Data is the new oil. It’s valuable, but if unrefined it cannot really be used. It has to be changed into gas, plastic, chemicals and so on to create a valuable entity that drives profitable activity; so must **data be broken down, analyzed** for it to have value.”
— Clive Humby, 2006.

- Essentially, all data are just a collection of observations represented by numbers and conveniently stored in some formats. There are many python packages that help us visualize various types of data in different styles.
- The most commonly used package is **Matplotlib**, which was started two decades ago as competant to MATLAB and has been under a constant development since then.
- There are other packages such as **Seaborn**,**Cartopy** and **plotnine** which are built on top of Matplotlib and customized for specifi domains.
- **Seaborn** is specifically built for statistical data visualizations that includes various plots such as Lin, bar chart,histograms and scatter plots.
- We use both **Seaborn** and **Matplotlib** and the choice is based on the type of plot and the ease of interface provided by the packages.(Of course, seaborn provides a simple interface as it is a high-level API 😃)

Data

- Well, let us import the oil first!.The data might be stored in different formats. Most of the commonly used formats are

1. csv
2. xlsx
3. hdf5
4. pkl (pickle).

- Once we know the format we can read and store them into the local variable for further processing.
- Since the emphasize is on **visualization**, we are going to use a moderate size datasets (either create one or get it from the public repository) and use those for different types of visualizations.
- As usual, let us first `import` all the necessary packages and do some global configurations.

```
import numpy as np
import pandas as pd

# visualization packages
```

```
import seaborn as sns
from matplotlib import pyplot as plt
```

```
np.random.seed(1729)
```

Line Plot

Let's start with plotting a simple function $f(x)$. We would like to visualize how the function looks like for various values of x.

$$f(x) = \sin(2\pi\omega x)$$

. Since it is function of x we fix $\omega = 1, 2$ and plot both of them in a single figure (graph sheet 😃).

```
PI = np.pi
w1 = 1
w2 = 2
x = np.linspace(0,1,100)
f_x1 = np.sin(2*PI*w1*x)
f_x2 = np.sin(2*PI*w2*x)
```

Ensure that the shape of `x` and `f_x1,f_x2` are of same length.

```
print(x.shape)
print(f_x1.shape)
print(f_x2.shape)
```

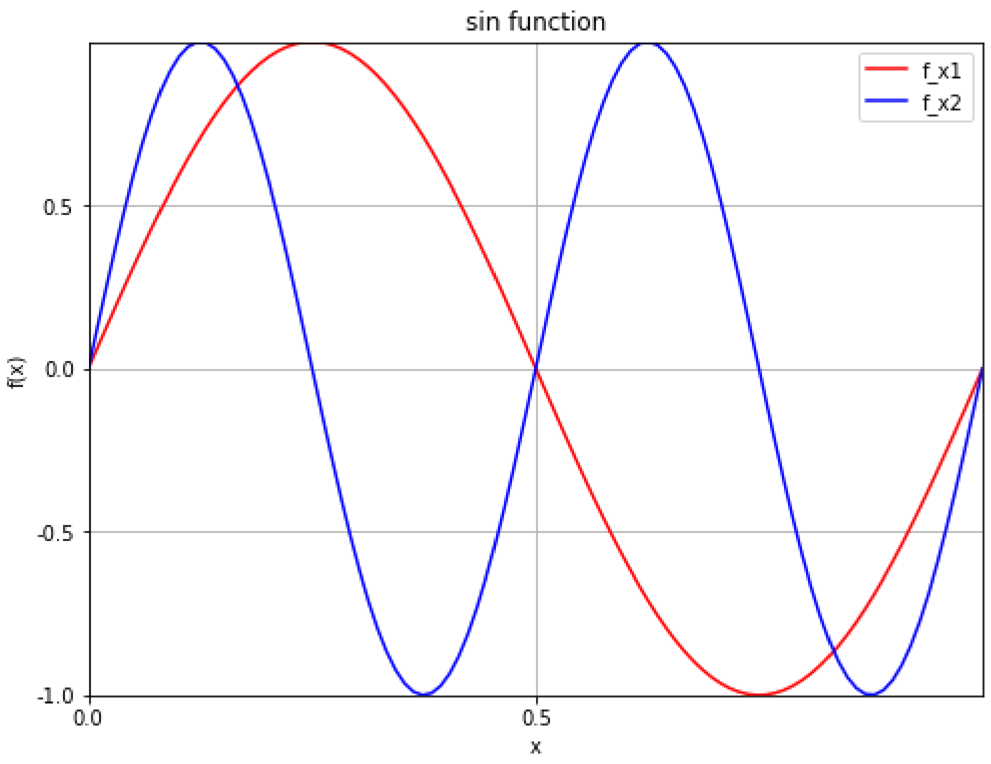
```
(100,)
(100,)
(100,)
```

matplotlib:

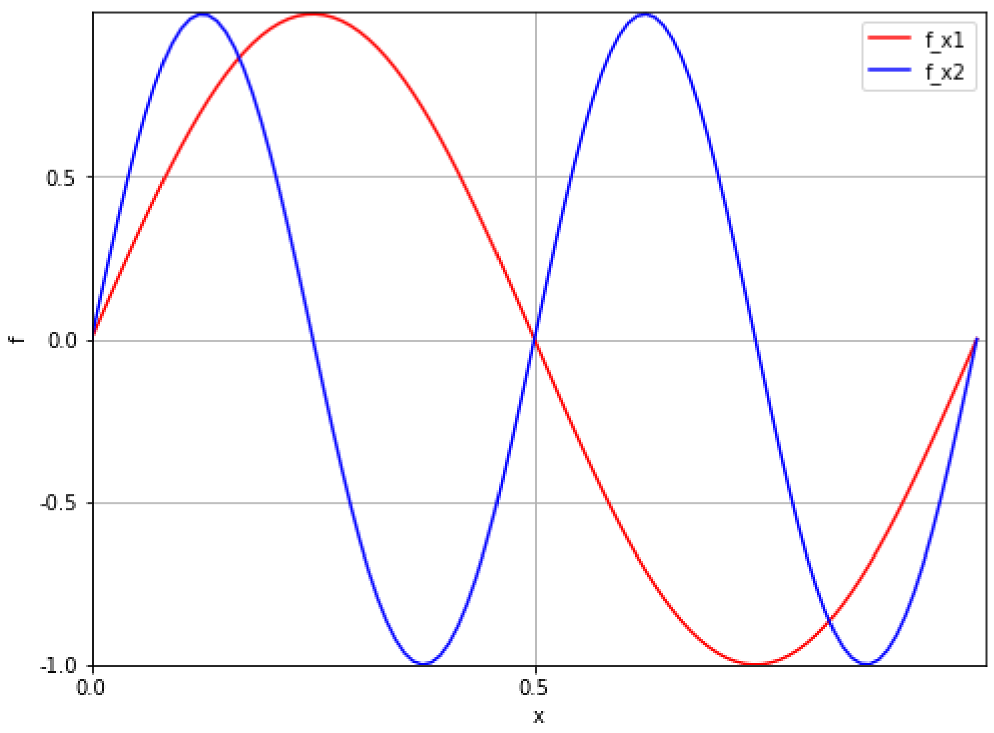
- There are two ways of plotting (visualizing) data in matplotlib.
    - One is using MATLAB (MATLAB is an another scripting language) style (functional approach).
    - The other one is using object oriented approach. We first create an axes object and add components to those axes.

We should keep in mind that functions can take always many optional arguments which provides a lot of control that aids customization of the plot.Feel free to comment out a particular line(s) and see what happens.
To know more about the keyword arguments see here

```
# MATLAB style
plt.figure(figsize=(8,6)) # create a figure with the given size
plt.plot(x,f_x1,color='r',label='f_x1') # 'r'--> red color
plt.plot(x,f_x2,color='b',label='f_x2')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('sin function')
plt.legend()
plt.grid() # turn the grid on
plt.xlim((0,1)) # limit the xaxis
plt.ylim((-1,1)) # limit the yaxis
plt.xticks(ticks=np.arange(0,1,0.5),labels=np.arange(0,1.5,0.5))
plt.yticks(ticks=np.arange(-1,1,0.5),labels=np.arange(-1,1.5,0.5))
plt.show()
```



```
# Object style
fig,ax= plt.subplots(nrows=1,ncols=1,figsize=(8,6)) # get the figure and axes objects
ax.plot(x,f_x1,color='r',label='f_x1')
ax.plot(x,f_x2,color='b',label='f_x2')
ax.grid()
ax.set_xlim((0,1.01))
ax.set_ylim((-1,1.01))
ax.set_xlabel('x')
ax.set_ylabel('f')
ax.set_xticks(ticks=np.arange(0,1,0.5))
ax.set_xticklabels(np.arange(0,1,0.5))
ax.set_yticks(ticks=np.arange(-1,1,0.5))
ax.set_yticklabels(np.arange(-1,1,0.5))
ax.legend()
fig.show()
```



If you think setting all the parameters such as xlim,xlabel,xticks .. is time consuming you can use `ax.set()` and pass these parameters as argument!

So we were able to produce the expected plot with both the approaches.However, object oriented approach is more suitable

- when the function we wish to plot changes its value during the plotting (some dynamism is there).
- Passing axes to another function as an argument.

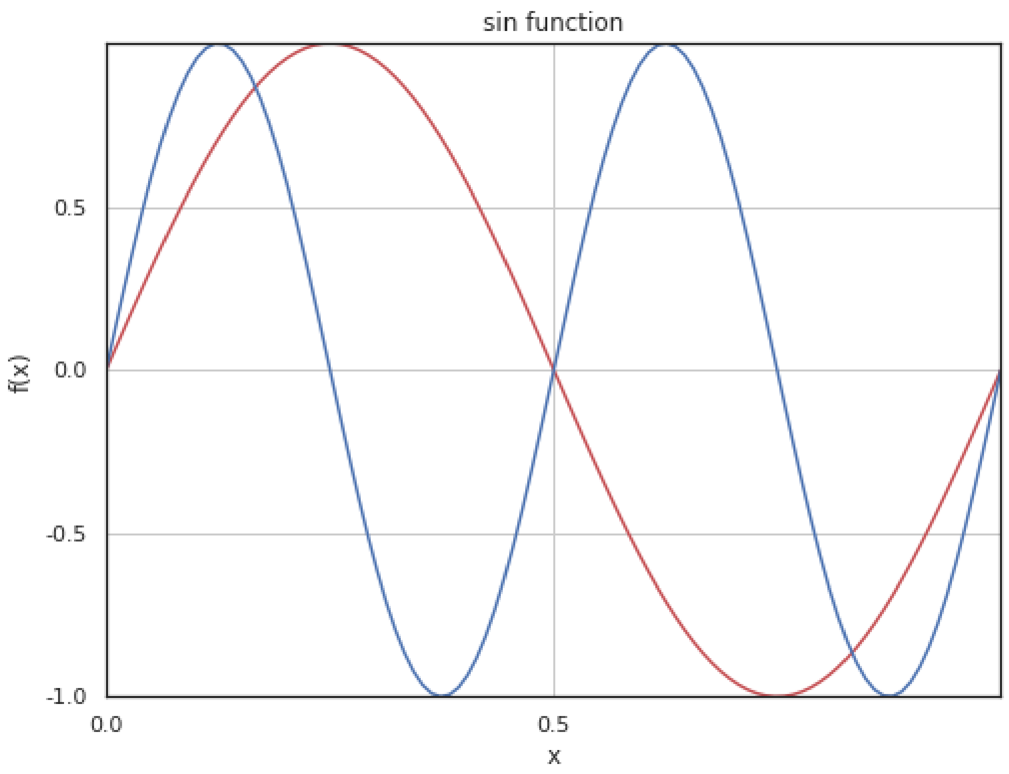
For simple plots, functional approach is more easier to work with.

Note: We can modify the type of line: dashed, dotted.. width of line, legend location and so on and so forth by passing an optional keyword arguments!.Try yourself!

Seaborn

Of course, you can do the same with seaborn. However, this example is not sufficient to explain the reason we go for seaborn. Here is a code just for completeness.

```
sns.set_theme(style='white') # It effects the matplotlib all matplotlib axe.
plt.figure(figsize=(8,6))
sns.lineplot(x=x,y=f_x1,color='r') # it returs a matplotlib axes
sns.lineplot(x=x,y=f_x2,color='b') # it returs a matplotlib axes
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('sin function')
plt.grid() # turn the grid on
plt.xlim((0,1)) # limit the xaxis
plt.ylim((-1,1)) # limit the yaxis
plt.xticks(ticks=np.arange(0,1,0.5),labels=np.arange(0,1.1,0.5))
plt.yticks(ticks=np.arange(-1,1,0.5),labels=np.arange(-1,1.1,0.5))
plt.show()
```



Let us download a different dataset to demonstrate how effortlessly seaborn makes various kinds of plots.(Example taken as is from the documentation)

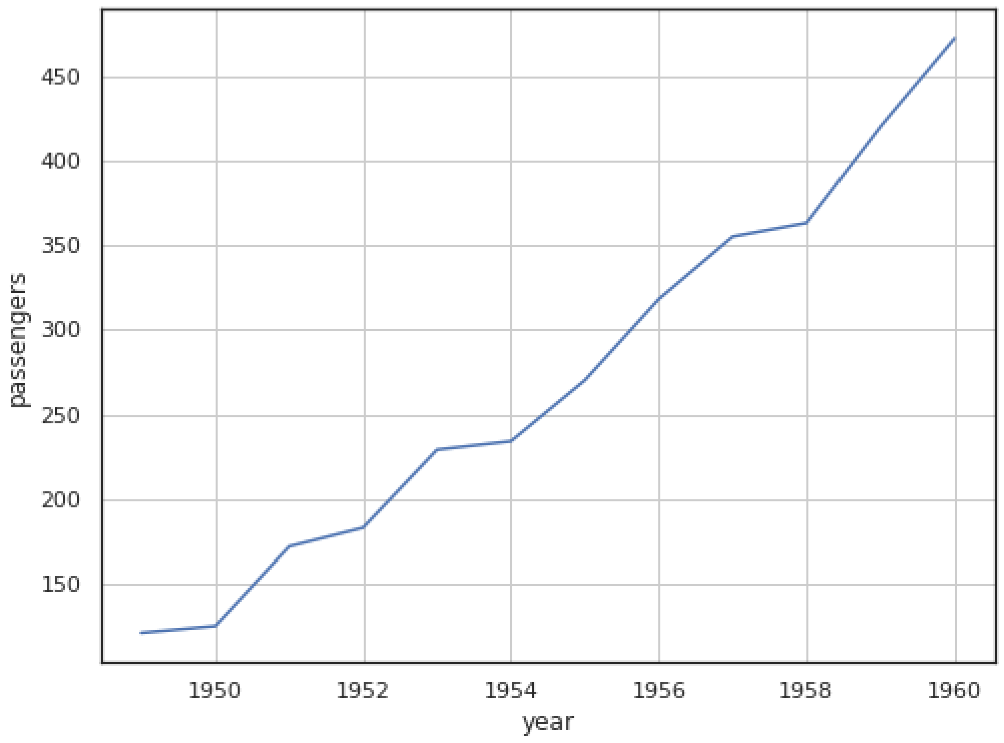
```
flights = sns.load_dataset("flights")
flights # pandas dataframe
```

```
          year  month  passengers
```

The dataset contains 144 records.Each record contains three attributes **(year,month, number of passangers**).

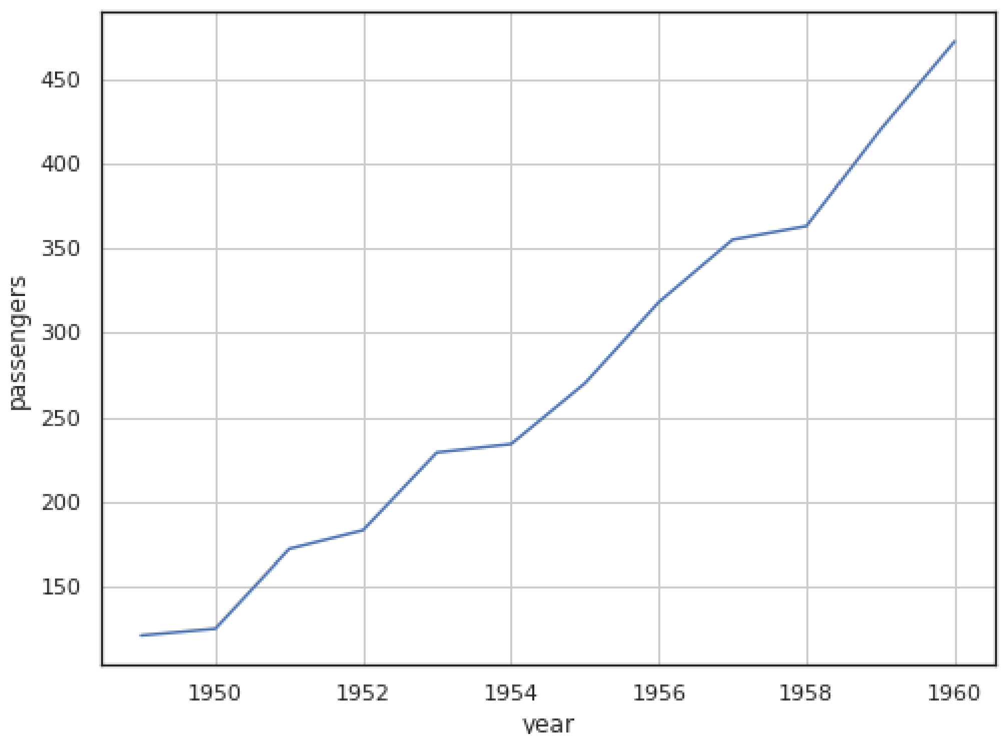
- Shall we answer the question whether the passanger count on the month of May for each year increases or decreases?

```
# using matplotlib
may_flights=flights.query("month == 'May'") #filters data along the month column
year = may_flights['year'].to_numpy() # extract the years
passengers = may_flights['passengers'].to_numpy()



plt.figure(figsize=(8,6))
plt.plot(year,passengers)
plt.grid()
plt.xlabel('year')
plt.ylabel('passengers')
plt.show()
```



Let us use seaborn. There is an additional optional argument `data` which takes the pandas dataframe as input.Then we specify the columns in the dataframe as an argument to `x` and `y`.

```
plt.figure(figsize=(8,6))
sns.lineplot(data=may_flights, x="year", y="passengers")
plt.grid()
plt.show()
```



Now you can see the difference: It seamlessly works with pandas dataframes with a very minimal efforts from our side(esspecially for formats like csv,Excel..). Now Let's change the problem statement a little bit.

- Study the trend in passangers count from 1949 to 1960 for each month.
- If you think about extending the above lines of code for each month, then that's fine.But, Painstaking work.
- Instead, change the data representation to a form called "wide-form representation"

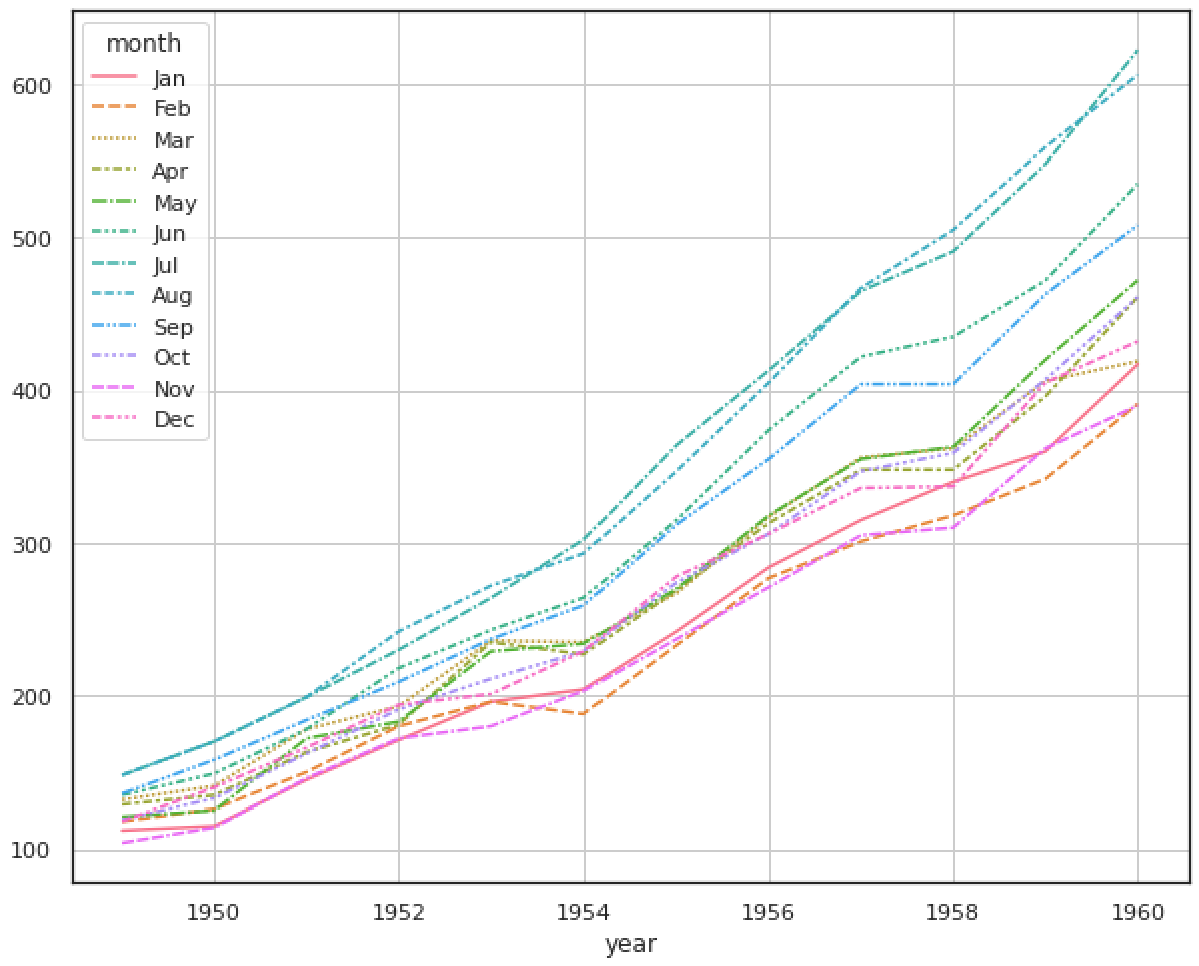
```
flights_wide = flights.pivot("year", "month", "passengers")
flights_wide.head()
```

month	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**
year												
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
1951	145	150	178	163	172	178	199	199	184	162	146	166
1952	171	180	193	181	183	218	230	242	209	191	172	194
1953	196	196	236	235	229	243	264	272	237	211	180	201

Now, do the magic with four lines of code!

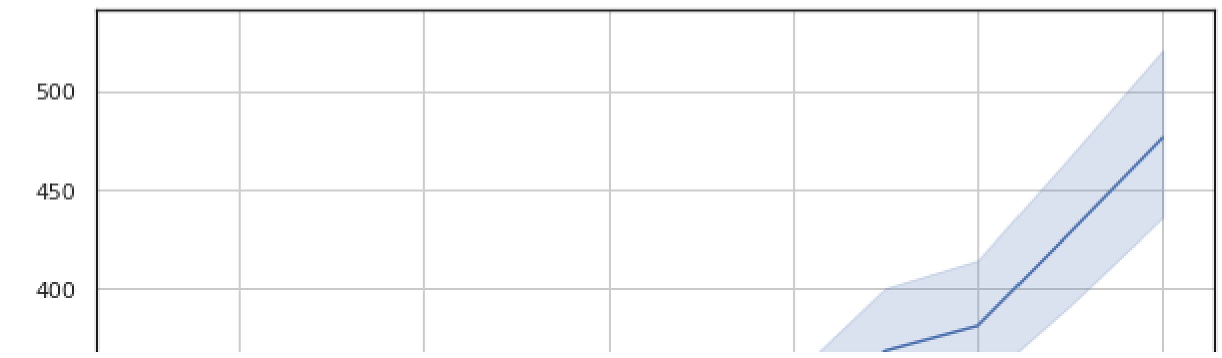
```
plt.figure(figsize=(10,8))
sns.lineplot(data=flights_wide)
```

```
plt.grid()
plt.show()
```



Not only that. Passing the entire dataset in long-form mode will aggregate over repeated values (each year) to show the mean and 95% confidence interval as shown below

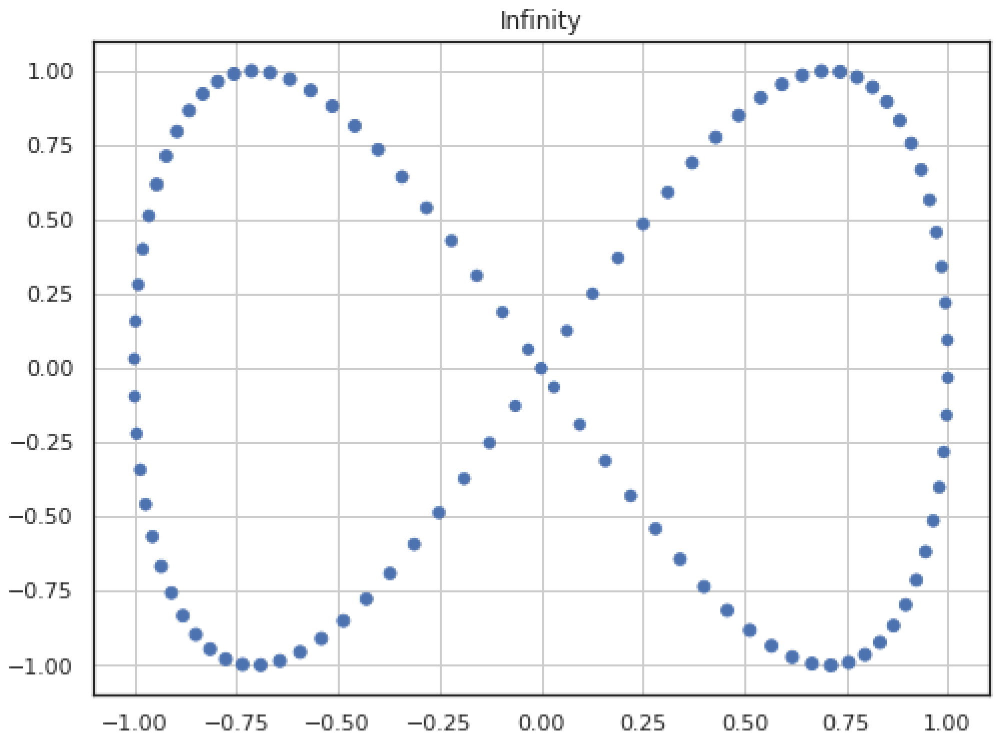
```
plt.figure(figsize=(10,8))
sns.lineplot(data=flights,x='year',y='passengers')
plt.grid()
plt.show()
```



Well, now you should be more curious to learn a lot about seaborn. Welcome, you can take a look into the well documented seaborn tutorials.

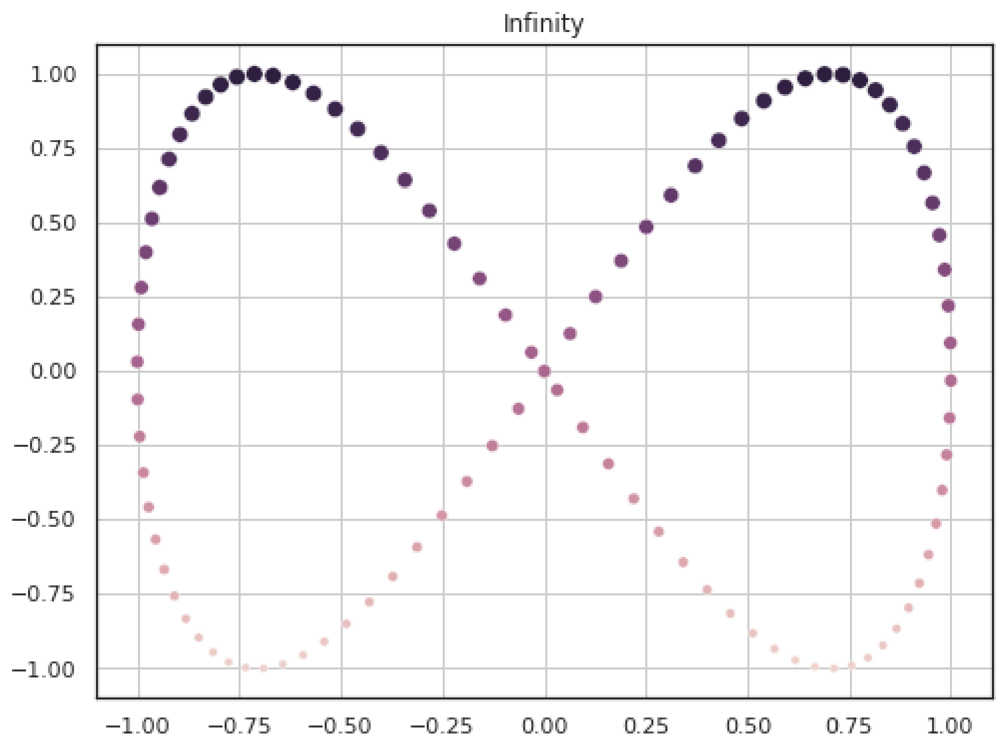
scatter plot

```
plt.figure(figsize=(8,6))
plt.scatter(f_x1,f_x2,linewidths=f_x2)
plt.title('Infinity')
plt.grid()
plt.show()
```



For some reasons, I would like to change the size and hue of the points according to some rules. It is quite a work in matplotlib. Let's do that in seaborn.

```
plt.figure(figsize=(8,6))
sns.scatterplot(x=f_x1,y=f_x2,hue=f_x2,size=f_x2,legend=None)
plt.grid()
plt.title('Infinity')
plt.show()
```



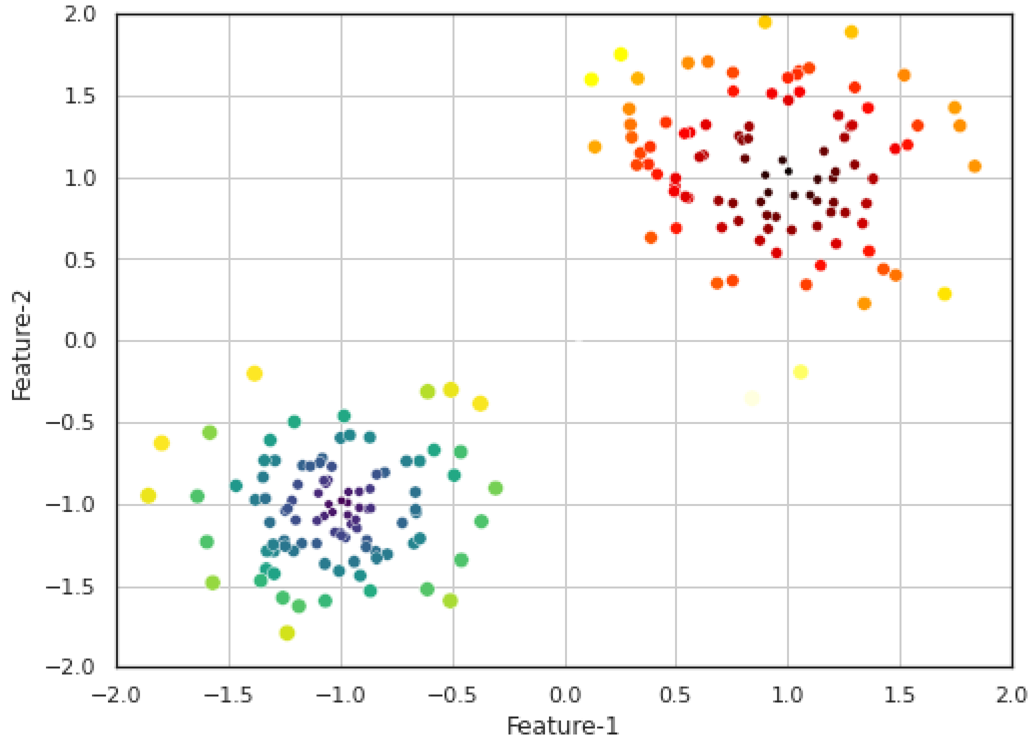
- Most of the time real world data are not deterministic in nature.There is always a noise associated with all observations. Further, the datapoints themsleves might sampled from some unknown distribution.
- Therefore, We assume that the noise follow some probability distribution such as normla distribution.
- Well, Is it possible to guess the distribution by looking at the datapoints?

Of course, yes. In order to do that, let us sample datapoints from the know distribution by changing the parameters of the pdf and see how the datapoints are **scattered** in the plane.

Generate two clusters of datapoints from normal distribution. Cluster 1 is with mean=0 and variance=0.5 and cluster 2 is with mean=1, variance=0.5. **Change the size,hue and colors** of the datapoints as a function of radius measured from the mean.

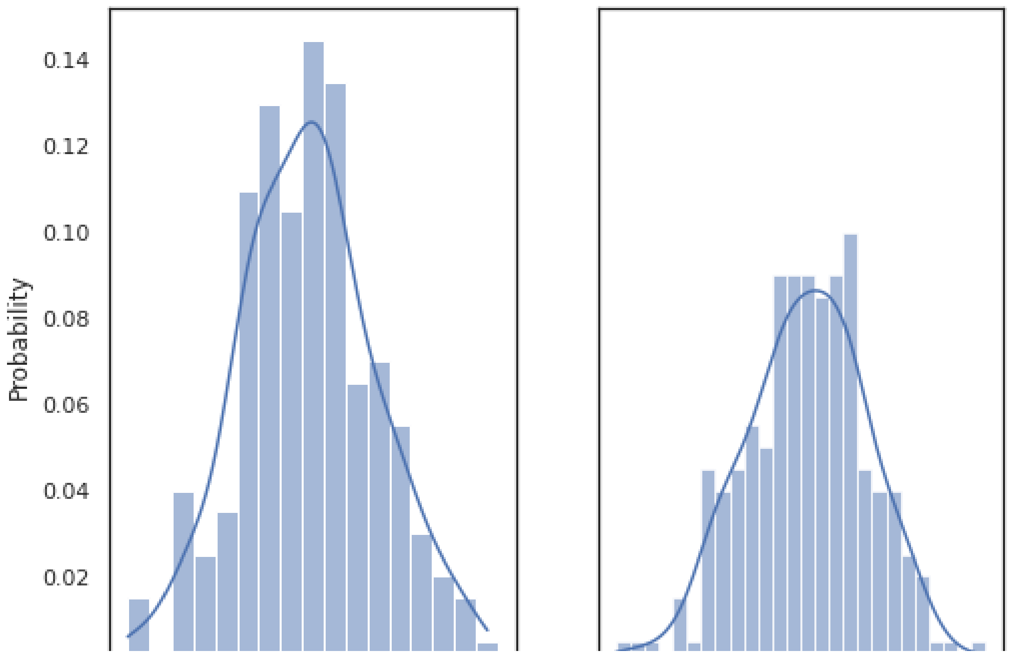
```
#cluster-1
mean_1 = -1
var_1 = 0.3
x1 = np.random.normal(loc=mean_1,scale=var_1,size=(2,100))
dist_1 =  np.sqrt((x1[0,:]-mean_1)**2+(x1[1,:]-mean_1)**2)
#cluster-2
mean_2 = 1
var_2 = 0.5
x2 = np.random.normal(loc=mean_2,scale=var_2,size=(2,100))
dist_2 =  np.sqrt((x2[0,:]-mean_2)**2+(x2[1,:]-mean_2)**2)


plt.figure(figsize=(8,6))
```

```
sns.scatterplot(x=x1[0,:],y=x1[1,:],hue=dist_1,size=dist_1,palette='viridis',legend=None)
sns.scatterplot(x=x2[0,:],y=x2[1,:],hue=dist_2,size=dist_2,palette='hot',legend=None)
plt.xlim((-2,2))
plt.ylim((-2,2))
plt.grid()
plt.xlabel('Feature-1')
plt.ylabel('Feature-2')
plt.show()
```



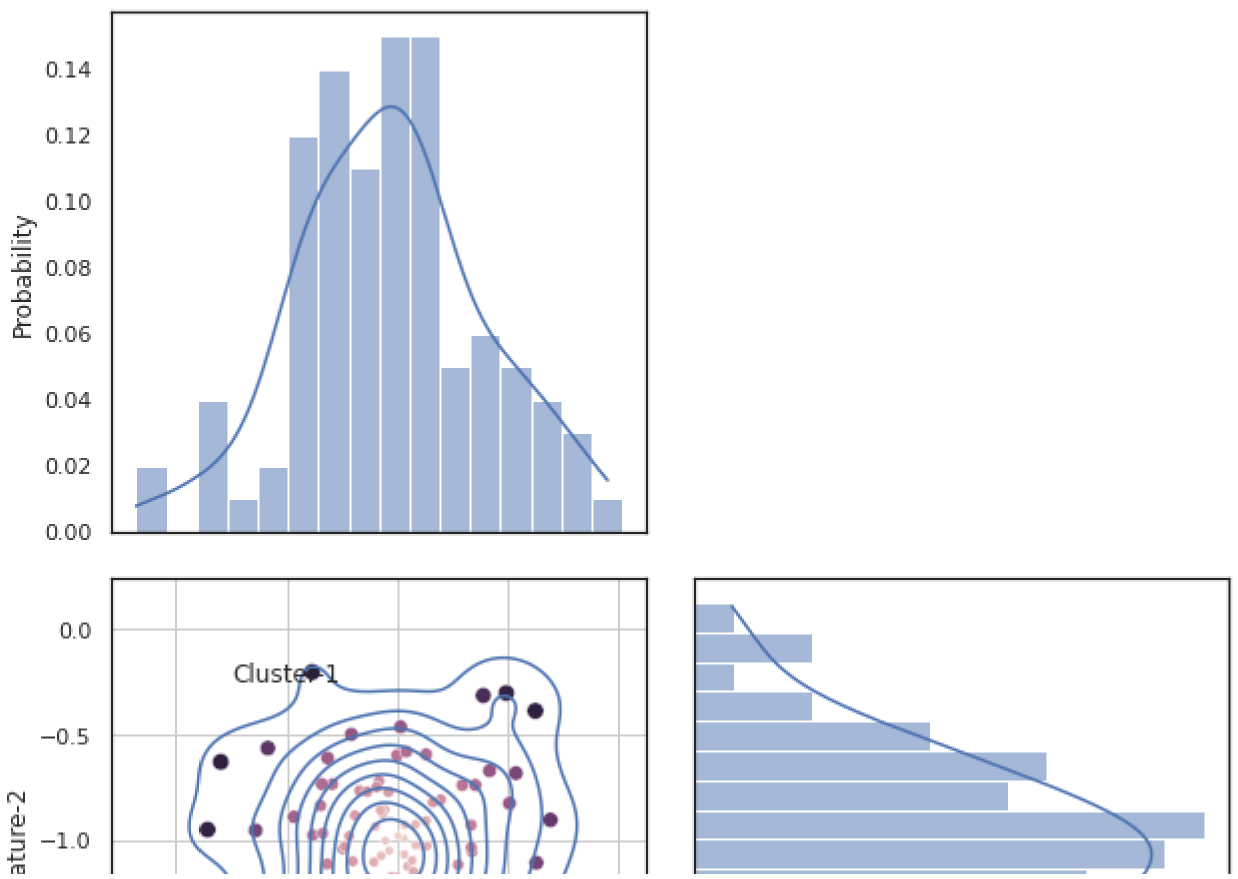
- Well, can we show the distribution (i.e, the bell curve) and convince ourselves it follows normal distribution.
- Instead of plotting the distribution function of `(x1,x2)` on two separate figures, it would be nice if plot them in a same figure in a two separate axes.
- Since there are two featues, then the PDF could be plot on 3D figure. However, before doing it, we combine the features in a single vector and plot them. We are allowed to do that as the features are independent and follows normal distribution.
- Followed by, we visualize the distributions in 3D

```
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(8,6),sharey=True)
sns.histplot(x=x1.ravel(),stat='probability',binwidth=0.1,ax=ax[0],kde=True)
ax[0].set_xlabel('x1')
sns.histplot(x=x2.ravel(),stat='probability',binwidth=0.1,ax=ax[1],kde=True)
ax[1].set_xlabel('x2')
fig.show()
```



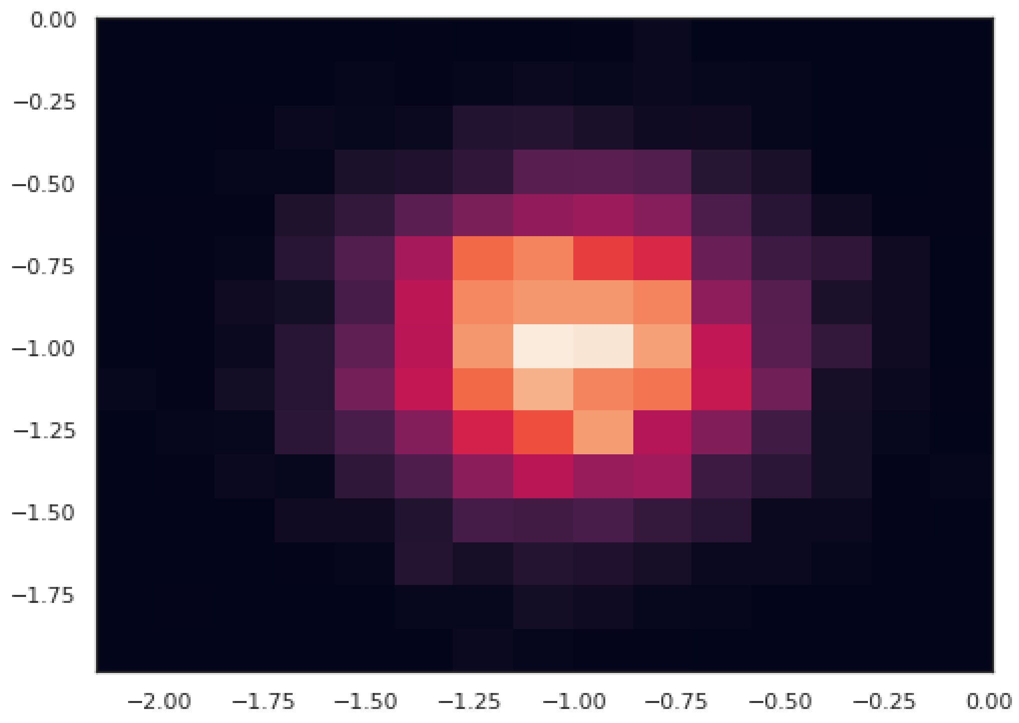
- Well, It would be great if we display the marginal density functions on the scatter plot figure itself. Then plot the contours of Joint PDF on the datapoints.
- However, it takes a littke effort from our side to use `subplot2grid` function which provides a fine control on the positions of the subplots.

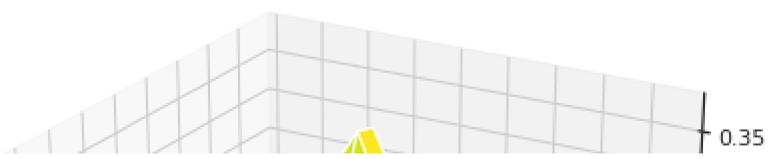
```
gridsize = (4,4)
fig = plt.figure(figsize=(10,10))
ax1 = plt.subplot2grid(gridsize,(2,2),colspan=2,rowspan=2)
ax2 = plt.subplot2grid(gridsize,(0,0),colspan=2,rowspan=2)
ax3 =  plt.subplot2grid(gridsize,(2,0),colspan=2,rowspan=2)

sns.scatterplot(x=x1[0,:],y=x1[1,:],ax=ax3,hue=dist_1,size=dist_1,legend=None)
sns.kdeplot(x=x1[0,:],y=x1[1,:])
ax3.set_xlabel('Feature-1')
ax3.set_ylabel('Feature-2')
ax3.grid()
ax3.text(x=-1.75,y=-0.25,s='Cluster-1') # wrt to data coordinates

sns.histplot(x=x1[0,:],stat='probability',binwidth=0.1,ax=ax2,kde=True)
ax1.axes.get_yaxis().set_visible(False)

sns.histplot(y=x1[1,:],stat='probability',binwidth=0.1,ax=ax1,kde=True)
ax2.axes.get_xaxis().set_visible(False)

fig.show()
```



Well, we were expecting smooth contours (circles) instead we got some rough curves. The reason is associated with the number of datapoints. Higher the better estimation.
Plot the 3D PDF with more number of datapoints (to have a smooth surface).

```
plt.figure(figsize=(8,6))
x1 = np.random.normal(loc=mean_1,scale=var_1,size=(2,5000))
h,x,y,_=plt.hist2d(x1[0,:],x1[1,:],bins=15,density=True)
x,y = np.meshgrid(x,y)
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111,projection='3d')
ax.plot_surface(x[:15,:15],y[:15,:15],h/(np.sqrt(2)*np.pi),cmap='viridis')
ax.set_xlabel('Feature-1')
ax.set_ylabel('Feature-2')
ax.set_zlabel('P(x1,x2)')
plt.show()
```





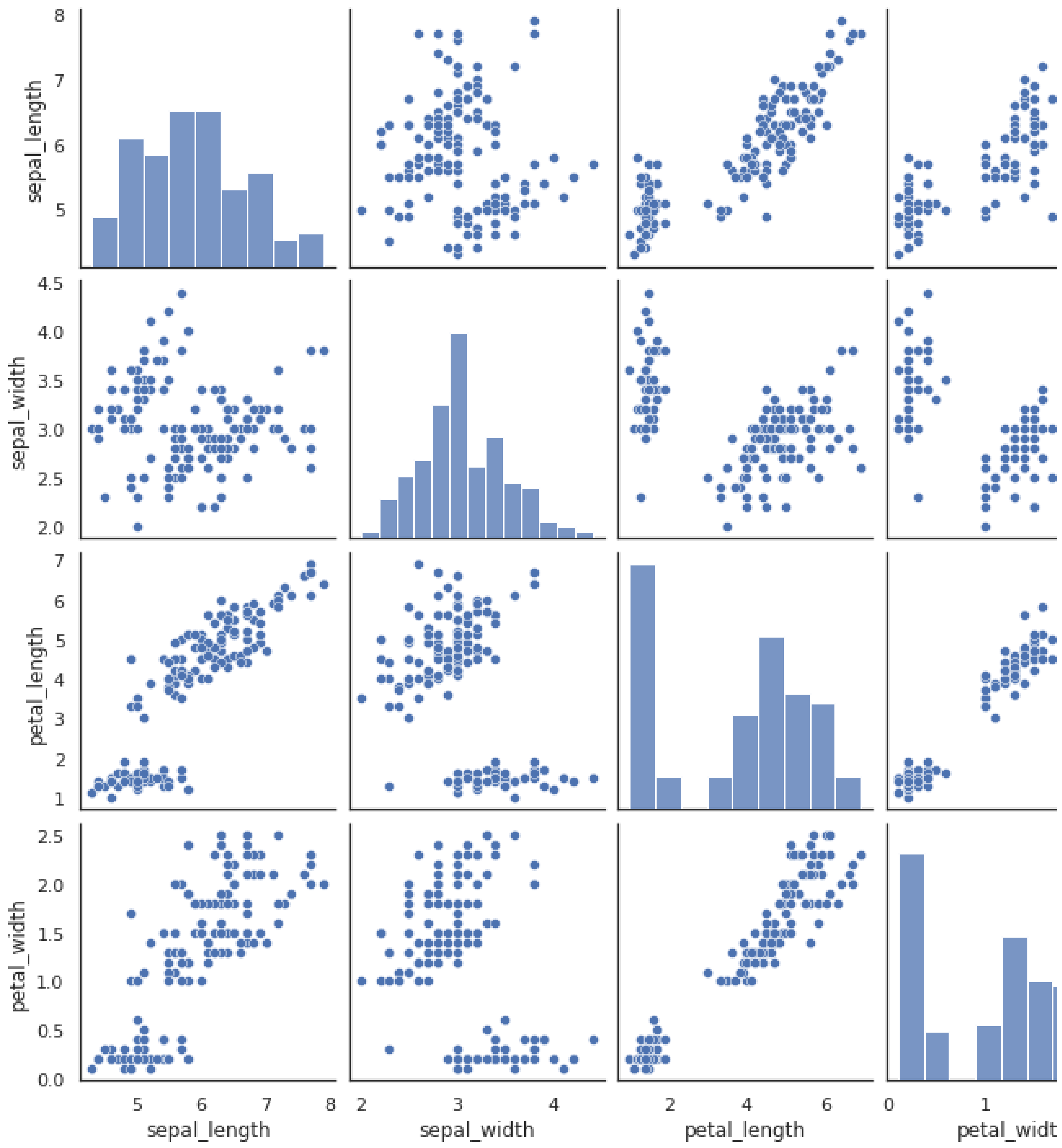
- Now we know how to create more than one plot in a single figure window.Suppose that we have a dataset with 4 features and we want to know the relation between the feature by using scatter plot.
- Therefore, we need to create 16 sub-plots for each pair-waise features.
- Of course, we can do that with matplotlib. Again, it involves a bit more effort.
- Seaborn provides a function called `pairplot` to create these types of plots quickly.(Behind the scene, it calls `PairGrid` function)
- Let's load the **iris** dataset that contains four features.

```
iris= sns.load_dataset("iris")
iris # pandas dataframe
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

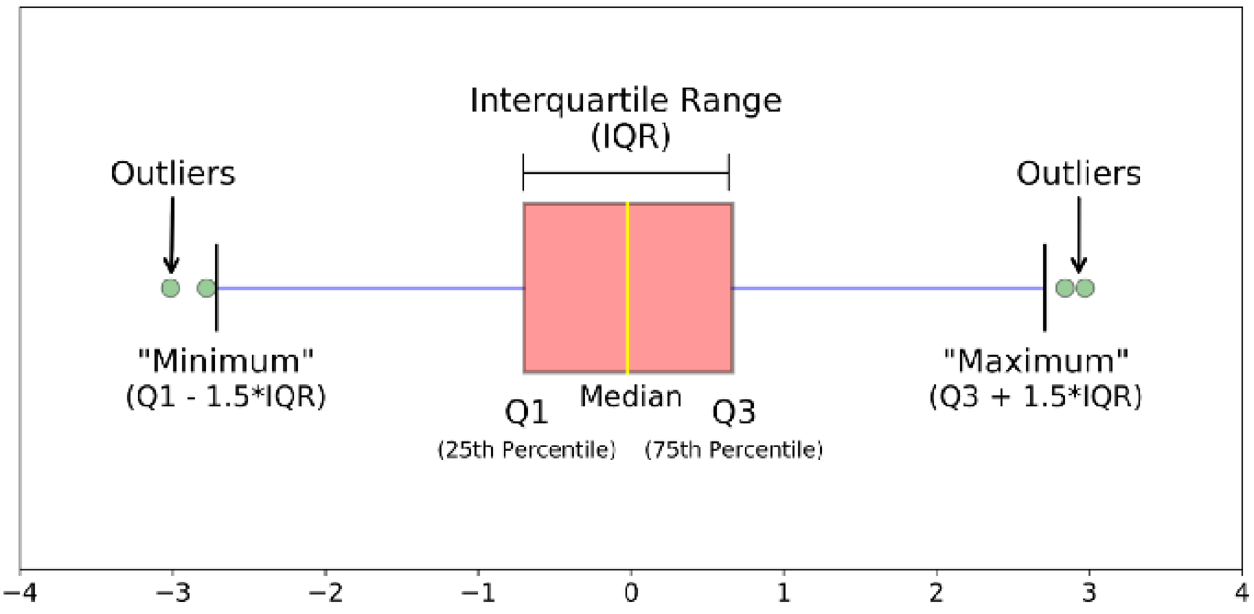
```
plt.figure(figsize=(8,6))
sns.pairplot(data=iris)
plt.show()
```

```
<Figure size 576x432 with 0 Axes>
```



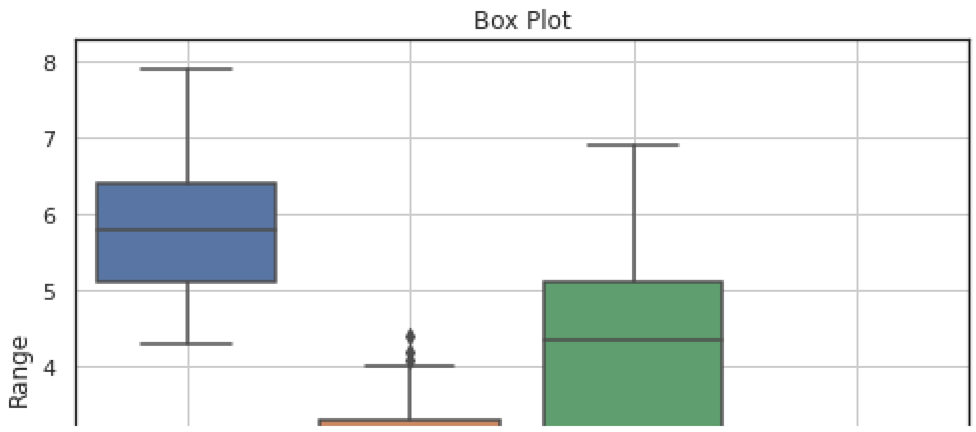
Box plot

- Histograms gives us the distribution of datapoints. (i.e.,the count of samples values that falls within in a bin). This helps us understand whether the distribution is skewed.Also, we can divde the entire distribution into "**quartiles**". What if we have more than one feature and wanted to display the distribution function for all the features in a single plot for comparision?
- That's where boxplot can be useful.It also reveals same information with less space using 5-point summary : "min,First Quartile (Q1),median,Third Quartile (Q3),and maximum" as shown below, (Recall that mean is versy senesitive to outliers whereas median is not!)



- This helps us to identify the outliers in the dataset. Please refer to the article to understand the need for boxplot in detail. https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

```
plt.figure(figsize=(8,6))
sns.boxplot(data=iris,)
plt.grid()
plt.title('Box Plot')
plt.ylabel('Range')
plt.show()
```



- Compare the above boxplot with the histogram plots along the diagonal of immediately above figure.Are you able to relate it with the histogram?
- Here is an other way to get all these (5 number summary) details.

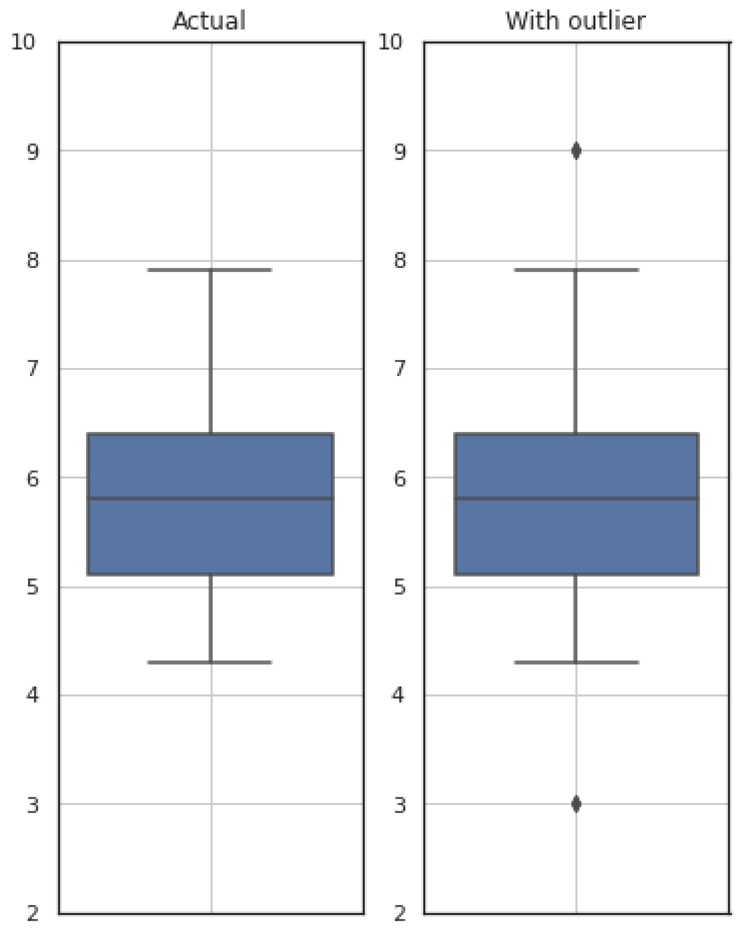
```
iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Well, what happens if there are some outliers in the data?. Let's add some outlier to the feature `sepal_length` ans see how boxplot looks like.

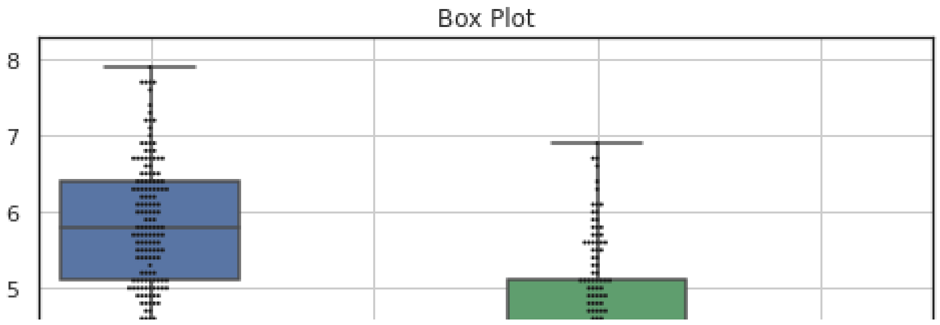
```
s_len = iris['sepal_length'].to_numpy()
plt.figure(figsize=(6,8))
plt.subplot(1,2,1)
sns.boxplot(y=s_len)
plt.ylim((2,10))
plt.title('Actual')
plt.grid()

# Add outlier
outliers = np.array([9,9,3,3])
s_len_new = np.concatenate((s_len,outliers),axis=0)
plt.subplot(1,2,2)
sns.boxplot(y=s_len_new)
```

```
plt.ylim((2,10))
plt.title('With outlier')
plt.grid()
plt.show()
```



We can also display the datapoints on the boxplot (by adjusting the x-axis a bit to avoid overlapping) as show below.

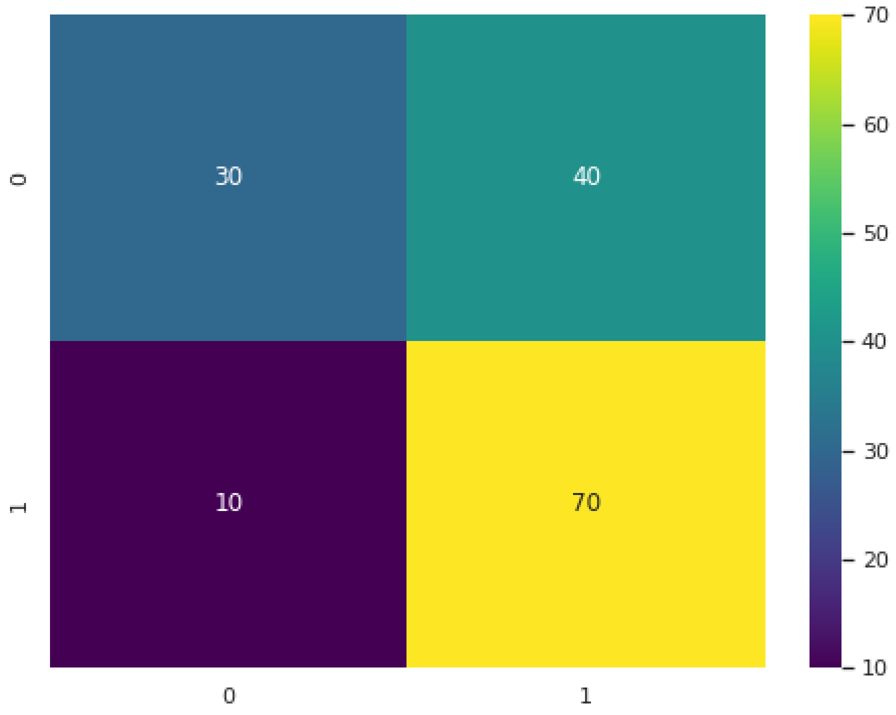
```
plt.figure(figsize=(8,6))
sns.boxplot(data=iris)
sns.swarmplot(data=iris,size=2,color='k')
plt.grid()
plt.title('Box Plot')
plt.ylabel('Range')
plt.show()
```



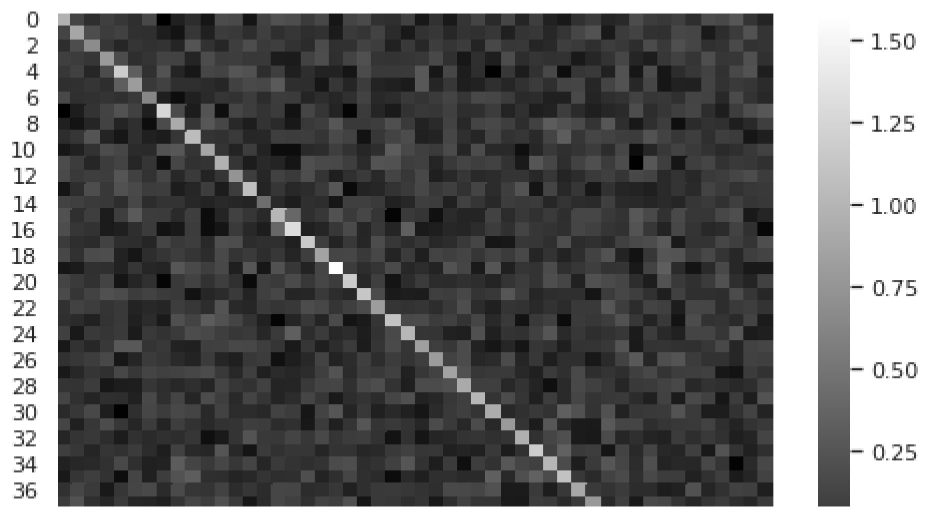
Matrix Visualization

- Often times, we need to display the matrix elemenets as an image to see the bigpicture.
- Examples : Image matrix, confusion matrix and covariance matrix

```
cm = np.array([[30,40],[10,70]])
plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,cmap='viridis')
plt.show()
```



```
x = np.random.randn(50,50)
cov = np.cov(x)
plt.figure(figsize=(8,6))
sns.heatmap(cov,cmap='gray')
plt.show()
```



```
from sklearn.datasets import load_sample_image
```

```
img = load_sample_image('flower.jpg')
plt.figure(figsize=(8,6))
plt.imshow(img) # can't use heatmap as image contains 3 channels
plt.show()
```