Contents

[Recap of last class – 2](#_Toc95703804)

[EDA, Matplotlib and Seaborn – An introduction 2](#_Toc95703805)

[Seaborn and Matplotlib - 3](#_Toc95703806)

[What is the significance of OOPS? 3](#_Toc95703807)

[Scatter Plot & Basic Pyplot functions - 3](#_Toc95703808)

[Magic function %matplotlib inline – 4](#_Toc95703809)

[Plotting the Scatter chart using plt.scatter() - 4](#_Toc95703810)

[Using Shift+tab – 5](#_Toc95703811)

[Changing marker and colour – 6](#_Toc95703812)

[Basic Pyplot Functions – 7](#_Toc95703813)

[Line Chart – 9](#_Toc95703814)

[Bar Chart – 10](#_Toc95703815)

[Variations in Bar chart | Stacked Bar Chart – 13](#_Toc95703816)

[Variations in Bar chart | Dodged Bar Chart – 13](#_Toc95703817)

[Pie Chart – 15](#_Toc95703818)

[Variations: 15](#_Toc95703819)

[Hiding extra objects information displayed above chart : 16](#_Toc95703820)

[Histogram – 16](#_Toc95703821)

[Boxplot – 17](#_Toc95703822)

[Multiple boxplots in same figure – 18](#_Toc95703823)

[Heatmap – 18](#_Toc95703824)

[Subplots – 20](#_Toc95703825)

[Axes Object – 21](#_Toc95703826)

[Axes Object with Subplots: 23](#_Toc95703827)

[MCQ- 25](#_Toc95703828)

# Recap of last class –

* Visualisations are helpful to derive strong insights from data. The applications are in business decision making, consulting, EDA, Machine learning etc.
* We discussed some basic rules for a good visualisation such as –
  + Making sure it is exhaustive with all the labels and appropriate title
  + It is suitable to the intended audience
  + The chart type is suitable to data elements used
  + Colors are not flashy or have unintended meanings
  + Data-Ink ratio should be high
* We learnt some basic OOPS concepts and 4 pillars of OOPS-
  + Class: A blueprint for object
  + Object: An instance of class
  + Method: A function of a class
  + 4 Pillars of OOPS-
    - Inheritance: Ability to create child classes out of a class (Parent Class). The child class have access to methods of its Parent class
    - Encapsulation: Ability to define private variables for enhanced layer of security
    - Polymorphism: Ability to use same name with different meanings. Example – Same method name of different Class/Objects will behave differently
    - Abstraction: Ability to hide the inner mechanism of something and only dealing with the input and output. Analogy – Driving a car without knowing how engine and all instruments of a car work
* We learnt Python syntaxes of all these concepts and practiced some problems

# EDA, Matplotlib and Seaborn – An introduction

EDA stands for Exploratory data analysis. EDA is extremely important component in Data Science/ Data Analytics workspace as in all project, almost 60-70% of the time is consumed in EDA process.

EDA consists of following components –

1. **Data Wrangling –** Making sure the data is in good shape for further consumption (Already covered last week)
2. **Analysis of variables –** This is of 3 types:
   1. **Univariate Analysis:** Analysis involving 1 variable. Example – Histogram of “books sold in a day” for a 1-year data , boxplot, bar chart using 1 variable etc.
   2. **Bivariate Analysis:** Analysis involving 2 variables. Example – A scatter plot of Temperature Vs Altitude in a country, dodged bar chart, line chart
   3. **Multivariate Analysis –** A heat map showing correlation among various quantitative variables in a dataset

While we will have a dedicate class this week for EDA where we’ll discuss a dataset and do live EDA, for now it is easy to understand that visualizations are integral part of EDA. If one is able to master visualisations, a good EDA becomes easier.

Thus, in next 2 classes, we’ll simply learn about how we can build different charts in Python. We will not go into the technical details and applications of charts since that’s already been covered under statistics component of this course, rather our focus will be on to the python implementation of those charts.

In Python, we have 2 main libraries for building charts –

1. Matplotlib
2. Seaborn

## Seaborn and Matplotlib -

Seaborn is built on top of Matplotlib. This is a really great library which gives us stunning visuals with minimal effort, and students will be using this 80-90% of the time.

Although we will start our earning journey from Matplotlib since it is the backbone of Seaborn. It will give students a great platform to build upon their expertise in Seaborn.

## What is the significance of OOPS?

We learnt so much about OOPs yesterday but today we will simply use the Pyplot module and use simple functions to do all the jobs. One might wonder why did we learn such complex concepts when it is not needed and we can simply use Pyplot to build charts ?

The answer lies in the fact that we are aiming for deep expertise in this course, and knowing the backend workings of things helps a lot when we get stuck on some error, or if we want to try and do something additional than what Pyplot has to offer.

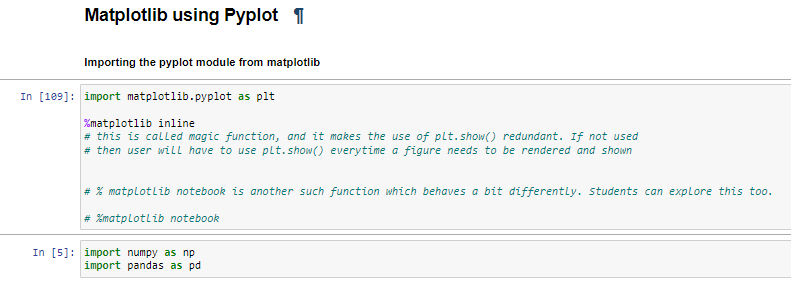
We will touch base upon the OOP aspect of Pyplot/Matplotlib too in the end of this class for 10-15 minutes, but that’s just to showcase how much work is being taken care of in the backend by Pyplot.

# Scatter Plot & Basic Pyplot functions -

Scatter Plot, as we already know, are a way to visually demonstrate 2 quantitative variables on a 2D plane.

The function to generate this plot is scatter () function of Pyplot.

But before we can use this function, we’ll need to import Pyplot module (generally imported as plt ) as shown below –

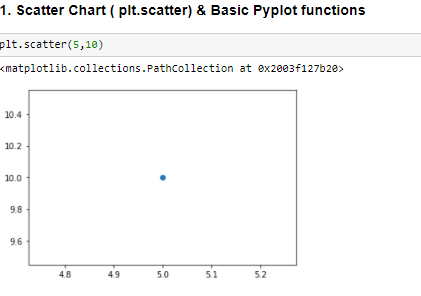


## Magic function %matplotlib inline –

This is called magic function, and it makes the use of plt.show() redundant. If not used, then user will have to use plt.show() every time a figure needs to be rendered and shown.

*%matplotlib notebook* is another such function which behaves a bit differently. Students should explore this too.

## Plotting the Scatter chart using plt.scatter() -



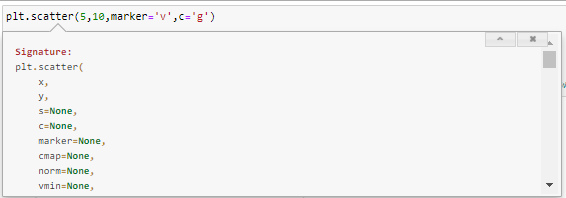
## Using Shift+tab –

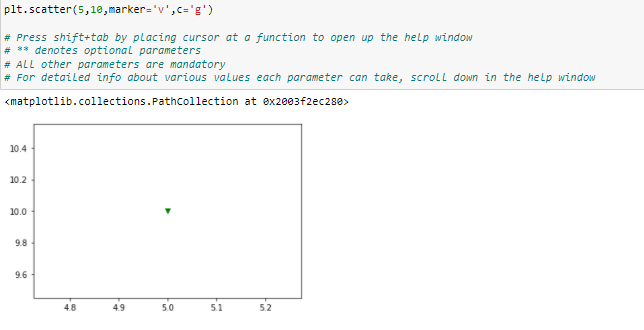
Students can place cursor on a function and press shift+tab (And then click on + sign on top right of help window) to get all the required details regarding function such as –

* Mandatory Parameters
* Optional Parameters
* Default values
* Different values a parameter can take and their values

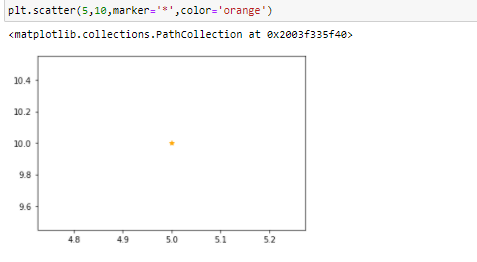
For example, in below screenshot we can see the same for Scatter function. We have given x as 5, y as 10, marker as v (Scroll down in this window in Jupyter to see that v stands for triangles, and color as green).

For all subsequent coding snippets in today’s lecture, we will not explain these parameter as students should make a habit of checking these on their own by using shift+tab .





## Changing marker and colour –



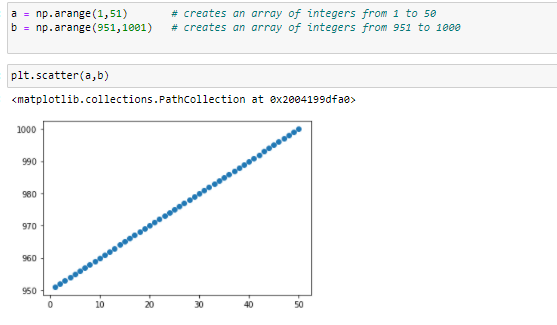
Scatter plot takes x & y arguments as data arrays in most cases.

Let’s see one such example where we have 2 data arrays using Numpy.

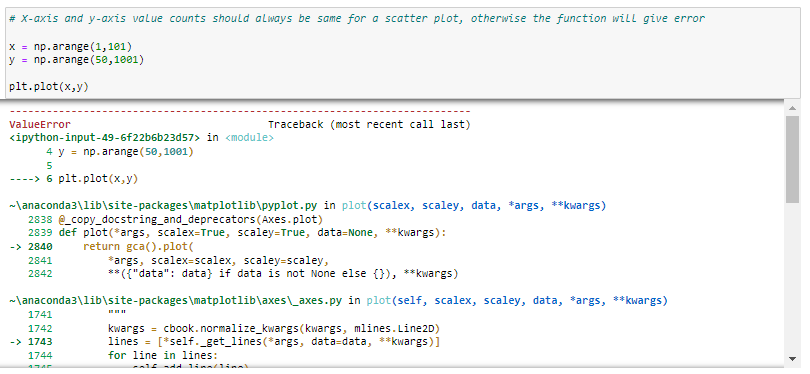
a contains integers from 1 50

b contains integer from 951 to 1000

Plotting a scatter for this data -

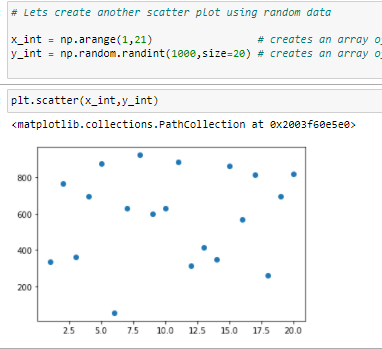


**X and Y data array should always have same counts otherwise scatter function will show error. Please take special care about this.**



## Basic Pyplot Functions –

Let’s consider below scatter plot –

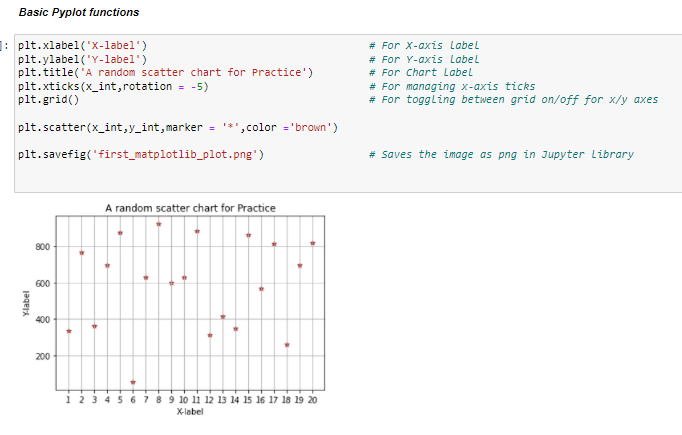


As we learnt in last class, there are following issues with this chart:

There are following issues in above plot-

1. No X-axis title
2. No Y-axis title
3. No chart title
4. X-axis ticks can be better represented if it doesn't include fractional values

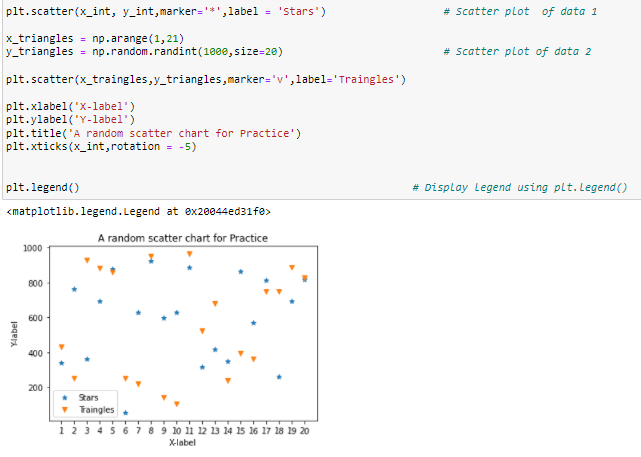
Let's resolve these issue with the help of some Pyplot functions



Functions used and their description are summarised below –

* Plt.xlabel(): To define X Axis label
* Plt.ylabel(): To define Y Axis label
* Plt.title(): To define chart title
* Plt.xticks(): To edit X axis ticks, rotate those
* Plt.grid(): Toggling background grid on
* Plt.scatter(): Plotting the scatter plot
* Plt.savefig(): Saving the figure in Jupyter directory
* Plt.legend(): To display legend

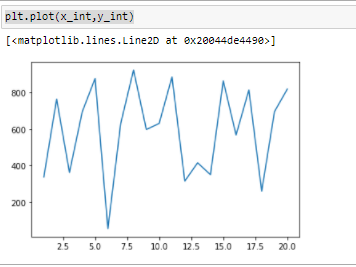
**Multiple data can be plotted on the same figure. The legend can also be used to categorize those:**



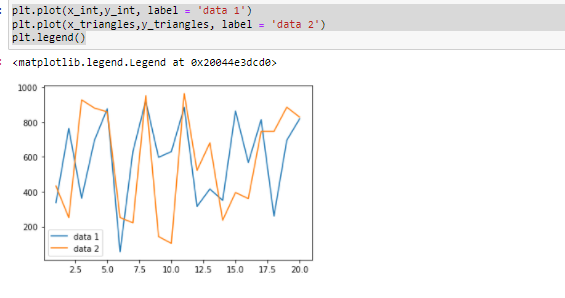
Thus, as long as we are in the same cell, all the plots will be overlayed on the same canvas.

# Line Chart –

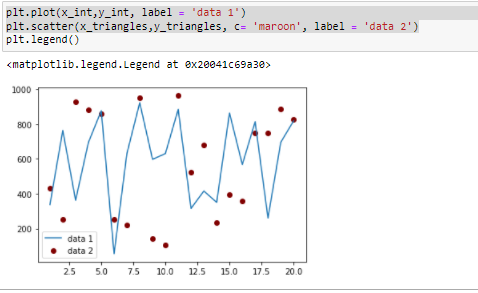
Plt.plot() is used to make a line chart.



Multiple line charts in same figure can be displayed as we saw in case of scatter plots –



A combination of different chart can also be made on same canvas/plot. A scatter+line works fine for example –



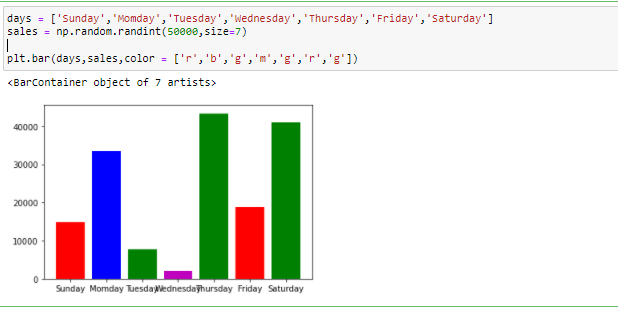
# Bar Chart –

Bar chart can be plotted by plt.bar() function.

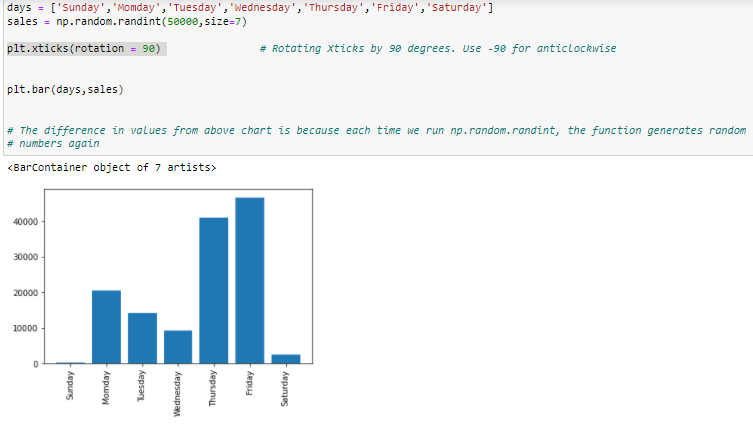
The data and labels need to be separately provided in this function.

We can see an example below where we are plotting bar chart of weekly sales of a product.

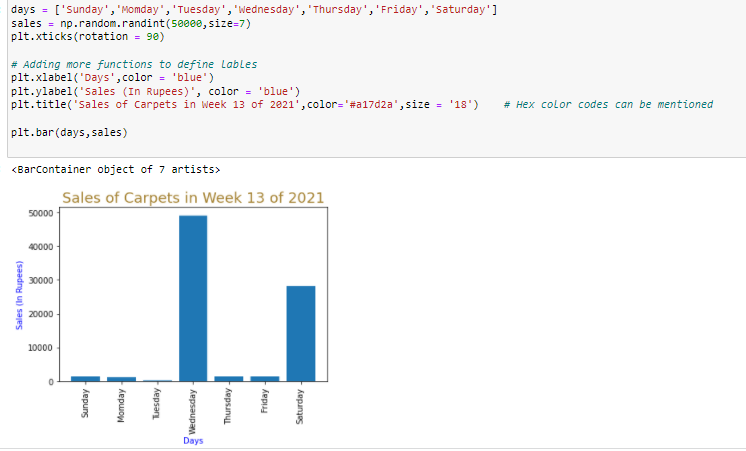
It can be seen that day names and sales data has been put into different arrays and provided to plt.bar() function -



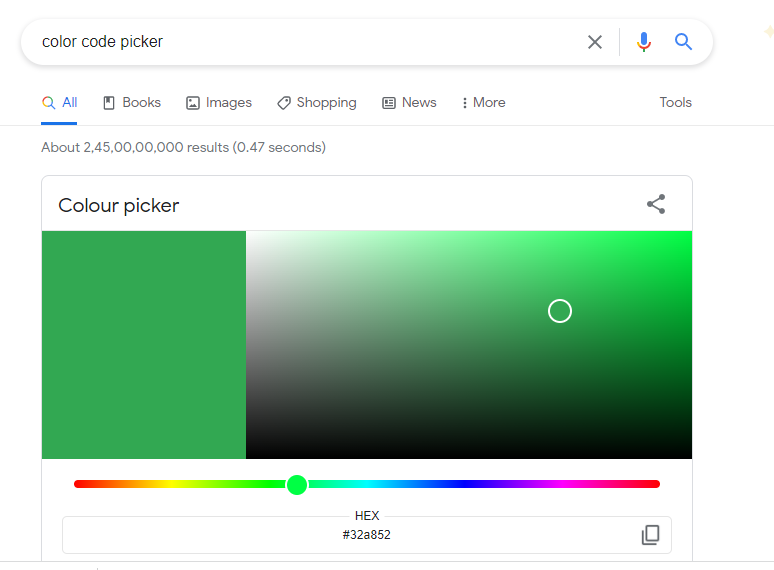
Problem in this chart – Category names are overlapping. It is a common problem with charts where categories are involved, and can be fixed by rotating the x-ticks by an angle as below –



Adding axis labels and chart-title by basic pyplot functions we’ve discussed before –

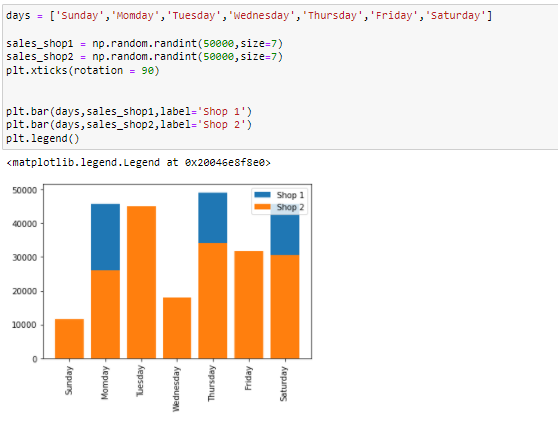


Hex code of any color can be found easily by Google search ‘Color Hex code’ –



## Variations in Bar chart | Stacked Bar Chart –

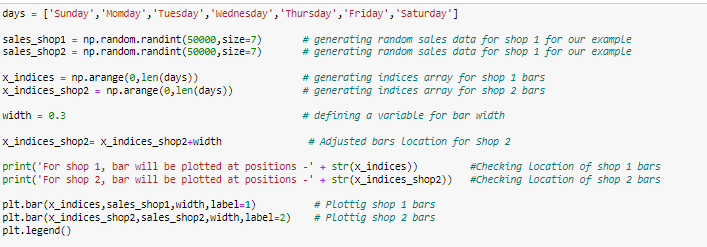
Suppose there are sales of 2 shops in a given week. If we are interested to see how total says were on each day, along with the split of each shop, a stacked bar chart can be used as below-

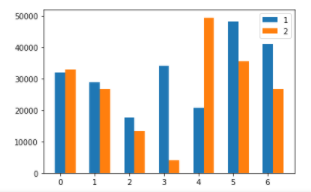


## Variations in Bar chart | Dodged Bar Chart –

But if we want to see the individual sales of shops, above chart will not be the desired output. We don't want the bar to be stacked.

This problem can be fixed as per below –





Explanation of above code –

By default, a second bar chart is plotted in same chart, the bars are plotted on same position, making it a stacked bar chart.

Hence we needed to adjust the position of second bar series to start where the first bar ends.

It would require following operations –

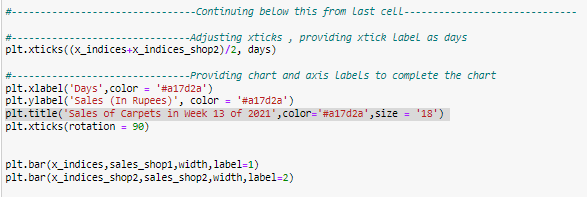
* Getting number of bars that will be plotted
* Have an array from 0 to number of bars so that we have an index for our 1st series bar
* Have an another indices array for our 2nd series – but this time the indices will have to shift by the width of bars of first series so that the bars don’t overlap.

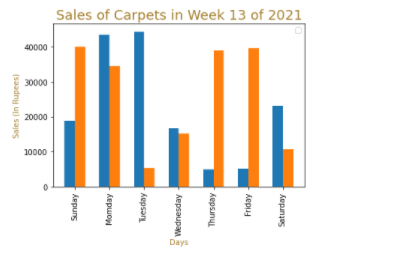
This is done by adding width of bars to first indices array

* We can plot the bar chart now. But there will be a small problem with x-ticks – By default the x-ticks will be centred on 1st bar series, we need to re-centre the xticks

Next operations to adjust xticks and xlabel are further performed in next screenshot below.

* This is done by shifting the xticks to mean of 2 indices. For example, for bar1, the xtick location can be (0+0.3)/2 instead of 0. (Assuming 0.3 is width)
* Xlabel are provided explicitly as days again. And rotated by 90 degrees to avoid overlap.

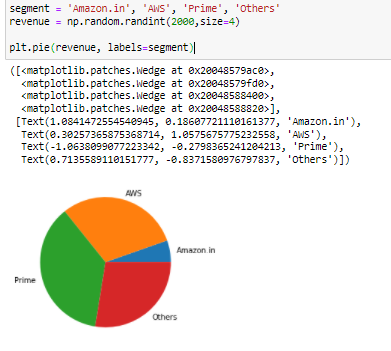




We’ve got our desired Dodged bar chart.

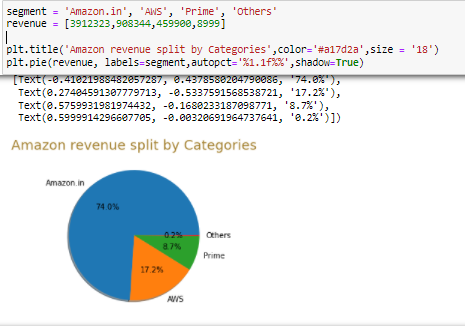
# Pie Chart –

Plt.pie() is used to generate pie charts. Let’s see an example of Amazon revenue by its product segments :



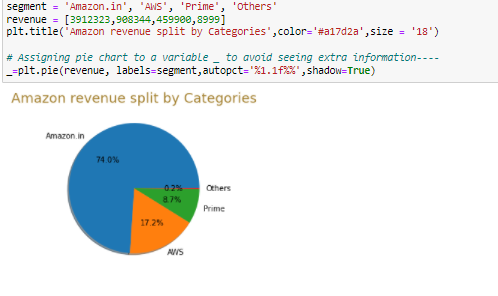
## Variations:

Explode parameter can make a segment split out from main chart colours can be explicitly specified using color parameter autopct is a really useful parameter which automatically calculated % of each segment and displays it shadow can be added using shadow parameter.



## Hiding extra objects information displayed above chart :

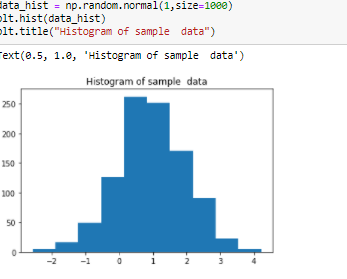
The object information can be hidden by assigning the chart to some variable as shown below –



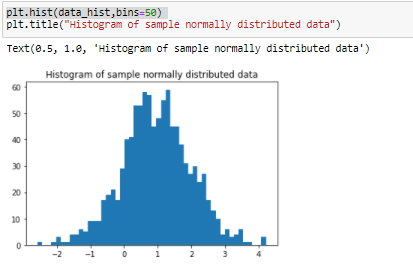
# Histogram –

Histograms show us information regarding the distribution of data. We have already discussed about this in statistics part of this course.

Plt.hist() is used to make histograms.



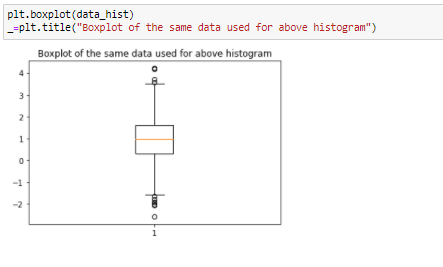
Bins size can be adjusted by bins parameter as shown –



# Boxplot –

Like histogram, boxplot has already been covered earlier in this course under statistics. We all know that boxplot and histograms represent similar information in different ways.

Plt.boxplot() is the function for boxplot.

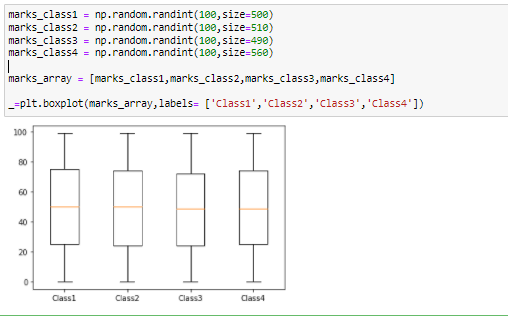


## Multiple boxplots in same figure –

A lot of times we would want to see the box plots for multiple quantitative variables in a data. We can also plot box plots for all the variables in same figure.

In below example, we plot boxplots of students' marks for 4 classes.

The marks here can be between 0 to 100 and each class has a strength close to 500



# Heatmap –

Heatmaps can be used to demonstrate data visually where 3 variables are used The 2 variables take up X and Y axis and third variable is represented by Intensity of color on the chart.

Heatmaps are applicable to very specific situations where there is a clear relationship between variables. It is easy to make a really bad heatmap with no meaning, please be careful about that.

Heatmaps are definitely a bad option where categorical variables are involved

Some good examples of situations where heatmaps can be used/have been used are -

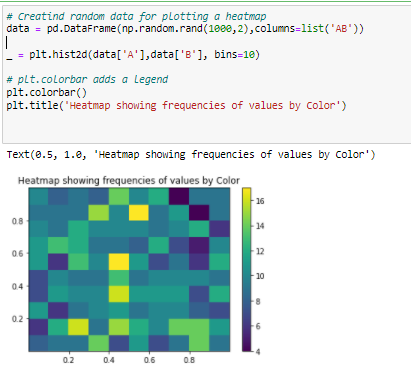
* Temperature(coloured) across longitude and latitude
* Probability of a missing plane(coloured) across longitude and latitude
* Correlation(Coloured) among different quantitative variables present in a data

Let’s make a sample heatmap below for a random data. We'll have 2 sets of 1000 random numbers all in the range between 0 to 1 (total 2000 numbers).

The color of a section in Heatmap will depict the frequency of numbers falling in that bracket while the range of 2 sets are represented on X and Y axis.

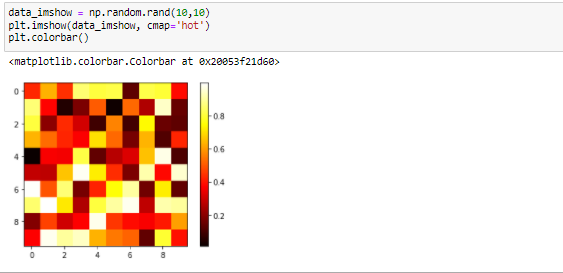
Let’s make a sample heatmap below for a random data. We'll have 2 sets of 1000 random numbers all in the range between 0 to 1 (total 2000 numbers). We’ll use ***hist2D()*** function here.

The color of a section in Heatmap will depict the frequency of numbers falling in that bracket while the range of 2 sets are represented on X and Y axis.



There is an another function to make a heatmap via Pyplot ***- imshow()*** Lets make a heatmap using imshow function.

The drawback here is that imshow doesn't work with Panda dataframes and adjusting the bins is a bit complicated. However, it can be used where data to be visualised is not huge.



# Subplots –

Subplots, as the name suggests, are multiple plots on the same canvas.

These are extremely useful to show multiple charts in same frame, especially if those are interlinked and a story can be told by the looking at a picture collectively.

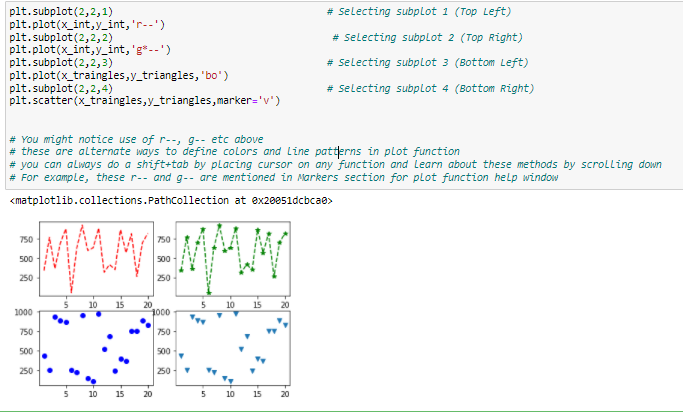
Subplot function takes in 3 parameters -

* 1st one denotes number of rows
* 2nd denotes number of columns
* 3rd one selects the particular subplot

For example, subplot (2,2,3) means that we are defining 2 rows and 2 columns of subplots, which means there will be total 2\*2 = 4 subplots on the canvas. 3 in last parameter means that we are selecting the 3rd subplot, and all the functions applied after this will work on 3rd subplot.

We can toggle between subplots by mentioning the correct subplot code like (2,2,1) or (2,2,2) or (2,2,3) or (2,2,4)

In this way we can make different charts on same canvas.



By the way, by default Pyplot assumes a canvas as (1,1,1) subplot. Which means a single plot.

# Axes Object –

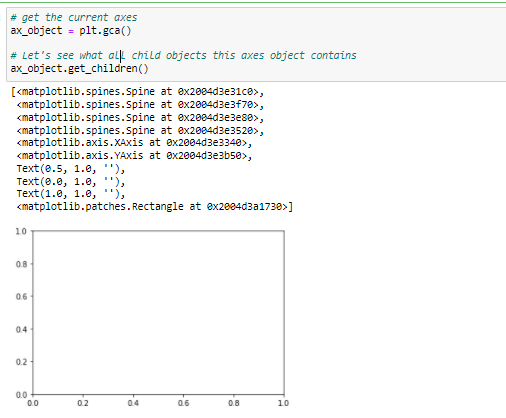
We had learnt about OOPS yesterday. You might wonder that we have not used any of that today, so what is the significance and why did we go through those concepts.

In reality, all the functions that we have used today are using OOPS at the backend. It’s just that the Pyplot scripting layer is so great at a high level that we do not even get a hint of how much work is going on at the backend.

In background, pyplot is maintaining track of multiple object a figure contains, and modifying different components as per our execution.

For example, a simple task of assigning the X-axis label involves checking if a figure exists or not (and creating a new one in case it doesn't), finding the right object which deals with axes, then finding the child object which keeps track of X-axis, then subsequently changing the value of label. In the end, it also ensures multiple different objects are modified accordingly (like size of label, font etc.) so that the figure is rendered properly without any error.

One of the key objects Pyplot interacts a lot with is Axes object. Let’s dive a bit into it and see what all different child object comes under axes object.

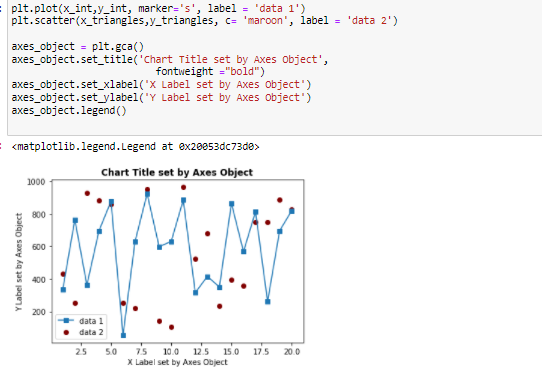


As we can see, it contains about 10 child components-

1. 4 spine line objects which keep track of area boundaries
2. 1 X Axis object
3. 1 Y Axis object
4. 3 Text objects
5. 1 rectangle canvas object

Although we have already learnt to use pyplot for most of our needs, but we can also perform all those functions directly with the help of Axes object too without the use of Pyplot.

Again, we are discussing this because you'll see a lot of discussion about these on forums like stackoverflow and its always good to be aware of all the different approaches.

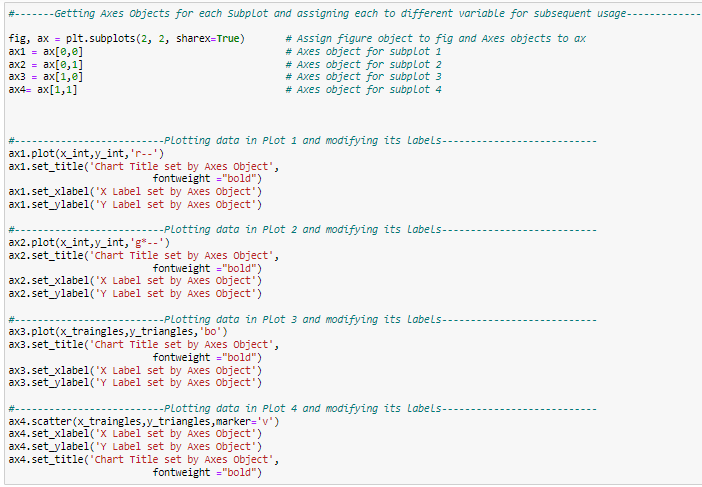


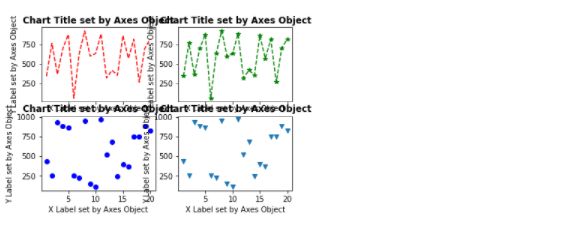
## Axes Object with Subplots:

In case of subplots, we can get separate axes objects for each subplot using syntax given below.

Each subplot then can be individually modified using the related axes object.

For 99% of the time, we will not need to use this. But knowing that such objects exists below Pyplot helps a lot in understanding and mastering Pyplot.





We can see that we have been able to modify the parameters using axes object.

But it is clearly visible that there is a lot of overlapping going on.

We are not going to fix it right now; maybe we can do that as part of class hands-on exercise.

But the purpose of this section today was to show how much work Pyplot is doing at the backend. We do get more options to tweak little things using axes object but it gets a bit harder since we'll have to manage more things manually if we don't use Pyplot.

Again, going ahead we’ll see that even Pyplot is redundant when compared to Seaborn module, which our topic for next class!

# MCQ-

Q1. Consider following code:

plt.bar(days, sales, color = ['r','b','g','m','g','r',m])

What will be the color of last bar?

1. Red
2. Black
3. Blue
4. Magenta

Q2. Which of the following is correct method for defining number of bins for histogram:

1. plt.hist(data\_hist,number\_of\_bins=50)
2. plt.hist(data\_hist, bins= [0,50])
3. plt.hist(data\_hist, bins=50)
4. plt.hist(data\_hist, binsize=50)

Q3. Which is the scripting layer for plotting visuals using Matplotlib:

1. plot
2. plotly
3. Plt
4. pyplot

Q4. Which of the following can be made using matplotlib:

1. Dodged bar chart
2. Scatter Chart
3. Heatmap
4. All of above

Q5. Consider following code:

plt.scatter(data1, data2, c = ‘black’, marker = "s")

What will be marker type in this chart :

1. Dot
2. Square
3. Circle
4. Triangle

Q6. Consider following code:

plt.scatter(data1, data2, c = ‘black’, marker = "v")

What will be marker type in this chart:

1. Dot
2. Square
3. Circle
4. Triangle

Q7. Which of the following chart cannot take more than 1 data input :

1. Pie Chart
2. Bar Chart
3. Stacked bar chart
4. Line Chart

Q8. In a pie chart, which function is used to automatically calculate and display percentages:

1. Percentage()
2. Autopercentage()
3. Autopct()
4. Colormap()

Q9. Which function is used to edit the x-axis label using Pyplot:

1. Plt.yplot()
2. Plt.x\_label()
3. Plt.xlabel()
4. Plt.xaxis()

Q10. Consider following code:

plt.plot(x1,y1, label = 'data 1')

plt.plot(x2,y2, label = 'data 2')

The output of this would be -

1. 2 different line plots
2. 1 plot with 2 different lines
3. 1 plot with 2 bars
4. 2 plots with 2 bars