

INDUSTRIAL TRAINING DAILY DIARY

DAY 14

10 July, 2025

Topic : Introduction to Numpy and MatPlot library

Objectives:

- To understand the basics and importance of NumPy in data analysis and scientific computing.
 - To learn about NumPy arrays and their advantages over Python lists.
 - To perform basic operations using NumPy such as array creation, reshaping, indexing, and slicing.
 - To understand the need for data visualization in data science and analytics.
 - To get introduced to the MatPlot library and its role in creating visualizations.
 - To learn how to plot basic charts using MatPlot, such as line graphs, bar charts, histograms, and scatter plots.
 - To implement a small exercise demonstrating NumPy operations along with graphical output using MatPlot.
-

NumPy:

NumPy (Numerical Python) is an open source Python library that's widely used in science and engineering. The NumPy library contains multidimensional array data structures, such as the homogeneous, N-dimensional `ndarray`, and a large library of functions that operate efficiently on these data structures.

How to import NumPy

After [installing NumPy](#), it may be imported into Python code like:

```
import numpy as np
```

\

Why use NumPy?

Python lists are excellent, general-purpose containers. They can be “heterogeneous”, meaning that they can contain elements of a variety of types, and they are quite fast when used to perform individual operations on a handful of elements.

Depending on the characteristics of the data and the types of operations that need to be performed, other containers may be more appropriate; by exploiting these characteristics, we can improve speed, reduce memory consumption, and offer a high-level syntax for performing a variety of common processing tasks. NumPy shines when there are large quantities of “homogeneous” (same-type) data to be processed on the CPU.

What is an “array”?

In computer programming, an array is a structure for storing and retrieving data. We often talk about an array as if it were a grid in space, with each cell storing one element of the data. For instance, if each element of the data were a number, we might visualize a “one-dimensional” array like a list:

1	5	2	0
---	---	---	---

A two-dimensional array would be like a table:

1	5	2	0
8	3	6	1
1	7	2	9

A three-dimensional array would be like a set of tables, perhaps stacked as though they were printed on separate pages. In NumPy, this idea is generalized to an arbitrary number of dimensions, and so the fundamental array class is called `ndarray`: it represents an “N-dimensional array”.

Most NumPy arrays have some restrictions. For instance:

- All elements of the array must be of the same type of data.
- Once created, the total size of the array can’t change.
- The shape must be “rectangular”, not “jagged”; e.g., each row of a two-dimensional array must have the same number of columns.

When these conditions are met, NumPy exploits these characteristics to make the array faster, more memory efficient, and more convenient to use than less restrictive data structures.

For the remainder of this document, we will use the word “array” to refer to an instance of `ndarray`.

Array attributes

This section covers the `ndim`, `shape`, `size`, and `dtype` attributes of an array.

The number of dimensions of an array is contained in the `ndim` attribute.

```
>>> a.ndim
2
```

The shape of an array is a tuple of non-negative integers that specify the number of elements along each dimension.

```
>>> a.shape
(3, 4)
>>> len(a.shape) == a.ndim
True
```

The fixed, total number of elements in array is contained in the `size` attribute.

```
>>> a.size
12
>>> import math
>>> a.size == math.prod(a.shape)
True
```

Arrays are typically “homogeneous”, meaning that they contain elements of only one “data type”. The data type is recorded in the `dtype` attribute.

```
>>> a.dtype
dtype('int64') # "int" for integer, "64" for 64-bit
```

How to create a basic array

This section

covers `np.zeros()`, `np.ones()`, `np.empty()`, `np.arange()`, `np.linspace()`.

Besides creating an array from a sequence of elements, you can easily create an array filled with 0's:

```
>>> np.zeros(2)
array([0., 0.])
```

Or an array filled with 1's:

```
>>> np.ones(2)
array([1., 1.])
```

Specifying your data type

While the default data type is floating point (`np.float64`), you can explicitly specify which data type you want using the `dtype` keyword.

```
>>> x = np.ones(2, dtype=np.int64)
>>> x
array([1, 1])
```

How do you know the shape and size of an array?

This section covers `ndarray.ndim`, `ndarray.size`, `ndarray.shape`

`ndarray.ndim` will tell you the number of axes, or dimensions, of the array.

`ndarray.size` will tell you the total number of elements of the array. This is the product of the elements of the array's shape.

`ndarray.shape` will display a tuple of integers that indicate the number of elements stored along each dimension of the array. If, for example, you have a 2-D array with 2 rows and 3 columns, the shape of your array is `(2, 3)`.

For example, if you create this array:

```
>>> array_example = np.array([[[0, 1, 2, 3],  
...                             [4, 5, 6, 7]],  
...                           [[0, 1, 2, 3],  
...                             [4, 5, 6, 7]],  
...                           [[0, 1, 2, 3],  
...                             [4, 5, 6, 7]])
```

To find the number of dimensions of the array, run:

```
>>> array_example.ndim  
3
```

To find the total number of elements in the array, run:

```
>>> array_example.size  
24
```

Matplotlib Library in Python

[Data visualization](#) helps you understand complex datasets. The Matplotlib library in Python is a key tool for creating plots, and this guide walks you through installation and basic plotting.

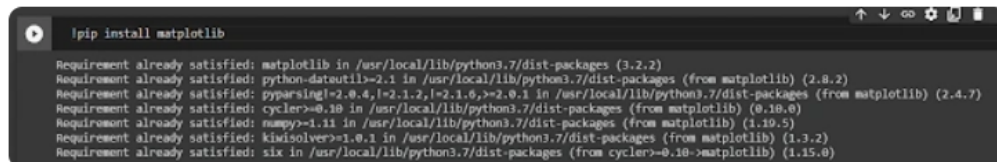
Matplotlib is a popular plotting library in Python used for creating high-quality visualizations and graphs. It offers various tools to generate diverse plots, facilitating data analysis, exploration, and presentation. Matplotlib is flexible, supporting multiple plot types and customization options, making it valuable for scientific research, data analysis, and visual communication. It can create different types of visualization reports like line plots, scatter plots, histograms, bar charts, pie charts, box plots, and many more different plots. This library also supports 3-dimensional plotting.

Installation of Matplotlib

Let's check how to set up the **Matplotlib** in Google Colab. Colab Notebooks are similar to Jupyter Notebooks except for running on the cloud. It is also

connected to our Google Drive, making it much easier to access our Colab notebooks anytime, anywhere, and on any system. You can install [Matplotlib](#) by using the **PIP** command.

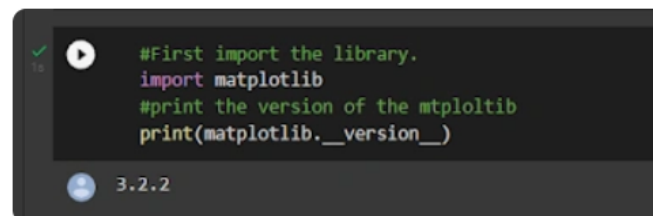
```
!pip install matplotlib
```

[Copy Code](#)

```
!pip install matplotlib
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (3.2.2)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (0.10.0)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.19.5)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cycler>=0.10->matplotlib) (1.15.0)
```

To verify the installation, you would have to write the following code chunk:

```
import matplotlib
print(matplotlib.__version__)
```

[Copy Code](#)

```
#First import the library.
import matplotlib
#print the version of the matplotlib
print(matplotlib.__version__)

3.2.2
```

Types of Plots in Matplotlib

Now that you know what Matplotlib function is and how you can install it in your system, let's discuss different kinds of plots that you can draw to analyze your data or present your findings. Also, you can go to the article to master in matplotlib

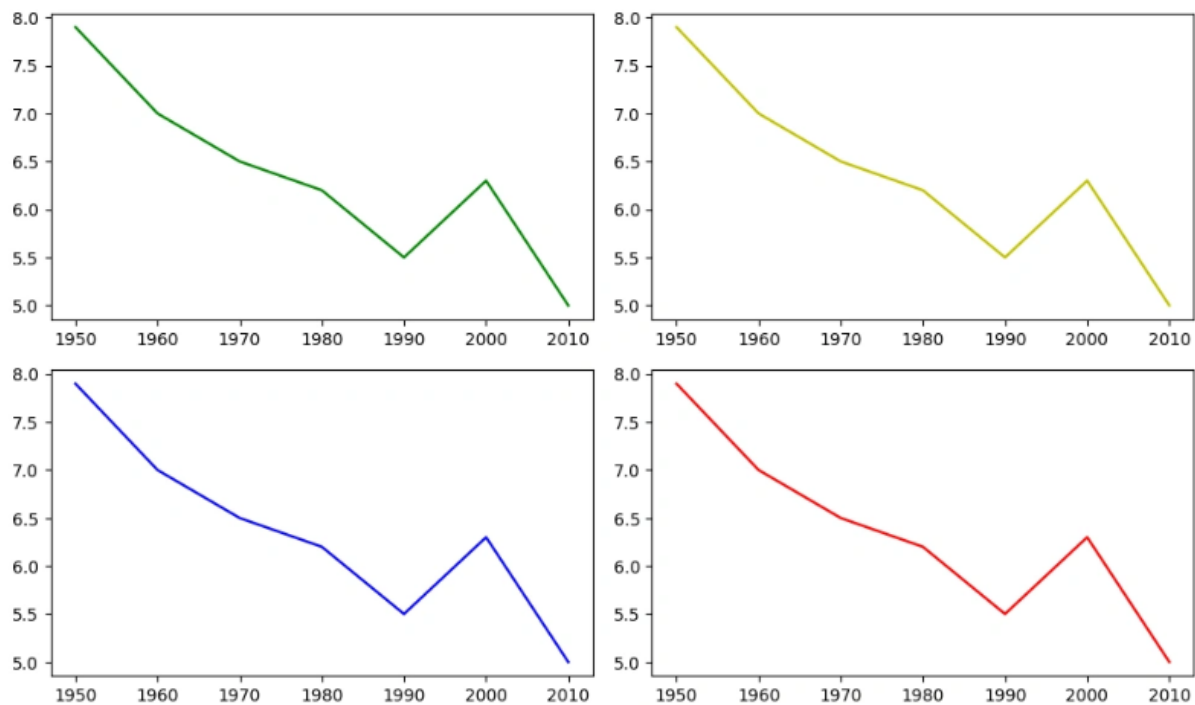
Sub Plots

Subplots() is a Matplotlib function that displays multiple plots in one figure. It takes various arguments such as many rows, columns, or sharex, sharey axis.

Code:

```
# First create a grid of plots
fig, ax = plt.subplots(2,2,figsize=(10,6)) #this will create the subplots with 2 rows and
#and the second argument is size of the plot
# Lets plot all the figures
ax[0][0].plot(x1, np.sin(x1), 'g') #row=0,col=0
ax[0][1].plot(x1, np.cos(x1), 'y') #row=0,col=1

ax[1][0].plot(x1, np.sin(x1), 'b') #row=1,col=0
ax[1][1].plot(x1, np.cos(x1), 'red') #row=1,col=1
plt.tight_layout()
#show the plots
plt.show()
```



Now, let's check the different categories of plots that Matplotlib provides.

- Line plot
- Histogram
- Bar Chart
- Scatter plot

- Pie charts
- Boxplot

Most of the time, we have to work with Pyplot as an interface for **Matplotlib**. So, we import Pyplot like this:

```
import matplotlib.pyplot
```

Line Plots

A line plot shows the relationship between the x and y-axis.

```
this line will create array of numbers between 1 to 10 of length 100
```

[Copy Code](#)

```
np.linspace(start,stop,num)
```

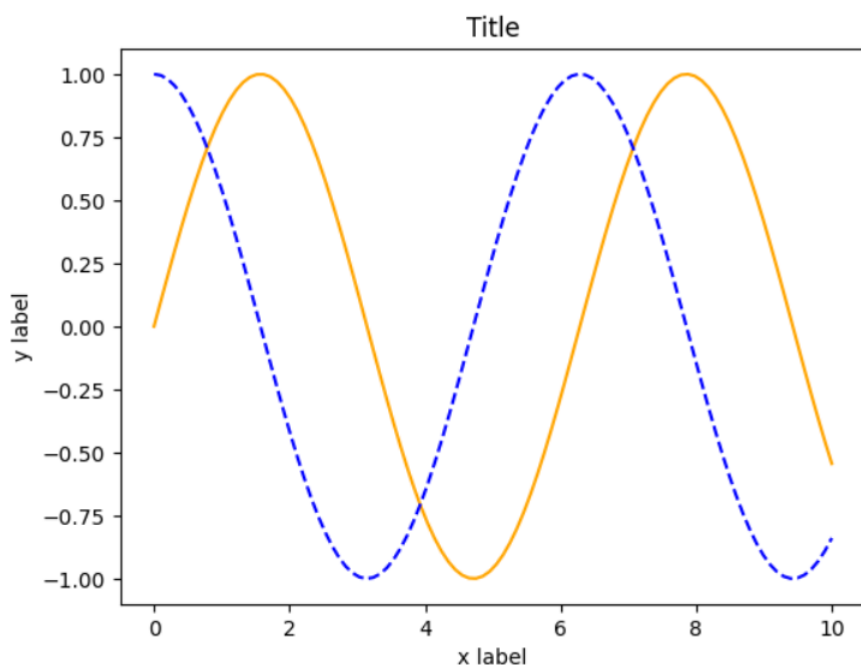
```
x1 = np.linspace(0, 10, 100) #line plot  
plt.plot(x1, np.sin(x1), '-',color='orange')  
plt.plot(x1, np.cos(x1), '--',color='b')
```

```
give the name of the x and y axis
```

```
plt.xlabel('x label')  
plt.ylabel('y label')
```

```
also give the title of the plot
```

```
plt.title("Title")  
plt.show()
```

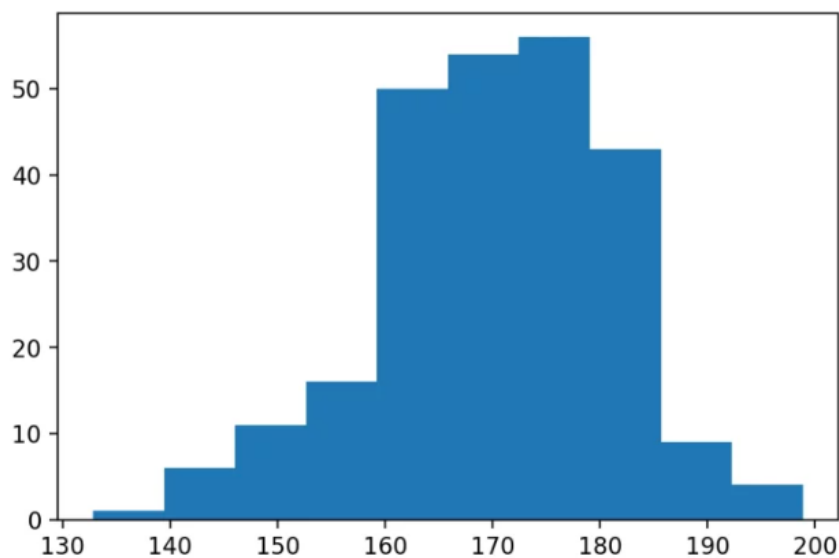


Histogram

The most common graph for displaying frequency distributions is a histogram. To create a histogram, the first step is to create a bin of ranges, then distribute the whole range of values into a series of intervals and count the value that will fall in the given interval. We can use `plt.hist()` function plots the histograms, taking various arguments like data, bins, color, etc.

Code:

```
#draw random samples from random distributions.  
x = np.random.normal(170, 10, 250)  
#plot histograms  
plt.hist(x) plt.show()
```

[Copy Code](#)

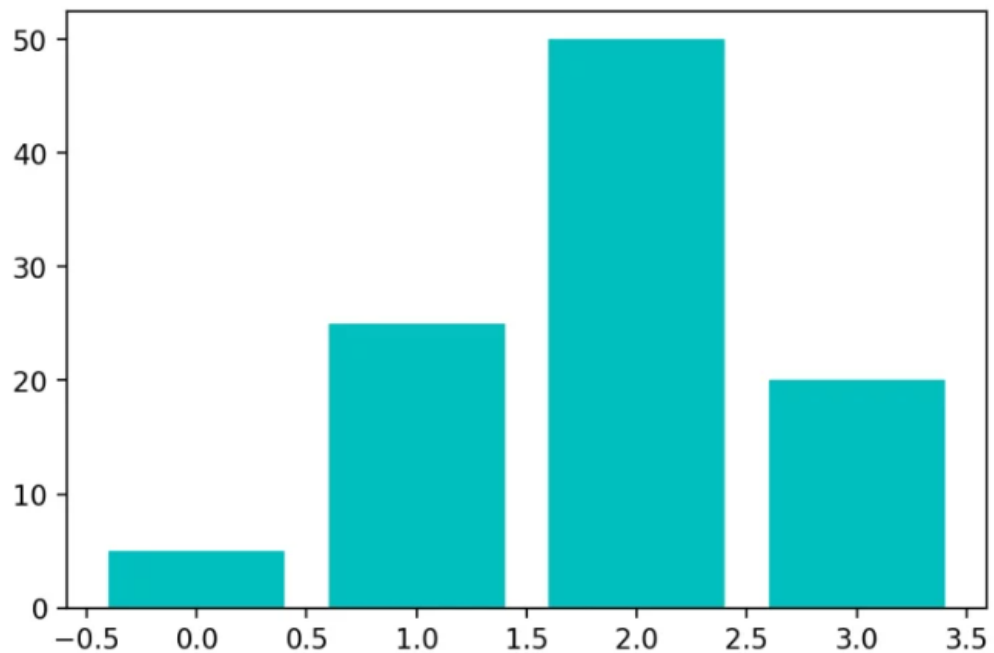
Bar Plot

Mainly, the barplot shows the relationship between the numeric and categoric values. In a bar chart, we have one axis representing a particular category of the columns and another axis representing the values or counts of the specific category. Barcharts can be plotted both vertically and horizontally using the following line of code:

```
plt.bar(x,height,width,bottom,align)
```

Code:

```
#define array  
data= [5. , 25. , 50. , 20.]  
plt.bar(range(len(data)), data,color='c')  
plt.show()
```

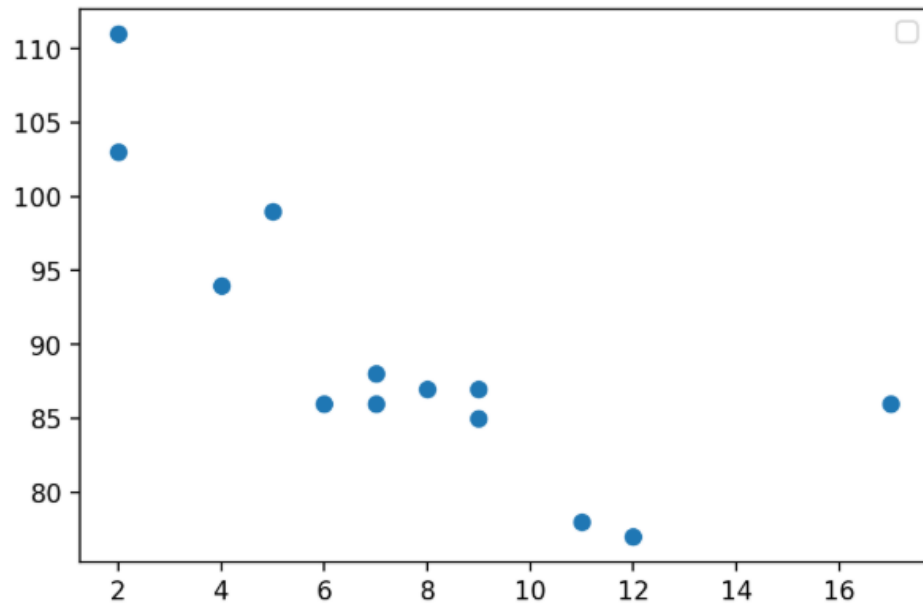


Scatter Plot

Scatter plots show the relationships between variables by using dots to represent the connection between two numeric variables.

The **scatter()** method in the **Matplotlib** [library](#) is used for plotting.

```
#create the x and y axis coordinates
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.legend()
plt.show()
```



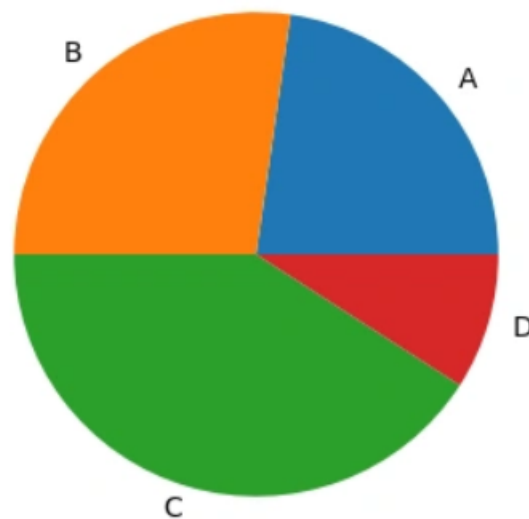
Pie Chart

A pie chart (or circular chart) is used to show the percentage of the whole.

Hence, it is used to compare the individual categories with the whole. **Pie()** will take the different parameters such as:

Code:

```
#define the figure size
plt.figure(figsize=(7,7))
x = [25,30,45,10]
#labels of the pie chart
labels = ['A','B','C','D']
plt.pie(x, labels=labels)
plt.show()
```

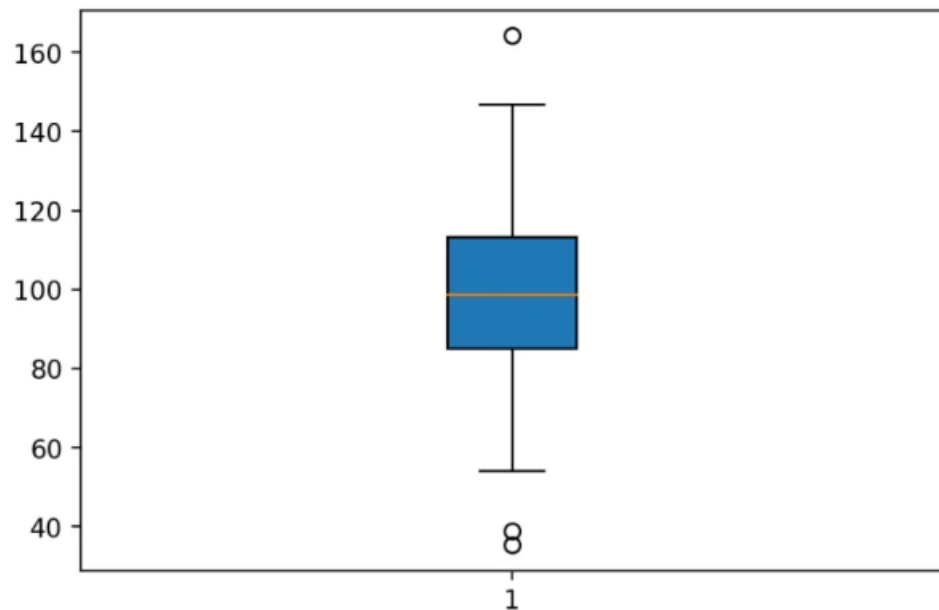


Box Plot

A Box plot in Python Matplotlib showcases the dataset's summary, encompassing all numeric values. It highlights the minimum, first quartile, median, third quartile, and maximum. The median lies between the first and third quartiles. On the x-axis, you'll find the data values, while the y-coordinates represent the frequency distribution.

Code:

```
#create the random values by using numpy
values= np.random.normal(100, 20, 300)
#creating the plot by boxplot() function which is available in matplotlib
plt.boxplot(values,patch_artist=True,vert=True)
plt.show()
```



Area Chart

An area chart or plot visualizes quantitative data graphically based on the line plot. **fill_between()** function is used to plot the area chart.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
y = [2, 7, 14, 17, 20, 27, 30, 38, 25, 18, 6, 1]
#plot the line of the given data
plt.plot(np.arange(12),y, color="blue", alpha=0.6, linewidth=2)
#decorate thw plot by giving the labels
plt.xlabel('Month', size=12)
plt.ylabel('Turnover(Cr.)', size=12) #set y axis start with zero
plt.ylim(bottom=0)
plt.show()
```

