eigs interface allows you to find the eigenvalues of real or complex nonsymmetric square matrices whereas the eigsh interface contains interfaces for real-symmetric or complex-hermitian matrices.

The eigh function solves a generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.

**EXAMPLE:**

**from scipy.linalg import eigh**

**import numpy as np**

**A = np.array([[1,2,3,4],[4,3,2,1],[1,4,6,3],[2,3,2,5]])**

**a, b = eigh(A)**

**print(“Selected eigenvalues :”,a)**

**print(“Complex ndarray :”,b)**

**OUTPUT:**

Selected eigenvalues : [-2.53382695 1.66735639 3.69488657 12.17158399]  
Complex ndarray : [[ 0.69205614 0.5829305 0.25682823 -0.33954321]  
[-0.68277875 0.46838936 0.03700454 -0.5595134 ]  
[ 0.23275694 -0.29164622 -0.72710245 -0.57627139]  
[ 0.02637572 -0.59644441 0.63560361 -0.48945525]]

## ****Spatial Data Structures and Algorithms:****

Spatial data basically consists of objects that are made up of lines, points, surfaces, etc. The scipy.spatial package of SciPy can compute Voronoi diagrams, triangulations, etc using the Qhull library. It also consists of KDTree implementations for nearest-neighbor point queries.

### ****Delaunay triangulations:****

Mathematically, Delaunay triangulations for a set of discrete points in a plane is a triangulation such that no point in the given set of points is inside the circumcircle of any triangle.

**EXAMPLE:**

**import matplotlib.pyplot as plt**

**from scipy.spatial import Delaunay**

**points=np.array([0,1],[1,1],[1,0],[0,0]])**

**a=Delaunay (points) #Delaunay object**

**print(a)**

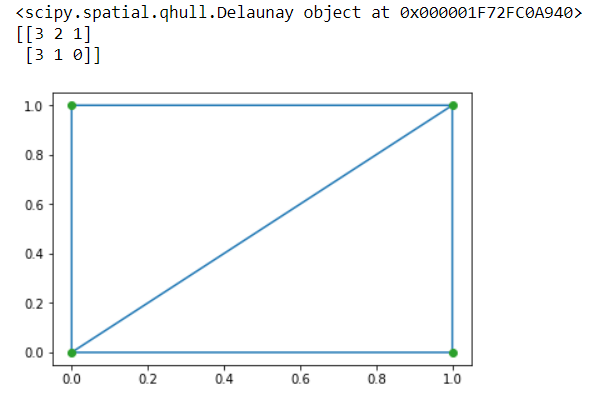
**print(a.simplices)**

**plt. Triplot(points[:,0],points,[:’1], a.simplices)**

**plt.plot(points[:,1], points [:,0],’o’)**

**plt.show( )**

**OUTPUT:**



## ****Multidimensional Image Processing Functions:****

Image processing basically deals with performing operations on an image to retrieve information or to get an enhanced image from the original one. The scipy.ndimage package consists of a number of image processing and analysis functions designed to work with arrays of arbitrary dimensionality.

### ****Convolution and correlation:****

SciPy provides a number of functions that allow correlation and convolution of images.

* The function correlate1d can be used to calculate one-dimensional correlation along a given axis
* The function correlate allows multidimensional correlation of any given array with the specified kernel
* The function convolve1d can be used to calculate one-dimensional convolution along a given axis
* The function convolve allows multidimensional convolution of any given array with the specified kernel

**EXAMPLE:**

**import numpy as np**

**from scipy.ndimage import correlate1d**

**correlate1d([3,5,1,7,2,6,9,4], weights=[1,2])**

**OUTPUT:**array([ 9, 13, 7, 15, 11, 14, 24, 17])

## ****File IO:****

The scipy.io package provides a number of functions that help you manage files of different formats such as MATLAB files, IDL files, Matrix Market files, etc.

To make use of this package, you will need to import it as follows:

import scipy.io as sio

For complete information on subpackage, you can refer to the official document on [File IO](https://docs.scipy.org/doc/scipy/reference/tutorial/io.html).

This brings us to the end of this SciPy Tutorial. I hope you have understood everything clearly. ***Make sure you practice as much as possible***.