

Pie plots, histograms, line plots, scatterplots, and boxplots

Working with matplotlib

1

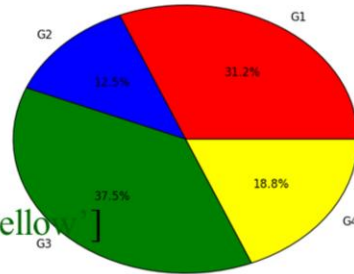
This video will discuss how to create pie plots, histograms, line plots, scatterplots, and boxplots with **matplotlib**.

Pie plot

```
dataLst = [250, 100, 300, 150]
```

```
labelLst = ['G1', 'G2', 'G3', 'G4']
```

```
colorLst = ['red', 'blue', 'green', 'yellow']
```



```
figure.pie (dataLst*, labels = labelLst, colors = colorLst,  
           autopct = '%0.1f%%')
```

* required parameter ↖ show percentages
with 1 decimal place

■ Parameters are matched based on their order in list...

i.e. dataLst[0] with labelLst[0] with colorLst[0]

2

This slide will show how to create the pie plot shown here. The dataLst contains the values that will be plotted. The labelLst contains the labels that will be used for each value. The colorLst defines the wedge colors for each value.

The **pie** method creates the pie plot. The first parameter indicates the data values and is the only required parameter.

The **autopct** parameter will calculate, for each data value, the percent the pie that is contained within each wedge. The percentage is formatted using a format string – the example shown here will display percentages with one digit after the decimal point.

Note that the values in each list are matched by their list positions. For example, the data value in the 1st position of dataLst is matched with the label in the 1st position of the labelLst and the color in the 1st position of the colorLst.

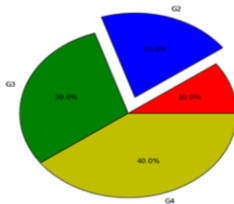
Pie plot – wedge offset

```
dataLst = [1, 2, 3, 4]
```

```
explodeLst = [0, 0.2, 0, 0]
```

```
figure.pie (dataLst , ..., explode = explodeLst)
```

list of fractions of
the radius with
which to offset
each wedge



offset 2nd
dataLst item by
20% of radius

■ Refer to docs for more options...

■ http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.pie

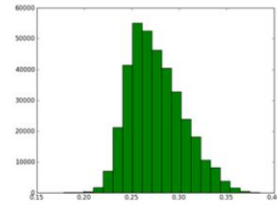
3

One or more wedges in a pie plot can be offset. The `explodeLst` defines the wedge offsets for each data value – offsets are specified in terms of fraction of the pie radius.

In this example, the wedge for the 2nd data value has an offset of 0.2 which corresponds to 20% of the pie's radius.

More information can be found for pie charts in the pyplot documentation.

Histogram



- Create a histogram...
`figure.hist(data, bins, color)`
* required parameter
list/array
single value, or list
- bins can indicate number of bins or bin boundaries...
`bins = 20` ← number of bins (i.e. number of bars)
`bins = [0, 0.1, 0.2, 0.3, ...]` ← boundaries of bins (bins can be different widths)
- To plot multiple datasets side-by-side...
 - nest a sub- list/array for each dataset within data, e.g.
`data = [dataset1, dataset2]`
 - use list to specify a color for each dataset...
`color= ['g', 'b']` ← specify in corresponding order 4

Histograms show the frequency with which values occur in a dataset. The **hist** method creates histograms.

To create a histogram, the values in the dataset are subset into bins. The **bins** can be specified as either an integer or a list of boundaries. If an integer value is used, then the dataset will be divided into the number of bins specified – each bin will have the same width. If a list is used, then consecutive values in the list will be used to define the lower and upper boundaries of the bin and the bins may be different widths.

Multiple datasets can be included in a single histogram plot by providing a 2-level list for the data parameter in the **hist** method. The top list should contain the lists for each dataset.

A list is used to define the colors when multiple datasets are plotted.

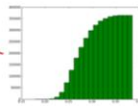
Histogram – additional options

normed = histtype = cumulative = align = } keywords

- True
- * False

show values
show as %

- * 'bar'
- 'barstacked'
- 'step'
- 'stepfilled'



- True
- * False

- 'left'
- * 'mid'
- 'right'

bar center relative to bin edge } options

- Labels for legend...
label = ['dataset 1', 'dataset 2'] ← one string per dataset
- Limits for x-axis (use when bin boundaries not specified...
range = (lower, upper) ← lower and upper limits
- Refer to docs for more options...
▪ http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist

5

Histograms have a number of additional options – the keywords and values for a few of these options are listed here. The **normed** parameter determines if the y-axis shows absolute values or percentages. Note that asterisks indicate the default values.

The **histtype** parameter determines if side-by-side or stacked bars are used for multiple datasets or if line plots are used instead of bars.

The **cumulative** parameter determines if the histogram will show cumulative values.

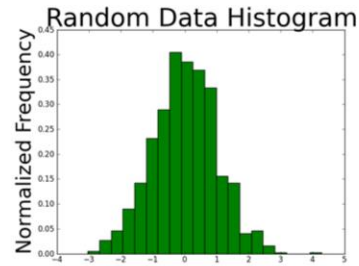
The **align** parameter determines the position of bars relative to the bin edges. The labels for the legend are specified through the **labels** parameter.

When an integer is used to defined the number of bins, then **range** parameter can be used to limit the range of values that are included in the histogram.

Refer to the pyplot documentation for further information on histograms.

Example script: Histogram

```
import numpy
import matplotlib.pyplot as figure
data =numpy.random.randn(1000)
figure.hist(data, bins = 20,
            normed = True, color = 'g')
figure.title('Random Data Histogram', size = 40)
figure.ylabel('Normalized Frequency', size = 32)
figure.show()
```



generate
random
numbers

This slide will show an example script that creates a histogram. I've imported the numpy module just to use it for generating some random data to use in the example.

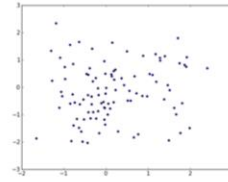
Here, I used numpy's **randn** function to create an array containing 1000 random numbers. Note that an array can be substituted for a list in any matplotlib statement.

The **hist** method creates a histogram with 20 equal sized bins. The y-axis values are set to display percentages rather than absolute values.

These statements set the figure and y-axis titles are set.

The **show** method displays the figure.

Scatter plot



- Create a scatter plot...

```
figure.scatter(xData*, yData*, s = 15, c = 'b', marker = 'o')
```

x value list/array y value list/array marker size marker color marker symbol

* required parameter

- To plot multiple datasets in one plot, run one scatter statement for each dataset...

```
figure.scatter(xData1, yData1, c = 'b')
```

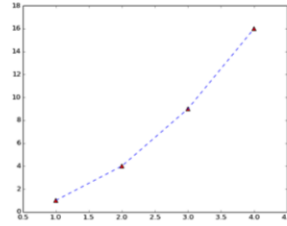
```
figure.scatter(xData2, yData2, c = 'g')
```

7

A scatter plot plots pairs of data values from two different datasets to allow correlations between the datasets to be assessed. The values from one dataset are plotted on the x-axis and values from the second dataset are plotted on the y-axis. The values in each pair are related in some way. The **scatter** method is used to create a scatterplot in matplotlib. The first parameter is the dataset that will be plotted on the x-axis, the 2nd parameter is the dataset that will be plotted on the y-axis. The two datasets are paired based on their positions in their respective datasets – so both the xData and yData lists must contain the same number of values.

Multiple pairs of datasets can be plotted in the same figure by repeating the **scatter** statement with different X and Y datasets.

Line plot



- Create a line plot...

```
figure.plot(xData,* yData,* color = 'g', markerfacecolor  
= 'b', linestyle = '--', marker = 'o', markersize = 6)
```

dashed
line style

* required parameter

- Refer to docs for more options on scatter and line plots...
 - http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.scatter
 - http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot

8

A line plot is similar to a scatter plot except that it plots sequential data such as a time series. The line plot is created using the **plot** method as shown here.

Refer to the matplotlib documentation for further information on scatterplot or line plot parameters.

Example script: Scatter\Line plot

```
import matplotlib.pyplot as figure
```

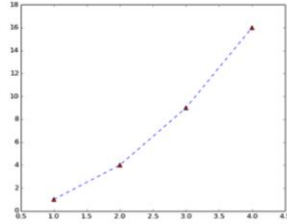
```
import numpy
```

```
xData = numpy.array(range(1, 5))
```

```
yData = xData ** 2
```

```
figure.plot(xData, yData, 'b--', marker = '^')
```

```
figure.show()
```



can combine color and
linestyle (blue dash line)

9

This slide will show an example of a script that creates a line plot.

Numpy's **array** method is used to convert the list created by **range** into an array.

Math operators work on arrays – the operation is applied to each value in the array. In this case, the values in the xData array are squared to create values for the yData array.

The **plot** method creates the line plot with a blue dashed line.

Box plot

- Create a box plot...
 - list/array →
 - outlier color →
 - outlier symbol →

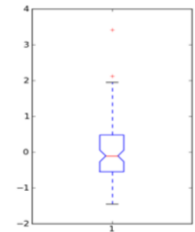
```
figure.boxplot(dataLst, *sym = 'r+', vert = True, notch = True)
```

shows notches in the boxplot (default is False)

False for horizontal (default is True)

* required parameter

- To plot multiple datasets side-by-side...
 - nest a sub- list/array for each dataset within dataLst
`dataLst = [dataset1, dataset2]`
- Refer to docs for more options...
 - http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.boxplot



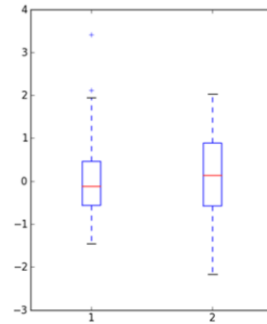
Boxplots are used to visualize the distribution of a dataset – this dataset is often the differences between two paired datasets. The **boxplot** method creates a boxplot.

The **sym** parameter specifies the color and symbol that will be used for outliers. The **vert** parameter determines whether the boxplot is vertical or horizontal. The **notch** parameter determines if the boxplot will be notched at the median.

Boxplots can be plotted side-by-side if multiple datasets are included. When multiple datasets are included, the dataLst should contain 2 levels with the top level containing a list for each dataset.

Example script: Box plot

```
import matplotlib.pyplot as figure
import numpy
data1 = numpy.random.randn(1000)
data2 = numpy.random.randn(1000)
figure.boxplot([data1, data2])
figure.show()
```



11

This slide will show an example script that creates a boxplot.

Numpy's **randn** function are used to generate random data for two datasets.

The **boxplot** method is used to create side-by-side boxplots for the two datasets.