A Review of Data Visualization Methods in Python

USHA MANJARI SIKHARAM

ABSTRACT

Data visualization involves presenting data in graphical or pictorial form which makes the information easy to understand. It helps to explain facts and determine courses of action. It will benefit any field of study that requires innovative ways of presenting large, complex information. The advent of computer graphics has shaped modern visualization. This paper presents a Python Data Visualization Libraries.

Keywords: Data visualization, Information Visualization, Scientific Visualization, Big data.

1. INTRODUCTION

Data visualization is an important skill in applied statistics and machine learning. Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a qualitative understanding. This can be helpful when exploring and getting to know a dataset and can help with identifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral to yourself and Stakeholders than measures of association or significance. We'll learn more about how to visualize data using the Python programming language below.

Python offers multiple great graphing libraries that come packed with lots of different features. No matter if you want to create interactive, live or highly customized plots python has an excellent library for you.

There are five key plots that you need to know well for basic data visualization. They are:

1. Line Plot
2. Bar Chart
3. Histogram Plot
4. Box and Whisker Plot
5. Scatter Plot

With knowledge of these plots, you can quickly get a qualitative understanding of most data that you come across.

1.1 Matplotlib

Matplotlib is a python two-dimensional plotting library for data visualization and creating interactive graphics or plots. Using python's matplotlib, the data visualization of large and complex data becomes easy.

1.2 Matplotlib Advantages

There are several advantages of using matplotlib to visualize data.

- A multi-platform data visualization tool built on the numpy and sidepy framework. Therefore, it's fast and efficient.
- It possesses the ability to work well with many operating systems and graphic back ends.
- It possesses high-quality graphics and plots to print and view for a range of graphs such as histograms, bar charts, pie charts, scatter plots and heat maps.
- With Jupyter notebook integration, the developers have been free to spend their time implementing features rather than struggling with compatibility.
- It has large community support and cross-platform support as it is an open source tool.
- It has full control over graph or plot styles such as line properties, thoughts, and access properties.

The context can be accessed via functions on *pyplot*. The context can be imported as follows

```
from matplotlib import pyplot
```

There is some convention to import this context and name it *plt*; for example:

```
import matplotlib.pyplot as plt
```

We will not use this convention; instead we will stick to the standard Python import convention.

Charts and plots are made by making and calling on context; for example:

```
pyplot.plot(...)
```

Elements such as axis, labels, legends, and so on can be accessed and configured on this context as separate function calls.

The drawings on the context can be shown in a new window by calling the show() function:

```
pyplot.show()
```

Alternately, the drawings on the context can be saved to file, such as a PNG formatted image file. The savefig() function can be used to save images.

```
pyplot.savefig('my_image.png')
```

This is the most basic crash course for using the matplotlib library.

1.3 Line Plot

A line plot is generally used to present observations collected at regular intervals.

The x-axis represents the regular interval, such as time. The y-axis shows the observations, ordered by the x-axis and connected by a line. A line plot can be created by calling the plot() function and passing the x-axis data for the regular interval, and y-axis for the observations.

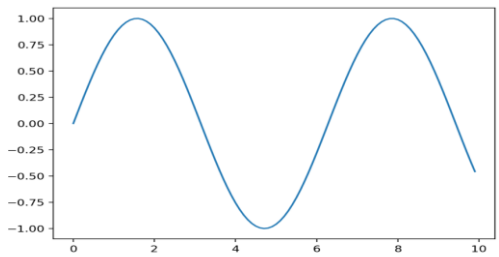
```
# create line plot
pyplot.plot(x, y)
```

Line plots are useful for presenting time series data as well as any sequence data where there is an ordering between observations.

The example below creates a sequence of 100 floating point values as the x-axis and a sine wave as a function of the x-axis as the observations on the y-axis. The results are plotted as a line plot.

```
# example of a line plot
from numpy import sin
from matplotlib import pyplot
# consistent interval for x-axis
x = [x*0.1 for x in range(100)]
# function of x for y-axis
y = sin(x)
# create line plot
pyplot.plot(x, y)
# show line plot
pyplot.show()
```

Running the example creates a line plot showing the familiar sine wave pattern on the y-axis across the x-axis with a consistent interval between observations.



1.4 Bar Chart

A bar chart is generally used to present relative quantities for multiple categories.

The x-axis represents the categories and are spaced evenly.

The y-axis represents the quantity for each category and is drawn as a bar from the baseline to the appropriate level on the y-axis.

A bar chart can be created by calling the bar() function and passing the category names for the x-axis and the quantities for the y-axis.

```
# create bar chart
pyplot.bar(x, y)
```

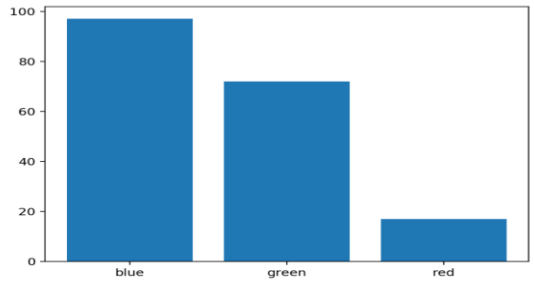
Bar charts can be useful for comparing multiple point quantities or estimations.

The example below creates a dataset with three categories, each defined with a string label. A single random integer value is drawn for the quantity in each category.

```
# example of a bar chart
from random import seed
from random import randint
from matplotlib import pyplot
# seed the random number generator
seed(1)
# names for categories
x = ['red', 'green', 'blue']
# quantities for each category
y = [randint(0, 100), randint(0, 100),
randint(0, 100)]
# create bar chart
pyplot.bar(x, y)
# show line plot
```

```
pyplot.show()
```

Running the example creates the bar chart showing the category labels on the x-axis and the quantities on the y-axis.



Example of a Bar Chart

1.5 Histogram Plot

A histogram plot is generally used to summarize the distribution of a data sample.

The x-axis represents discrete bins or intervals for the observations. For example observations with values between 1 and 10 may be split into five bins, the values [1,2] would be allocated to the first bin, [3,4] would be allocated to the second bin, and so on.

The y-axis represents the frequency or count of the number of observations in the dataset that belong to each bin.

Essentially, a data sample is transformed into a bar chart where each category on the x-axis represents an interval of observation values.

A histogram plot can be created by calling the hist() function and passing in a list or array that represents the data sample.

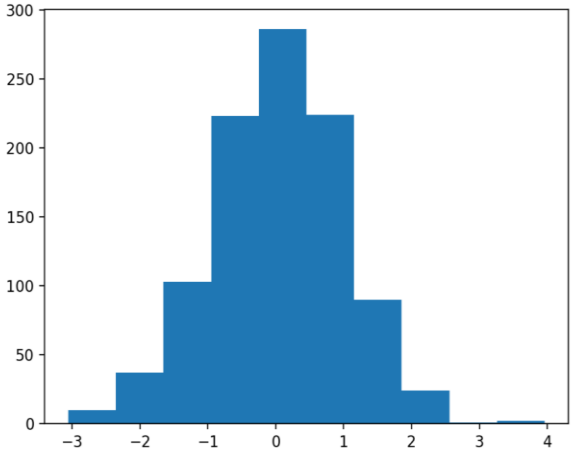
```
# create histogram plot
pyplot.hist(x)
```

Histograms are valuable for summarizing the distribution of data samples.

The example below creates a dataset of 1,000 random numbers drawn from a standard Gaussian distribution, then plots the dataset as a histogram.

```
# example of a histogram plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# random numbers drawn from a Gaussian distribution
x = randn(1000)
# create histogram plot
pyplot.hist(x)
# show line plot
pyplot.show()
```

Running the example, we can see that the shape of the bars shows the bell-shaped curve of the Gaussian distribution. We can see that the function automatically chose the number of bins, in this case splitting the values into groups by integer value.



Example of a Histogram Plot

1.6 Box and Whisker Plot

A box and whisker plot, or box plot for short, is generally used to summarize the distribution of a data sample.

The x-axis is used to represent the data sample, where multiple box plots can be drawn side by side on the x-axis if desired.

The y-axis represents the observation values. A box is drawn to summarize the middle 50% of the dataset starting at the observation at the 25th percentile and ending at the 75th percentile. This is called the interquartile range, or IQR. The median, or 50th percentile, is drawn with a line.
Lines called whiskers are drawn extending from both ends of the box calculated as (1.5 x IQR) to demonstrate the expected range of sensible values in the distribution. Observations outside the whiskers might be outliers and are drawn with small circles.

Box plots can be drawn by calling the boxplot() function passing in the data sample as an array or list.

```
# create box and whisker plot
pyplot.boxplot(x)
```

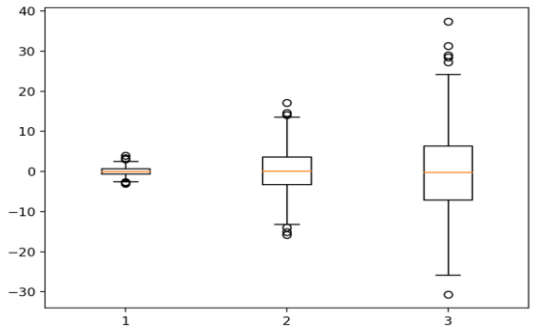
Boxplots are useful to summarize the distribution of a data sample as an alternative to the histogram. They can help to quickly get an idea of the range of common and sensible values in the box and in the whisker respectively. Because we are not looking at the shape of the distribution explicitly, this method is often used when the data has an unknown or unusual distribution, such as non-Gaussian.

The example below creates three boxplots in one chart, each summarizing a data sample drawn from a slightly different Gaussian distribution. Each data sample is created as an array and all three data sample arrays are added to a list that is padded to the plotting function.

```
# example of a box and whisker plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# random numbers drawn from a Gaussian distribution
x = [randn(1000), 5 * randn(1000), 10 * randn(1000)]
# create box and whisker plot
pyplot.boxplot(x)
# show line plot
pyplot.show()
```

Running the example creates a chart showing the three box and whisker plots. We can see that the same scale is used on the y-axis for each, making the first plot look squashed and the last plot look spread out.

In this case, we can see the black box for the middle 50% of the data, the orange line for the median, the lines for the whiskers summarizing the range of sensible data, and finally dots for the possible outliers.



Example of a Box and Whisker Plot

1.6 Scatter Plot

A scatter plot (or 'scatterplot') is generally used to summarize the relationship between two paired data samples.

Paired data samples means that two measures were recorded for a given observation, such as the weight and height of a person.

The x-axis represents observation values for the first sample, and the y-axis represents the observation values for the second sample. Each point on the plot represents a single observation.

Scatter plots can be created by calling the scatter() function and passing the two data sample arrays.

```
# create scatter plot
pyplot.scatter(x, y)
```

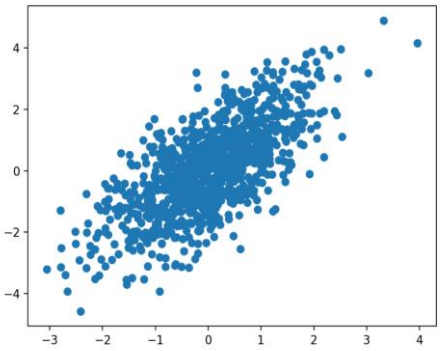
Scatter plots are useful for showing the association or correlation between two variables. A correlation can be quantified, such as a line of best fit, that too can be drawn as a line plot on the same chart, making the relationship clearer.

A dataset may have more than two measures (variables or columns) for a given observation. A scatter plot matrix is a cart containing scatter plots for each pair of variables in a dataset with more than two variables.

The example below creates two data samples that are related. The first is a sample of random numbers drawn from a standard Gaussian. The second is dependent upon the first by adding a second random Gaussian value to the value of the first measure.

```
# example of a scatter plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# first variable
x = 20 * randn(1000) + 100
# second variable
y = x + (10 * randn(1000) + 50)
# create scatter plot
pyplot.scatter(x, y)
# show line plot
pyplot.show()
```

Running the example creates the scatter plot, showing the positive relationship between the two variables.



Example of a Scatter Plot

CONCLUSION

Data visualization is the process of representing data in a graphical or pictorial way in a clear and effective manner. It has emerged as a powerful and widely applicable tool for analyzing and interpreting large and complex data. It has become a quick, easy means of conveying concepts in a universal format. It must communicate complex ideas with clarity, accuracy, and efficiency. These benefits have allowed data visualization to be useful in many fields of study.

REFERENCES:

1. https://www.simplilearn.com/data-visualization-in-python-using-atplotlib-tutorial
2. https://www.edureka.co/blog/needs-and-benefits-of-data-visualization/
3. https://towardsdatascience.com/introduction-to-data-visualization-in-python-89a54c97fbedhttps://www.cs.uic.edu/~kzhao/Papers/00_course_Data_visualization.pdf

Usha Manjari Sikharam received a degree in M.Tech Computer Science and Engineering from university of JNTUH and currently working as Assistant Professor in Sreenidhi Institute of Science and Technology. Research interest includes parallel Computing, Big Data, Python programming, R Programming and Data Visualization.