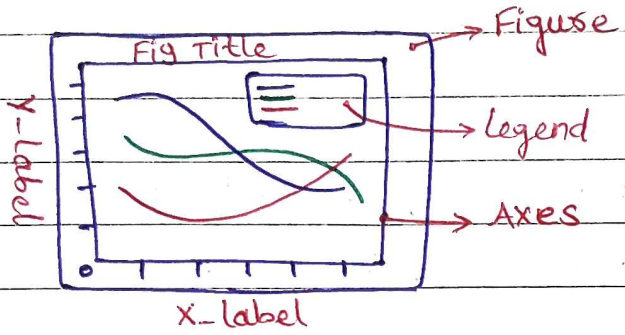
Notes

__/__/____

Matplotlib

- Matplotlib Graphs your data on **Figures**, each of which contain one or more **Axes**, an area where points can be specified in terms of x-y coordinates.



- # Installing Matplotlib :

```
! pip install matplotlib      → In Jupyter Notebook
  pip install matplotlib      → In Command line (or) CMD.
```

Importing Matplotlib :

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

Note : In Matplotlib we have two types of plotting techniques are these,

1. Implicit plotting Technique
2. Explicit plotting Technique (Recommended by Matplotlib)

- 4-steps to create plots :

1. Initialize figure
2. Generate plots
3. Labelling the plots
4. Annotating the plots
5. Display the figure.

__/__/____

Implicit plotting Technique :

Initialize a Figure :

```
plt.figure (Figsize = (3,3), constrained_layout = True)
```

Generate plots :

```
plt.plot (data1, data2, 'bo', label = 'scatter')
plt.plot (data1, data2, 'k-', label = 'line')
```

Note : Here, 'bo', 'k-' are format strings:
A Format string consists of a part for color, markers and line:
fmt = '[markers][line][colors]'
Each of them is Optional. If not provided, the value from the style cycle is used.
Exception : If line is given, but no markers, the data will be a line without markers.

Other combinations such as [colors][markers][line] are also supported, but note that their passing may be ambiguous.

- Markers :

'.'	point	'1'	tri_down
','	pixel	'2'	tri_up
'o'	circle	'3'	tri_left
'v'	triangle	'4'	tri_right
'^'		'8'	Octagon
'<'		'*'	star etc...
'>'			

__/__/____

Line styles :

'-' - solid line style.
'--' - dashed line style.
'-.' - dash-dot line style.
':' - dotted line style.

Colors :

'b' - Blue
'g' - Green
'r' - Red
'c' - cyan
'm' - magenta
'y' - yellow
'k' - black
'w' - white.

Labelling the plot :

```
plt.title("Lineplot")
plt.xlabel('x label')
plt.ylabel('y label')
```

putting limits to x & y :

```
plt.xlim([0, 1])
plt.ylim([0, 1])
```

Rotate x-ticks :

```
plt.xticks(rotate=45)
```

__/__/____

Annotating :

```
plt.text(s='My text', x=0.5, y=0.6)        (or)
plt.text(s='My text', x=10, y=50, ha='left', color='red')
                                   'right'   'green'
                                   'center'  'blue'
                                   :
```

(or)

```
plt.annotate('My text', xytext=(-2, 3.5), color='red',
             xy=(-0.9, 2.8),
             arrowprops={'width': 6, 'headwidth': 15,
                         'headlength': 15,
                         'arrowstyle': '->',
                         'connectionstyle': 'arc3, rad=0.3'})
```

Display the legend :

```
plt.legend()
```
← Display on the image.

```
plt.legend(bbox_to_anchor=(1, 1))
```
← Display the legend outside the Image

Display the Grid :

```
plt.grid(True)
```

Display the figure :

```
plt.show()
```

__/__/____

Creating Subplots - Single and Mutiple Axes :

Multiple Axes in a single figure.

Initializing a Figure :

```
plt.figure ( figsize = (5,3), constrained_layout = True )
```

creating a plot with 1 row and 2 columns :

```
plt.subplot(1, 2, 1)
plt.title ('y = log(x)')
plt.plot (data1, np.log(data1), label = 'log', color = 'green')
plt.legend ( )

plt.subplot (1, 2, 2)
plt.title ('y = exp(x)')
plt.plot (data1, np.exp(data1), label = 'exp', color = 'violet')
plt.legend ( )
plt.xlabel ('x')
plt.ylabel ('exp(x)')


plt.show ( )
```

__/__/____

Explicit plotting Technique :

- creating Figure :

```
fig = plt.figure (figsize = (2,2), constrained_layout = True)
plt.show()
```

- creating Figure & Adding Axes (Explicitly) :

Syntax - 1 :

```
# Initializing figure without Axes.

fig = plt.figure (figsize = (3,3), facecolor = 'light yellow')

# Adding Axes (Generates 1 axes or subplot)

ax = fig.add_subplot( )

# Display the figure
plt.show( )
```

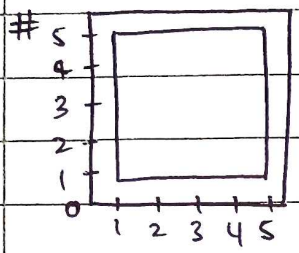
Syntax - 2 :

```
# Initializing a figure (without Axes)

fig = plt.figure (figsize = (3,3), facecolor = 'light skyblue')

# Adding Axes (Generates 1 axes (or) subplot)
ax = fig.add_subplots (1,1 )
# Display the figure
plt.show( )
```

__/__/____

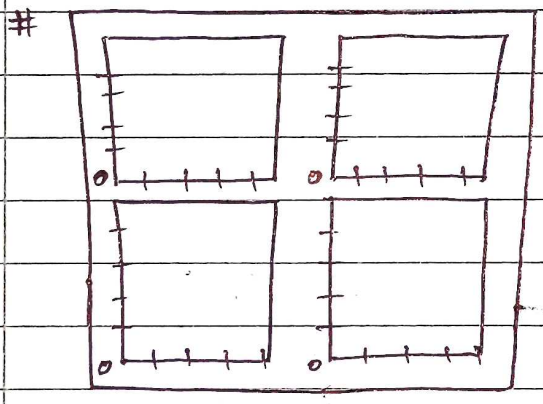
Syntax-2a :

```
fig = plt.figure(figsize = (3,3), facecolor = 'lightskyblue')

ax = fig.subplots()
plt.show()
```



Syntax-2b :

```
fig = plt.figure(figsize = (3,3), facecolor = 'lightskyblue')

ax = fig.subplots(2, 2)
plt.show()
```



Creating Figure and Axes using Subplots :

```
fig, axs = plt.subplots(2, 3, figsize = (5,5),
                        facecolor = 'lightskyblue',
                        constrained_layout = True)
plt.show()
```

__/__/____

Labeling and Annotating subplots :

```
fig, ax = plt.subplots(figsize=(3,3), facecolor='lightskyblue')
fig.suptitle("Figure Title")

ax.set_title("Axes Title", loc='left', fontsize='small')

ax.grid(True)

ax.set_facecolor("yellow")

ax.annotate('My text', xytext=(0.2,0.2), ha='center',
            xy=(0.5,0.5), arrowprops={})

plt.show()
```

Add Horizontal & Vertical line :

```
ax.axhline(1, ls='--', color='red')
ax.axvline(1, ls='--', color='black')
```

Using Helper Methods :

```
def my_plotter(axes, data1, data2, kind)
    if kind=='line':
        out = axes.plot(data1, data2)
    else:
        out = axes.scatter(data1, data2)
    return out

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(5,2.7))
```

__/__/____

```
my_plotter(ax1, data1, data2, kind='line')
my_plotter(ax2, data1, data2, kind='scatter')
plt.show()
```

Customizing Matplotlib with style sheets:

- List of all the Available styles:

```
print(plt.style.available)
```

Temporary Styling:

```
with plt.style.context('ggplot'):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(5, 2.7))
    my_plotter(ax1, data1, data2, kind='line')
    my_plotter(ax2, data1, data2, kind='scatter')
plt.show()
```

permanent styling: (complete Notebook)

```
# write the following code in the beginning after
  importing pyplot.
plt.style.use('ggplot')

# following code will reset the style to defaults.

plt.rcdefaults()
```

__/__/____

Saving the plot :

- Supported files :

```
print ( fig.canvas.get_supported_filetypes( ))
```

- dpi controls the Resolution of the output.

```
fig.savefig ('path/fig_name.png', dpi = 80 )
fig.savefig ('path/fig_name.png', dpi = 800 )
```

- transparent makes the Background of the saved figure transparent if the format supports it.

```
fig.savefig ('path/fig_name.png', dpi = 800, transparent = True)
```

- # bbox_inches = 'tight' fits the bounds of the figure to our plot

```
fig.savefig ('path/fig_name.png', transparent = True,
             dpi = 80 , bbox_inches = 'tight' )
```

Basic Matplotlib plots :

- Univariate plotting for Numerical columns - Histogram, Box

```
fig, ax = plt.subplots (figsize=(5,5))

ax.hist (df['col-3'] )

ax.boxplot (df['col-2'])
```

__/__/____

- univariate plotting for categorical columns - Bar plot, pie chart

```
ax.bar(df['col-2'].value_counts(),
bar_plot_data = df['col-2'].value_counts(normalize=True)
ax.pie(bar_plot_data.values, labels=bar_plot_data.index,
       autopct='%1.1f%%')
```

- Bi-variate plotting for Num vs Num - Scatter plot, Hexbin plot

```
ax.scatter(df['col-1'], df['col-3'])

hb1 = ax.hexbin(df['col-3'], df['col-4], bins=)
fig.colorbar(hb1, ax=ax)
```

Advance plotting with Matplotlib & Seaborn

Install seaborn :

pip install Seaborn - In command line (cmd)

Import seaborn :

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
df = pd.read_csv('data/tips.csv')
df
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	NO	Sun	Dinner	2
1	10.34	1.66	male	NO	Mon	Dinner	3
2	21.01	3.50	male	yes	Sat	Lunch	3
3	23.68	3.31	male	yes	fri	Dinner	2
4	24.59	3.61	Female	NO	Sun	Dinner	4

Uni-variate plotting for Numerical columns :

- Histogram :

```
sns.histplot(data = df, x = 'total_bill', ax = )
```

- Box plot :

```
sns.boxplot(data = df, x = 'total_bill', ax = )
```

- KDE :

```
sns.kdeplot(data = df, x = 'total_bill', ax = )
```

__/__/____

- ecdf plot :

```
sns.ecdfplot(data = df, x = 'total_bill', ax = )
```

- Boxen plot :

```
sns.boxenplot(data = df, x = 'total_bill', ax = )
```

Example :

```
fig, axs = plt.subplots(1, 5, figsize = (10, 3), constrained_layout = True)
fig.suptitle('visualization with Seaborn')

axs[0].set_title('Histogram')
sns.histplot(data = df, x = 'total_bill', ax = axs[0])

axs[1].set_title('KDE')
sns.kdeplot(data = df, x = 'total_bill', ax = axs[1])

axs[2].set_title('ECDF')
sns.ecdfplot(data = df, x = 'total_bill', ax = axs[2])

axs[3].set_title('Box plot')
sns.boxplot(data = df, x = 'total_bill', ax = axs[3])

axs[4].set_title('Boxen plot')
sns.boxenplot(data = df, x = 'total_bill', ax = axs[4])
```

__/__/____

Uni-variate plotting for Categorical columns : count plot

```
sns.countplot(data = df , x = 'day' , ax = )
```

Bi-variate plotting For Num vs Num columns :-

- scatter plot :

```
sns.scatterplot(data = df , x = 'total_bill' y = 'tip' ,ax = ax )
```

- line plot :

```
sns.lineplot(data = df , x = 'total_bill', y = 'tip' ,ax = )
```

- Heatmap :

```
sns.heatmap(df.select_dtypes(include = ['int64', 'float64']).corr(
            , annot = True , linewidths = 0.8, ax = axs[2])
```

- pair-plot : (Multi-variate)

```
sns.pairplot(df , hue = 'sex')
```

Bivariate plotting for Num vs Categorical columns :-

- Box plot :

```
sns.boxplot(data = df, x = 'day', y = 'total_bill' ax = axs[0])
```

- violinplot :

```
sns.violinplot(data = df, x = 'day', y = 'total_bill' ax = axs[1])
```

- Boxen plot :

```
sns.boxenplot(data = df, x = 'day', y = 'total_bill', ax = axs[2])
```

__/__/____

- Histogram plot :

```
sns.histplot(data = df.loc[df['day']=='sat', x = 'total_bill', alpha=0.3,
                           ax = axs[3], label = 'sat' )
sns.histplot(data = df.loc[df['day']=='Fri', x = 'total_bill',
             alpha = 0.3, ax = axs[3], label = 'Fri' )
```

- Multi-plot Grid using Facet Grid :

```
grid = sns.FactGrid(df, col = 'sex', hue = 'smokes'
grid.map( sns.histplot, 'total_bill', alpha = 0.3 )
grid.add_legend( )
```

Bivariate plotting For cat vs categorical columns :

```
fig, ax = plt.subplots( figsize = (3,3 ))
ax.set_title ("count plot")
sns.countplot(data = df, x = 'sex' , hue = 'smokes', ax = ax)
plt.show( )
```