3D plotting in Python using matplotlib

Shahbaz Khan Published: February 26, 2021 Last updated: February 27, 2021

ata visualization is one such area where a large number of libraries have been developed in Python.
mong these, Matplotlib is the most popular choice for data visualization.
hile initially developed for plotting 2-D charts like histograms, bar charts, scatter plots, line plots, etc., Matplotlib has extende
ts capabilities to offer 3D plotting modules as well.

n this tutorial, we will look at various aspects of 3D plotting in Python.

e will begin by plotting a single point in a 3D coordinate space. We will then learn how to customize our plots, and then we'll
nove on to more complicated plots like 3D Gaussian surfaces, 3D polygons, etc. Specifically, we will look at the following
opics:

Table of Contents

Plot a single point in a 3D space

.et us begin by going through every step necessary to create a 3D plot in Python, with an example of plotting a point in 3D space.

S: Import the libraries

```
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D
```

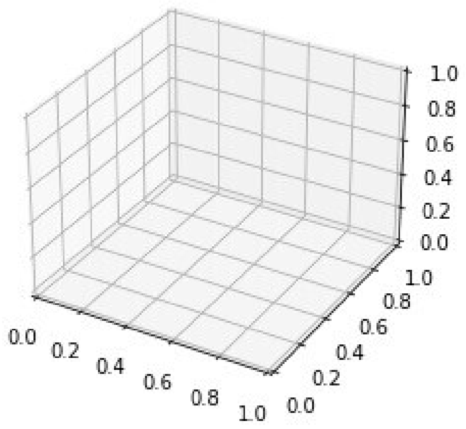
he first one is a standard import statement for plotting using matplotlib, which you would see for 2D plotting as well.
he second import of the `Axes3D` class is required for enabling 3D projections. It is, otherwise, not used anywhere else.

lote that the second import is required for Matplotlib versions before 3.2.0. For versions 3.2.0 and higher, you can plot 3D lots without importing `mpl_toolkits.mplot3d.Axes3D` .

tep 2: Create figure and axes

```
 plt.figure(figsize=(4,4))

fig.add_subplot(111, projection='3d')
```

utput:



H are first creating a figure of size 4 inches X 4 inches.
Ve then create a 3-D axis object by calling the `add_subplot` method and specifying the value ‘3d’ to the `projection`parameter.
Ve will use this axis object ‘ax’ to add any plot to the figure.

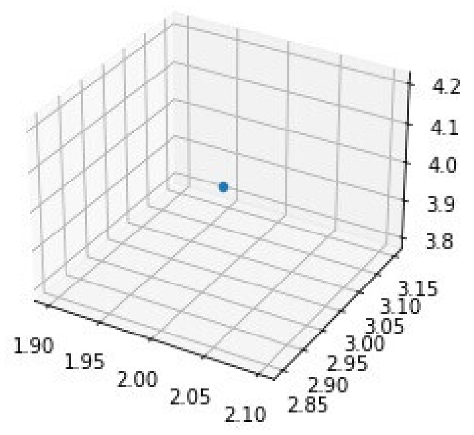
lote that these two steps will be common in most of the 3D plotting you do in Python using Matplotlib.

tep 3: Plot the point

fter we create the axes object, we can use it to create any type of plot we want in the 3D space.
o plot a single point, we will use the `scatter()` method, and pass the three coordinates of the point.

```
fig = plt.figure(figsize=(4,4))

ax = fig.add_subplot(111, projection='3d')

ax.scatter(2,3,4) # plot the point (2,3,4) on the figure

plt.show()
```

)utput:



\s you can see, a single point has been plotted (in blue) at (2,3,4).

Plotting a 3D continuous line

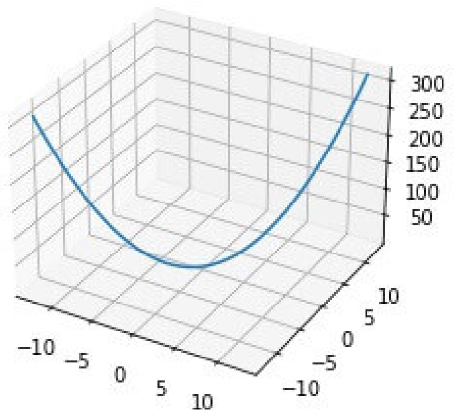
Now that we know how to plot a single point in 3D, we can similarly plot a continuous line passing through a list of 3D coordinates.

We will use the `plot()` method and pass 3 arrays, one each for the x, y, and z coordinates of the points on the line.

```
import numpy as np

x = np.linspace(-4*np.pi,4*np.pi,50)

y = np.linspace(-4*np.pi,4*np.pi,50)

z = x**2 + y**2

fig = plt.figure()
```

```
ax = fig.add_subplot(111, projection='3d')

ax.plot(x,y,z)

plt.show()
```

)utput:



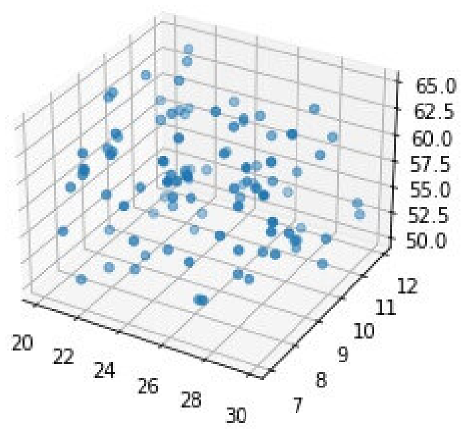
/Ve are generating x, y, and z coordinates for 50 points.
˜he x and y coordinates are generated using `np.linspace` to generate 50 uniformly distributed points between -4π and +4π. The
: coordinate is simply the sum of the squares of the corresponding x and y coordinates.

Customizing a 3D plot

Let us plot a scatter plot in 3D space and look at how we can customize its appearance in different ways based on our preferences. We will use NumPy random seed so you can generate the same random number as the tutorial.

```
np.random.seed(42)

xs = np.random.random(100)*10+20

ys = np.random.random(100)*5+7

zs = np.random.random(100)*15+50

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.scatter(xs,ys,zs)

plt.show()
```

)utput:



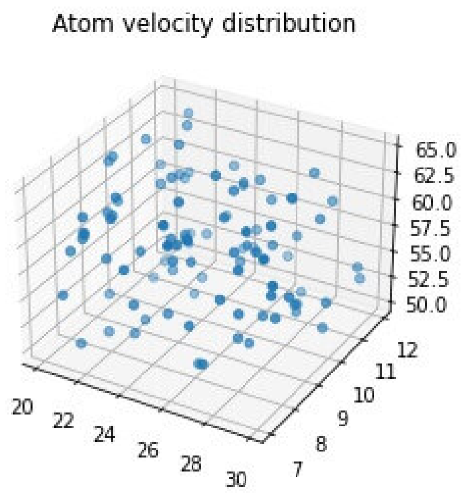
.e. .. ..)w add a title to this plot

Adding a title

Ve will call the `set_title` method of the axes object to add a title to the plot.

```
ax.set_title("Atom velocity distribution")

plt.show()
```

utput:



[illegible]at I have not added the preceding code (to create the figure and add scatter plot) here, but you should do it.

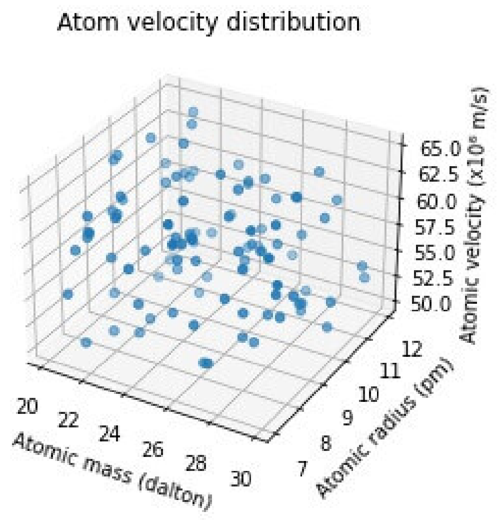
et us now add labels to each axis on the plot.

Adding axes labels

We can set a label for each axis in a 3D plot by calling the methods `set_xlabel` , `set_ylabel` and `set_zlabel` on the axes object.

```
ax.set_xlabel("Atomic mass (dalton)")

ax.set_ylabel("Atomic radius (pm)")

ax.set_zlabel("Atomic velocity (x10⁶ m/s)")

     how()
```

Output:



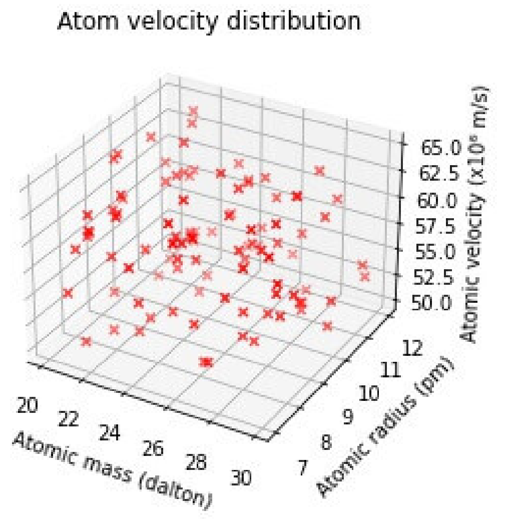
Modifying the markers

As we have seen in our previous examples, the marker for each point, by default, is a filled blue circle of constant size. We can alter the appearance of the markers to make them more expressive.

Let us begin by changing the color and style of the marker

```
ax.scatter(xs,ys,zs, marker="x", c="red")

     how()
```

Output:



We have used the parameters `marker` and `c` to change the style and color of the individual points

lodifying the axes limits and ticks

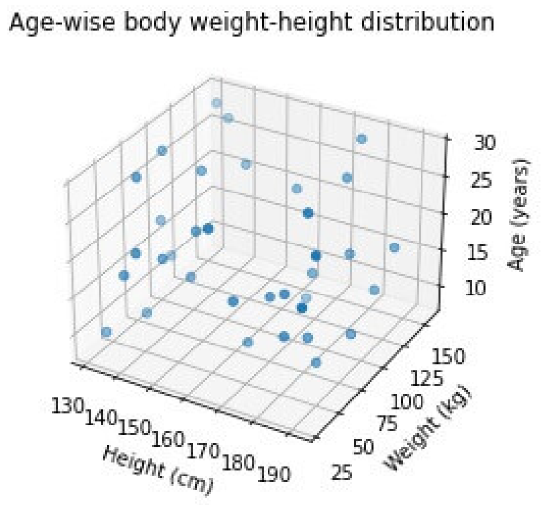
'he range and interval of values on the axes are set by default based on the input values.
Ve can however alter them to our desired values.

.et us create another scatter plot representing a new set of data points, and then modify its axes range and interval.

```
np.random.seed(42)

ages = np.random.randint(low = 8, high = 30, size=35)

heights = np.random.randint(130, 195, 35)

weights = np.random.randint(30, 160, 35)

fig = plt.figure()
```

```
ax = fig.add_subplot(111, projection='3d')

ax.scatter(xs = heights, ys = weights, zs = ages)

ax.set_title("Age-wise body weight-height distribution")

ax.set_xlabel("Height (cm)")

ax.set_ylabel("Weight (kg)")

ax.set_zlabel("Age (years)")

plt.show()
```

Output:



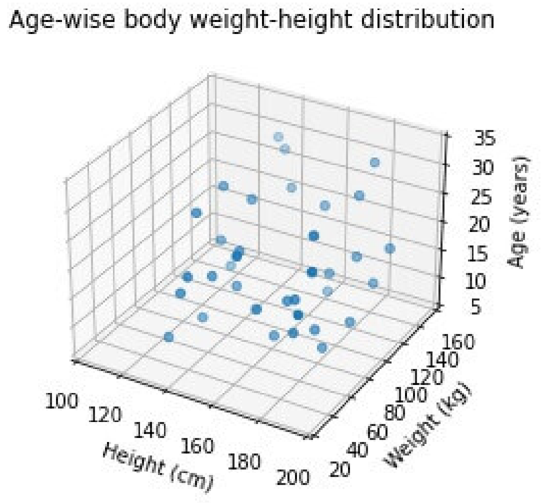
Ve have plotted data of 3 variables, namely, height, weight and age on the 3 axes.

\s you can see, the limits on the X, Y, and Z axes have been assigned automatically based on the input data.

.et us modify the minimum and maximum limit on each axis, by calling the `set_xlim` , `set_ylim` , and `set_zlim`methods.

```
ax.set_xlim(100,200)

ax.set_ylim(20,160)

     t_zlim(5,35)

     how()
```

)utput:

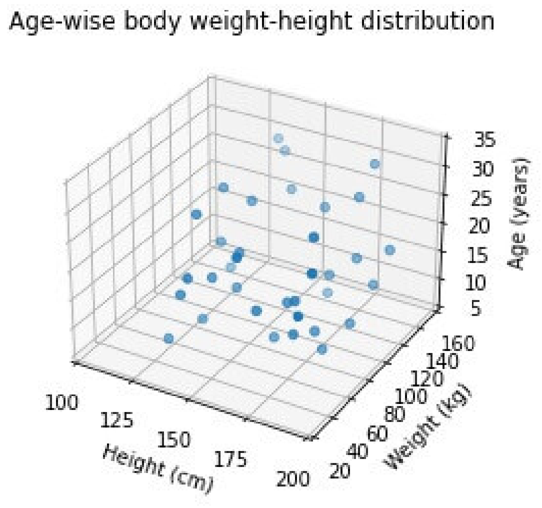


'he limits for the three axes have been modified based on the min and max values we passed to the respective methods.

Ve can also modify the individual ticks for each axis. Currently, the X-axis ticks are [100,120,140,160,180,200].

.et us update this to [100,125,150,175,200]

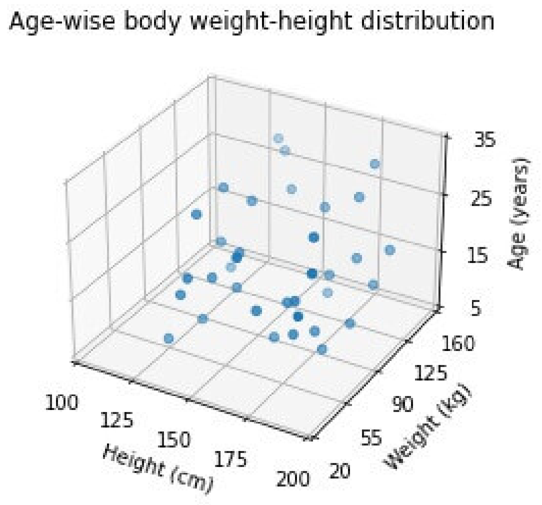
```
ax.set_xticks([100,125,150,175,200])

plt.show()
```



Similarly, we can update the Y and Z ticks using the `set_yticks` and `set_zticks` methods.

```
ax.set_yticks([20,55,90,125,160])
ax.set_zticks([5,15,25,35])

plt.show()
```

)utput:



:hange the size of the plot

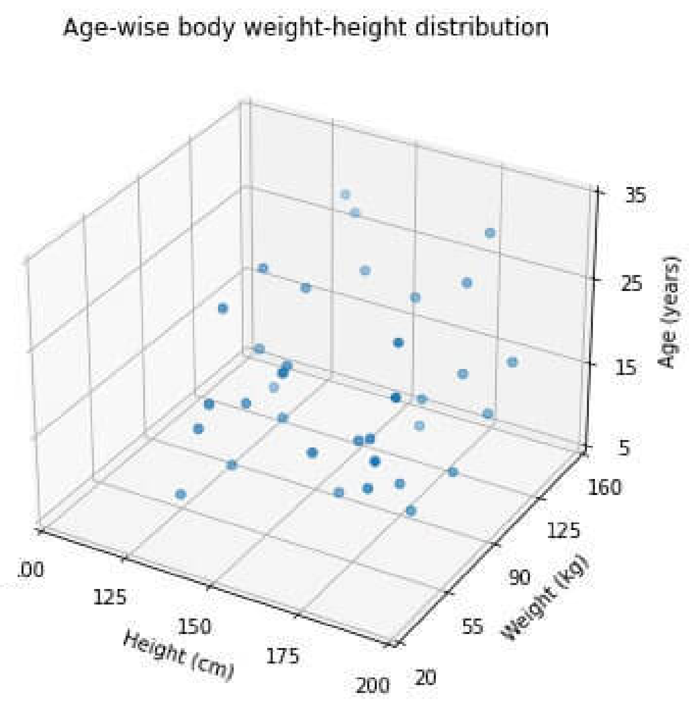
f we want our plots to be bigger or smaller than the default size, we can easily set the size of the plot either when initializing he figure – using the `figsize` parameter of the `plt.figure` method,
›r we can update the size of an existing plot by calling the `set_size_inches` method on the figure object.
n both approaches, we must specify the width and height of the plot in inches.

;ince we have seen the first method of specifying the size of the plot earlier, let us look at the second approach now i.e nodifying the size of an existing plot.

/Ve will change the size of our scatter plot to 6×6 inches.

```
fig.set_size_inches(6, 6)

plt.show()
```

)utput:



'he size of our scatter plot has been increased compared to its previous default size.

'urn off/on gridlines

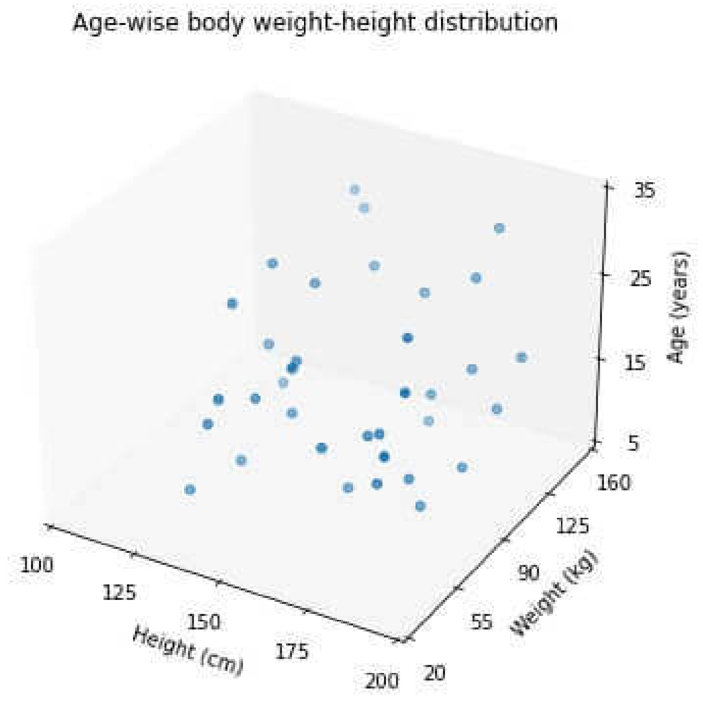
\ll the plots that we have plotted so far have gridlines on them by default.

/e can change this by calling the `grid` method of the axes object, and pass the value 'False.'

f we want the gridlines back again, we can call the same method with the parameter 'True.'.

```
ax.grid(False)

plt.show()
```

Output:



Set 3D plot colors based on class

.et us suppose that the individuals represented by our scatter plot were further divided into two or more categories.

Ve can represent this information by plotting the individuals of each category with a different color.

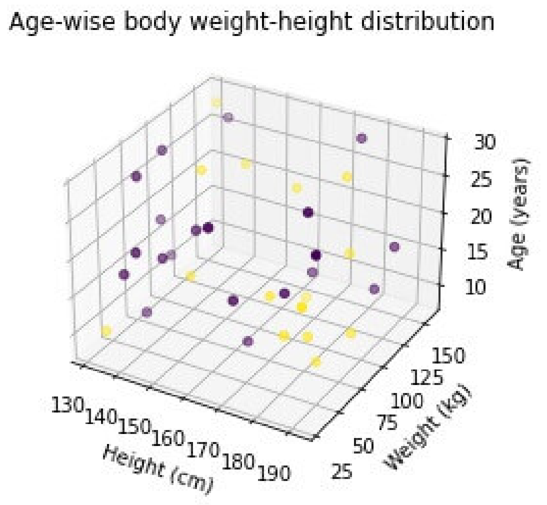
For instance, let us divide our data into 'Male' and 'Female' categories.

Ve will create a new array of the same size as the number of data points, and assign the values 0 for 'Male' and 1 for the Female' category.

Ve will then pass this array to the color parameter `c` when creating the scatter plot.

```
np.random.seed(42)

    = np.random.randint(low = 8, high = 30, size=35)

    ts = np.random.randint(130, 195, 35)

weights = np.random.randint(30, 160, 35)

gender_labels = np.random.choice([0, 1], 35) #0 for male, 1 for female

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.scatter(xs = heights, ys = weights, zs = ages, c=gender_labels)

ax.set_title("Age-wise body weight-height distribution")

ax.set_xlabel("Height (cm)")
ax.set_ylabel("Weight (kg)")

ax.set_zlabel("Age (years)")

plt.show()
```

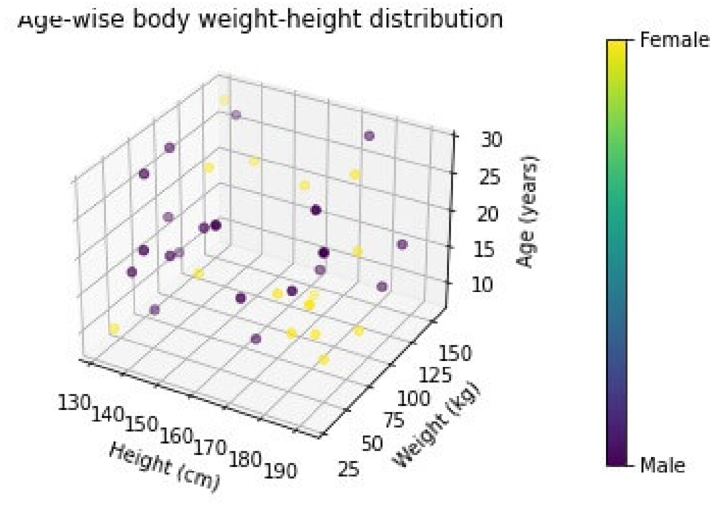
utput:



he plot now shows each of the two categories with a different color.
ut how would we know which color corresponds to which category?

Ve can add a ‘colorbar’ to solve this problem.

```
scat_plot = ax.scatter(xs = heights, ys = weights, zs = ages, c=gender_labels)

cb = plt.colorbar(scat_plot, pad=0.2)

cb.set_ticks([0,1])

cb.set_ticklabels(["Male", "Female"])

plt.show()
```



Putting legends

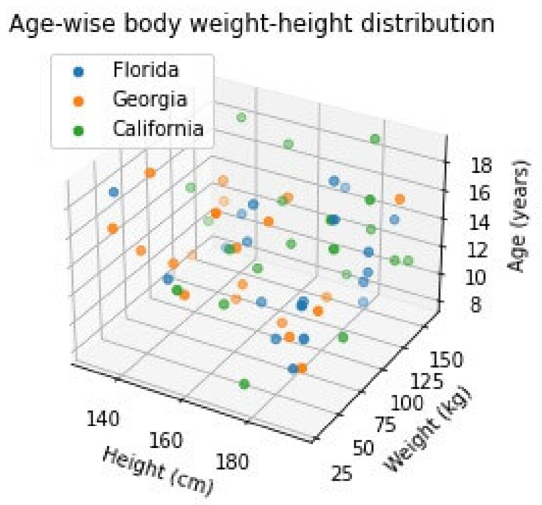
Often we have more than 1 set of data that we want to plot on the same figure.
n a situation, we must assign labels to each plot and add a legend to the figure to distinguish the different plots from each other.

For eg, let us suppose that our age-height-weight data were collected from 3 states of the United States, namely, Florida, Georgia and California.
We want to plot scatter plots for the 3 states and add a legend to distinguish them from each other.

Let us create the 3 plots in a for-loop and assign a different label to them each time.

```
labels = ["Florida", "Georgia", "California"]

for l in labels:

    ages = np.random.randint(low = 8, high = 20, size=20)
    heights = np.random.randint(130, 195, 20)

    weights = np.random.randint(30, 160, 20)

    ax.scatter(xs = heights, ys = weights, zs = ages, label=l)

ax.set_title("Age-wise body weight-height distribution")

ax.set_xlabel("Height (cm)")

ax.set_ylabel("Weight (kg)")

ax.set_zlabel("Age (years)")

ax.legend(loc="best")

plt.show()
```

Output:



Plot markers of varying size

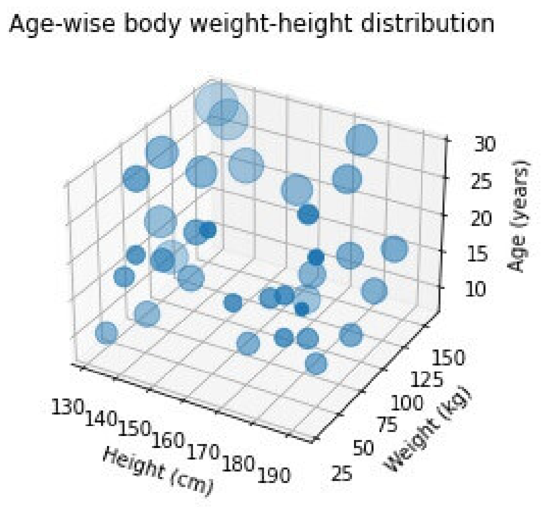
n the scatter plots that we have seen so far, all the point markers have been of constant sizes.

We can alter the size of markers by passing custom values to the parameter `s` of the scatter plot.
We can either pass a single number to set all the markers to a new fixed size, or we can provide an array of values, where each value represents the size of one marker.

n our example, we will calculate a new variable called 'bmi' from the heights and weights of individuals and make the sizes of individual markers proportional to their BMI values.

```
np.random.seed(42)

ages = np.random.randint(low = 8, high = 30, size=35)

heights = np.random.randint(130, 195, 35)

weights = np.random.randint(30, 160, 35)

bmi = weights/((heights*0.01)**2)

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.scatter(xs = heights, ys = weights, zs = ages, s=bmi*5 )

ax.set_title("Age-wise body weight-height distribution")

ax.set_xlabel("Height (cm)")

ax.set_ylabel("Weight (kg)")

ax.set_zlabel("Age (years)")

plt.show()
```

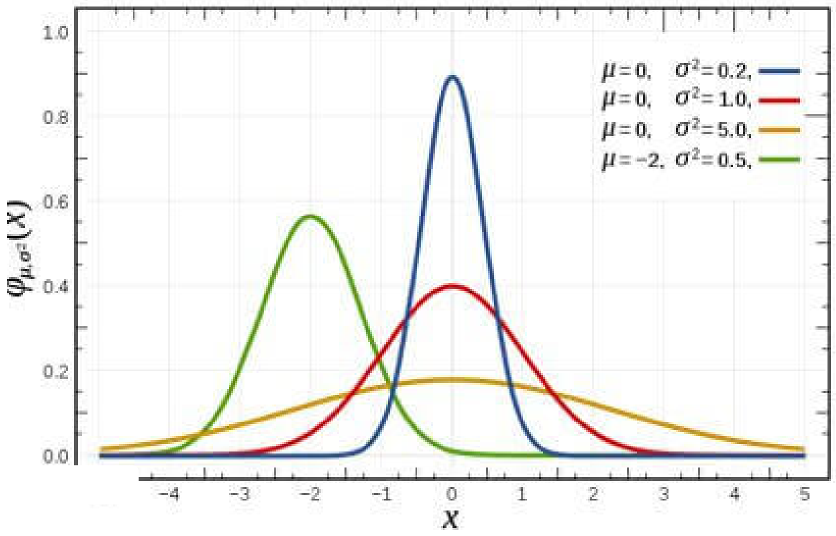
Output:



The greater the sizes of markers in this plot, the higher are the BMI’s of those individuals, and vice-versa.

Plotting a Gaussian distribution

/ou may be aware of a univariate Gaussian distribution plotted on a 2D plane, popularly known as the ‘bell-shaped curve.’



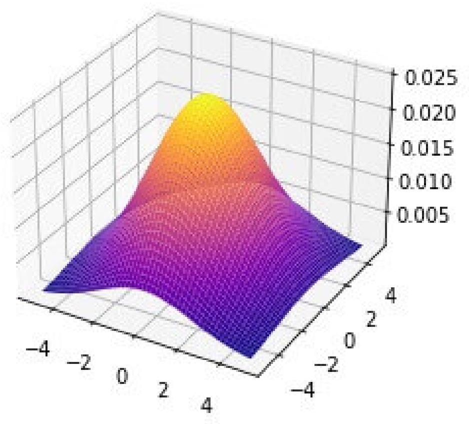
;(ıttps://en.wikipedia.org/wiki/File:Normal_Distribution_PDF.svg

/Ve can also plot a Gaussian distribution in a 3D space, using the multivariate normal distribution.
/Ve must define the variables X and Y and plot a probability distribution of them together.

```
from scipy.stats import multivariate_normal

X = np.linspace(-5,5,50)

Y = np.linspace(-5,5,50)

X, Y = np.meshgrid(X,Y)

X_mean = 0; Y_mean = 0
```

```
X_var = 5; Y_var = 8

pos = np.empty(X.shape+(2,))

pos[:,:,0]=X

pos[:,:,1]=Y

rv = multivariate_normal([X_mean, Y_mean],[[X_var, 0], [0, Y_var]])

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, rv.pdf(pos), cmap="plasma")

plt.show()
```

utput:



[illegible] `plot_surface` method, we can create similar surfaces in a 3D space.

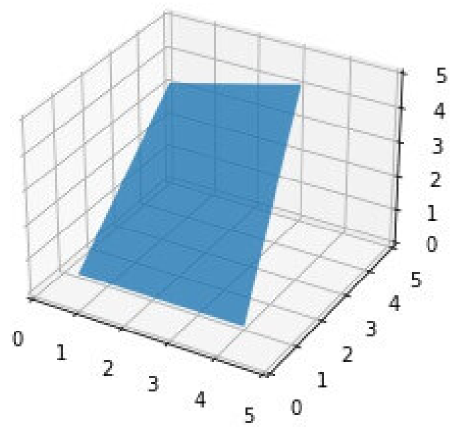
Plotting a 3D Polygon

Ve can also plot polygons with 3-dimensional vertices in Python.

```
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

x = [1, 0, 3, 4]
```

```
y = [0, 5, 5, 1]

z = [1, 3, 4, 0]

vertices = [list(zip(x,y,z))]

poly = Poly3DCollection(vertices, alpha=0.8)

ax.add_collection3d(poly)

ax.set_xlim(0,5)

ax.set_ylim(0,5)

ax.set_zlim(0,5)
```

)utput:



Rotate a 3D plot with the mouse

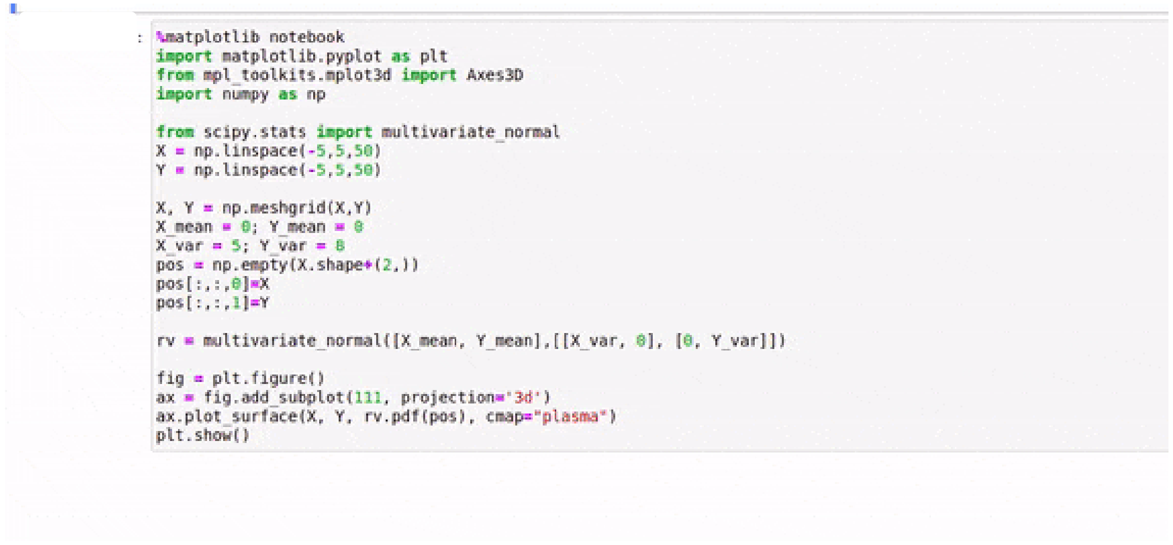
o create an interactive plot in a **Jupyter Notebook**, you should run the nagic command `%matplotlib notebook` at the beginning of the notebook.

This enables us to interact with the 3D plots, by zooming in and out of the plot, as well as rotating them in any direction.

```
%matplotlib notebook
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

import numpy as np

from scipy.stats import multivariate_normal

X = np.linspace(-5,5,50)

Y = np.linspace(-5,5,50)

X, Y = np.meshgrid(X,Y)

X_mean = 0; Y_mean = 0

X_var = 5; Y_var = 8

pos = np.empty(X.shape+(2,))

pos[:,:,0]=X

pos[:,:,1]=Y
```

```
rv = multivariate_normal([X_mean, Y_mean],[[X_var, 0], [0, Y_var]])

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, rv.pdf(pos), cmap="plasma")

plt.show()
```

Output:



```
%matplotlib notebook
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

from scipy.stats import multivariate_normal
X = np.linspace(-5,5,50)
Y = np.linspace(-5,5,50)

X, Y = np.meshgrid(X,Y)
X_mean = 0; Y_mean = 0
X_var = 5; Y_var = 8
pos = np.empty(X.shape+(2,))
pos[:,:,0]=X
pos[:,:,1]=Y

rv = multivariate_normal([X_mean, Y_mean],[[X_var, 0], [0, Y_var]])

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, rv.pdf(pos), cmap="plasma")
plt.show()
```

Plot two different 3D distributions

Ve can add two different 3D plots to the same figure, with the help of the `fig.add_subplot` method.
he 3-digit number we supply to the method indicates the number of rows and columns in the grid and the position of the
urrent plot in the grid.
he first two digits indicate the total number of rows and columns we need to divide the figure in.
he last digit indicates the position of the subplot in the grid.

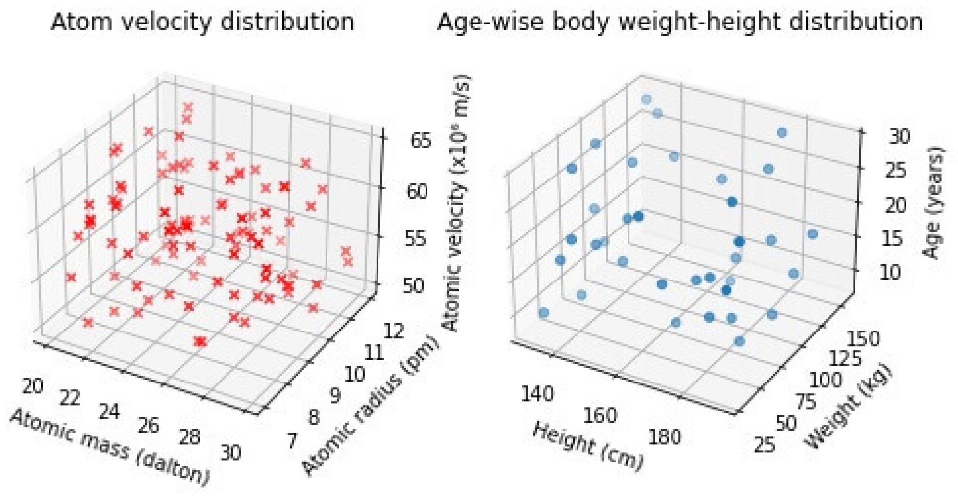
or example, if we pass the value 223 to the `add_subplot` method, we are referring to the 3rd plot in the 2×2 grid (considering
ow-first ordering).

.ɛ ɔw look at an example where we plot two different distributions on a single plot.

```
#data generation for 1st plot
np.random.seed(42)

xs = np.random.random(100)*10+20

ys = np.random.random(100)*5+7

zs = np.random.random(100)*15+50

#data generation for 2nd plot
np.random.seed(42)

ages = np.random.randint(low = 8, high = 30, size=35)

heights = np.random.randint(130, 195, 35)
```

```
weights = np.random.randint(30, 160, 35)

fig = plt.figure(figsize=(8,4))

#First plot
ax = fig.add_subplot(121, projection='3d')

ax.scatter(xs,ys,zs, marker="x", c="red")

ax.set_title("Atom velocity distribution")

ax.set_xlabel("Atomic mass (dalton)")

ax.set_ylabel("Atomic radius (pm)")

ax.set_zlabel("Atomic velocity (x10⁶ m/s)")

#Second plot
ax = fig.add_subplot(122, projection='3d')

ax.scatter(xs = heights, ys = weights, zs = ages)

ax.set_title("Age-wise body weight-height distribution")

ax.set_xlabel("Height (cm)")

ax.set_ylabel("Weight (kg)")
ax.set_zlabel("Age (years)")

plt.show()
```

)utput:



Ve can plot as many subplots as we want in this way, as long as we fit them right in the grid.

)utput Python 3D plot to HTML

f we want to embed a 3D plot figure to an HTML page, without first saving it as an image file,
ve can do so by encoding the figure into ‘base64’ and then inserting it at the correct position in an HTML `img` tag

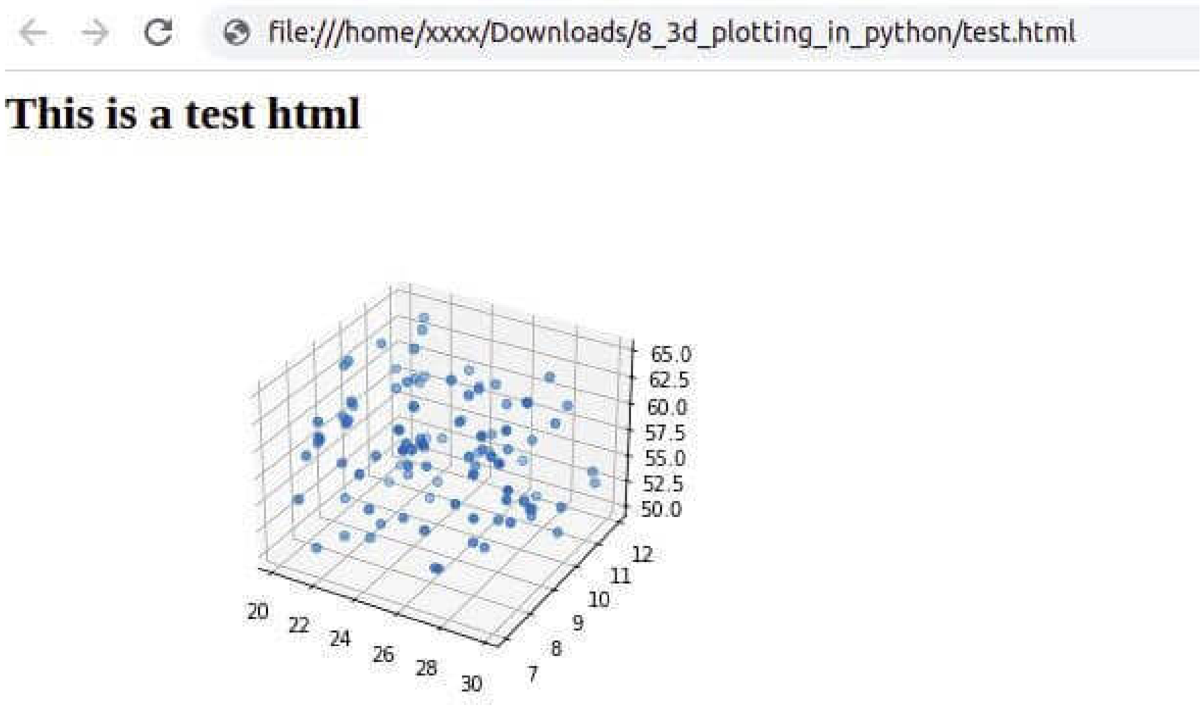
```
import base64

from io import BytesIO

np.random.seed(42)

xs = np.random.random(100)*10+20

ys = np.random.random(100)*5+7

zs = np.random.random(100)*15+50

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.scatter(xs,ys,zs)

#encode the figure
temp = BytesIO()

fig.savefig(temp, format="png")

fig_encode_bs64 = base64.b64encode(temp.getvalue()).decode('utf-8')

html_string = """
<h2>This is a test html</h2>
```

```
<img src = 'data:image/png;base64,{}'/>
""".format(fig_encode_bs64)
```

Ve can now write this HTML code string to an html file, which we can then view in a browser

```
with open("test.html", "w") as f:

    f.write(html_string)
```

)utput:



file:///home/xxxx/Downloads/8_3d_plotting_in_python/test.html

This is a test html

Conclusion

n this tutorial, we learned how to plot 3D plots in Python using the matplotlib library.

Ve began by plotting a point in the 3D coordinate space, and then plotted 3D curves and scatter plots.

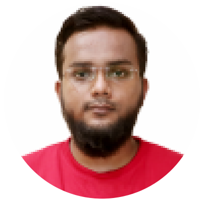
hen we learned various ways of customizing a 3D plot in Python, such as adding a title, legends, axes labels to the plot, esizing the plot, switching on/off the gridlines on the plot, modifying the axes ticks, etc.
Ve also learned how to vary the size and color of the markers based on the data point category.

fter that, we learned how to plot surfaces in a 3D space. We plotted a Gaussian distribution and a 3D polygon in Python.

Ve then saw how we can interact with a Python 3D plot in a Jupyter notebook.

inally, we learned how to plot multiple subplots on the same figure, and how to output a figure into an HTML code.

Share on Facebook Tweet on Twitter



Shahbaz Khan

Machine Learning Engineer & Software Developer working on challenging problems in Computer Vision at IITK Research and Development center.
3+ years of coding experience in Python, 1+ years of experience in Data Science and Machine Learning.
Skills: C++, OpenCV, Pytorch, Darknet, Pandas, ReactJS, Django.