2 LAB 4: Data Pre-Processing

2.0.1 Feature Engineering in DataScience

- Data Visualization
- Data Pre-processing
- Dimension Reduction
- Feature Extraction
- Feature Selection

```
from datetime import datetime

try:
  from google.colab import drive
  %tensorflow_version 2.x
  COLAB = True
  print("Hello World")
  print("Note: using Google CoLab")
except:
  print("Hello NITD")
  print("Note: not using Google CoLab")
  COLAB = False
# Print your name and Roll No.
print('Rohit Byas Sherwan 181210043')

#print current time
now = datetime.now()

current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
```

```
Hello World
Note: using Google CoLab
Rohit Byas Sherwan 181210043
Current Time = 09:24:26
```

```
! pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /u
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
```

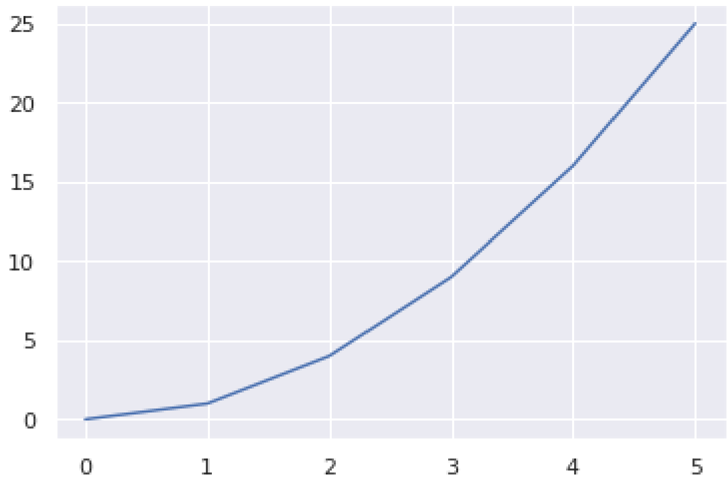
```
import numpy as np
import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator, MultipleLocator, FuncFormatter

np.random.seed(19680801)

X = np.linspace(0.5, 3.5, 100)
Y1 = 3+np.cos(X)
Y2 = 1+np.cos(1+X/0.75)/2
Y3 = np.random.uniform(Y1, Y2, len(X))

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect=1)


def minor_tick(x, pos):
    if not x % 1.0:
        return ""
    return "%.2f" % x

ax.xaxis.set_major_locator(MultipleLocator(1.000))
ax.xaxis.set_minor_locator(AutoMinorLocator(4))
ax.yaxis.set_major_locator(MultipleLocator(1.000))
ax.yaxis.set_minor_locator(AutoMinorLocator(4))
ax.xaxis.set_minor_formatter(FuncFormatter(minor_tick))

ax.set_xlim(0, 4)
ax.set_ylim(0, 4)

ax.tick_params(which='major', width=1.0)
ax.tick_params(which='major', length=10)
ax.tick_params(which='minor', width=1.0, labelsize=10)
ax.tick_params(which='minor', length=5, labelsize=10, labelcolor='0.25')

ax.grid(linestyle="--", linewidth=0.5, color='.25', zorder=-10)

ax.plot(X, Y1, c=(0.25, 0.25, 1.00), lw=2, label="Blue signal", zorder=10)
ax.plot(X, Y2, c=(1.00, 0.25, 0.25), lw=2, label="Red signal")
ax.plot(X, Y3, linewidth=0,
        marker='o', markerfacecolor='w', markeredgecolor='k')

ax.set_title("Anatomy of a figure", fontsize=20, verticalalignment='bottom')
ax.set_xlabel("X axis label")
ax.set_ylabel("Y axis label")

ax.legend()


def circle(x, y, radius=0.15):
    from matplotlib.patches import Circle
    from matplotlib.patheffects import withStroke
    circle = Circle((x, y), radius, clip_on=False, zorder=10, linewidth=1,
                    edgecolor='black', facecolor=(0, 0, 0, .0125),
                    path_effects=[withStroke(linewidth=5, foreground='w')])
    ax.add_artist(circle)
```

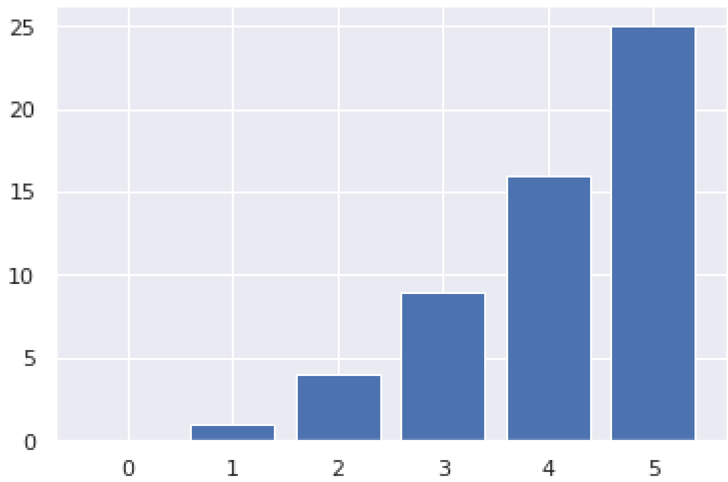
```
def text(x, y, text):
    ax.text(x, y, text, backgroundcolor="white",
            ha='center', va='top', weight='bold', color='blue')


# Minor tick
circle(0.50, -0.10)
text(0.50, -0.32, "Minor tick label")

# Major tick
circle(-0.03, 4.00)
text(0.03, 3.80, "Major tick")

# Minor tick
circle(0.00, 3.50)
text(0.00, 3.30, "Minor tick")

# Major tick label
circle(-0.15, 3.00)
text(-0.15, 2.80, "Major tick label")

# X Label
circle(1.80, -0.27)
text(1.80, -0.45, "X axis label")

# Y Label
circle(-0.27, 1.80)
text(-0.27, 1.6, "Y axis label")

# Title
circle(1.60, 4.13)
text(1.60, 3.93, "Title")

# Blue plot
circle(1.75, 2.80)
text(1.75, 2.60, "Line\n(line plot)")

# Red plot
circle(1.20, 0.60)
text(1.20, 0.40, "Line\n(line plot)")

# Scatter plot
circle(3.20, 1.75)
text(3.20, 1.55, "Markers\n(scatter plot)")

# Grid
circle(3.00, 3.00)
text(3.00, 2.80, "Grid")

# Legend
circle(3.70, 3.80)
text(3.70, 3.60, "Legend")

# Axes
circle(0.5, 0.5)
```

```
circle(0.5, 0.5)
text(0.5, 0.3, "Axes")

# Figure
circle(-0.3, 0.65)
text(-0.3, 0.45, "Figure")

color = 'blue'
ax.annotate('Spines', xy=(4.0, 0.35), xytext=(3.3, 0.5),
            weight='bold', color=color,
            arrowprops=dict(arrowstyle='->',
                            connectionstyle="arc3",
                            color=color))

ax.annotate('', xy=(3.15, 0.0), xytext=(3.45, 0.45),
            weight='bold', color=color,
            arrowprops=dict(arrowstyle='->',
                            connectionstyle="arc3",
                            color=color))

ax.text(4.0, -0.4, "Made with http://matplotlib.org",
        fontsize=10, ha="right", color='.5')

plt.show()

## https://matplotlib.org/3.1.1/gallery/showcase/anatomy.html
```

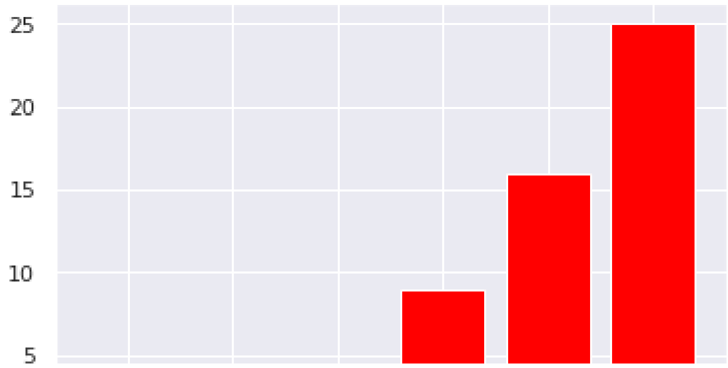
```
import matplotlib.pyplot as plt

# get the data ready
x = [0,1,2,3,4,5]
y = [0,1,4,9,16,25]

# a simple graph
plt.plot(x, y)

# show the graph
plt.show()
```



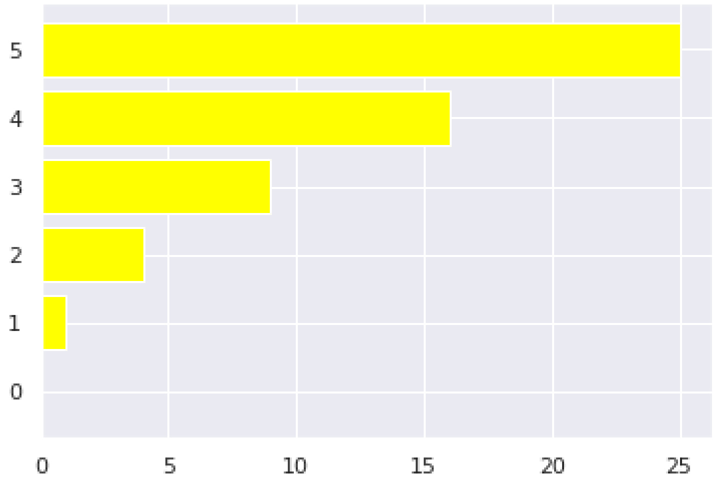

```
# make it a bar plot
plt.bar(x, y)
plt.show()
```



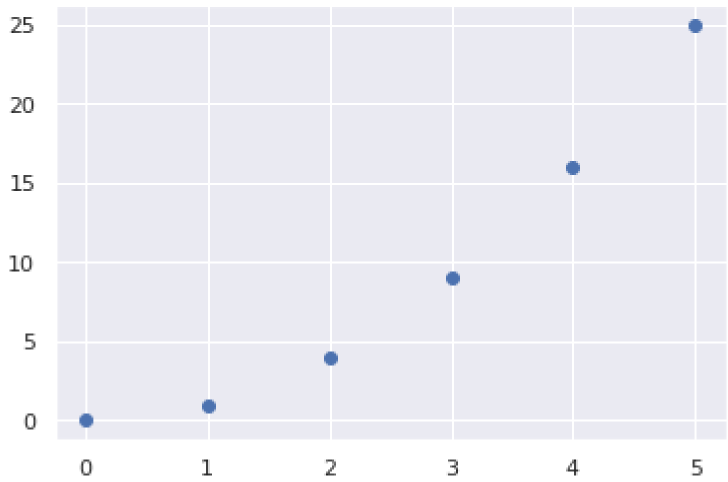

```
# change the color of bars
plt.bar(x, y, color='red')
plt.show()
```



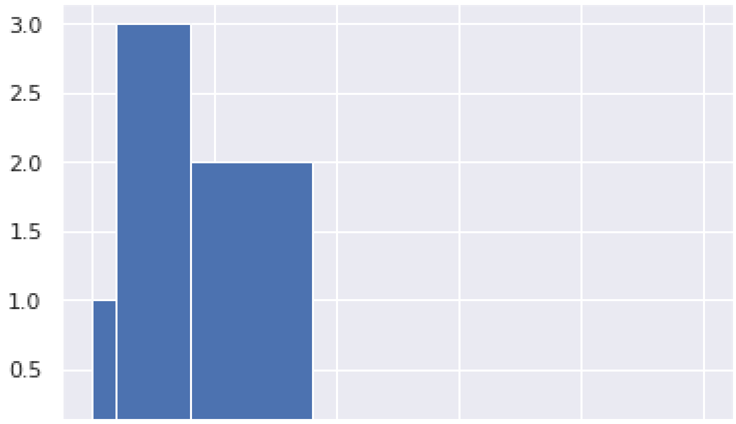

```
# make it horizontal bar
plt.barh(x, y, color='yellow')
plt.show()
```



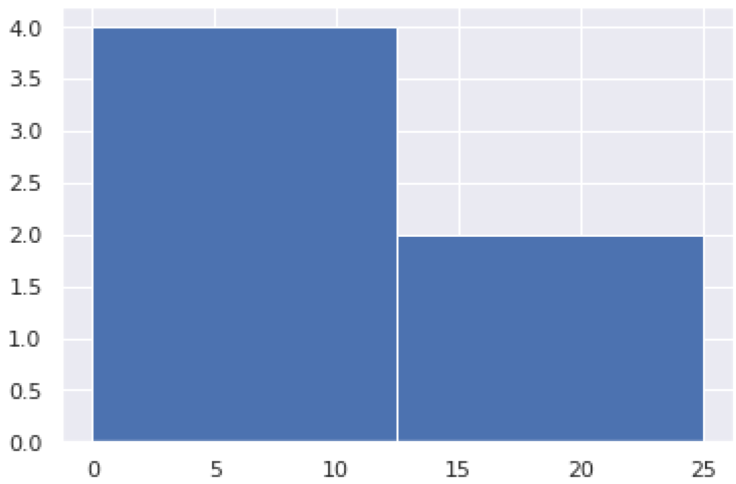

```
# make a scatter plot
plt.scatter(x, y)
plt.show()
```




```
# make those histograms
plt.hist(x, y)
plt.show()
```



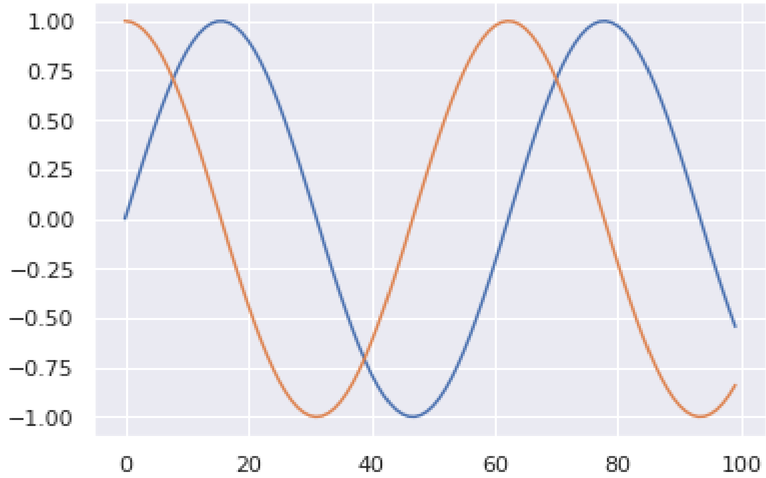

```
# make a proper histograms
plt.hist(y, bins=2)
plt.show()
# identify the number of the bin that disctirubte is same ?
# 25
```



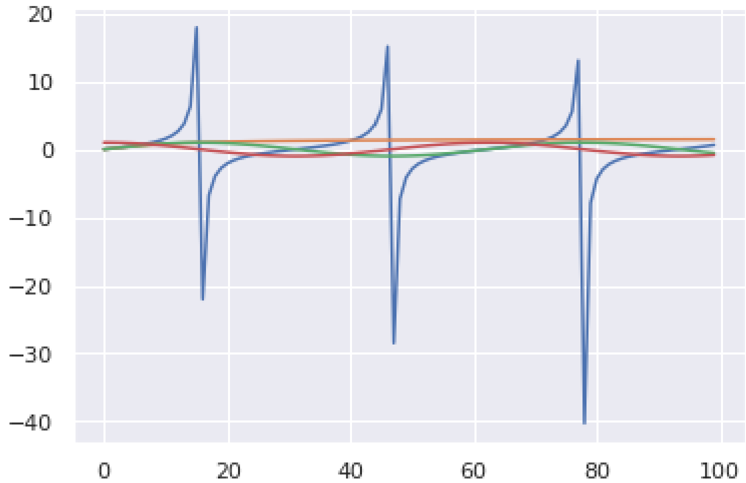

```
# Now lets explore with continuous data
import numpy as np
x = np.linspace(0, 10, 100)
print(x)
```

```
[ 0.          0.1010101   0.2020202   0.3030303   0.4040404   0.50505051
  0.60606061  0.70707071  0.80808081  0.90909091  1.01010101  1.11111111
  1.21212121  1.31313131  1.41414141  1.51515152  1.61616162  1.71717172
  1.81818182  1.91919192  2.02020202  2.12121212  2.22222222  2.32323232
  2.42424242  2.52525253  2.62626263  2.72727273  2.82828283  2.92929293
  3.03030303  3.13131313  3.23232323  3.33333333  3.43434343  3.53535354
  3.63636364  3.73737374  3.83838384  3.93939394  4.04040404  4.14141414
  4.24242424  4.34343434  4.44444444  4.54545455  4.64646465  4.74747475
  4.84848485  4.94949495  5.05050505  5.15151515  5.25252525  5.35353535
  5.45454545  5.55555556  5.65656566  5.75757576  5.85858586  5.95959596
  6.06060606  6.16161616  6.26262626  6.36363636  6.46464646  6.56565657
  6.66666667  6.76767677  6.86868687  6.96969697  7.07070707  7.17171717
  7.27272727  7.37373737  7.47474747  7.57575758  7.67676768  7.77777778
  7.87878788  7.97979798  8.08080808  8.18181818  8.28282828  8.38383838
  8.48484848  8.58585859  8.68686869  8.78787879  8.88888889  8.98989899
  9.09090909  9.19191919  9.29292929  9.39393939  9.49494949  9.5959596
  9.6969697   9.7979798   9.8989899  10.        ]
```

```
# Try with some real data
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.sin(x))
plt.plot(np.cos(x))
plt.show()

# Which one is sin cureve ??
```

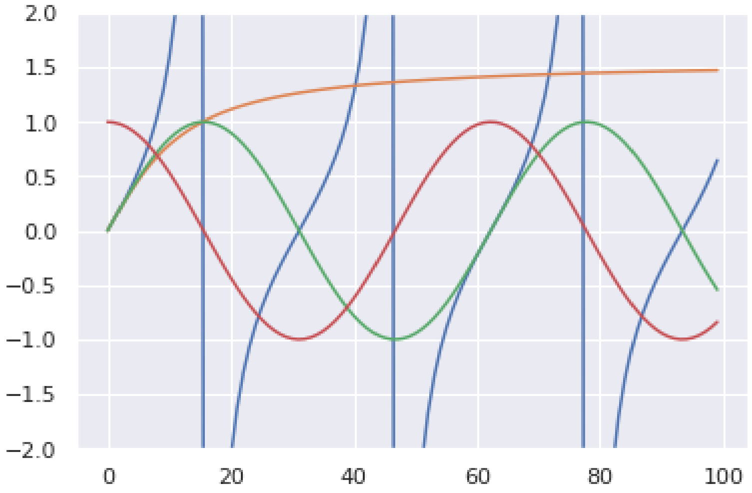



```
# do some real data [kind of]
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.tan(x))
plt.plot(np.arctan(x))
plt.plot(np.sin(x))
plt.plot(np.cos(x))
plt.show()
```

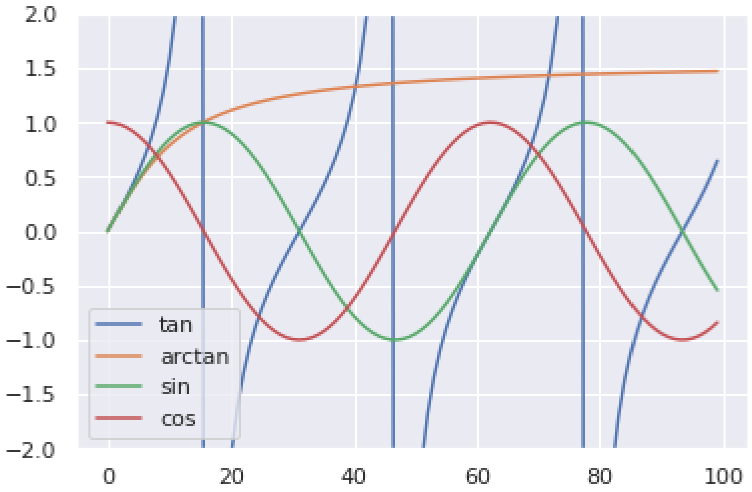



```
# make this look better
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.tan(x), label='tan')
plt.plot(np.arctan(x), label='arctan')
plt.plot(np.sin(x), label='sin')
plt.plot(np.cos(x), label='cos')
plt.ylim(-2, 2) ## Look here
```

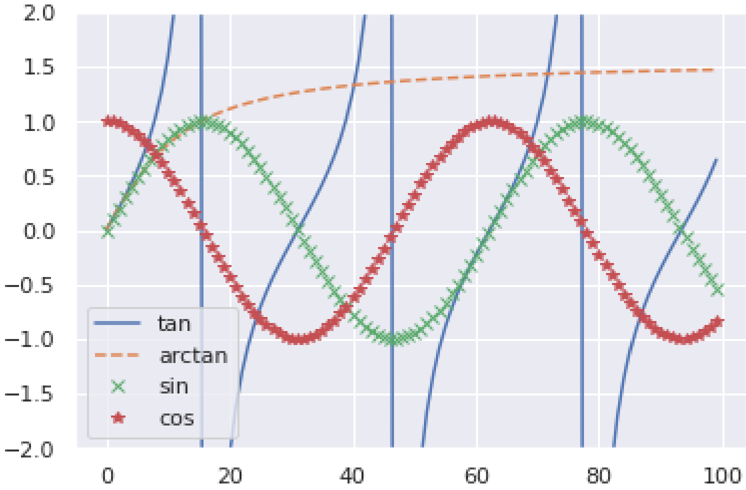
```
plt.ylim(-2, 2) ## Look here
plt.show()
```



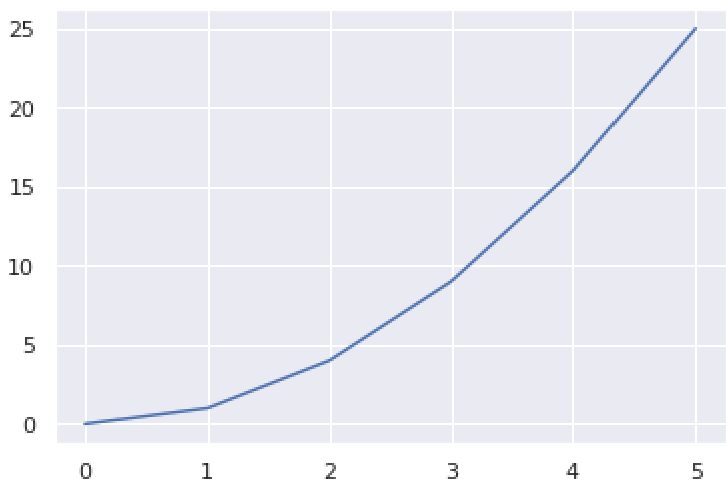

```
# lets give a legend
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.tan(x), label='tan')
plt.plot(np.arctan(x), label='arctan')
plt.plot(np.sin(x), label='sin')
plt.plot(np.cos(x), label='cos')
plt.ylim(-2, 2)
plt.legend()
plt.show()
```



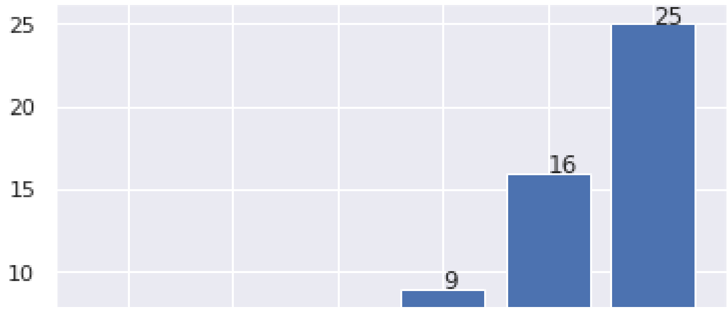

```
# Change dash styles
import numpy as np
x = np.linspace(0, 10, 100)
6
plt.plot(np.tan(x), '-', label='tan')
plt.plot(np.arctan(x), '--', label='arctan')
plt.plot(np.sin(x), 'x', label='sin')
plt.plot(np.cos(x), '*', label='cos')
plt.ylim(-2, 2)
plt.legend()
plt.show()
```



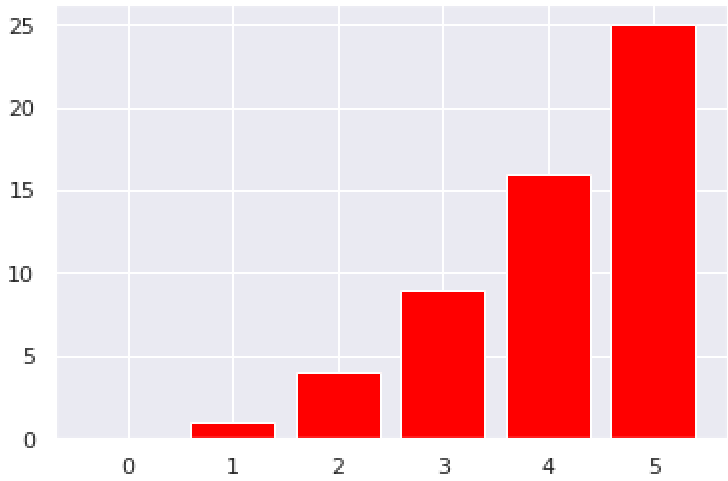

```
#Get necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
# get the data ready
x = [0,1,2,3,4,5]
y = [0,1,4,9,16,25]
# a simple graph
plt.plot(x, y)
# show the graph
plt.show()
```



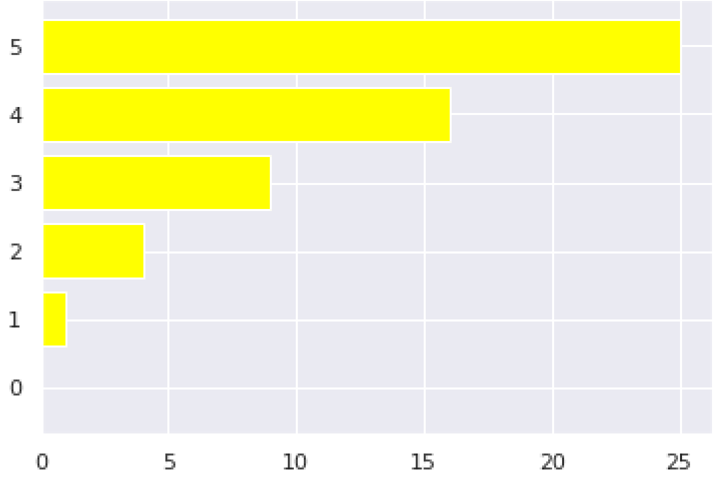

```
# make it a bar plot
plt.bar(x, y)
for index, value in enumerate(y):
  # print(index, value) # 3 rd value
  plt.text(index, value, str(value))
plt.show()
```




```
# change the color of bars
plt.bar(x, y, color='red')
plt.show()
```

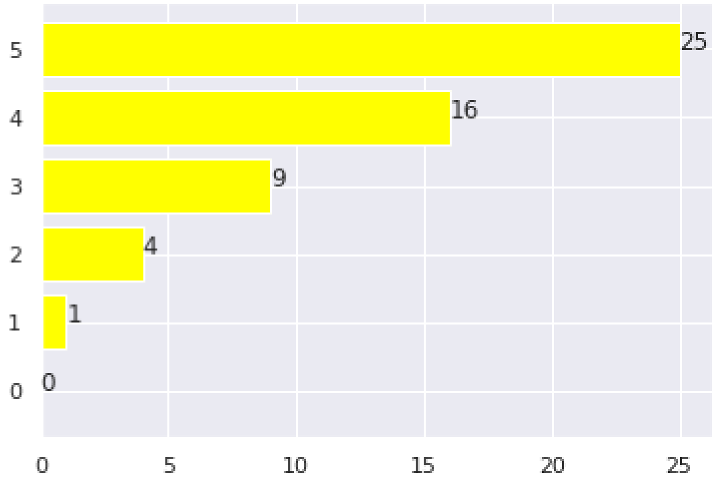



```
# make it horizontal bar
plt.barh(x, y, color='yellow')
plt.show()
```



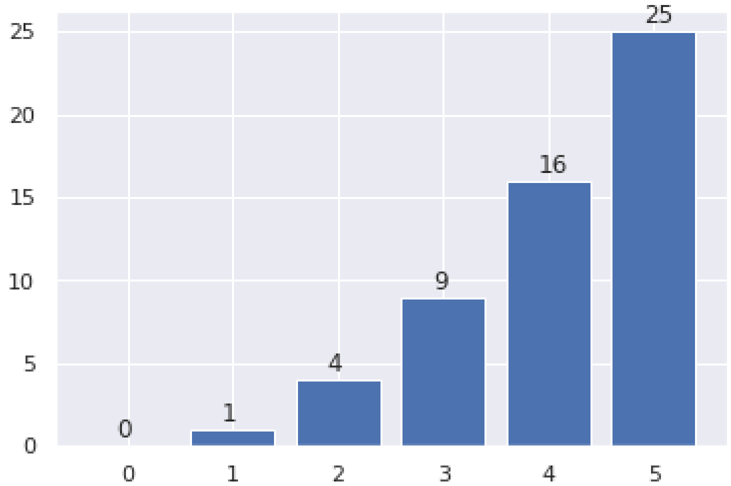

```
# make it horizontal bar
plt.barh(x, y, color='yellow')
for index, value in enumerate(y):
  print(index, value) # 3 rd value
  plt.text(value, index, str(value))
plt.show()
```

```
0 0
1 1
2 4
3 9
4 16
5 25
```

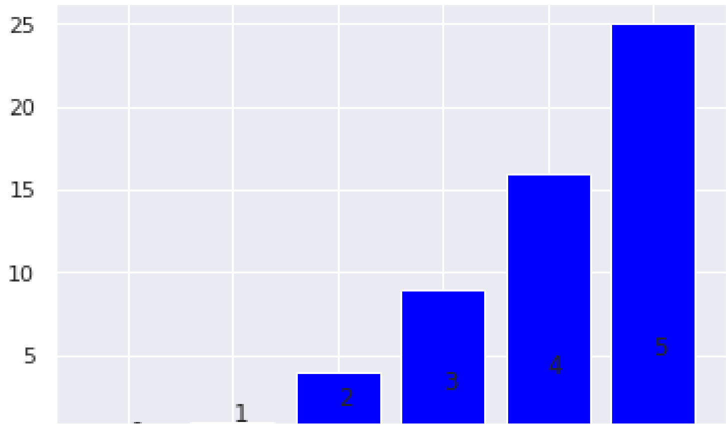



```
# make it horizontal bar
plt.bar(x, y)
for index, value in enumerate(y):
  print(index, value) # 3 rd value
  plt.text(index - 0.1, value + 0.5, str(value))
# plt.text(0, 15, 'hey')
plt.show()
```

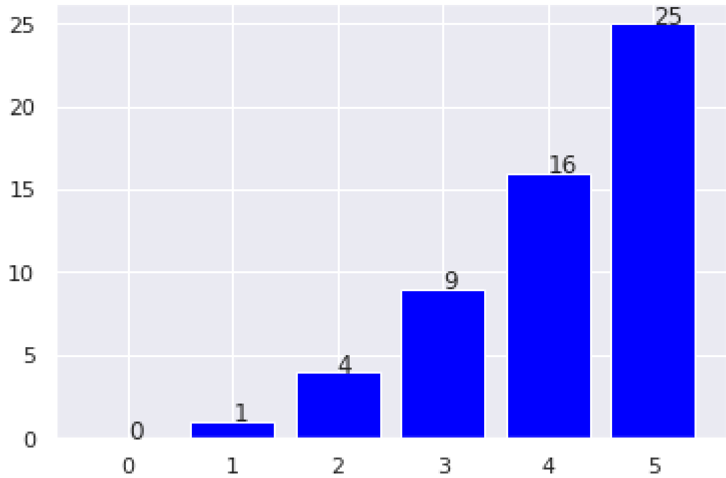
```
0 0
1 1
2 4
3 9
4 16
5 25
```



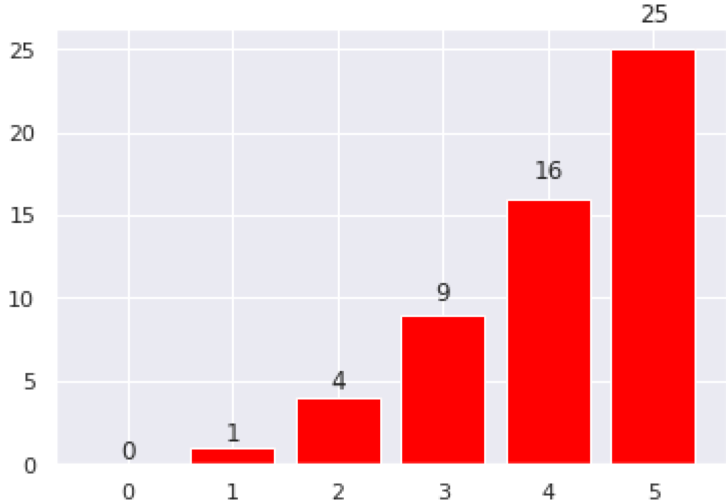

```
plt.bar(x, y, color='blue')
for index, value in enumerate(x):
  plt.text(value, index, str(value))
plt.show()
```



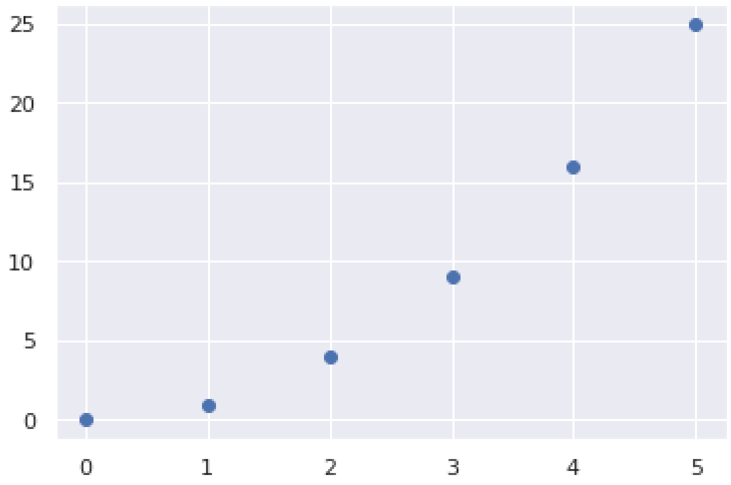

```
plt.bar(x, y, color='blue')
for index, value in enumerate(y):
  plt.text(index, value, str(value))
plt.show()
```



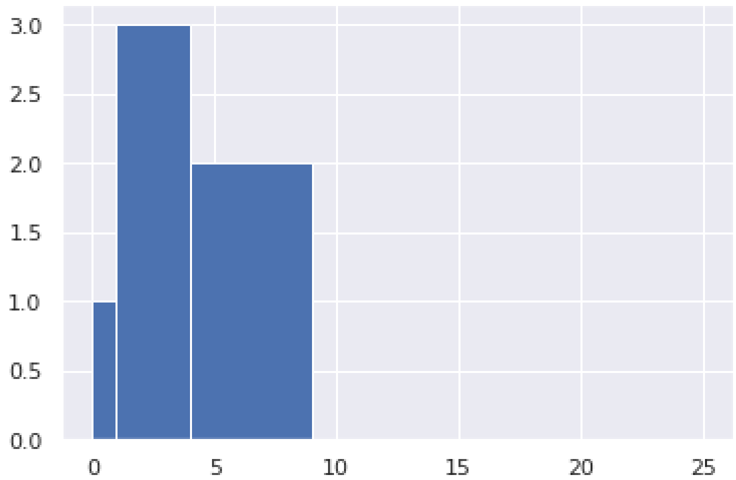

```
# Print the values a littile above
fig, ax = plt.subplots()
bars = ax.bar(x, y, color='red')
for bar in bars:
  height = bar.get_height()
  ax.text(bar.get_x() + bar.get_width()/2., 1.05*height,'%d' % int(height), ha='cer
plt.show()
```



```
# make a scatter plot
plt.scatter(x, y)
plt.show()
```

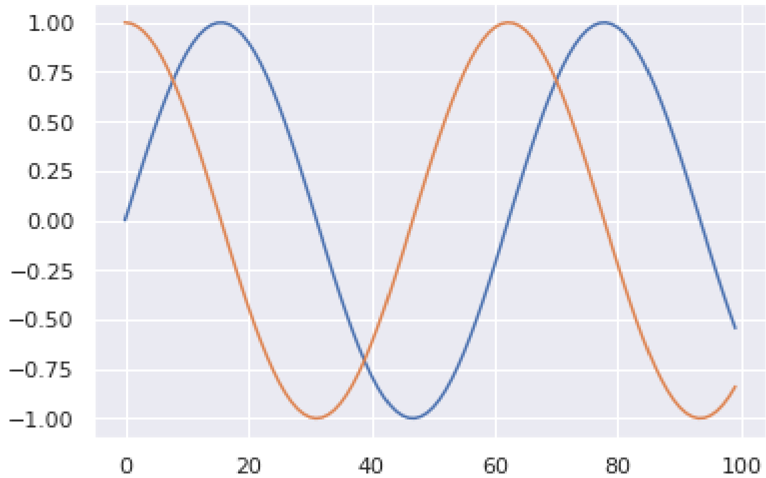



```
# make those histograms
plt.hist(x, y)
plt.show()
```

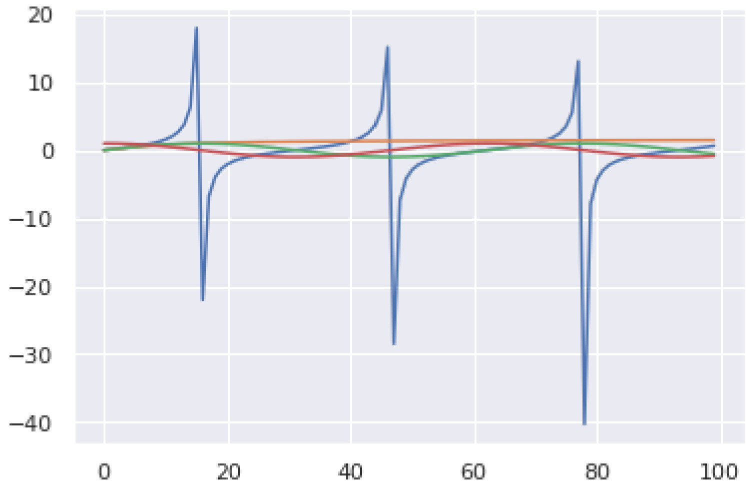



```
# make a proper histograms
plt.hist(y, bins=2)
plt.show()
```

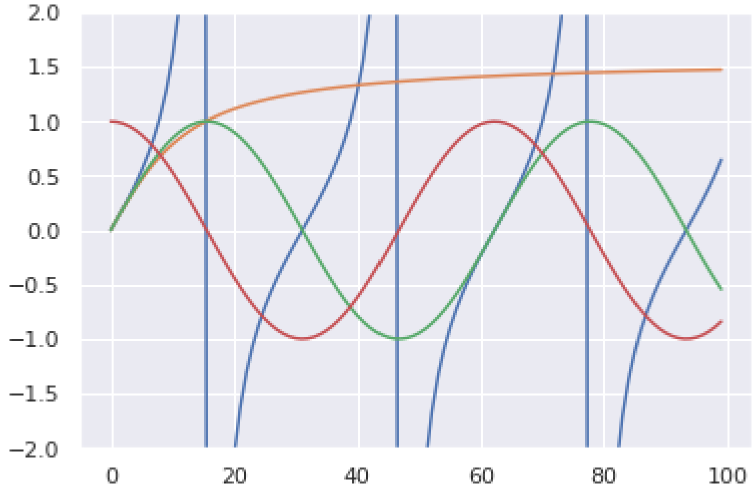

```
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.sin(x))
plt.plot(np.cos(x))
plt.show()
```



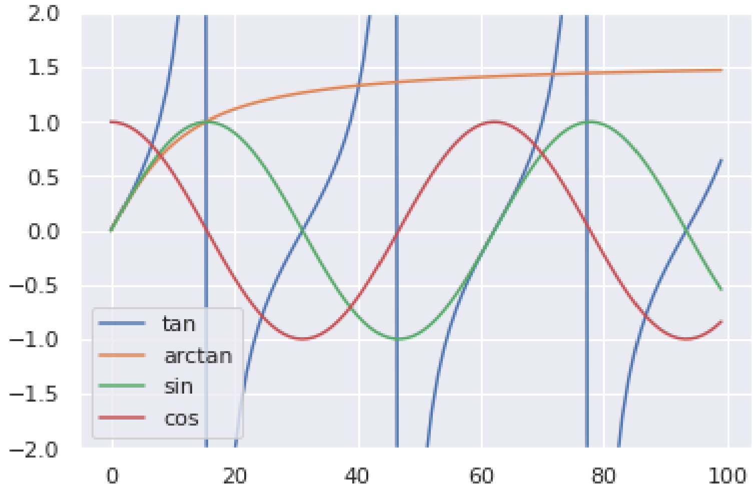

```
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.tan(x))
plt.plot(np.arctan(x))
plt.plot(np.sin(x))
plt.plot(np.cos(x))
plt.show()
```



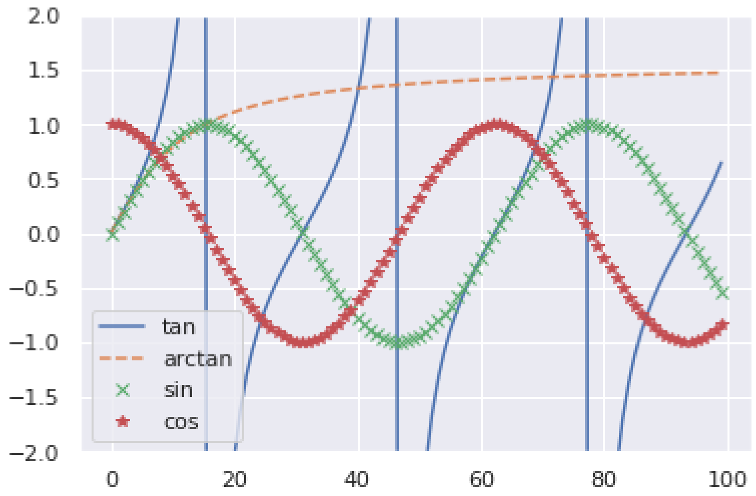

```
# make this look better
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.tan(x), label='tan')
plt.plot(np.arctan(x), label='arctan')
plt.plot(np.sin(x), label='sin')
plt.plot(np.cos(x), label='cos')
plt.ylim(-2, 2)
plt.show()
```



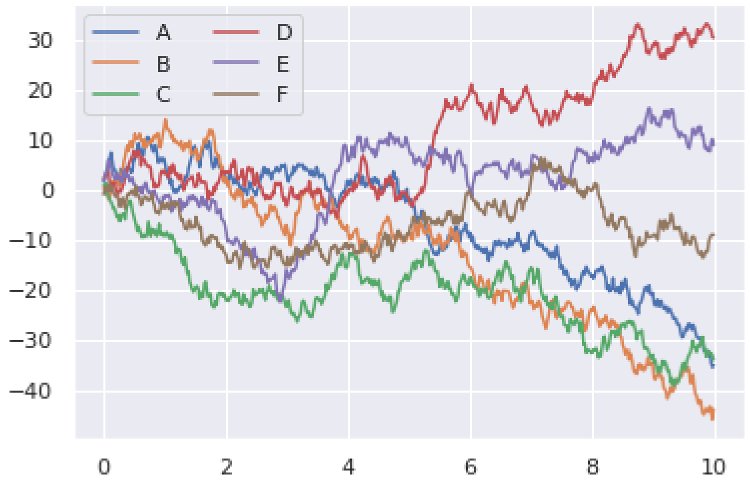

```
# lets give a legend
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.tan(x), label='tan')
plt.plot(np.arctan(x), label='arctan')
plt.plot(np.sin(x), label='sin')
plt.plot(np.cos(x), label='cos')
plt.ylim(-2, 2)
plt.legend()
plt.show()
```



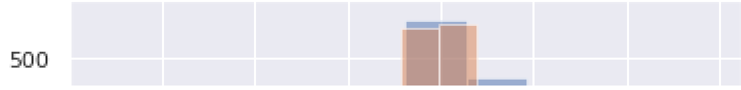

```
# change dash styles
# make this look better
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(np.tan(x), '-', label='tan')
plt.plot(np.arctan(x), '--', label='arctan')
plt.plot(np.sin(x), 'x', label='sin')
plt.plot(np.cos(x), '*', label='cos')
plt.ylim(-2, 2)
plt.legend()
plt.show()
```



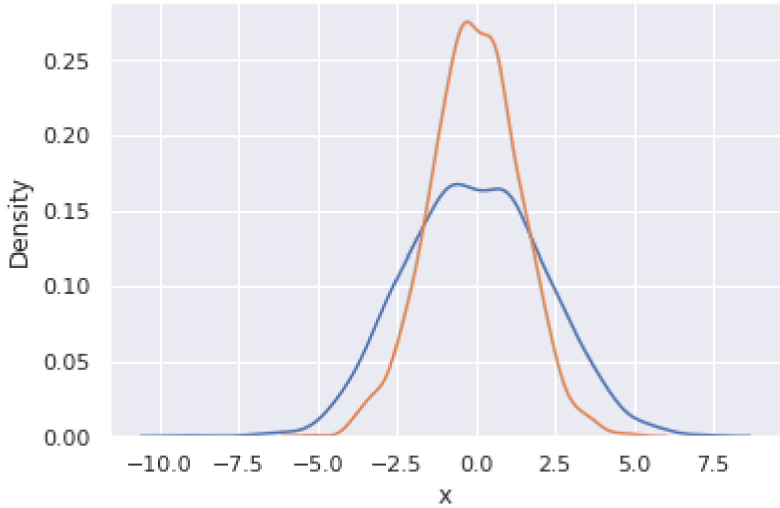

```
# generate some random data
# Create some data
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left')
plt.show()
```




```
import pandas as pd
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])
for col in 'xy':
  plt.hist(data[col], alpha=0.5)
```

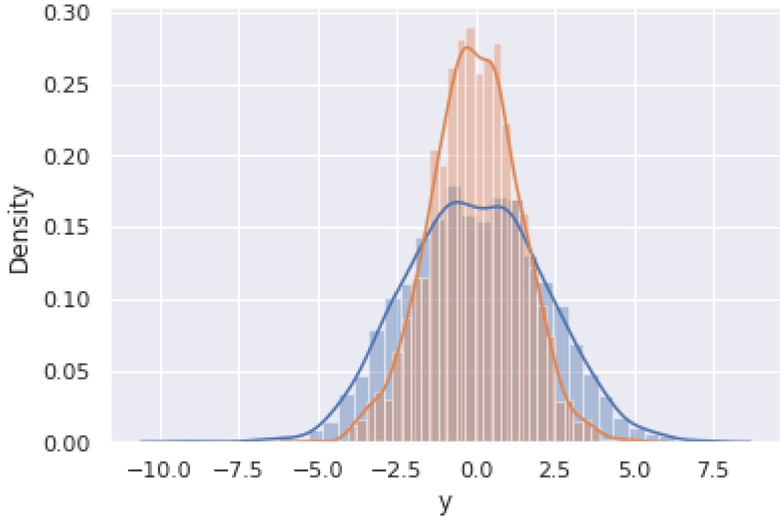



```
# KDE Plot shows the correclation between the data varaibles
#KDE Plot described as Kernel Density Estimate is used for visualizing the Probabi
# It depicts the probability density at different values in a continuous variable.
for col in 'xy':
  sns.kdeplot(data[col])
```

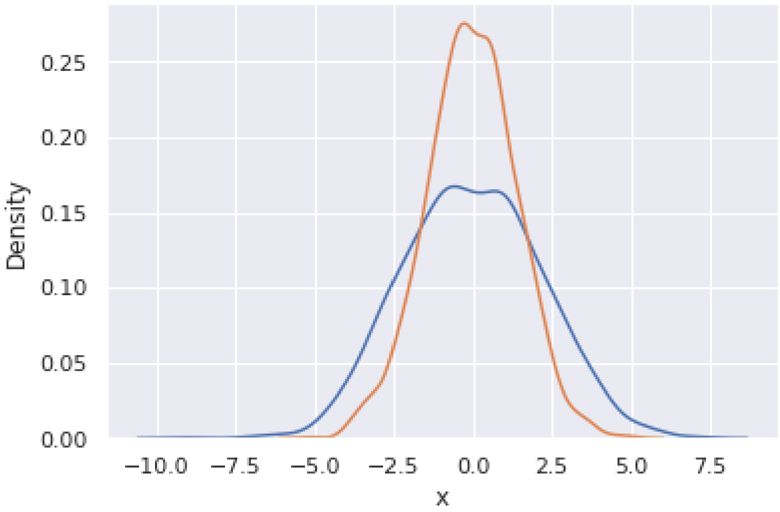



```
sns.distplot(data['x'])
sns.distplot(data['y'])
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
  warnings.warn(msg, FutureWarning)
```

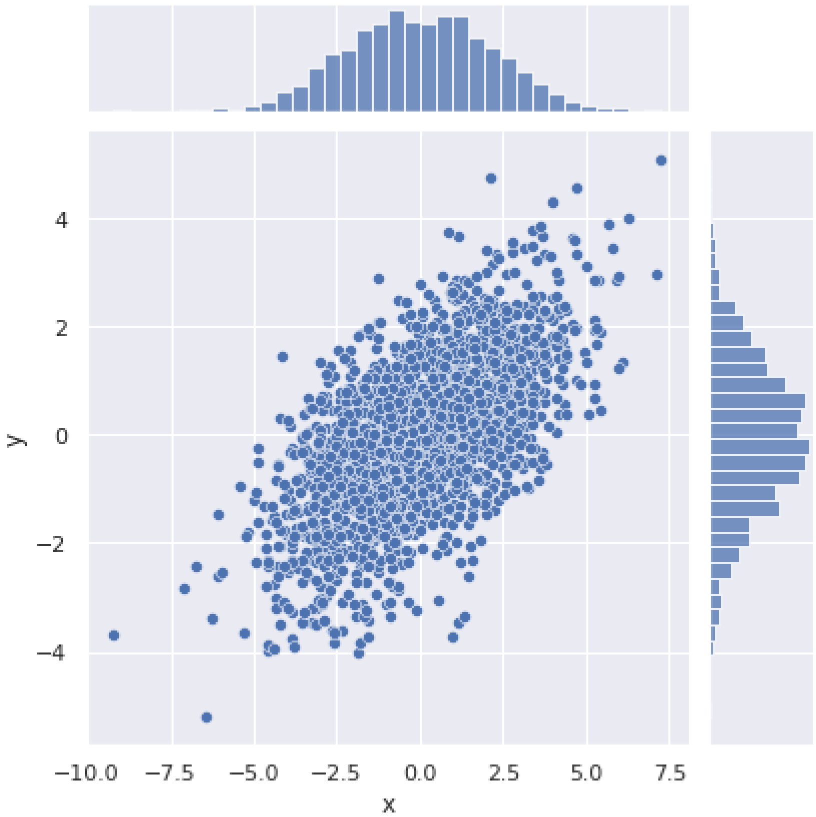



```
# MAx difference between two variables in the plot
for col in 'xy':
  sns.kdeplot(data[col])
plt.show()
```



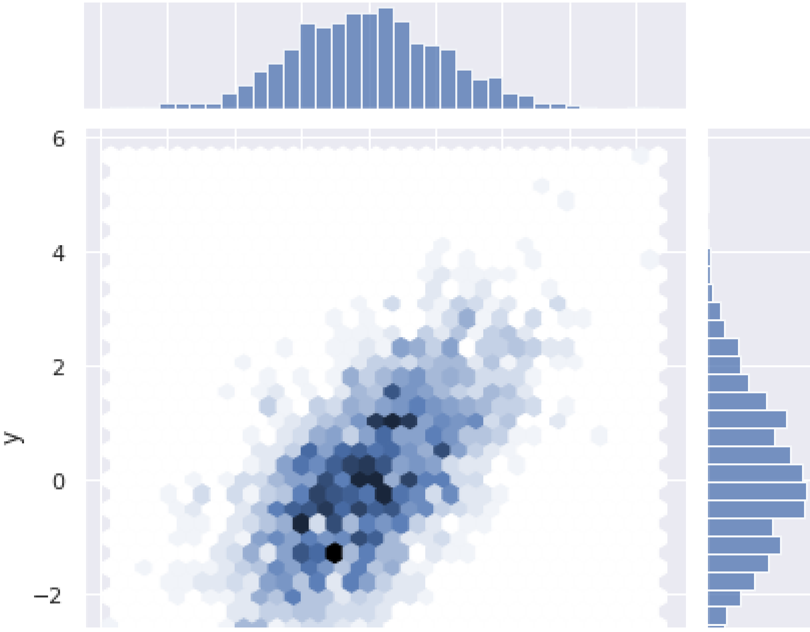

```
sns.jointplot('x', 'y', data)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
  FutureWarning
```



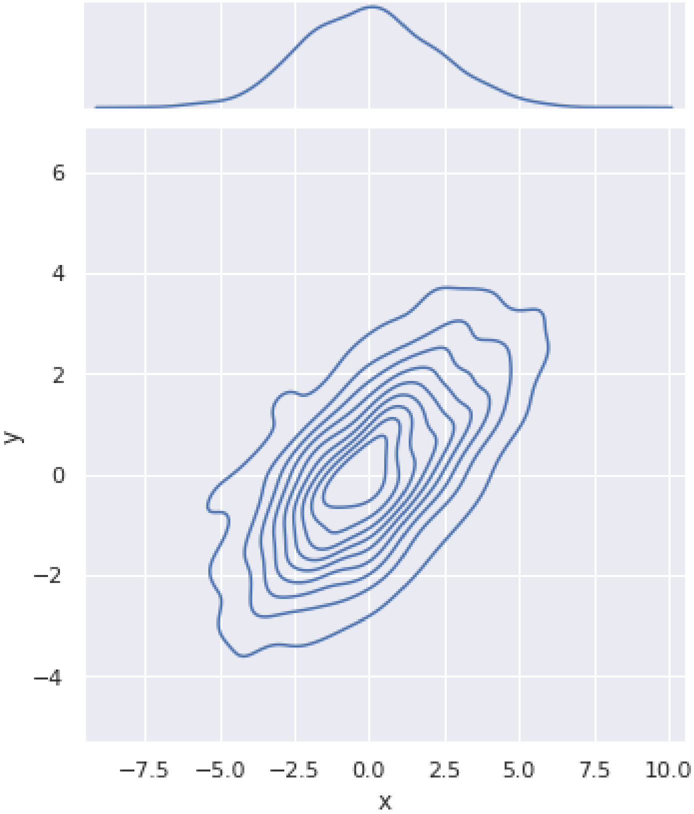

```
sns.jointplot('x', 'y', data, kind='hex')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
  FutureWarning
```




```
sns.jointplot('x', 'y', data, kind='kde')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
  FutureWarning
```



2.3 HandsOn on real dataset : IRIS Visualization

```
df = sns.load_dataset('iris')
df.head()
```

	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

```
df.shape
```

```
(150, 5)
```

```
df.info
```

```
<bound method DataFrame.info of      sepal_length  sepal_width  petal_length
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
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]>
```

```
df.describe()
```

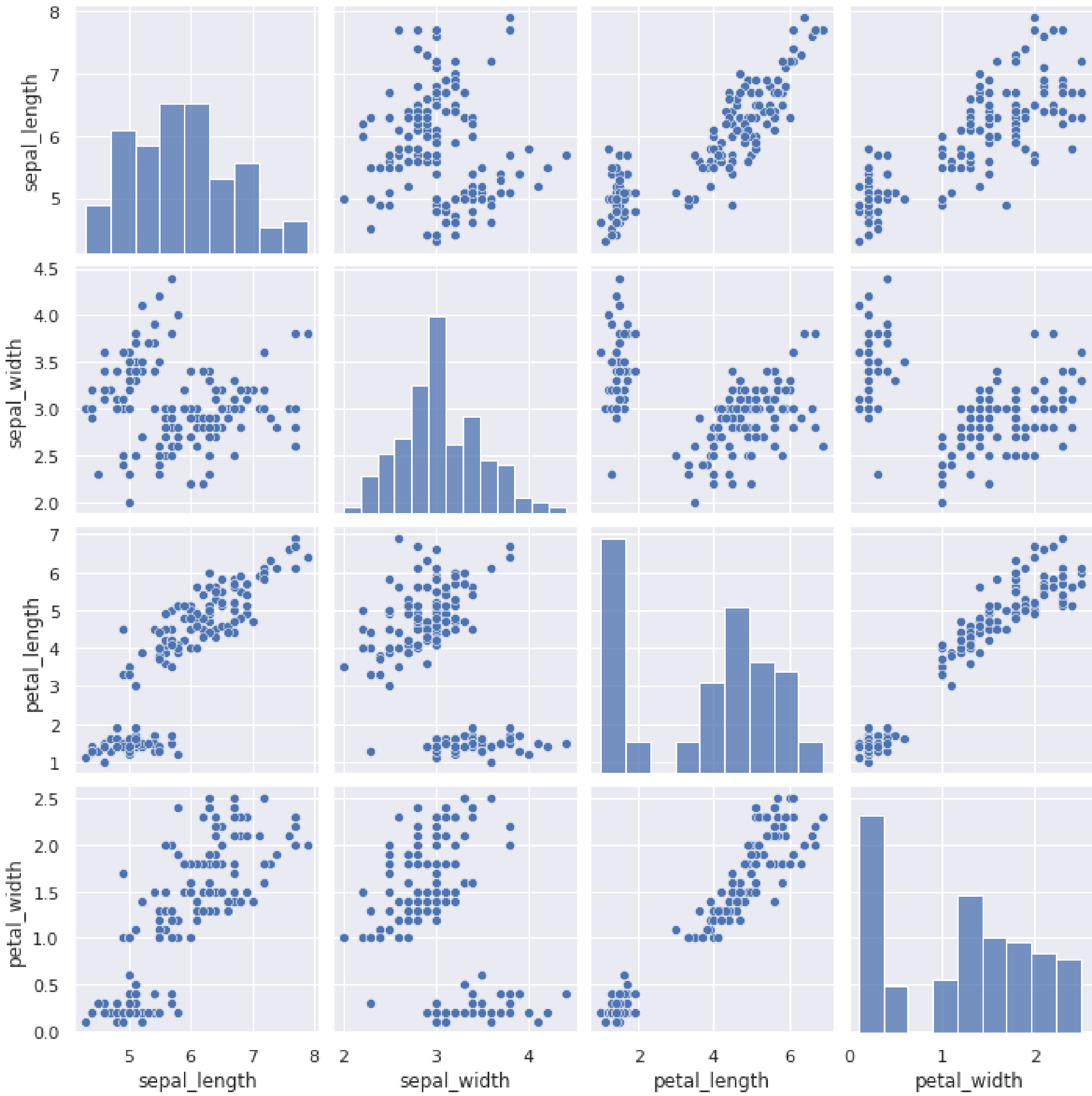
	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000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

```
df["sepal_length"].describe()
```

```
count    150.000000
mean       5.843333
```

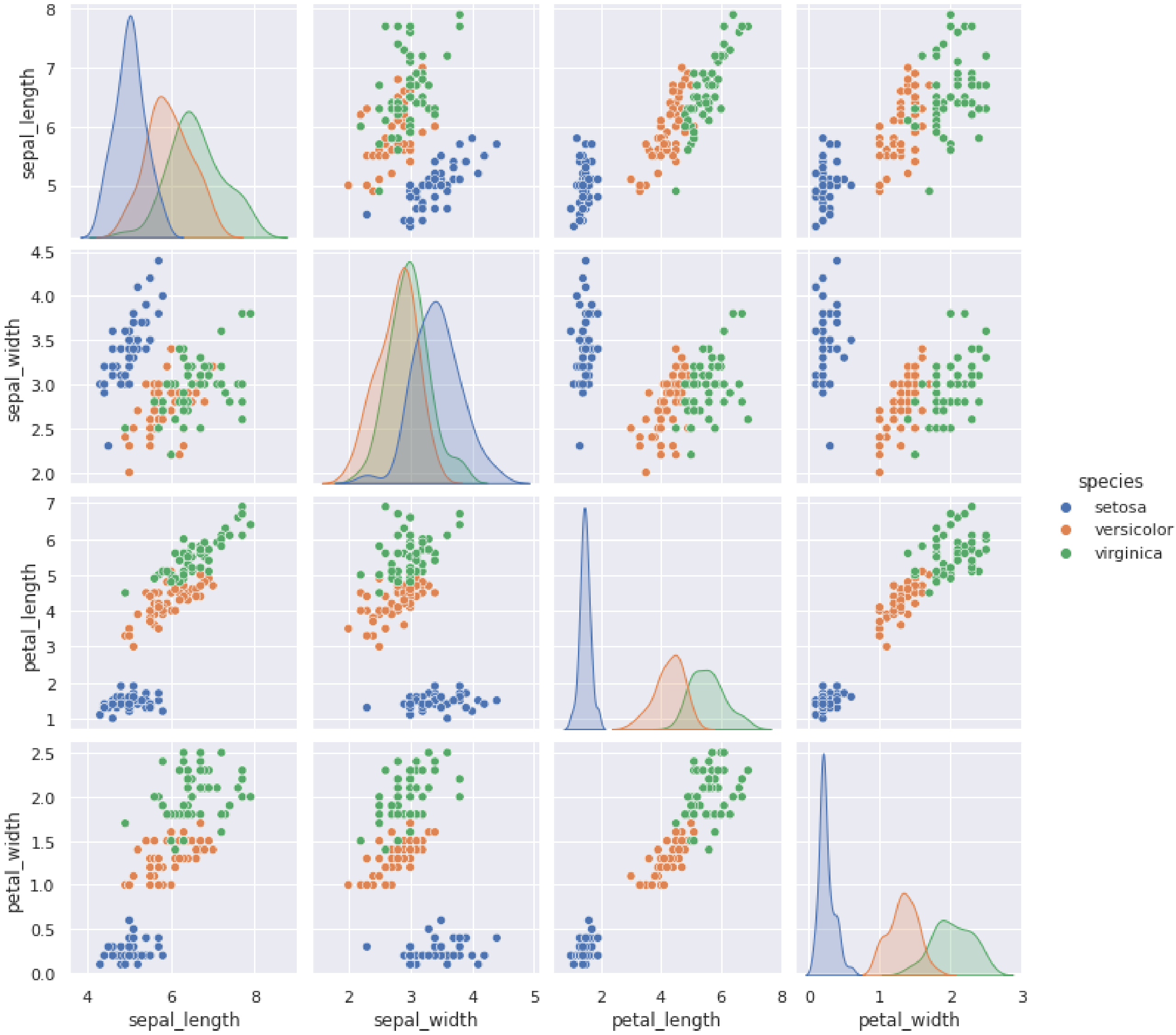
```
std        0.828066
min        4.300000
25%        5.100000
50%        5.800000
75%        6.400000
max        7.900000
Name: sepal_length, dtype: float64
```

```
sns.pairplot(df)
plt.show()
```



3 GOOD WAY TO See The relation

```
sns.pairplot(df, hue='species')
plt.show()
```

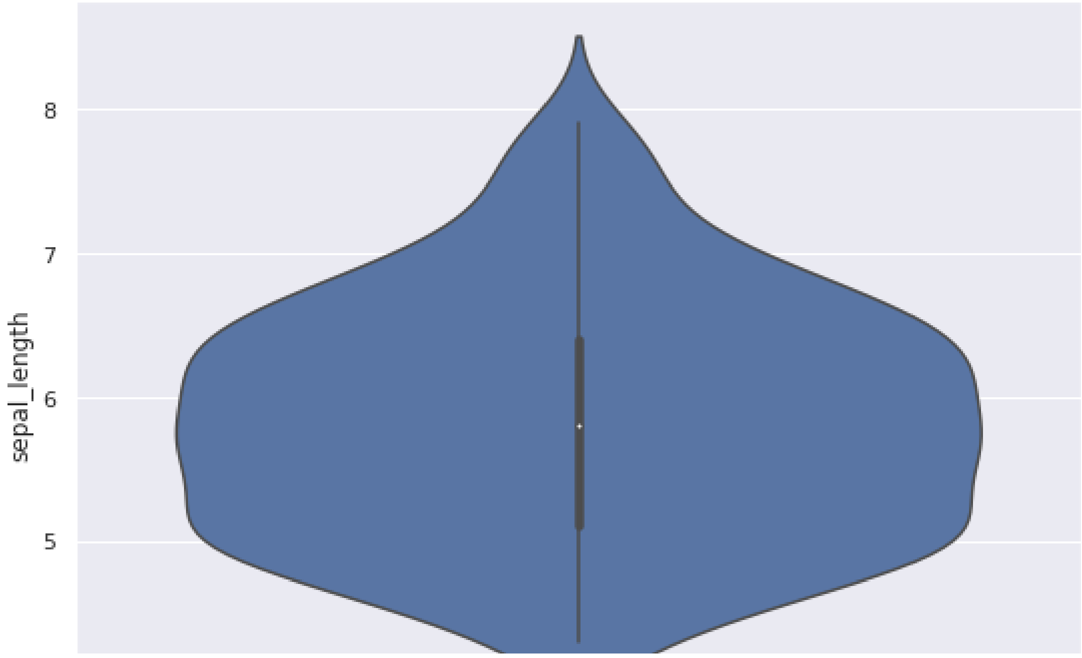


3.0.1 sns.violinplot()

Violin plot is a method of plotting numeric data. It is similar to a box plot, with the addition of a rotated kernel density plot on each side. Violin plots are similar to box plots, except that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator. Wikipedia

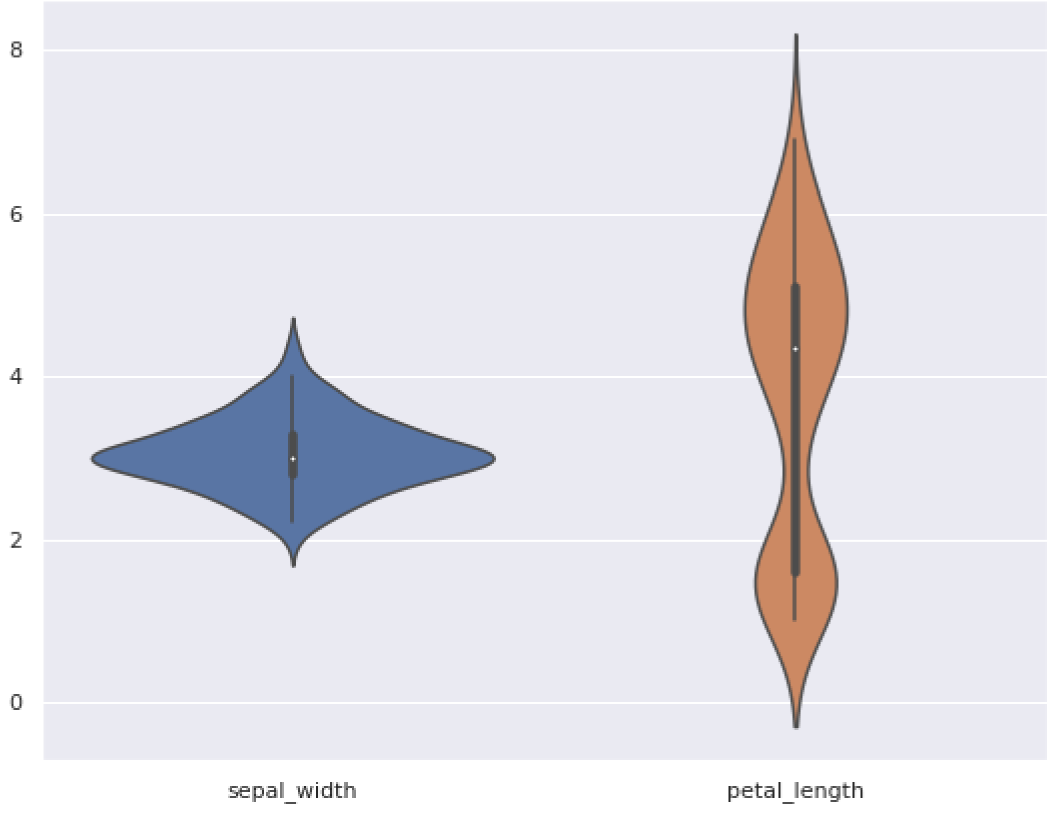
```
fig, ax = plt.subplots(figsize =(9, 7))
sns.violinplot( ax = ax, y = df["sepal_length"] )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f197e33d2e8>
```




```
fig, ax = plt.subplots(figsize =(9, 7))
sns.violinplot(ax = ax, data = df.iloc[:, 1:3])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f197e316fd0>
```




```
!wget https://www.dropbox.com/s/go8m1pbuiphzgi8/mobile_cleaned.csv
```

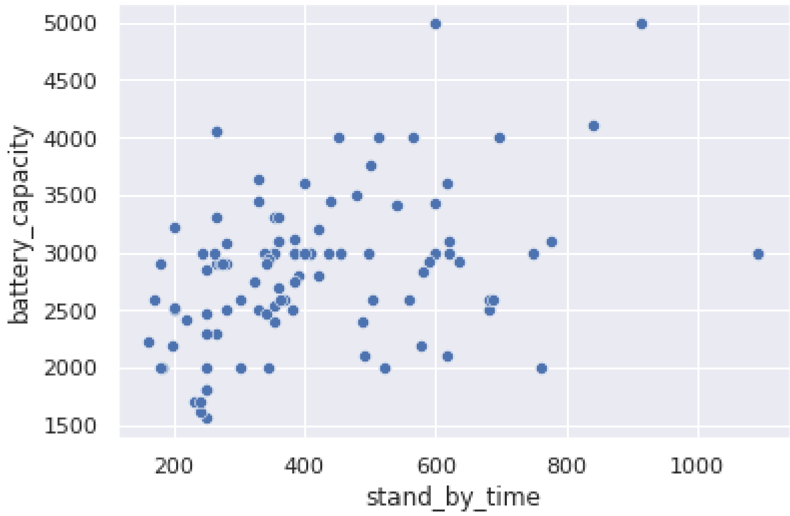
```
--2021-01-25 09:14:56--  https://www.dropbox.com/s/go8m1pbuiphzgi8/mobile_cle
Resolving www.dropbox.com (www.dropbox.com)... 162.125.1.18, 2620:100:6016:18
Connecting to www.dropbox.com (www.dropbox.com)|162.125.1.18|:443... connecte
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/go8m1pbuiphzgi8/mobile_cleaned.csv [following]
--2021-01-25 09:14:56--  https://www.dropbox.com/s/raw/go8m1pbuiphzgi8/mobile
Reusing existing connection to www.dropbox.com:443.
```

```
HTTP request sent, awaiting response... 302 Found
Location: https://uc280ea4b0f476a7c5be1b9d5891.dl.dropboxusercontent.com/cd/0
--2021-01-25 09:14:56--  https://uc280ea4b0f476a7c5be1b9d5891.dl.dropboxuserc
Resolving uc280ea4b0f476a7c5be1b9d5891.dl.dropboxusercontent.com (uc280ea4b0f
Connecting to uc280ea4b0f476a7c5be1b9d5891.dl.dropboxusercontent.com (uc280ea
HTTP request sent, awaiting response... 200 OK
Length: 14044 (14K) [text/plain]
Saving to: 'mobile_cleaned.csv'

mobile_cleaned.csv  100%[===================>]  13.71K  --.-KB/s    in 0s

2021-01-25 09:14:57 (286 MB/s) - 'mobile_cleaned.csv' saved [14044/14044]
```

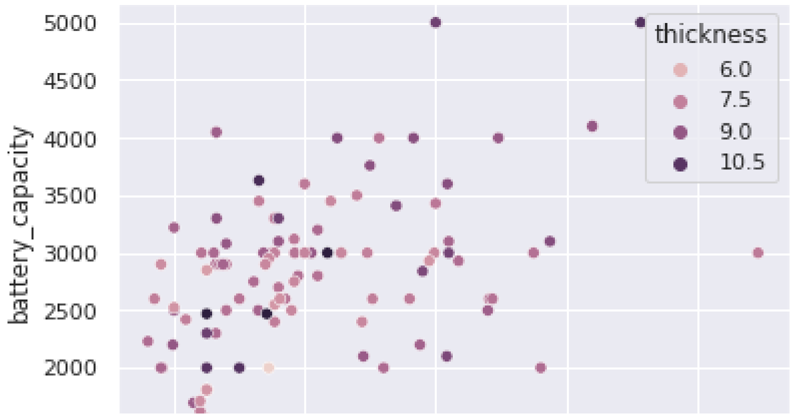
```
import pandas as pd
import seaborn as sns
data = pd.read_csv('mobile_cleaned.csv')
data.head()
```

	sim_type	aperture	gpu_rank	weight	stand_by_time	processor_frequency	t
0	0	12	55	155.0	250	1.3	
1	0	1	55	132.0	300	1.3	
2	0	9	55	142.0	329	1.5	
3	0	8	55	152.0	385	1.3	
4	1	1	55	234.0	385	1.3	

```
ax = sns.scatterplot(x="stand_by_time", y="battery_capacity", data=data)
```

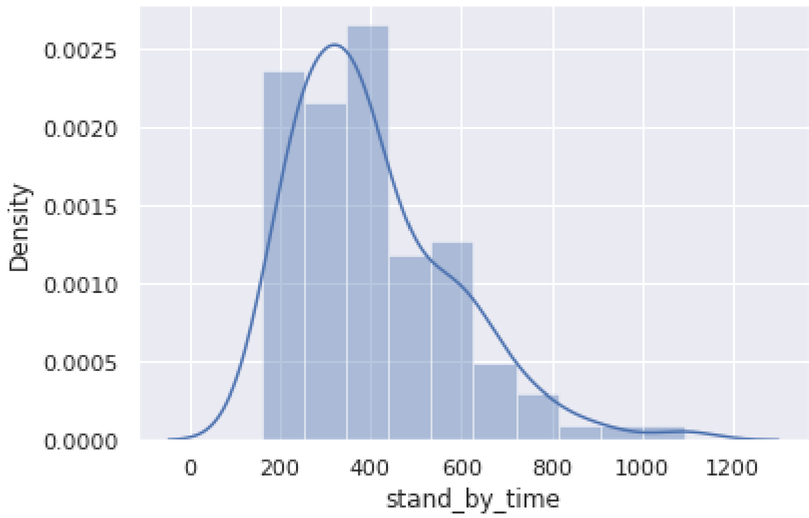



```
ax = sns.scatterplot(x = "stand_by_time", y = "battery_capacity",hue="thickness", 
```



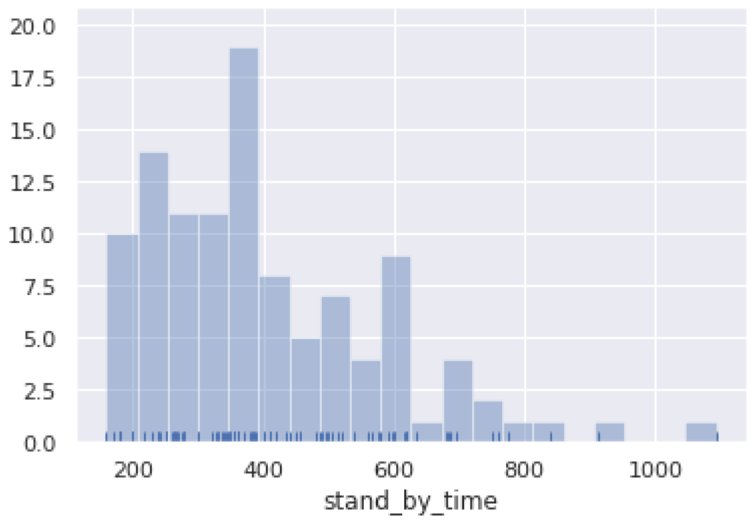

```
ax = sns.distplot(data["stand_by_time"])
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
  warnings.warn(msg, FutureWarning)
```

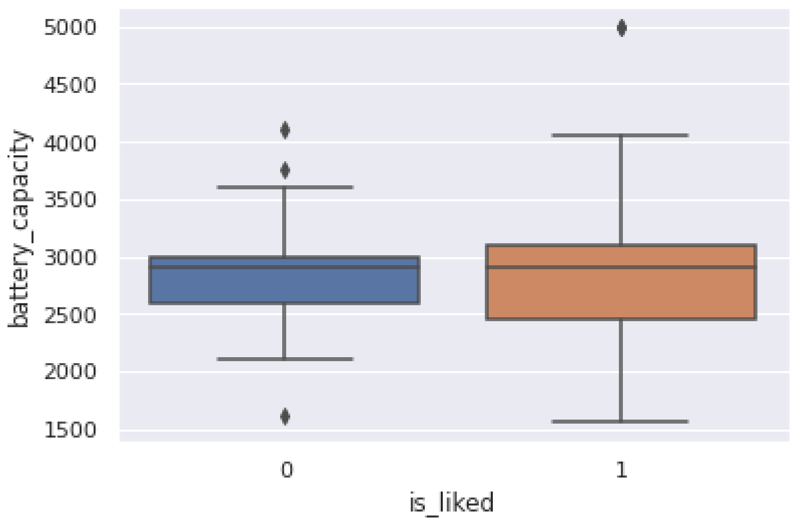



```
ax = sns.distplot(data["stand_by_time"], kde=False, rug=True, bins = 20)
```

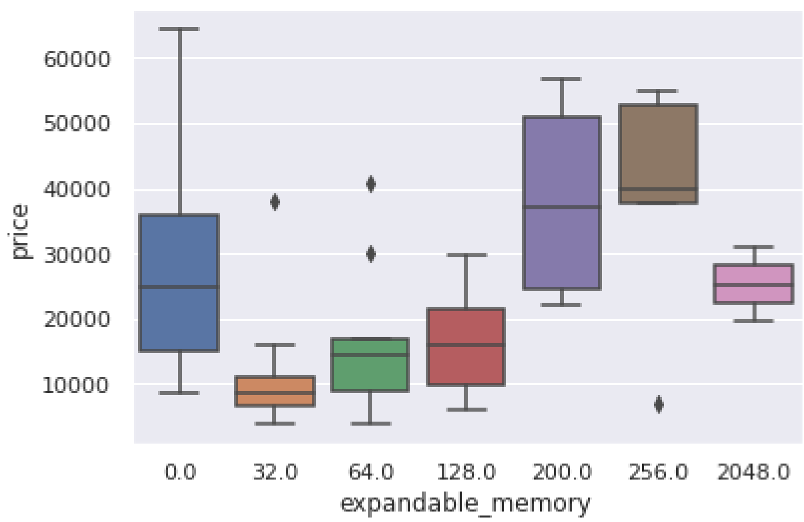
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2056: FutureW
  warnings.warn(msg, FutureWarning)
```



```
ax = sns.boxplot(x="is_liked", y="battery_capacity", data=data)
```




```
ax = sns.boxplot(x = "expandable_memory", y = "price", data=data)
```




```
import numpy as np
uniform_data = np.random.rand(10, 12)
print(uniform_data)
```

```
[[0.66598831 0.47645545 0.9499697  0.30286815 0.33934392 0.30778577
  0.10654643 0.13952205 0.62852938 0.03902754 0.32664185 0.75021389]
 [0.07711365 0.9425221  0.60933812 0.04574301 0.89950115 0.70114991
  0.23454783 0.5890388  0.07517015 0.75194119 0.11291884 0.930454  ]
 [0.92527754 0.15410303 0.99874348 0.57746576 0.34353421 0.15998263
  0.78727324 0.8215636  0.21154059 0.13545222 0.69116786 0.69224481]
 [0.16876135 0.34861695 0.57160117 0.67307943 0.32054337 0.42447693
  0.25908689 0.4257711  0.18674472 0.35468475 0.15631576 0.61308739]
 [0.08642637 0.44088432 0.02466107 0.28500906 0.23411007 0.04053709
  0.39174814 0.05915203 0.73827179 0.43616846 0.18693014 0.13718455]
 [0.71294095 0.10699125 0.7280009  0.65779964 0.05224475 0.13560743
  0.89359824 0.69705022 0.02351442 0.41560058 0.19429251 0.02188504]
 [0.6732775  0.95836003 0.38657695 0.75174124 0.1705204  0.98251248
  0.85442847 0.65146354 0.11135823 0.00593975 0.93774397 0.30351887]
 [0.05666617 0.59287496 0.83181676 0.66776415 0.35628818 0.69397796
  0.65060678 0.92710565 0.14355164 0.48846045 0.27183107 0.67494332]
 [0.45177211 0.10910486 0.95678205 0.69173706 0.67297012 0.9017492
```

```
 0.46270328 0.90352536 0.74372098 0.73582472 0.83040545 0.93799957]
[0.13089929 0.01692451 0.65975744 0.02678213 0.64300637 0.94864874
 0.27897326 0.40665158 0.51521157 0.73945579 0.9260223  0.83290822]]
```

```
ax = sns.heatmap(uniform_data, cmap="YlGnBu")
```

4 Visulization Using Images

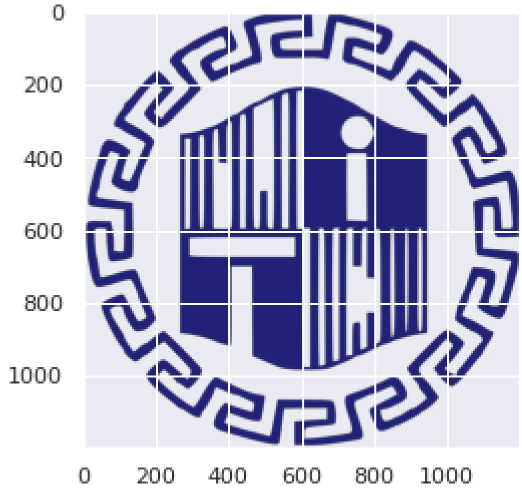
4.0.1 Image PRocessing using CV2 and PIL

```
## necessary libraries
import matplotlib.pyplot as plt
import matplotlib.image as img
from PIL import Image
import cv2
```

```
image= img.imread('NITD.jpeg')
```

```
# image= img.imread('S_1.png')
plt.imshow(image)
```

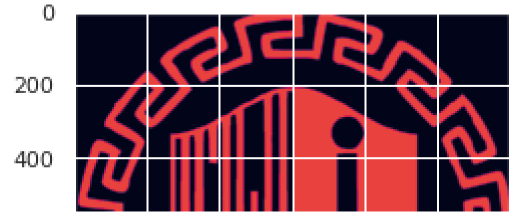
```
<matplotlib.image.AxesImage at 0x7f20eb32b4e0>
```




```
# img= Image.open('S_1.png').convert('LA')
img = Image.open('NITD.jpeg').convert('L')
img.save('greyscale.png')
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f20eb30c630>
```




```
print('size of the image (width, height):',img.size)
```

```
size of the image (width, height): (1200, 1200)
```

Note: anti-aliasing is a technique for minimizing the distortion artifacts known as aliasing when

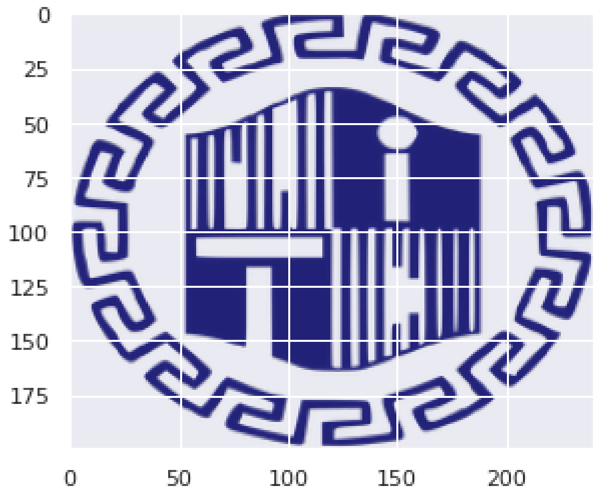
representing a high-resolution image at a lower resolution.

```
# img = Image.open('S_1.png')
img = Image.open('NITD.jpeg')
img = img.resize((240, 200), Image.ANTIALIAS)
img.size
```

```
(240, 200)
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f20eb2ed978>
```




```
! pip install opencv-python
```

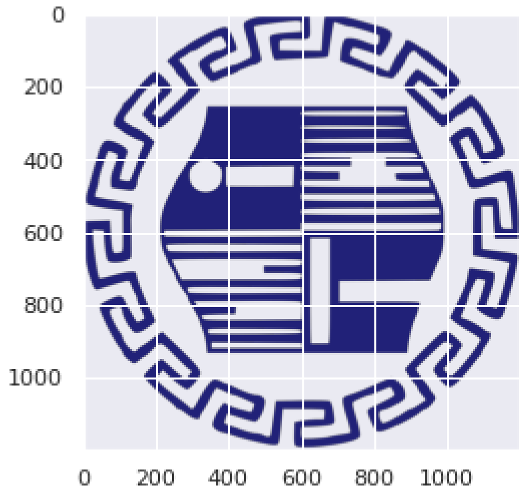
```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.6/dist
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist
```

4.0.5 3. Rotate image

```
# im = Image.open("S_1.png")
im = Image.open("NITD.jpeg")
```

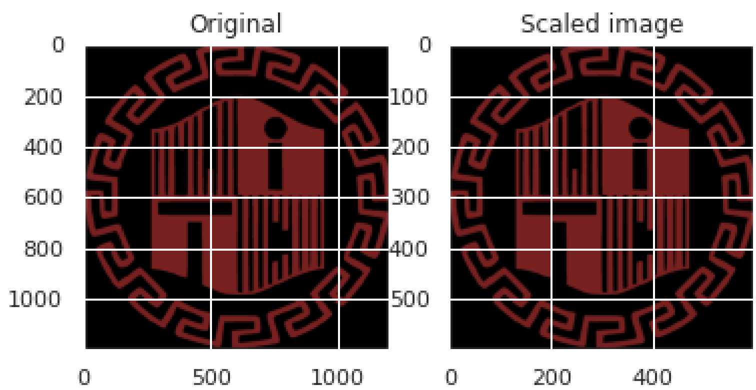
```
im = Image.open('NITD.jpeg')
img=im.rotate(90)


plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f20eb48ce10>
```




```
# img = cv2.imread('keras_sample.jpg')
img = cv2.imread('NITD.jpeg')


# Get number of pixel horizontally and vertically.
(height, width) = img.shape[:2]


# Specify the size of image along with interploation methods.
# cv2.INTER_AREA is used for shrinking, whereas cv2.INTER_CUBIC
# is used for zooming.
res = cv2.resize(img,(int(width / 2), int(height / 2)), interpolation = cv2.INTER_A
# Write image back to disk.
plt.subplot(121), plt.imshow(img),plt.title('Original')
plt.subplot(122), plt.imshow(res),plt.title('Scaled image')
```

```
(<matplotlib.axes._subplots.AxesSubplot at 0x7f20fe788048>,
 <matplotlib.image.AxesImage at 0x7f20fe882eb8>,
 Text(0.5, 1.0, 'Scaled image'))
```



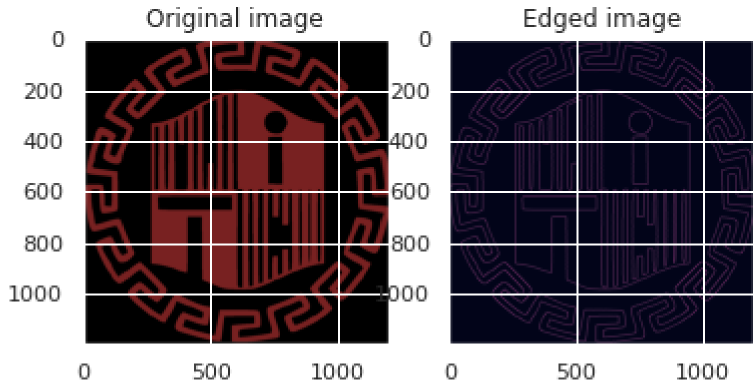

```
img.shape
```

```
(1200, 1200, 3)
```

4.0.7 Edge Detection

```
# Read image from disk.
img = cv2.imread('NITD.jpeg')
#img = cv2.imread('recess.png')
# Canny edge detection.
edges = cv2.Canny(img, 100, 200)
# Write image back to disk.
plt.subplot(121), plt.imshow(img),plt.title('Original image')
plt.subplot(122), plt.imshow(edges),plt.title('Edged image')
```

```
(<matplotlib.axes._subplots.AxesSubplot at 0x7f210d00cc50>,
 <matplotlib.image.AxesImage at 0x7f210d004da0>,
 Text(0.5, 1.0, 'Edged image'))
```



4.0.8 4. Applying Noise

```
import cv2
from matplotlib import pyplot as plt
%matplotlib inline
```

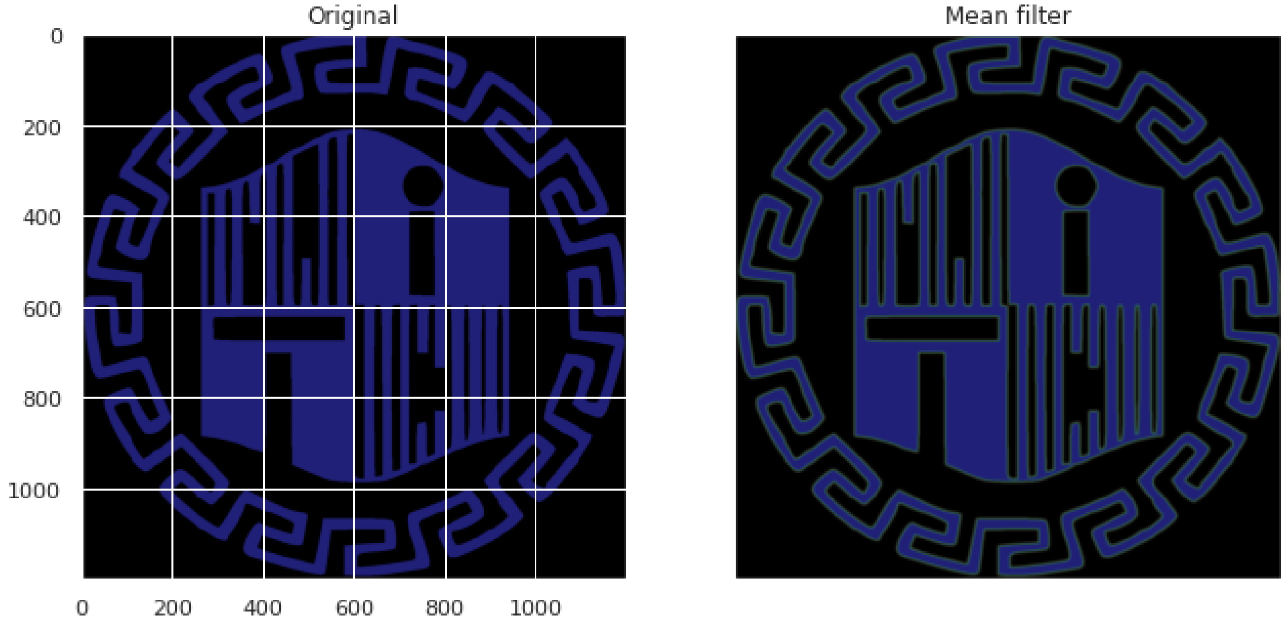
Color vision can be processed using RGB color space or HSV color space. RGB color space describes colors in terms of the amount of red, green, and blue present. HSV color space describes colors in terms of the Hue, Saturation, and Value. In situations where color description plays an integral role, the HSV color model is often preferred over the RGB model. The HSV model describes colors similarly to how the human eye tends to perceive color. RGB defines color in terms of a combination of primary colors, where as, HSV describes color using more familiar comparisons such as color, vibrancy and brightness.

The purpose of using plt.figure() is to create a figure object.

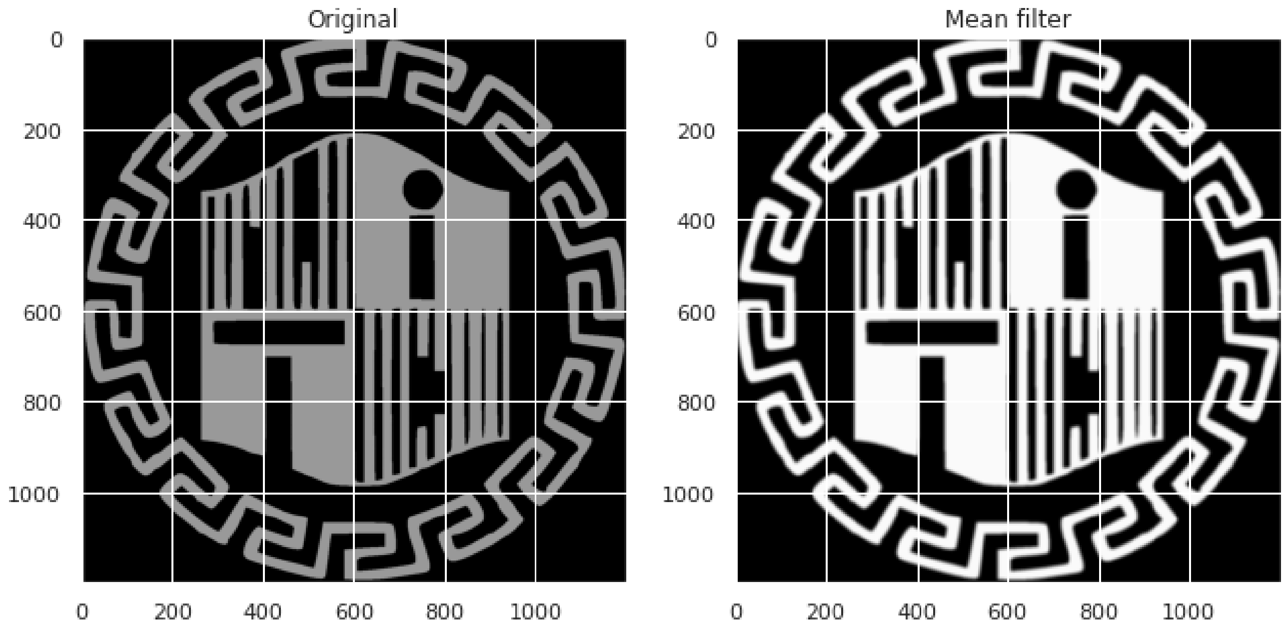
The whole figure is regarded as the figure object. It is necessary to explicitly use plt.figure() when we want to #tweak the size of the figure and when we want to add multiple Axes objects in a single figure.

```
image = cv2.imread('NITD.jpeg') # reads the image
image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV) # convert to HSV
figure_size = 9
```

```
new_image = cv2.blur(image,(figure_size, figure_size))
plt.figure(figsize=(11,6))
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_HSV2RGB)),plt.title('Or
plt.subplot(122), plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_HSV2RGB)),plt.title
plt.xticks([]), plt.yticks([])
plt.show()
```




```
# The image will first be converted to grayscale
image2 = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
figure_size = 9
new_image = cv2.blur(image2,(figure_size, figure_size))
plt.figure(figsize=(11,6))
plt.subplot(121), plt.imshow(image2, cmap='gray'),plt.title('Original')
plt.subplot(122), plt.imshow(new_image, cmap='gray'),plt.title('Mean filter')
plt.show()
```



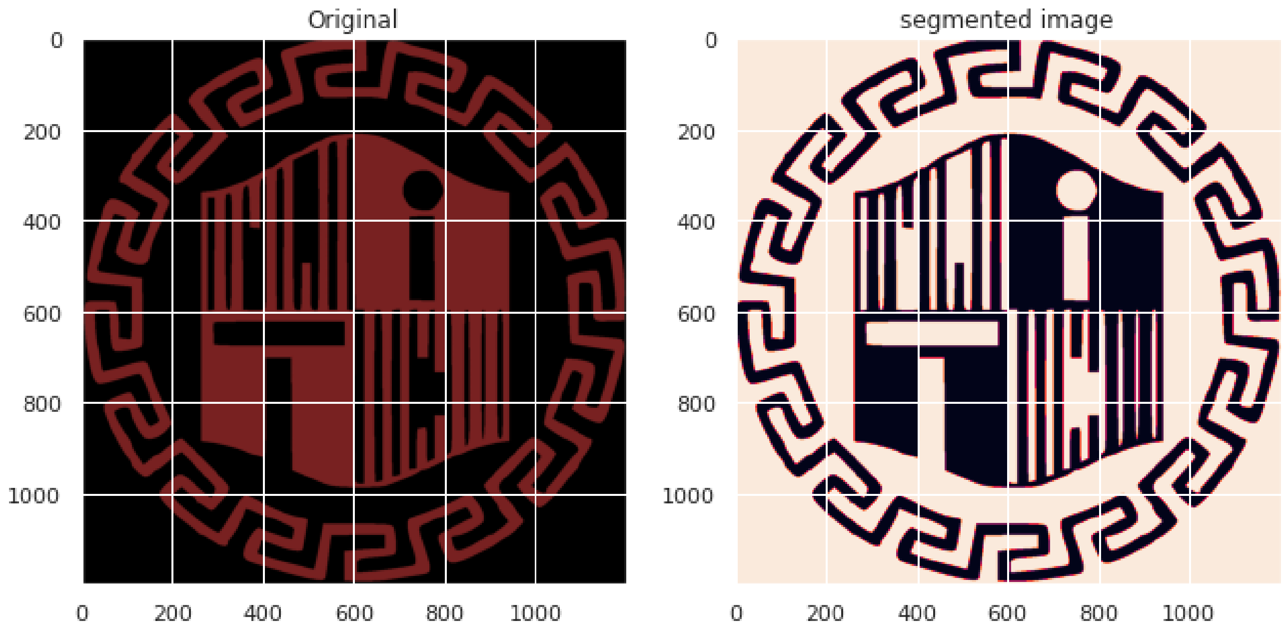
```
# reference: https://towardsdatascience.com/image-filters-in-python-26ee938e57d2
```

4.0.9 5. Segmentation

Image Segmentation. Image segmentation is the process of partitioning an image into multiple segments. Image segmentation is typically used to locate objects and boundaries in images.

```
image = cv2.imread('NITD.jpeg') # reads the image
# image = cv2.imread('S_1.png')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
plt.figure(figsize=(11,6))
# Displaying segmented images
plt.subplot(121), plt.imshow(image),plt.title('Original')
plt.subplot(122), plt.imshow(thresh),plt.title('segmented image')
```

```
(<matplotlib.axes._subplots.AxesSubplot at 0x7f20fe7b4b00>,
 <matplotlib.image.AxesImage at 0x7f20fd984ba8>,
 Text(0.5, 1.0, 'segmented image'))
```




```
# Reference: https://www.geeksforgeeks.org/python-thresholding-techniques-using-ope
```

4.0.10 6. Morphology

```
import numpy as np
image = cv2.imread('NITD.jpeg') # reads the image
# image = cv2.imread('recess.png') # reads the image
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
# defining the range of masking
blue1 = np.array([110, 50, 50])
blue2 = np.array([130, 255, 255])
# initializing the mask to be
# convoluted over input image
mask = cv2.inRange(hsv, blue1, blue2)
# passing the bitwise_and over
```

```
# each pixel convoluted
res = cv2.bitwise_and(image, image, mask = mask)
# defining the kernel i.e. Structuring element
kernel = np.ones((5, 5), np.uint8)
# defining the opening function
# over the image and structuring element
opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

plt.figure(figsize=(11,6))
plt.subplot(121), plt.imshow(image),plt.title('Original')
plt.subplot(122), plt.imshow(mask),plt.title('segmented image')
```

```
(<matplotlib.axes._subplots.AxesSubplot at 0x7f20fd9554e0>,
 <matplotlib.image.AxesImage at 0x7f20eb289710>,
 Text(0.5, 1.0, 'segmented image'))
```