Matplotlib

(Code: Subhajit Das)

Data Visualization:

- The process of finding trends and correlations in data by representing it pictorially is called Data Visualization.
- To perform data visualization in python, we can use various python data visualization modules such as Matplotlib, Seaborn, Plotly, Ggplot.

What is Matplotlib:

➤ Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

➤ Matplotlib is a python library used for Data Visualization.

➤ Matplotlib is 2D and 3D plotting python library.

➤ It was introduced by john hunter in the year 2002.

Matplotlib Graphs:

1. **Pairwise data**: Plots of pairwise (x,y), tabular (var_0,...var_n), and functional f(x) = y data:

➤ Line plot: **plt.plot(x, y)**

➤ Scatter plot: **plt.scatter(x, y)**

➤ Bar plot: **plt.bar(x, y)**

➤ Stem plot: **plt.stem(x, y)**

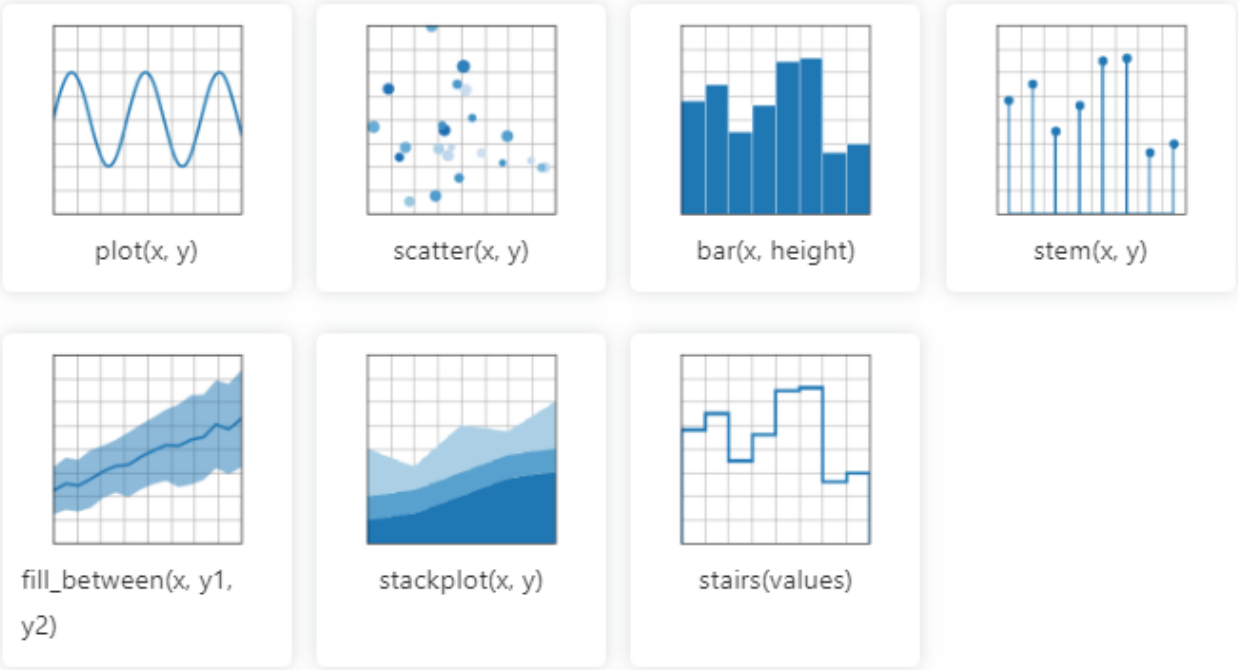
➤ Filled area between two horizontal curves: **plt.fill_between(x, y1, y2)**

➤ Stack plot: **plt.stackplot(x, y)**

➤ Step / Stairs plot: **plt.stairs(x) / plt.step(x)**

Pairwise data

Plots of pairwise (x, y), tabular $(var\_0, \cdots, var\_n)$, and functional $f(x) = y$ data.



2. **Statistical distributions**: Plots of the distribution of at least one variable in a dataset. Some of these methods also compute the distributions.

➤ Histogram: **plt.hist(X)**

➤ Box plot: **plt.boxplot(X)**

➤ Error bar plot: **plt.errorbar(x, y, yerr, xerr)**

➤ Violin plot: **plt.violinplot(D)**

➤ Event plot: plt.eventplot(D)

➤ Hist2D plot: plt.hist2D(x, y)

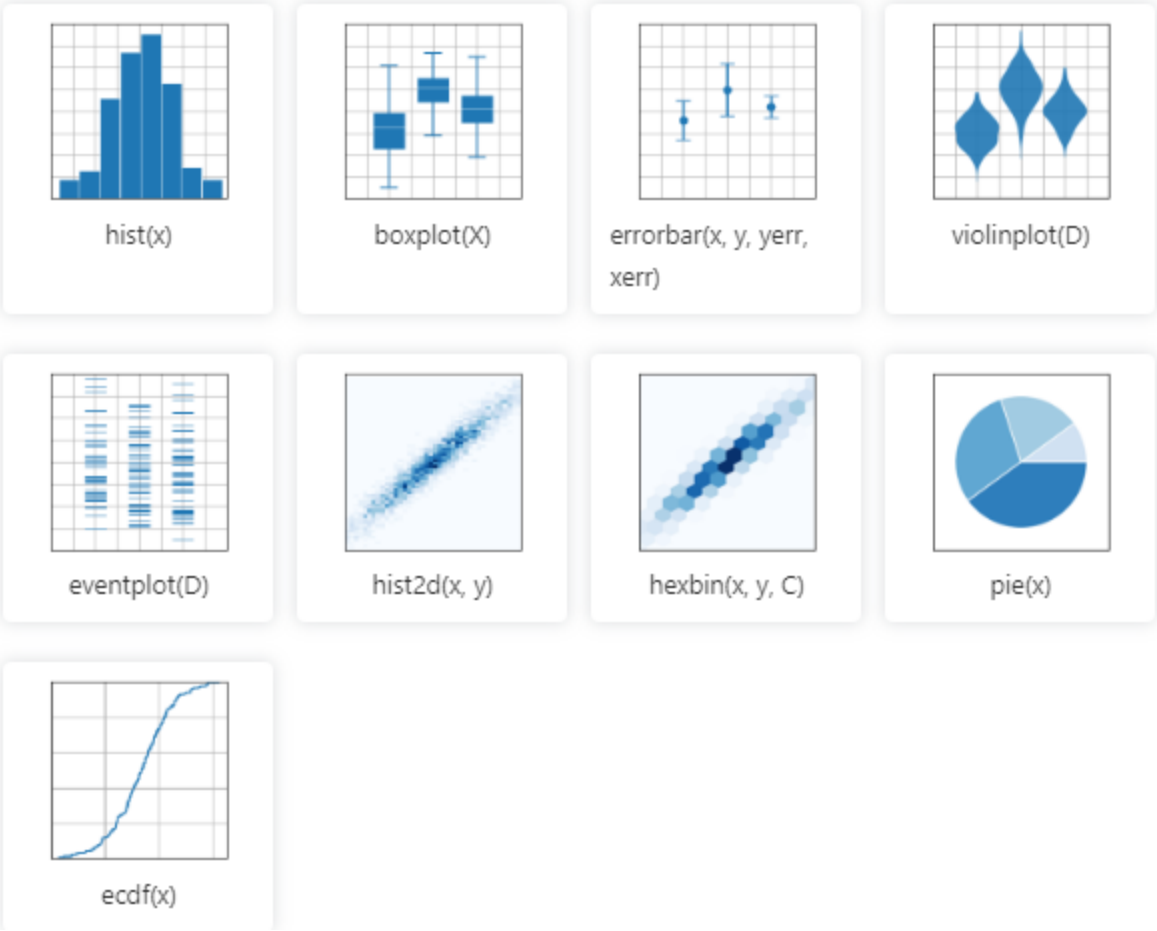
➤ Hexbin plot: plt.hexbin(x, y, C)

➤ Pie plot: **plt.pie(x)**

➤ Empirical Cumulative Distribution Function: plt.ecdf(x)

Statistical distributions

Plots of the distribution of at least one variable in a dataset. Some of these methods also compute the distributions.



3. **Gridded data**: Plots of arrays and images Z i,j and fields U i,j, Vi,j on regular grids and corresponding coordinate grids X i,j , Y i,j

➤ Display an image on the axes: **plt.imshow(Z)**

➤ Create a pseudocolor plot of a 2-D array: **plt.pcolormesh([X, Y, Z]) / plt.pcolormesh(Z)**

➤ Contour plot: **plt.contour([X, Y, Z]) / plt.contour(Z)**

➤ Filled contour plot: **plt.contourf([X, Y, Z]) / plt.contourf(Z)**

➤ Step plot: **plt.step(x, y)**

➤ Barbs plot: **plt.barbs(X, Y, U, V)**

➤ Quiver plot: **plt.quiver(X, Y, U, V)**

Gridded data:

Plots of arrays and images $Z_{i,j}$ and fields $U_{i,j}, V_{i,j}$ on regular grids and corresponding coordinate grids $X_{i,j}, Y_{i,j}$.

4. **Irregularly gridded data**: Plots of data Z x,y on unstructured grids, unstructured coordinate grids (x,y) and 2D functions f(x,y) = z

➤ Tricontour Plot: plt.tricontour(x, y, z)

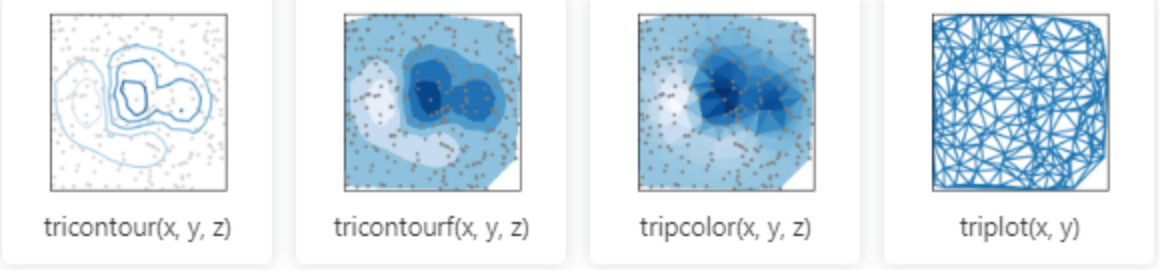
➤ Tricontour filled with Plot: plt.tricontourf(x, y, z)

➤ Tri Pcolor Plot: plt.tripcolor(x, y, z)

➤ Tri Plot: plt.triplot(x, y)

Irregularly gridded data

Plots of data $Z_{x,y}$ on unstructured grids , unstructured coordinate grids (x, y), and 2D functions $f(x, y) = z$.



5. **3D and volumetric data**: Matplotlib also supports 3D plots using the mpl_toolkits.mplot3d library.

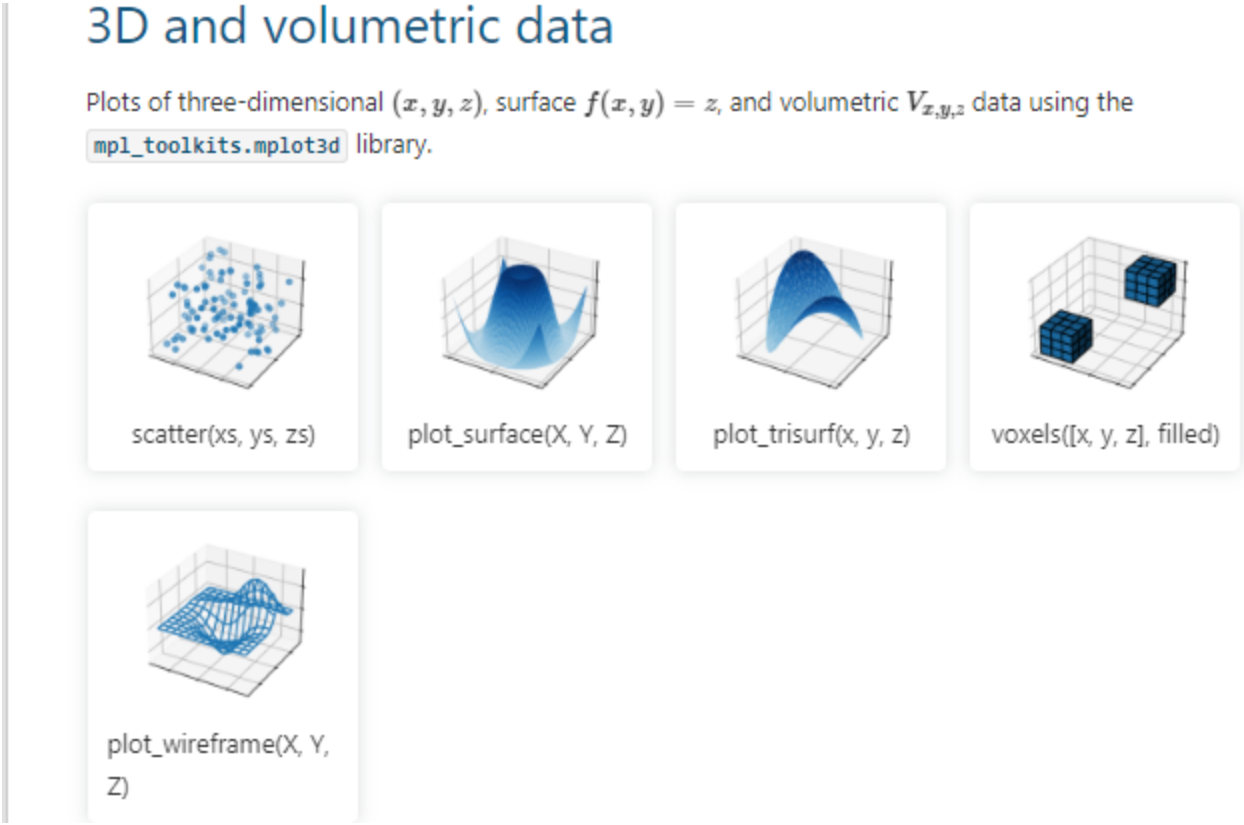
➤ Scatter 3D: **scatter(xs, ys, yz)**

➤ Surface 3D: **plot_surface(X, Y, Z)**

➤ Trisurf 3D: **plot_trisurf(x, y, z)**

➤ Voxels 3D: **voxels([x, y, z], field)**

➤ Wireframe 3D: **plot_wireframe(X, Y, Z)**



Install Matplotlib

```
In [1]: # pip install matplotlib
```

Classes of Matplotlib:

1. **Figure Class**: This is the top-level container in this hierarchy. It is the overall window/page that everything is drawn on. **Syntax: import matplotlib.figure**
2. **Axes Class**: This is what we commonly think of as 'a plot'. A given figure can contain many Axes, but a given Axes object can only be in one Figure[1]. The Axes contains two (or three in the case of 3D) Axis objects which take care of the data limits. **Syntax: import matplotlib.axes**
3. **Axis Class**: These are the number-line-like objects and take care of setting the graph limits and generating the ticks (the marks on the axis) and ticklabels (strings labeling the ticks). **Syntax: import matplotlib.axis**
4. **Artist Class**: Basically, everything visible on the figure is an artist (even the Figure, Axes, and Axis objects). This includes Text objects, Line2D objects, collections objects, Patch objects. **Syntax: import matplotlib.artist**
5. **Pyplot**: Pyplot is a Matplotlib module that provides a MATLAB-like interface. Pyplot provides functions that interact with the figure i.e., creates a figure, decorates the plot with labels, etc. **Syntax: import matplotlib.pyplot**
6. **Backend Layer Classes**: These include FigureCanvas (the surface on which the figure will be drawn), Renderer (the class that takes care of drawing on the surface), and Event (handles mouse and keyboard events). **Syntax: import matplotlib**

Import Matplotlib

In [2]:
```
import matplotlib.pyplot as plt
# from matplotlib import pyplot as plt
from mpl_toolkits import mplot3d

import pandas as pd
import numpy as np
```

Linking the dataset for Data Visualizations

In [3]:
```
file = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/3.2_NSCA_DC Matplotlib

file.tail(4)
```

Out[3]:

	Not_AvailableME_of_Responder	NUMBER_OF_MEMBERS_IN_HOUSEHOLD	AVG_HOUSEHOLD_INCOME
206	Ashish Chetri	3	50000
207	Ramesh BalakrishNot_Availablen	5	90000
208	Aaditya Varma	7	40000
209	Akhil Sharma	3	25000

4 rows × 25 columns

In [4]:
```
# Getting the datatype of column
print(file.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   Not_AvailableME_of_Responder    210 non-null    object
 1   NUMBER_OF_MEMBERS_IN_HOUSEHOLD  210 non-null    int64 
 2   AVG_HOUSEHOLD_INCOME            210 non-null    int64 
 3   HOUSE_ADDRESS                   210 non-null    object
 4   AGE_OF_MEMBERS                  210 non-null    object
 5   MALE_MEMBERS                    210 non-null    int64 
 6   FEMALE_MEMBERS                  210 non-null    int64 
 7   NO_OF_CHILDREN                  210 non-null    int64 
 8   STUDYING_CHILDREN               210 non-null    int64 
 9   NON_STUDYING_CHILDREN           210 non-null    int64 
 10  FOOD_CONSUMPTION_PATTERNS       208 non-null    object
 11  EXPENDITURES                    210 non-null    int64 
 12  HOUSEHOLD_ASSETS                210 non-null    object
 13  Fridge                          210 non-null    object
 14  AC                              210 non-null    object
 15  Personal_Vehicle                210 non-null    object
 16  Washing_Machine                 210 non-null    object
 17  TV                              210 non-null    object
 18  SOURCES_OF_WATER                209 non-null    object
 19  AVAILABILITY_OF_SMART_DEVICES   210 non-null    object
 20  Desktop_Laptop                  210 non-null    object
 21  Phone                           210 non-null    object
 22  Other_SmartDevices              210 non-null    object
 23  ACCESS_TO_INTERNET              210 non-null    object
 24  ANY_PREDOMINANT_AILMENT         210 non-null    object
dtypes: int64(8), object(17)
memory usage: 41.1+ KB
None
```

In [5]:
```
#to check if the dataset has any missing value
file.isna().sum()
```

Out[5]:
```
Not_AvailableME_of_Responder      0
NUMBER_OF_MEMBERS_IN_HOUSEHOLD    0
AVG_HOUSEHOLD_INCOME              0
HOUSE_ADDRESS                     0
AGE_OF_MEMBERS                    0
MALE_MEMBERS                      0
FEMALE_MEMBERS                    0
NO_OF_CHILDREN                    0
STUDYING_CHILDREN                 0
NON_STUDYING_CHILDREN             0
FOOD_CONSUMPTION_PATTERNS         2
EXPENDITURES                      0
HOUSEHOLD_ASSETS                  0
Fridge                            0
AC                                0
Personal_Vehicle                  0
Washing_Machine                   0
TV                                0
SOURCES_OF_WATER                  1
AVAILABILITY_OF_SMART_DEVICES     0
Desktop_Laptop                    0
Phone                             0
Other_SmartDevices                0
ACCESS_TO_INTERNET                0
ANY_PREDOMINANT_AILMENT           0
dtype: int64
```

In [6]:
```
# Male members in every household and that numbers
male = file.MALE_MEMBERS.value_counts()
print(male)
```

```
2    88
1    65
3    36
4     9
0     7
5     4
6     1
Name: MALE_MEMBERS, dtype: int64
```

In [7]:
```
# Max male members in household
max = file.MALE_MEMBERS.value_counts().max()
print(max)
```

```
88
```

In [8]:
```
# Viewing the particular number of male member
par = (file.MALE_MEMBERS == 4).sum()
print(par)
```

```
9
```

Linking another dataset for visualization

In [9]:
```
df_car = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/3.3_CarPriceData_Matp

df_car.head(4)
```

Out[9]:

	ID	Company	Model	Type	Fuel	Transmission	Engine	Mileage	Kms_driven	Buyers	Horsepo
0	1	Maruti	Alto	Hatchback	Petrol	Manual	796	19.7	45000	2	
1	2	Maruti	Wagon R	Hatchback	Petrol	Manual	998	20.5	40005	2	
2	3	Maruti	Wagon R	Hatchback	Petrol	Manual	998	20.5	40005	2	
3	4	Maruti	Ertiga	MUV	Petrol	Automatic	1462	18.5	28000	2	

Here are some of the functions in `matplotlib.pyplot` that are used to set properties and customizing the appearance of plot:

1. **plt.title()**: This function is used to set a title for the plot.
2. **plt.xlabel()**: This function is used to set a label for the x-axis.
3. **plt.ylabel()**: This function is used to set a label for the y-axis.
4. **plt.axis()**: This function is used to get or set some axis properties.
5. **plt.xticks()**: This function is used to get or set the current tick locations and labels of the x-axis.
6. **plt.yticks()**: This function is used to get or set the current tick locations and labels of the y-axis.
7. **plt.grid()**: This function is used to configure the grid lines.
8. **plt.legend()**: This function places a legend on the axes.
9. **plt.figure()**: This function creates a new figure, or activates an existing figure.
10. **plt.show()**: This function displays a figure.
11. **plt.colorbar()**: This will show a color pallate in scatter plot

Line Plot

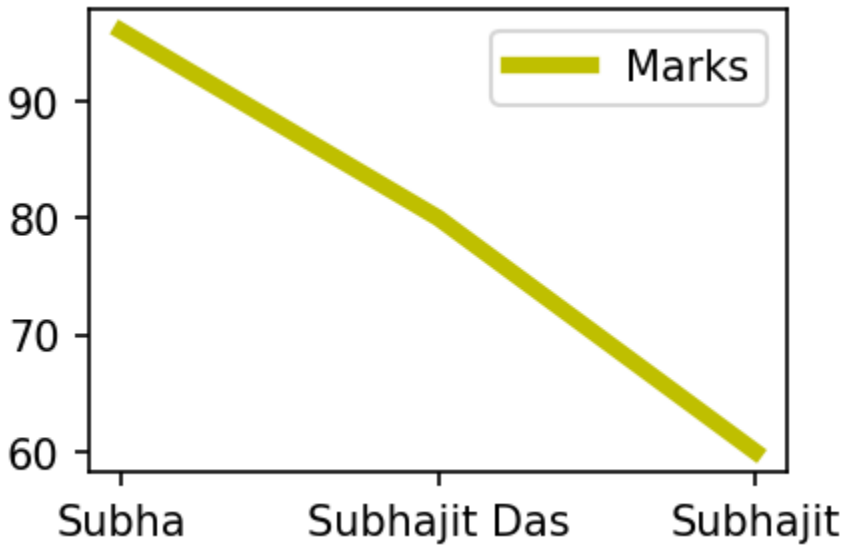
Syntax: plt.plot(x,y), where x and y can be list or tuple

The `plt.plot()` function in `matplotlib.pyplot` has the following parameters:

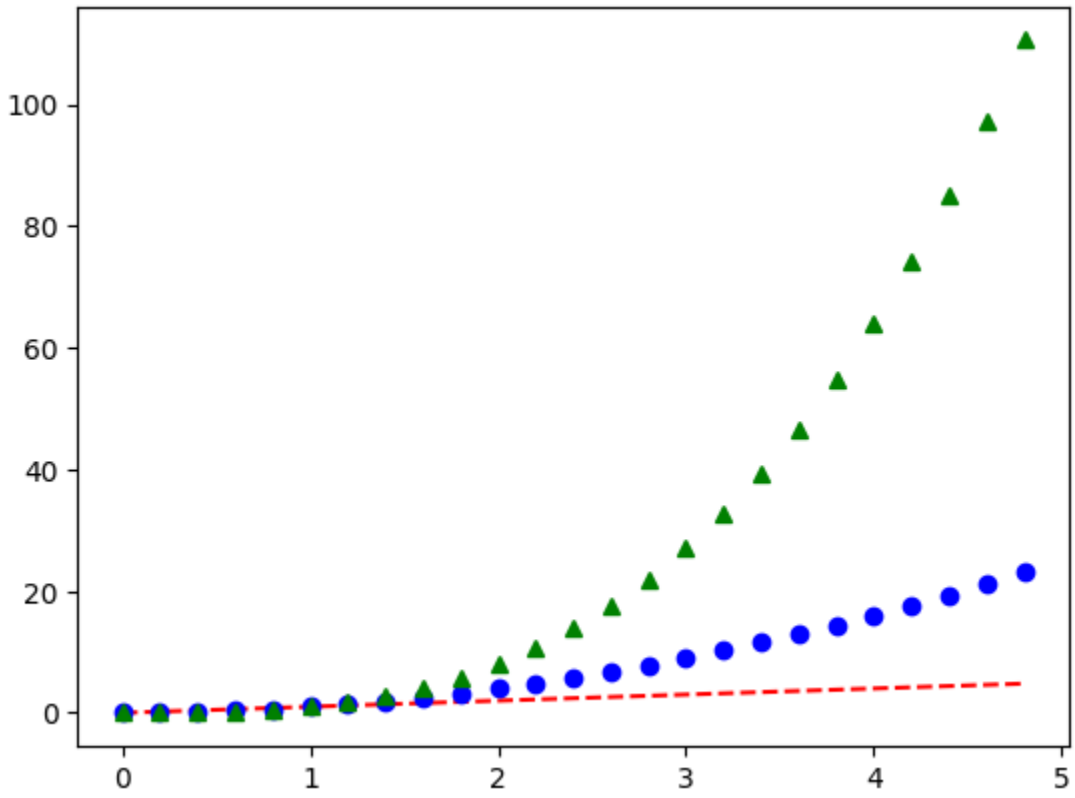
1. **args**: This is a variable length argument, allowing for multiple x, y pairs with an optional format string.
2. **scalex, scaley**: These are optional boolean values that, when set to True (default), will autoscale the x and/or y axis.
3. **data**: This is an optional parameter where you can pass a DataFrame.
4. **color**: color or list of color, optional. The colors of the line faces.
5. **linewidth**: float or array-like, optional. Width of the line edges.
6. **label**: str or list of str, optional. A label for the whole dataset or individual lines.

In [10]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

plt.plot(name, marks,
         color = 'y',
         label = 'Marks',
         linewidth = 4,
         scalex=True, # If, it's falese x axis line will not visible, by default it's
         scaley=True) # If, it's falese x axis line will not visible, by default it's

plt.legend()
```

Out[10]: <matplotlib.legend.Legend at 0x7931c58fdf60>

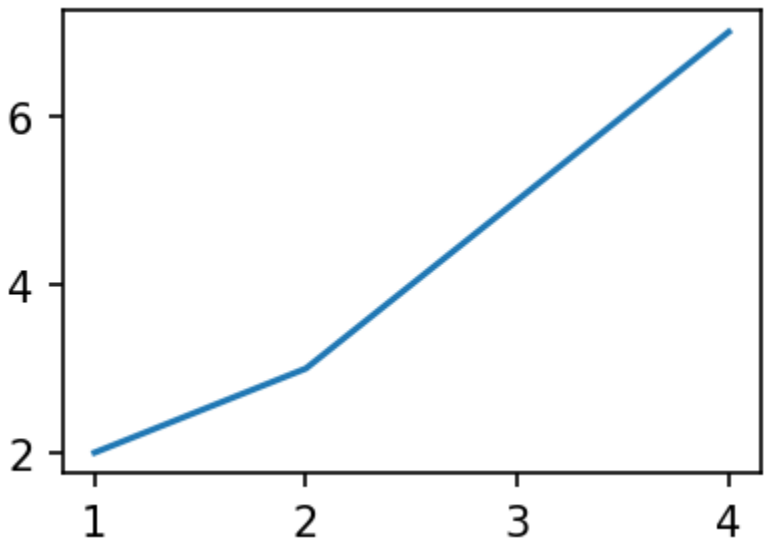


In [11]:
```
# Using *args concept

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--',
         t, t**2, 'bo',
         t, t**3, 'g^')

''' The first character ('r', 'b', 'g') specifies the color: red, blue, green.
The second character ('--', 's', '^', 'o') specifies the line type or marker style: d

plt.show()
```



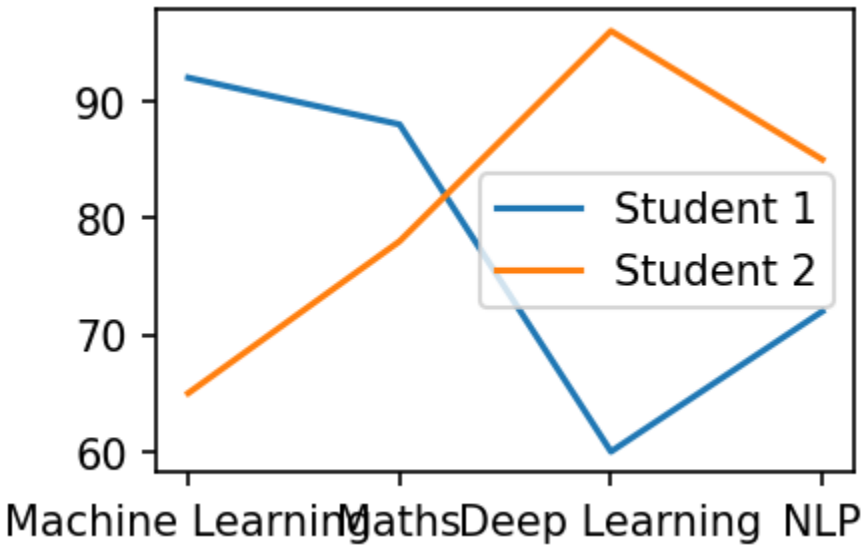
In [12]:
```
# Using data function
fig = plt.figure(figsize=(3, 2), dpi=150)
# Define data
data = {'a': [1, 2, 3, 4], 'b': [2, 3, 5, 7]}

# Plot 'b' against 'a'
plt.plot('a', 'b', data=data)
plt.show()
```

```
<ipython-input-12-bbdd178fa8c6>:7: RuntimeWarning: Second argument 'b' is ambiguous: could be a format string but is in 'data'; using as data.  If it was intended as data, set the format string to an empty string to suppress this warning.  If it was intended as a format string, explicitly pass the x-values as well.  Alternatively, rename the entry in 'data'.
  plt.plot('a', 'b', data=data)
```



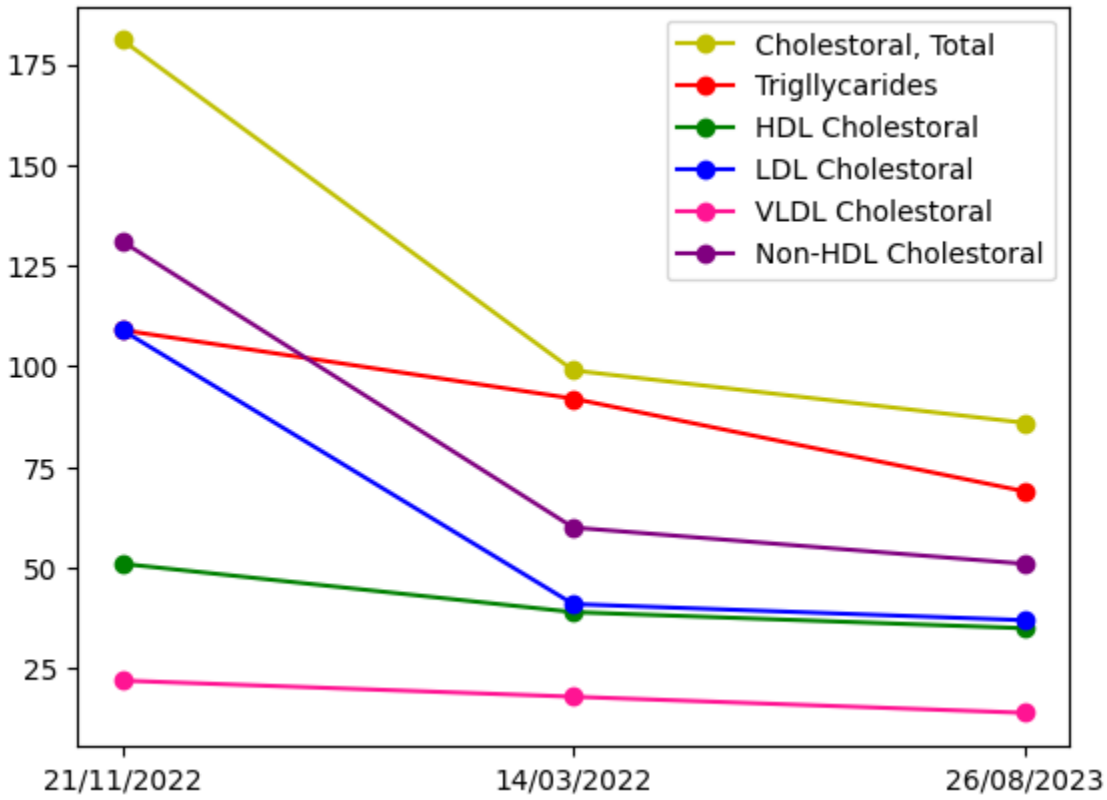
In [13]:
```
# Using multiple line plot in a single graph
fig = plt.figure(figsize=(3, 2), dpi=150)
# Define data
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]
std2 = [65, 78, 96, 85]

plt.plot(sub, std1, label='Student 1')
plt.plot(sub, std2, label='Student 2')
plt.legend()
```

Out[13]: <matplotlib.legend.Legend at 0x7931c3736d40>



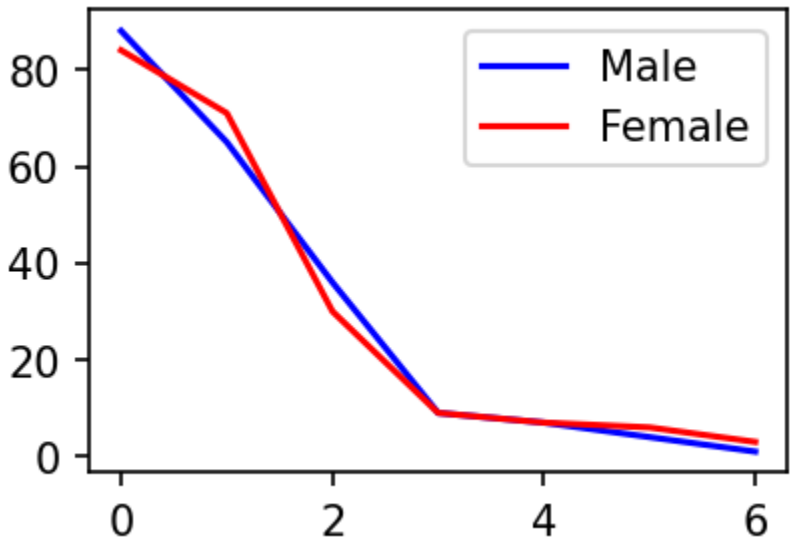
In [14]:
```
date = ('21/11/2022', '14/03/2022', '26/08/2023')
Cholestoral_Total = [181, 99, 86]
Trigllycarides = [109, 92, 69]
HDL_Cholestoral = [51, 39, 35]
LDL_Cholestoral = [109, 41, 37]
VLDL_Cholestoral = [22, 18, 14]
Non_HDL_Cholestoral = [131, 60, 51]

plt.plot(date, Cholestoral_Total, marker='o', color = 'y', label = 'Cholestoral, Tota
plt.plot(date, Trigllycarides, marker='o', color = 'r', label = 'Trigllycarides')
plt.plot(date, HDL_Cholestoral, marker='o', color = 'g', label = 'HDL Cholestoral')
plt.plot(date, LDL_Cholestoral, marker='o', color = 'b', label = 'LDL Cholestoral')
plt.plot(date, VLDL_Cholestoral, marker='o', color = 'deeppink', label = 'VLDL Choles
plt.plot(date, Non_HDL_Cholestoral, marker='o', color = 'purple', label = 'Non-HDL Ch

plt.legend()
```

Out[14]: <matplotlib.legend.Legend at 0x7931c35b7d00>



In [15]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
# Viewing male and female in line graph

x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

plt.plot(num, x, color='b', label= 'Male')
plt.plot(num, y, color='r', label= 'Female')

plt.legend()
```

Out[15]:
```
<matplotlib.legend.Legend at 0x7931c3657910>
```



Scatter Plot

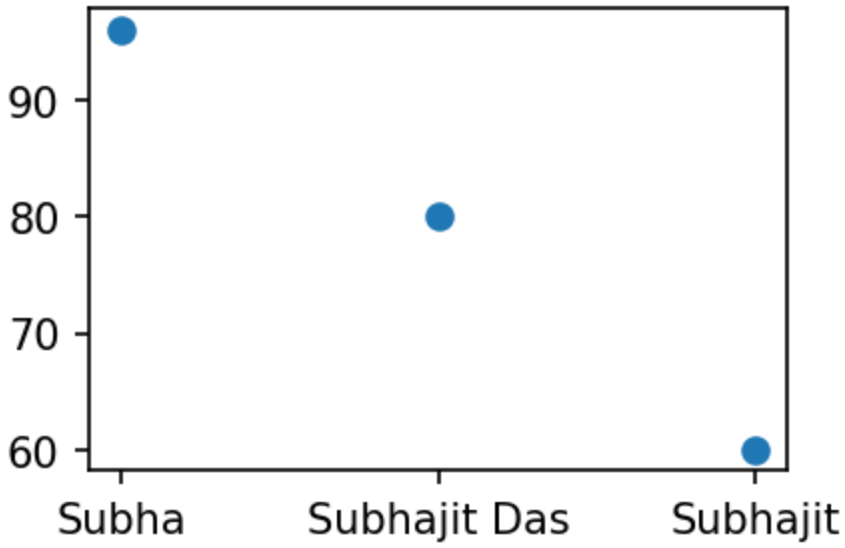
Syntax: plt.scatter(x,y), where x and y can be list or tuple

The `plt.scatter()` function in `matplotlib.pyplot` has the following parameters:

1. **x, y**: These are arrays or sequences of n numbers representing the data positions. This defines dot size
2. **s**: This is the marker size in points**2. It can be a scalar or an array-like shape (n,), and is optional.
3. **c or color**: This represents the marker colors. It can be an array-like or list of colors or color, and is optional.
4. **marker**: This is the marker style. It can be either an instance of the class or the text shorthand for a particular marker.
5. **cmap**: This is the Colormap instance or registered colormap name used to map scalar data to colors.
6. **norm**: This is the normalization method used to scale scalar data to the [0, 1] range before mapping to colors using cmap.
7. **vmin, vmax**: These are used in conjunction with norm to normalize luminance data.
8. **alpha**: This is the alpha blending value, between 0 (transparent) and 1 (opaque).
9. **linewidths**: This is the linewidth of the marker edges.
10. **edgecolors**: This is the color or sequence of colors of the marker edges.
11. **plotnonfinite**: This boolean value is used to indicate whether to plot the non-finite x, y values.
12. **data**: This is an optional parameter that can be a DataFrame.

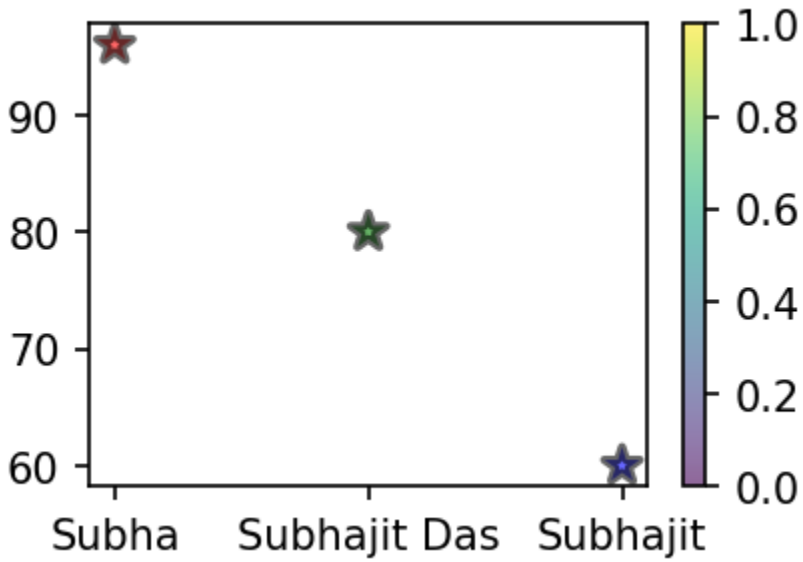
In [16]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

plt.scatter(name, marks)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x7931c34e3040>
```

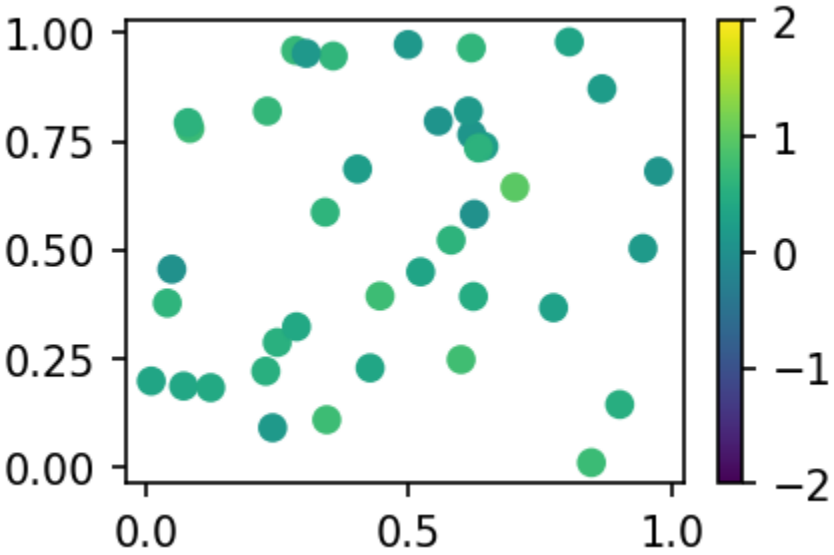


In [17]:
```
# Using the parameters of plt.scatter()
fig = plt.figure(figsize=(3, 2), dpi=150)

name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

c = ['r', 'g', 'b']# pass tuple or list for color


plt.scatter(name, marks,
        color = c,
        s = 80,    # Can also pass a specific color
        marker='*', # ‘o’ for a circle, ‘s’ for a square, ‘*’ for a star marker, etc
        alpha = 0.6,
        edgecolor = 'black',
        linewidth = 2,
        plotnonfinite=True) # not that much imp
plt.colorbar()
```

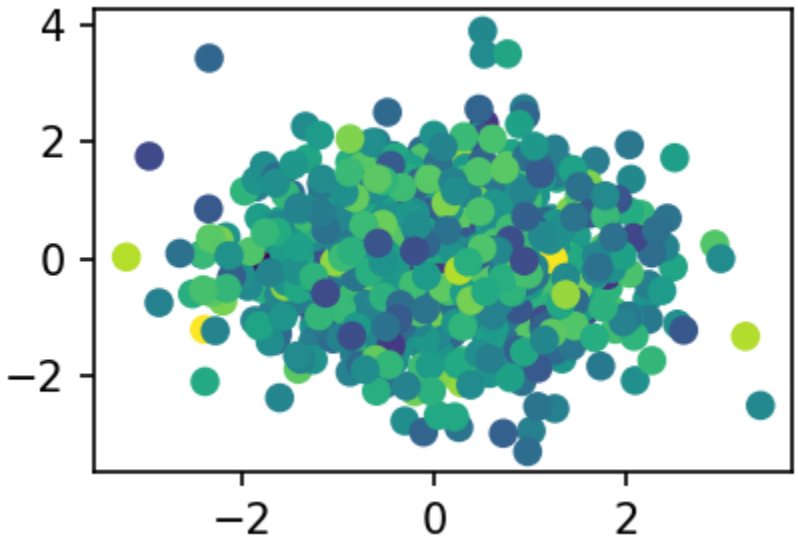
Out[17]: <matplotlib.colorbar.Colorbar at 0x7931c3562f50>



In [18]:
```
# Using vmin, vmax and norm parameter
import matplotlib.colors as mcolors

fig = plt.figure(figsize=(3, 2), dpi=150)

N = 40
x = np.random.rand(N)
y = np.random.rand(N)

colors = np.random.rand(N)
n = mcolors.Normalize(vmin=-2.,vmax=2.) # We have to use, mcolors to use, vmin and vm

plt.scatter(x, y, c=colors, norm=n, cmap='viridis') # In cmap we have huge color comb
plt.colorbar()
plt.show()
```

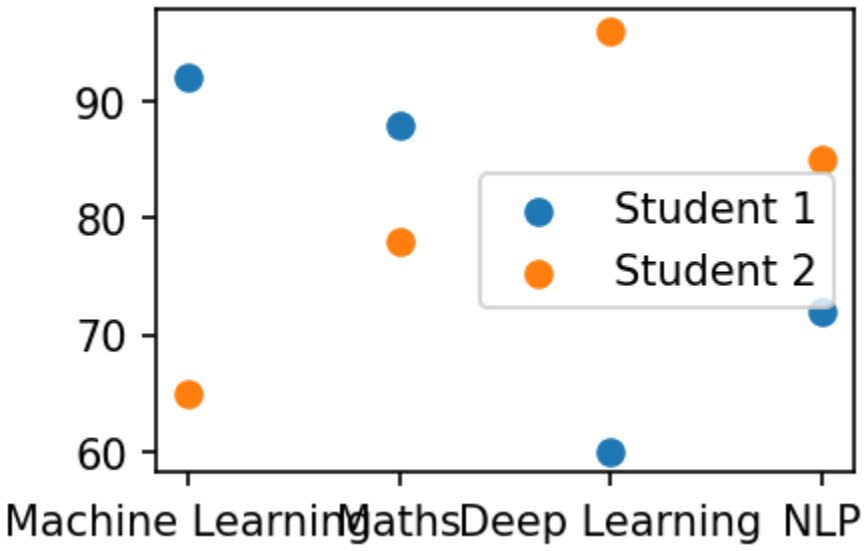


In [19]:
```
# Using data parameter
fig = plt.figure(figsize=(3, 2), dpi=150)
df = pd.DataFrame({
    'a': np.random.randn(1000),
    'b': np.random.randn(1000),
    'c': np.random.randn(1000)
})

plt.scatter('a', 'b', c='c', data=df) # The x and y coordinates of the points are the
                                      # and the colors of the points are specified by
plt.show()
```



In [20]:
```
# Using multiple scatter plot in a single graph
fig = plt.figure(figsize=(3, 2), dpi=150)
# Define data
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]
std2 = [65, 78, 96, 85]

# Plot std1 and std2 against sub
plt.scatter(sub, std1, label='Student 1')
plt.scatter(sub, std2, label='Student 2')

plt.legend()
```

Out[20]: <matplotlib.legend.Legend at 0x7931c3321d20>



Bar Plot

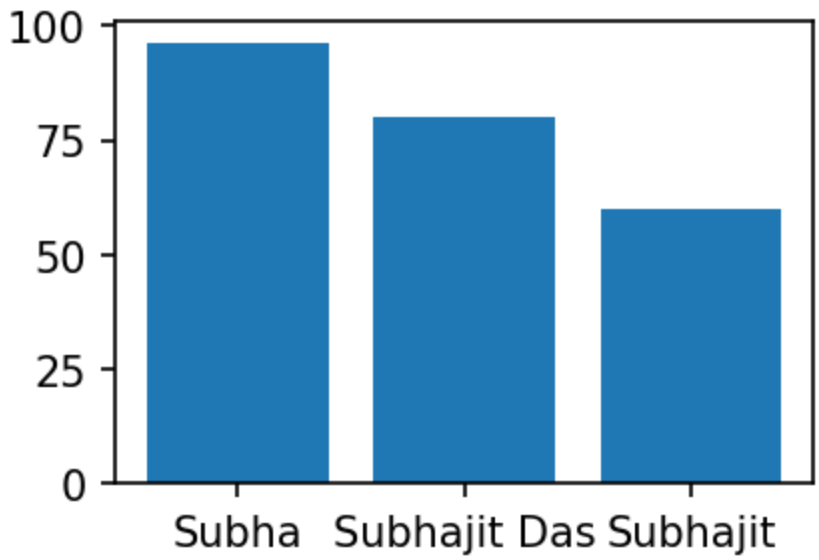
Syntax: plt.bar(x,y), where x and y can be list or tuple

The `plt.bar()` function in `matplotlib.pyplot` has the following parameters:

1. **x**: Float or array-like. The x coordinates of the bars.
2. **height / y**: Float or array-like. The heights of the bars.
3. **width**: Float or array-like, default: 0.8. The widths of the bars.
4. **bottom**: Float or array-like, default: 0. The y coordinates of the bottom sides of the bars.
5. **align**: {'center', 'edge'}, default: 'center'. Alignment of the bars to the x coordinates.
6. **color**: color or list of color, optional. The colors of the bar faces.
7. **edgecolor**: color or list of color, optional. The colors of the bar edges.
8. **linewidth**: float or array-like, optional. Width of the bar edges.
9. **linestyle**: Style of the bar graph. It can be "solid", "dotted", "dashed" or "dashdot". By default it's solid.
10. **tick_label**: str or list of str, optional. The tick labels of the bars.
11. **label**: str or list of str, optional. A label for the whole dataset or individual bars.
12. **alpha**: It's control the opacity of bar graph
13. **xerr, yerr**: float or array-like of shape (N,) or shape (2, N), optional. Add horizontal/vertical errorbars to the bar tips.

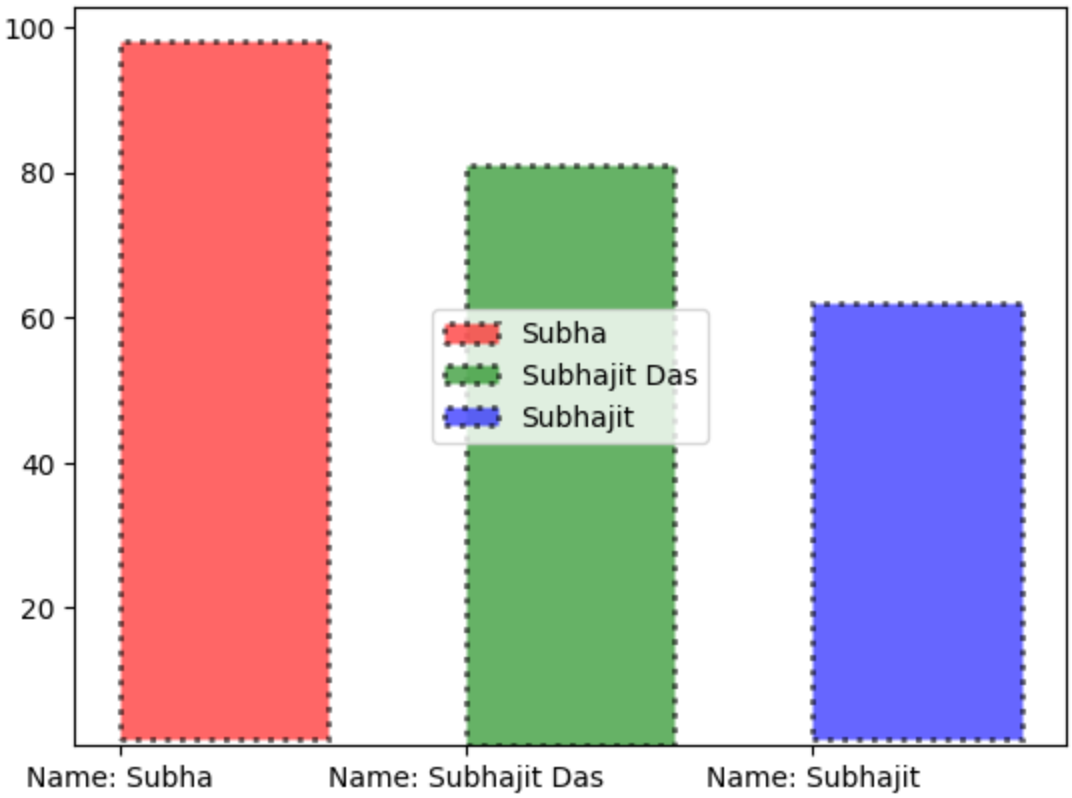
In [21]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

plt.bar(name, marks)
```

Out[21]:
```
<BarContainer object of 3 artists>
```



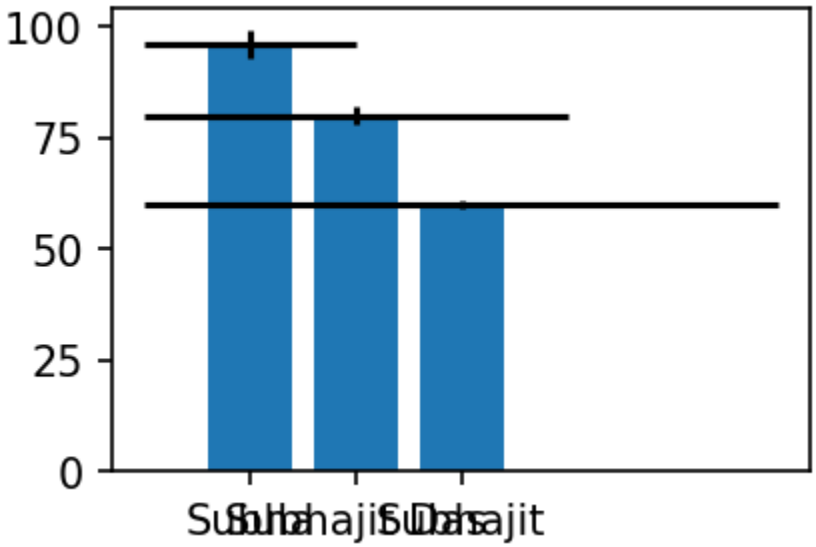
In [22]:
```
# Using the parameters of plt.bar()
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

b = (2, 1, 2) # we have to pass tuple or list for buttom
c = ['r', 'g', 'b']# pass tuple or list for color
tl = ('Name: Subha', 'Name: Subhajit Das', 'Name: Subhajit')
l = ('Subha', 'Subhajit Das', 'Subhajit')

plt.bar(name, marks,
        bottom= b,
        align = 'edge',
        width = 0.6, # Can also pass width of tuple or list
        color = c, # Can also pass a specific color
        alpha = 0.6, # Can also pass alpha of tuple or list
        edgecolor = 'black', # Can also pass color of tuple or list
        linewidth = 2, # Can also pass linewidth of tuple or list
        linestyle = "dotted",
        tick_label = tl,
        label = l)

plt.legend(loc = 'center') # We have to use this, when we are using label, Center is
# Also we have 'upper left', 'upper right', 'lower left', 'lower right', 'upper cente
```

Out[22]: <matplotlib.legend.Legend at 0x7931c33e3e20>



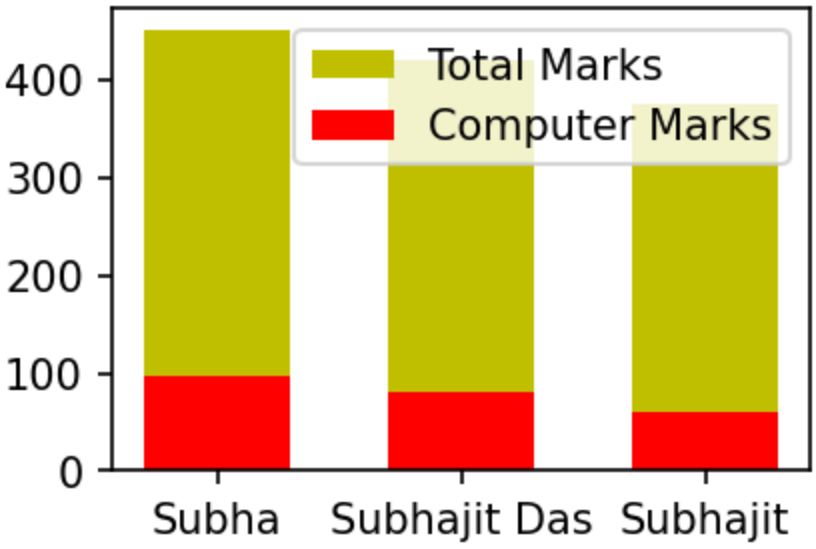
In [23]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
# Using xerr and yerr
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

x = (1, 2, 3)
y = (3, 2, 1)

plt.bar(name, marks, xerr = x, yerr = y)
```

Out[23]: <BarContainer object of 3 artists>



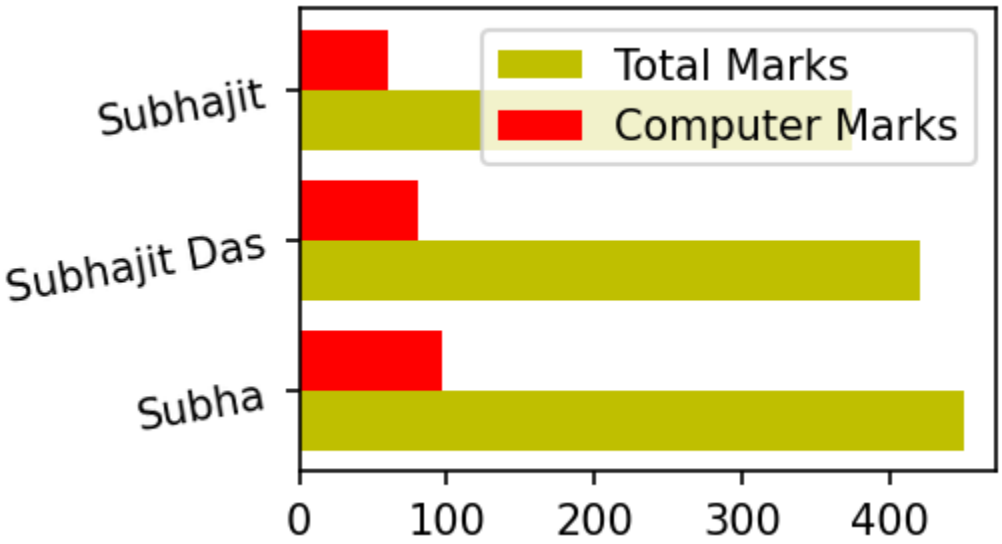
In [24]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
# Creating multiple bar graph which are overlaping
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]
total = [450, 420, 375]

plt.bar(name, total, color = 'y', width = 0.6, label = 'Total Marks')
plt.bar(name, marks, color = 'r', width = 0.6, label = 'Computer Marks')
plt.legend()
```

Out[24]: <matplotlib.legend.Legend at 0x7931c3404ee0>

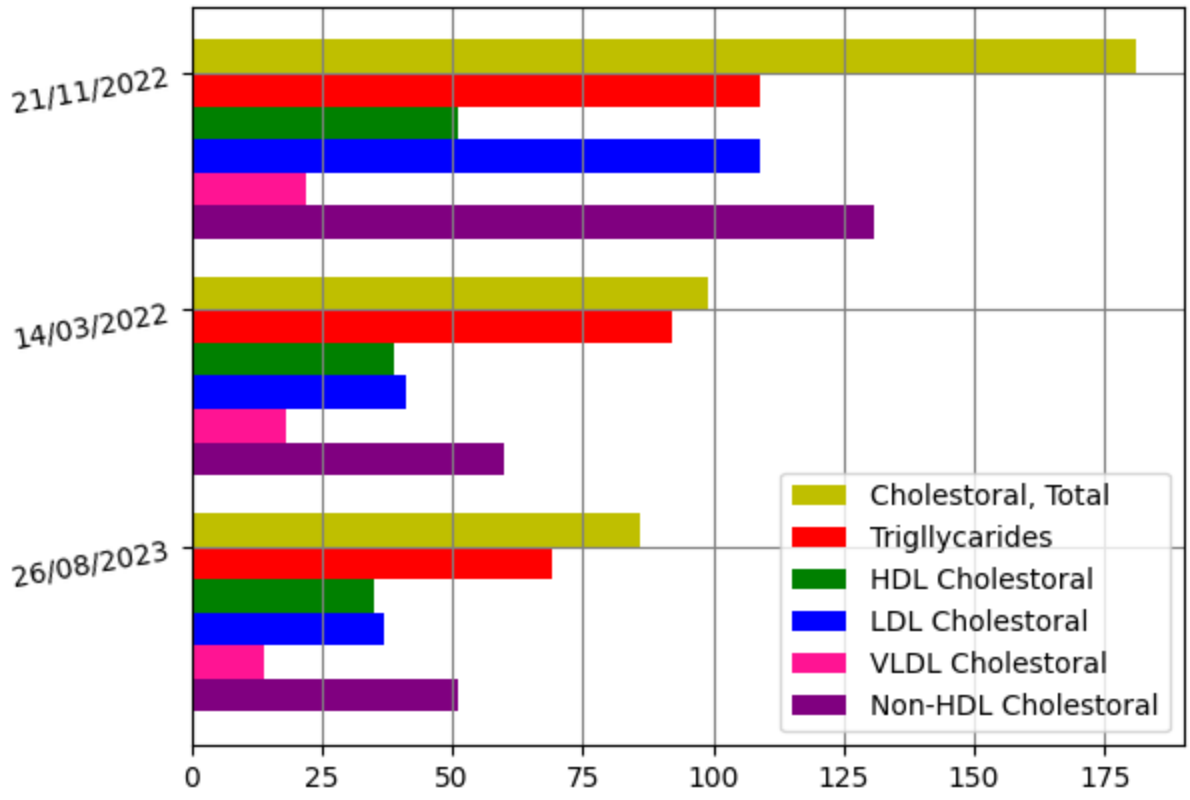


In [25]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
# Creating multiple horizontal bar graph, which are separate
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]
total = [450, 420, 375]

width = 0.4
n1 = np.arange(len(name))
n2 = [i+width for i in n1]

plt.yticks(n1+width/2, name, rotation = 10) # To print the names

plt.barh(n1, total, width, color = 'y', label = 'Total Marks')
plt.barh(n2, marks, width, color = 'r', label = 'Computer Marks')
plt.legend()
```

Out[25]: <matplotlib.legend.Legend at 0x7931c1959360>



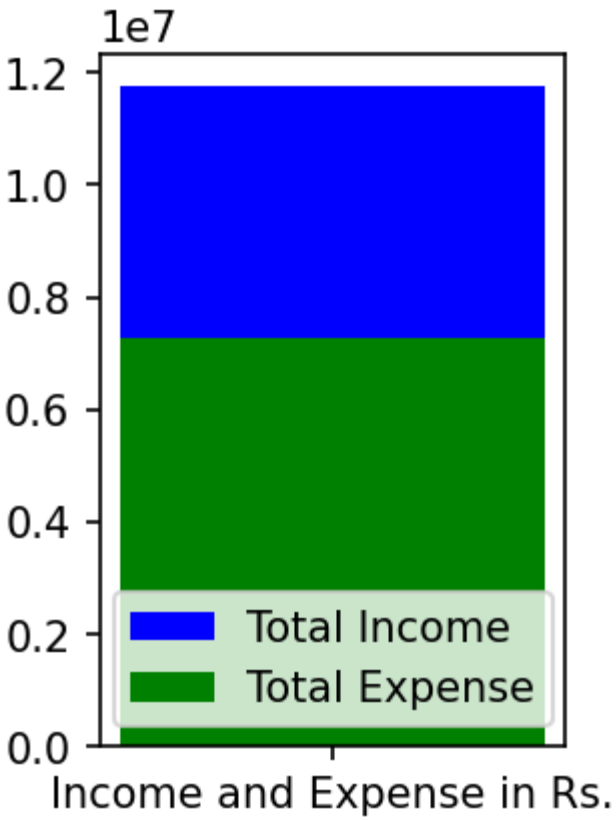
In [26]:
```
# Creating multiple horizontal bar graph, which are separate
date = ('21/11/2022', '14/03/2022', '26/08/2023')
Cholestoral_Total = [181, 99, 86]
Trigllycarides = [109, 92, 69]
HDL_Cholestoral = [51, 39, 35]
LDL_Cholestoral = [109, 41, 37]
VLDL_Cholestoral = [22, 18, 14]
Non_HDL_Cholestoral = [131, 60, 51]

width = 0.14
n1 = np.arange(len(date))
n2 = [i+width for i in n1]
n3 = [i+width for i in n2]
n4 = [i+width for i in n3]
n5 = [i+width for i in n4]
n6 = [i+width for i in n5]

plt.yticks(n1+width/2, date, rotation = 10) # To print the dates

plt.barh(n1, Cholestoral_Total, width, color = 'y', label = 'Cholestoral, Total')
plt.barh(n2, Trigllycarides, width, color = 'r', label = 'Trigllycarides')
plt.barh(n3, HDL_Cholestoral, width, color = 'g', label = 'HDL Cholestoral')
plt.barh(n4, LDL_Cholestoral, width, color = 'b', label = 'LDL Cholestoral')
plt.barh(n5, VLDL_Cholestoral, width, color = 'deeppink', label = 'VLDL Cholestoral')
plt.barh(n6, Non_HDL_Cholestoral, width, color = 'purple', label = 'Non-HDL Cholestor

plt.gca().invert_yaxis() # To reverse the graph
plt.legend()
plt.grid(color = 'gray')
```



In [27]:
```
# Calculate Total income and expenses and create the bar graph
fig = plt.figure(figsize=(2, 3), dpi=150)  # DPI stands for "Dots Per Inch". It is us

total_income = file['AVG_HOUSEHOLD_INCOME'].sum()
total_expense = file['EXPENDITURES'].sum()
money = 'Income and Expense in Rs.'

print(f'Total income is: {total_income}')
print(f'Total expense is: {total_expense}')
print()

plt.bar(money, total_income, color = 'b', width = 0.4, label = 'Total Income')
plt.bar(money, total_expense, color = 'g', width = 0.4, label = 'Total Expense')

plt.legend(loc = 'lower center')
```

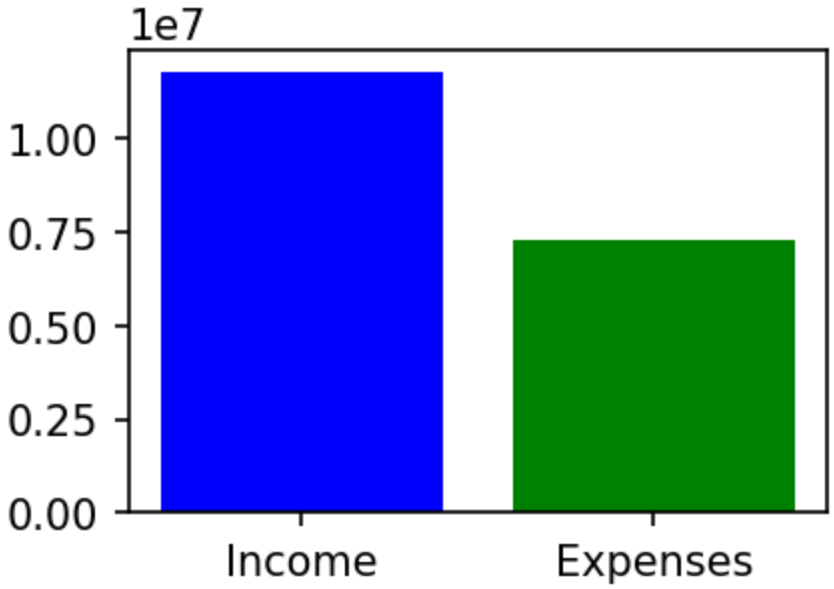
```
Total income is: 11743000
Total expense is: 7285500
```

Out[27]:
```
<matplotlib.legend.Legend at 0x7931c1958cd0>
```



In [28]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
# Create income and expense chart separetly
money = ['Income', 'Expenses']
total = [total_income, total_expense]
c = ['b', 'g']

plt.bar(money,total, color = c)
```

Out[28]: <BarContainer object of 2 artists>



Stem Plot

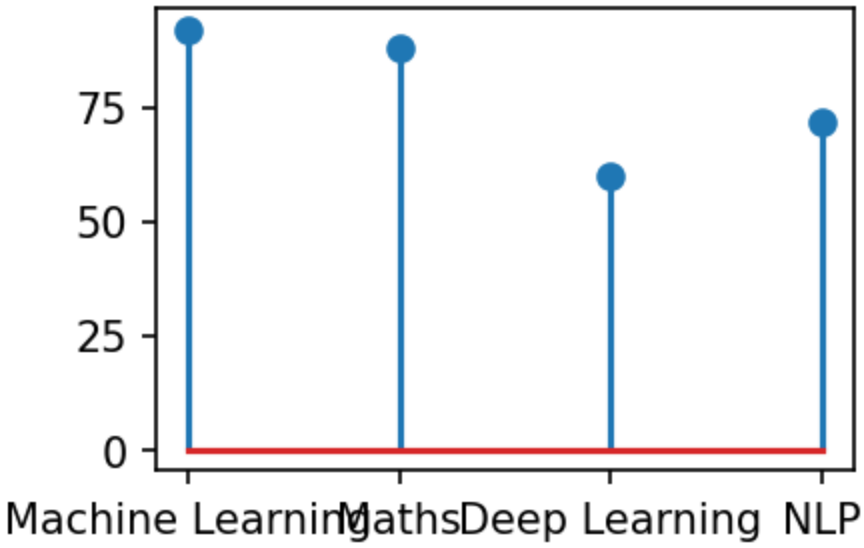
Syntax: plt.stem(x,y) / plt.stem(y,x), where x and y can be list or tuple

The `plt.stem()` function in `matplotlib.pyplot` has the following parameters:

1. **locs, x/y**: The x or y positions of the stem heads.
2. **heads, x/y**: The y or x positions of the stem heads.
3. **linefmt**: A string defining the color and style of the vertical lines.
4. **markerfmt**: A string defining the color and style of markers.
5. **basefmt**: A format for the baseline.
6. **bottom**: The y-position of the baseline.
7. **use_line_collection**: A boolean value indicating whether to create the stem lines as a LineCollection instead of individual lines.
8. **orientation**: If 'vertical', will produce a plot with stems oriented vertically, If 'horizontal', the stems will be oriented horizontally.

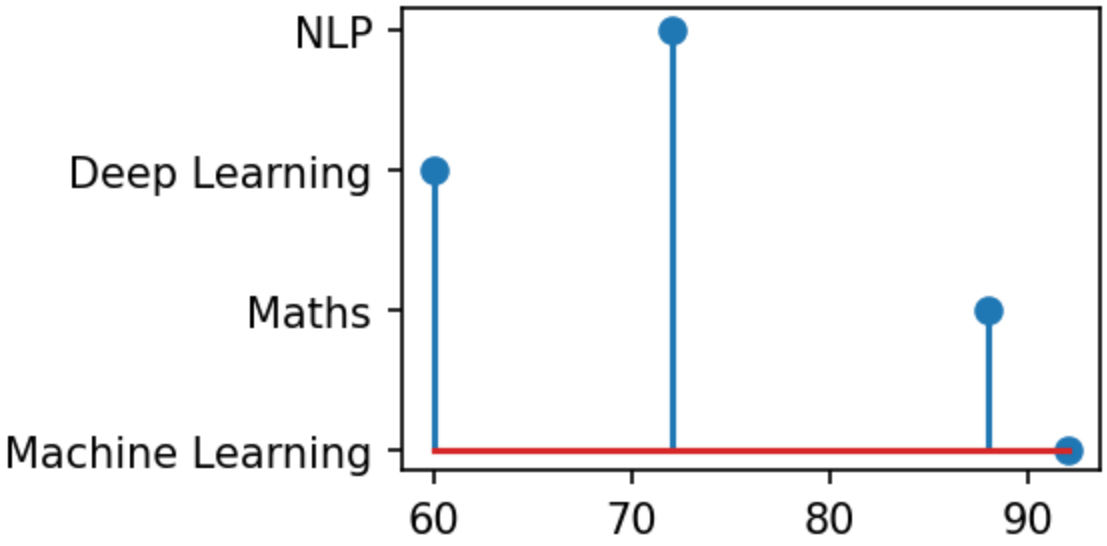
In [29]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
# Horizontal stem plot
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]

plt.stem(sub, std1)
```

Out[29]: <StemContainer object of 3 artists>



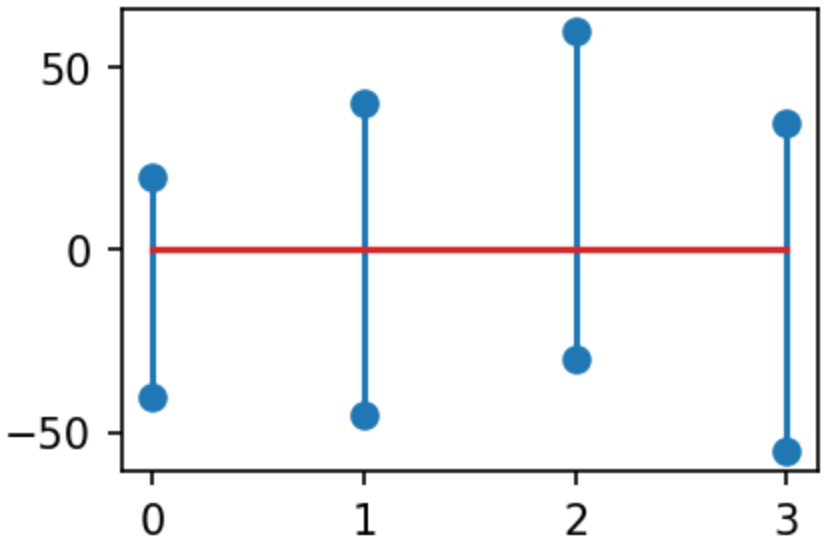
In [30]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
# Vertical stem plot
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]

plt.stem(std1, sub)
```

Out[30]: <StemContainer object of 3 artists>



In [31]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)
# With negative plotting
num = [0, 1, 2, 3]
x = [20, 40, 60, 35]
y = [-40, -45, -30, -55]

plt.stem(num, x)
plt.stem(num, y)
```

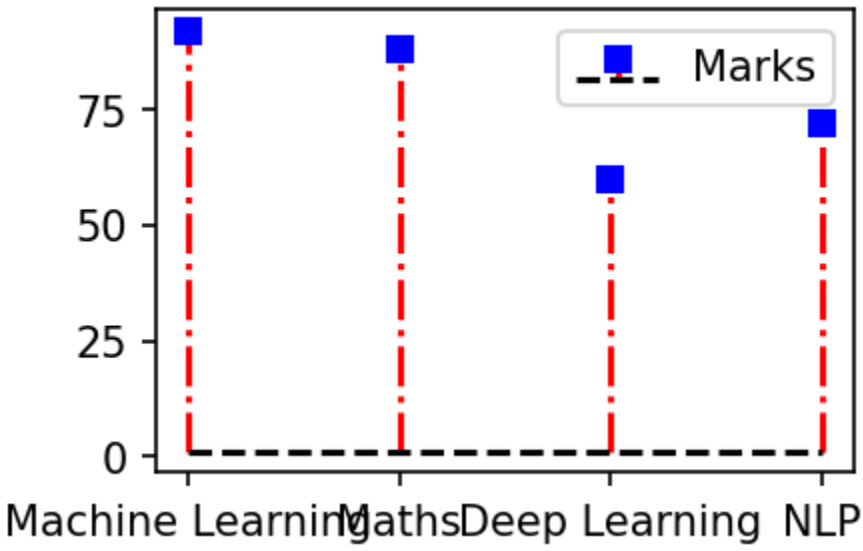
Out[31]: <StemContainer object of 3 artists>



In [32]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Parameters in stem plot
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]

plt.stem(sub, std1,
         linefmt='r-.',   # The first character ('r', 'b', 'g') specifies the color:
         markerfmt='bs',  # The second character ('--', 's', '^', 'o', '-.', ':') spe
         bottom=1, # Ploting from 1
         use_line_collection=True, # By default it's true. If it's false it will chan
         basefmt='k--',   # By default it's 'C3-'. 'k--', which means that the baseli
         orientation ='vertical', # By default it's 'vertical'
         label = 'Marks')

plt.legend()
```

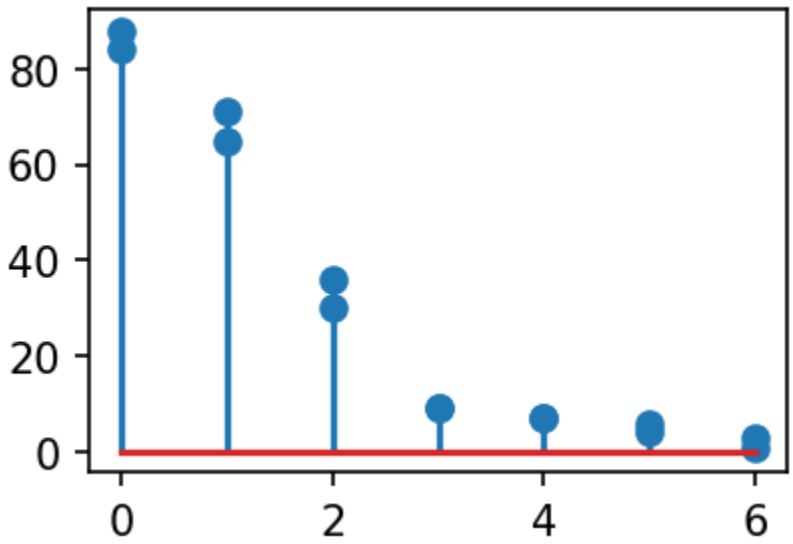
```
<ipython-input-32-e8cfc6e2531b>:7: MatplotlibDeprecationWarning: The 'use_line_colle
ction' parameter of stem() was deprecated in Matplotlib 3.6 and will be removed two
minor releases later. If any parameter follows 'use_line_collection', they should be
passed as keyword, not positionally.
  plt.stem(sub, std1,
```

Out[32]:
```
<matplotlib.legend.Legend at 0x7931c371b1c0>
```



In [33]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Viewing male and female stem graph
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

# Create a vertical stem plot.
plt.stem(num, x)
plt.stem(num, y)
```

Out[33]:
```
<StemContainer object of 3 artists>
```



Filled area between two horizontal curves:

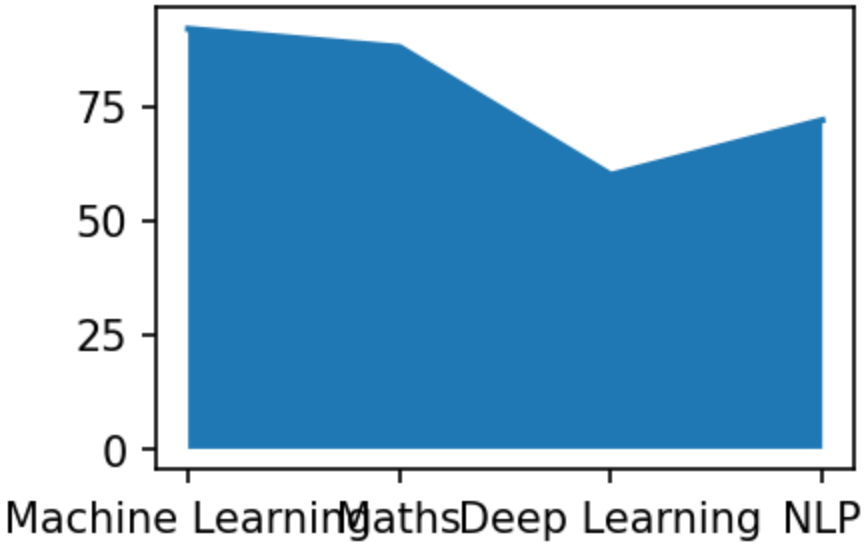
plt.fill_between(x, y1, y2)/ plt.plot(x,y) plt.fill_between(), where x and y can be list or tuple

The `plt.fill_between()` function in `matplotlib.pyplot` has the following parameters:

1. **x**: An array of length N representing the x-coordinates of the nodes defining the curves.
2. **y1**: An array of length N or a scalar representing the y-coordinates of the nodes defining the first curve.
3. **y2**: An array of length N or a scalar (default: 0) representing the y-coordinates of the nodes defining the second curve.
4. **color**: Color or array-like of color, default: None.
5. **where**: An array of boolean values of length N (optional) used to exclude some horizontal regions from being filled.
6. **interpolate**: An optional boolean parameter (default: False) that is relevant if where is used and the two curves are crossing each other.
7. **step**: An optional parameter that accepts one of the three values: 'pre', 'post', or 'mid'. It specifies where the steps will occur.

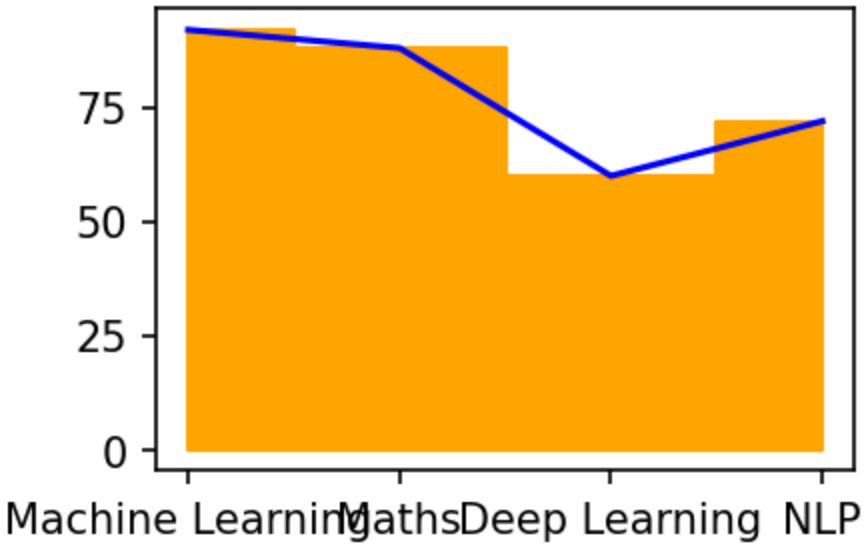
In [34]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Filled area between
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]

plt.plot(sub, std1)
plt.fill_between(sub, std1)
```

Out[34]: <matplotlib.collections.PolyCollection at 0x7931c1718670>

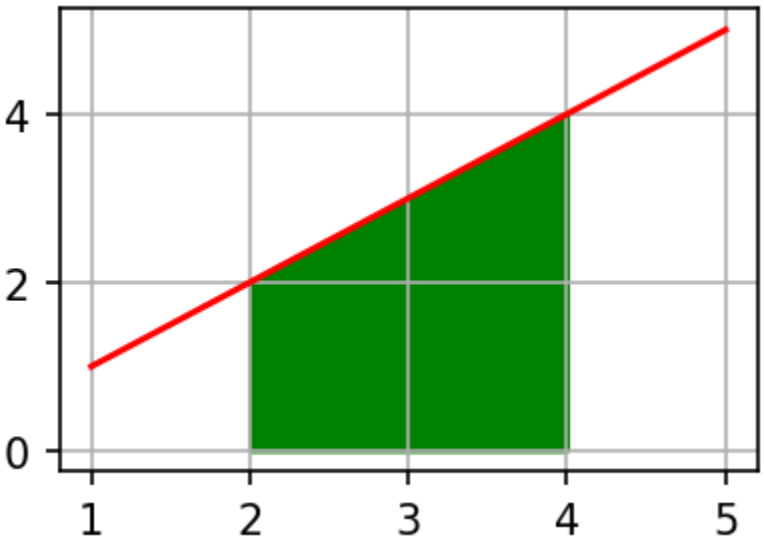


In [35]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# parameters used inside plt.fill_between()
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]

plt.plot(sub, std1, color='b')
plt.fill_between(sub, std1,
                 color= 'orange',
                 step='mid') # We have: pre, post and mid
```

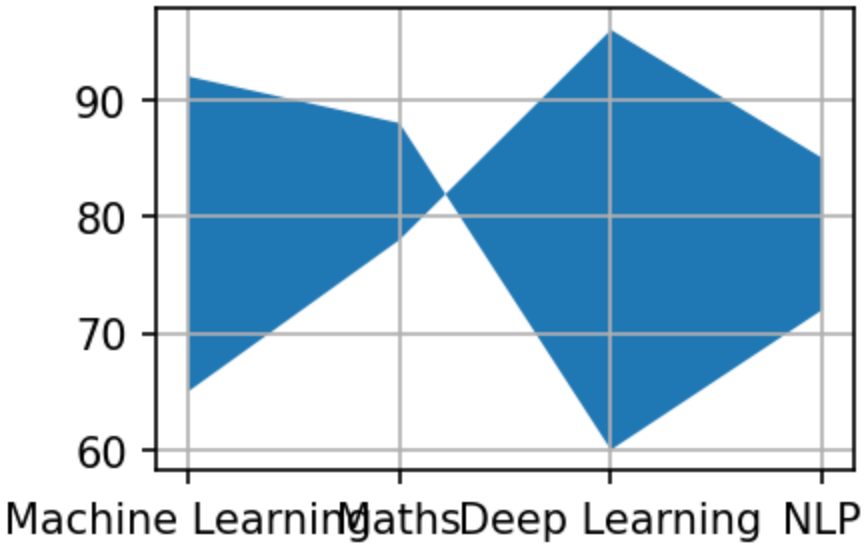
Out[35]: <matplotlib.collections.PolyCollection at 0x7931c1762b30>



In [36]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Using where function
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 2, 3, 4, 5])

plt.plot(x,y, color='r')

plt.fill_between(x, y,
                 color='g',
                 where = (x>=2) & (x<=4)) # For this calculation, we can't do it in l
plt.grid()
```

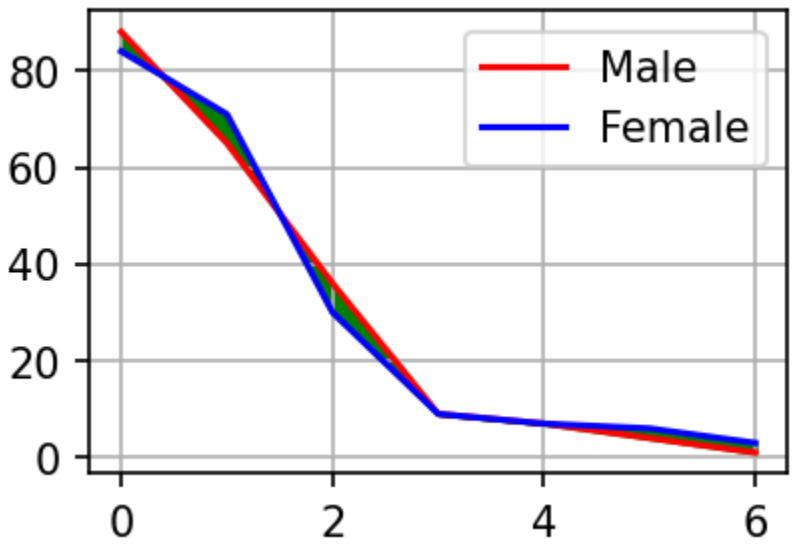


In [37]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Filled area between two horizontal curves
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]
std2 = [65, 78, 96, 85]

plt.fill_between(sub, std1, std2)
plt.grid()
plt.show()
```



In [38]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Viewing male and female in fill between graph
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

plt.plot(num, x, color='r', label= 'Male')
plt.plot(num, y, color='b', label= 'Female')

plt.fill_between(num, x, y, color= 'g')
plt.grid()
plt.legend()
```

Out[38]:
```
<matplotlib.legend.Legend at 0x7931c1673790>
```



Stack Area Plot:

plt.stackplot(x, y), where x and y can be list or tuple

The `plt.stackplot()` function in `matplotlib.pyplot` has the following parameters:

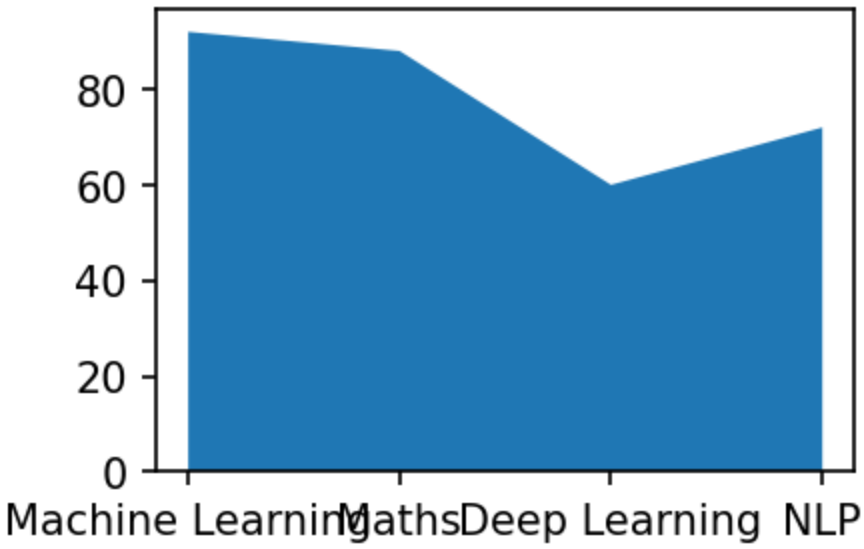
1. **x**: a 1-D array of values.
2. **y**: a 2-D array of values. The data is assumed to be unstacked. Each of the following calls is legal: stackplot(x, y) where y has shape (M, N) or stackplot(x, y1, y2, y3) where y1, y2, y3, y4 have length N.
3. **labels**: A sequence of labels to assign to each data series.
4. **colors**: A sequence of colors to be cycled through and used to color the stacked areas.
5. **baseline**: Method used to calculate the baseline. It can be ‘zero’, ‘sym’, ‘wiggle’, 'weighted_wiggle’.
6. **data**: If given, all parameters also accept a string s, which is interpreted as data [s] (unless this raises an exception).

The baseline parameter is used to denote the method used to calculate the baseline:

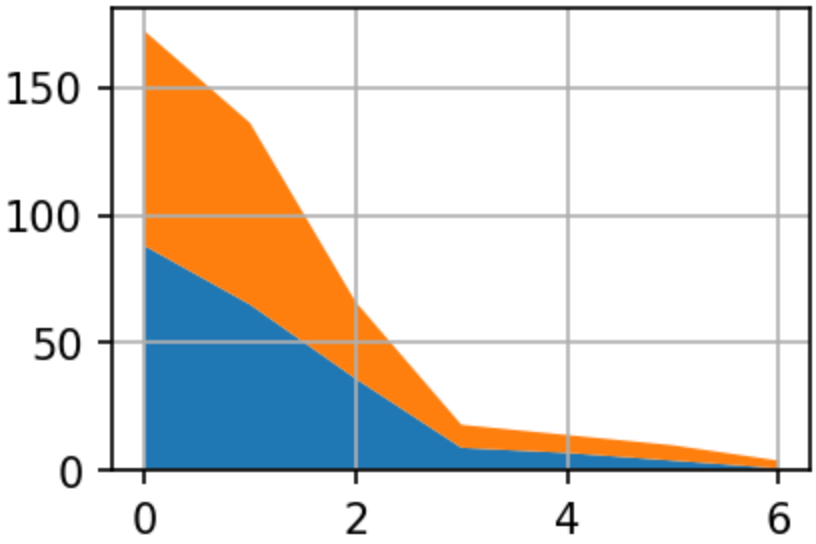
- **zero**: Constant zero baseline, i.e., a simple stacked plot.
- **sym**: Symmetric around zero and is sometimes called 'ThemeRiver’.
- **wiggle**: Minimizes the sum of the squared slopes.
- **weighted_wiggle**: Does the same but weights to account for the size of each layer. It is also called ‘Streamgraph’-layout.

In [39]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Single Stack Plot
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]

plt.stackplot(sub, std1)
```

Out[39]:
```
[<matplotlib.collections.PolyCollection at 0x7931c150b190>]
```

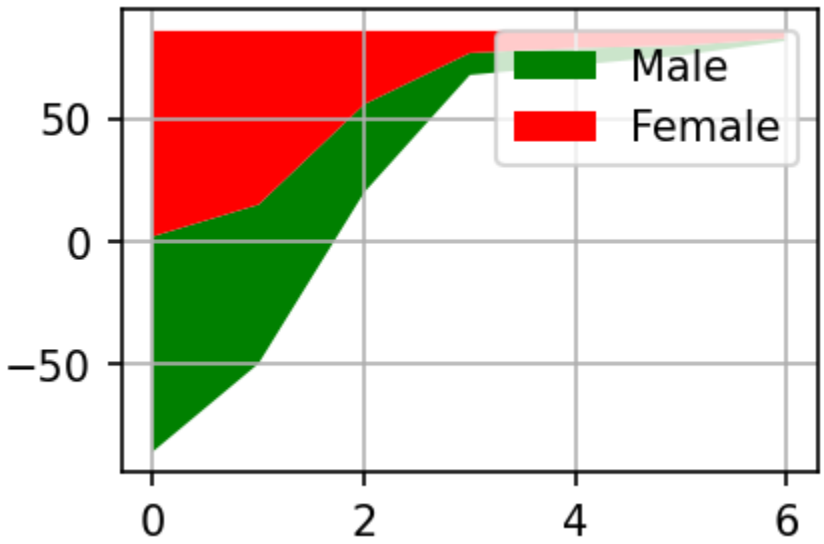


In [40]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Viewing male and female in Stack plot (multiple plot)
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

plt.stackplot(num, x, y)
plt.grid()
```



In [41]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Using parameters in the graph
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

l = ['Male', 'Female']
c = ['g','r']

plt.stackplot(num, x, y,
              labels = l,
              colors = c,
              baseline = 'weighted_wiggle')

plt.grid()
plt.legend()
```

Out[41]:
```
<matplotlib.legend.Legend at 0x7931c13d1b10>
```



Step Plot / Stair Plot

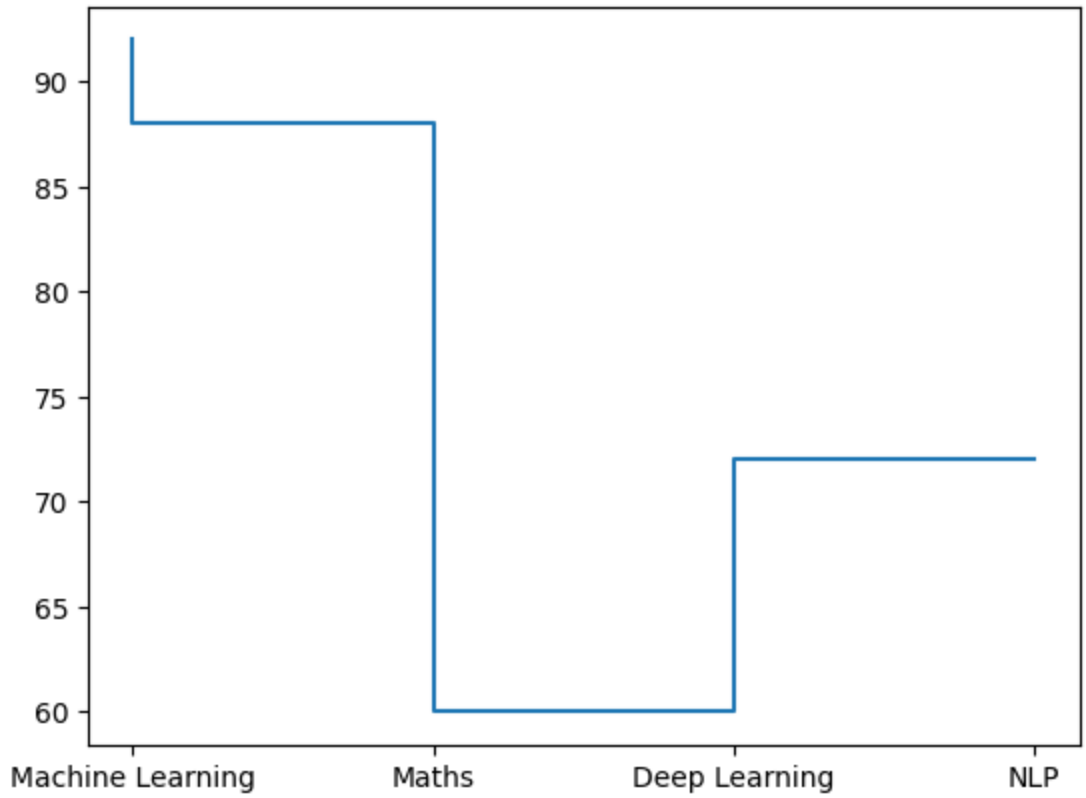
Syntax: plt.step(x,y) / plt.step(y,x), where x and y can be list or tuple

The `plt.step()` function in `matplotlib.pyplot` has the following parameters:

1. **x**: An 1-D array of values representing the X-axis.
2. **y**: An 1-D array of values representing the Y-axis.
3. **fmt**: A formatting string that specifies the line color, marker type, etc.
4. **data**: Two iterables containing the label names for labeled data.
5. **where**: A parameter used to decide the position of the vertical line, which can take one of the following values: 'pre', 'post', or 'mid'.
6. **marker**: A string defining the color and style of markers.
7. **ms**: Will define the marker size
8. **mfc**: Will define the marker color

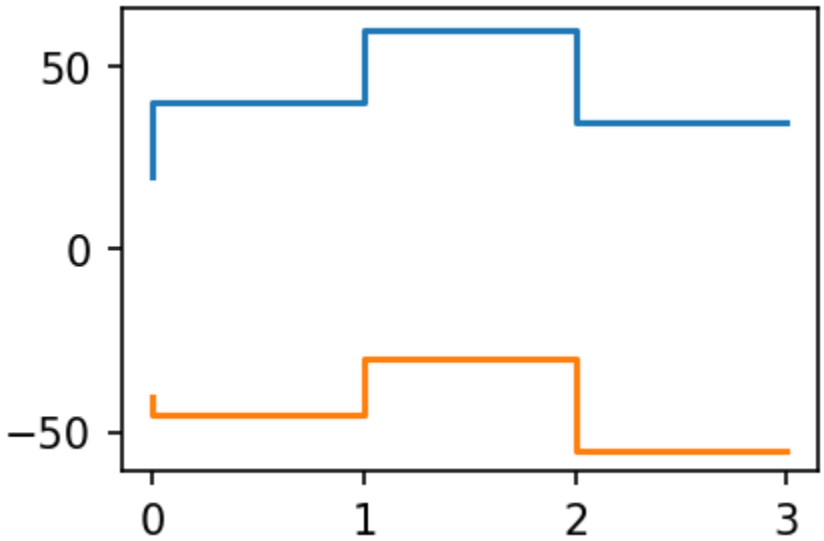
In [42]:
```
# Step plot
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]

plt.step(sub, std1)
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x7931c143f670>]
```

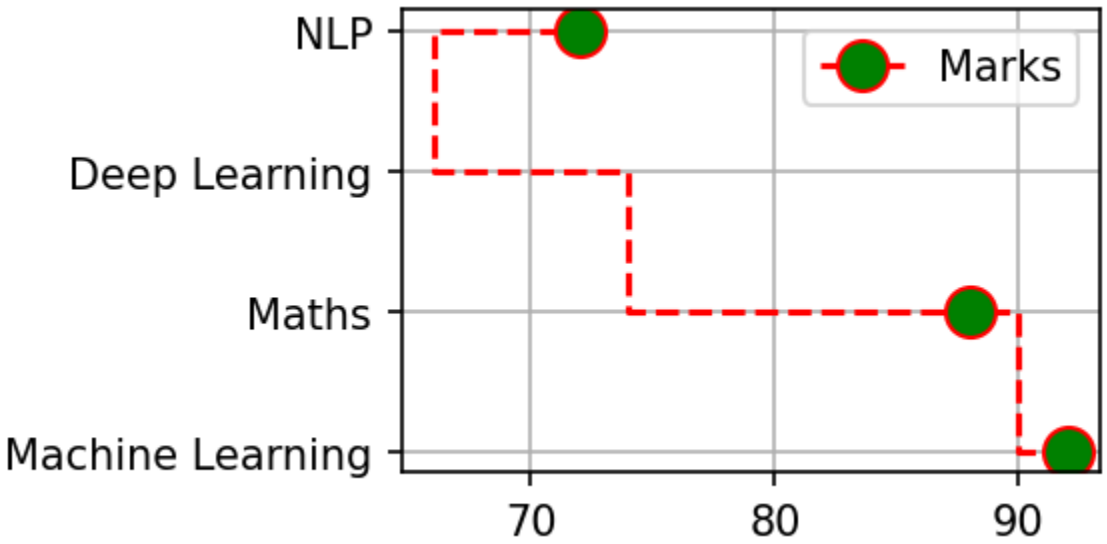


In [43]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# With negative plotting
num = [0, 1, 2, 3]
x = [20, 40, 60, 35]
y = [-40, -45, -30, -55]


plt.step(num, x)
plt.step(num, y)
```

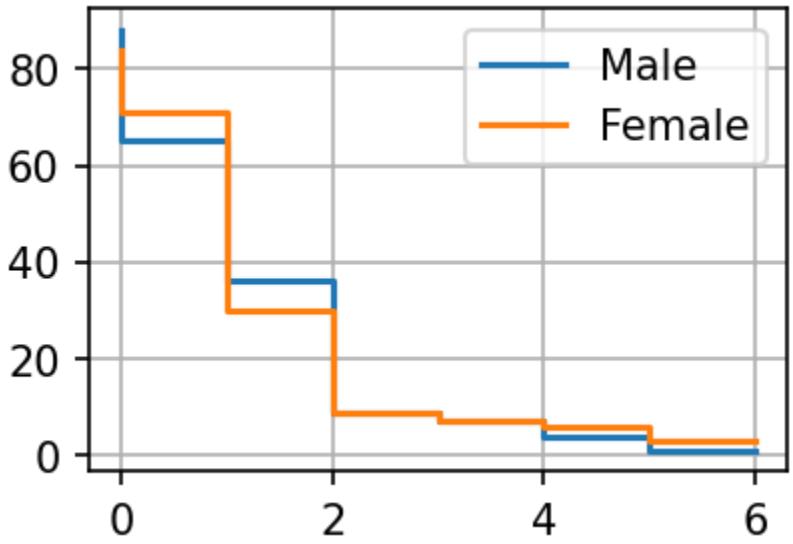
Out[43]:
```
[<matplotlib.lines.Line2D at 0x7931c12a7970>]
```



In [44]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Parameters in stem plot
sub = ["Machine Learning", "Maths", "Deep Learning", "NLP"]
std1 = [92, 88, 60, 72]

plt.step(std1, sub,
         'r--',  # Also we can use other fmt, and can use color parameter also
         where='mid', # We have: pre, post and mid
         marker= 'o',
         ms= 12,
         mfc= 'g',
         label= 'Marks')

plt.legend()
plt.grid() # It will show grid in background
```



In [45]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Viewing male and female step graph
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

# Create a step plot.
plt.step(num, x, label = 'Male')
plt.step(num, y, label = 'Female')

plt.legend()
plt.grid() # It will show grid in background
```



2. Statistical distributions

Histogram:

plt.hist(x)

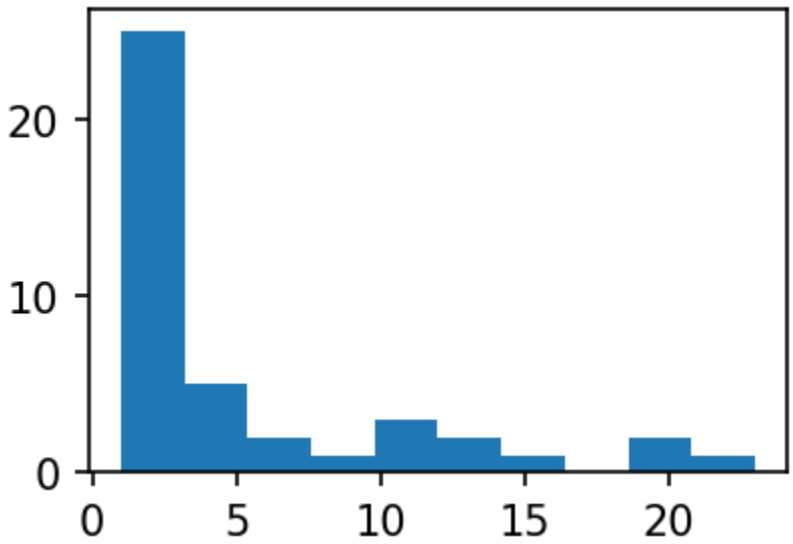
The `plt.hist()` function in `matplotlib.pyplot` has the following parameters:

1. **x**: This is the sequence of data that you want to plot in the histogram.
2. **bins**: This can be an integer, sequence, or string. If an integer, it defines the number of equal-width bins in the range. If a sequence, it defines the bin edges.
3. **range**: This is the lower and upper range of the bins.
4. **density**: If True, the function will draw and return a probability density.
5. **weights**: This is an array of weights, of the same shape as x.
6. **cumulative**: If True, then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values.
7. **bottom**: This is the location of the bottom baseline of each bin.
8. **histtype**: This parameter is used to draw type of histogram. Options include ‘bar’, ‘barstacked’, ‘step’, 'stepfilled’.
9. **align**: This controls how the histogram is plotted. Options include ‘left’, ‘mid’, 'right’.
10. **orientation**: This controls the orientation of the histogram. Options include ‘horizontal’, 'vertical’.
11. **rwidth**: This is the relative width of the bars as a fraction of the bin width.
12. **log**: If True, the histogram axis is set to a log scale.
13. **edgecolors**: This is the color or sequence of colors of the marker edges.
14. **color**: This is a color spec or sequence of color specs, one per dataset.
15. **label**: This is a string, or sequence of strings to match multiple datasets.

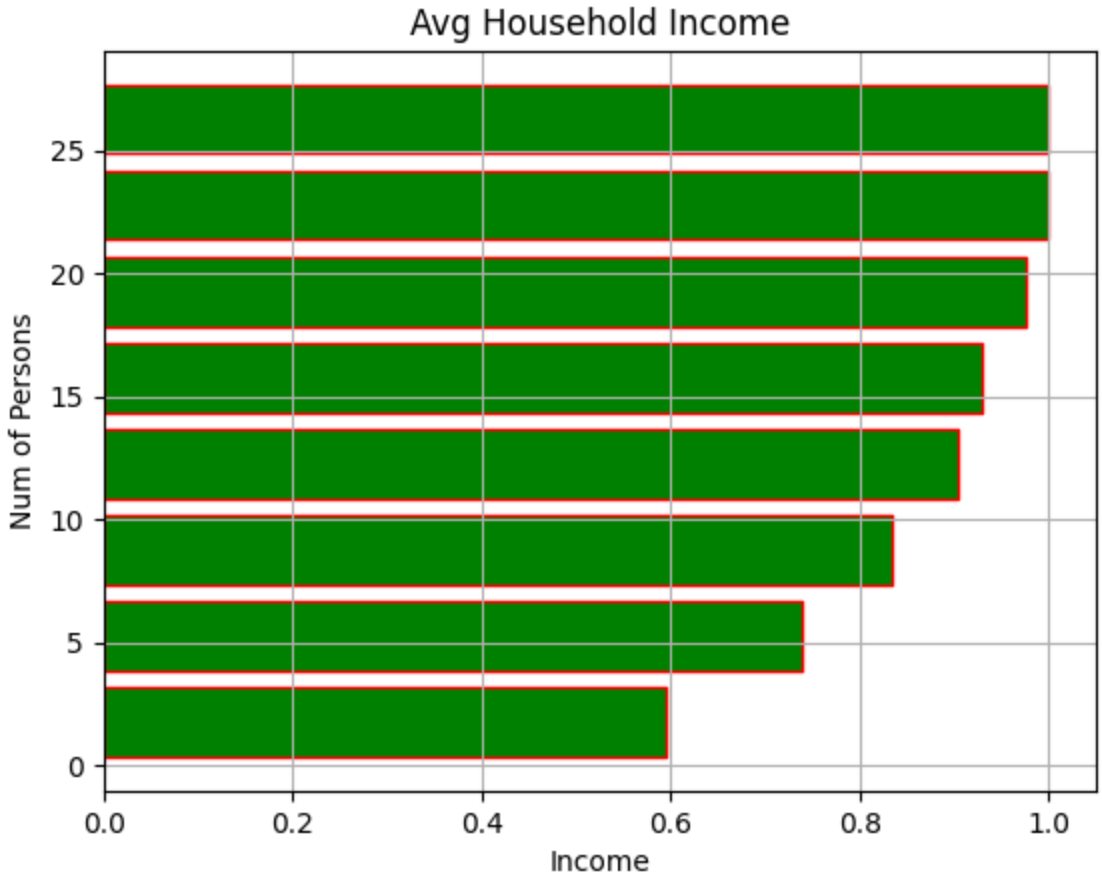
16. **stacked**: If True, multiple data are stacked on top of each other If false multiple data are arranged side by side if histtype is ‘bar’ or on top of each other if histtype is 'step’.
17. **data**: This parameter is used when the x values are a string, in which case this should be a data object that the string can be looked up in.

In [46]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Avg household income of the persons in the locality
inc = file.AVG_HOUSEHOLD_INCOME.value_counts().values

plt.hist(inc)
```

Out[46]:
```
(array([25.,  5.,  2.,  1.,  3.,  2.,  1.,  0.,  2.,  1.]),
 array([ 1. ,  3.2,  5.4,  7.6,  9.8, 12. , 14.2, 16.4, 18.6, 20.8, 23. ]),
 <BarContainer object of 10 artists>)
```

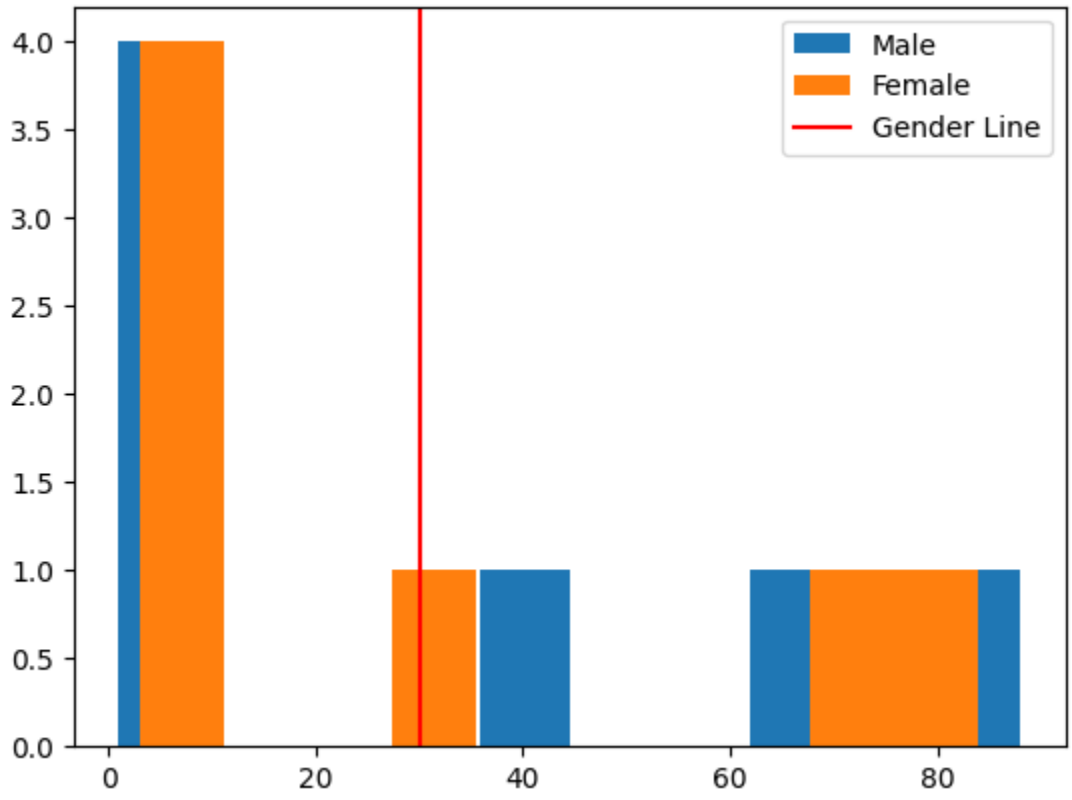


In [47]:
```
# Using parameters
inc = file.AVG_HOUSEHOLD_INCOME.value_counts().values

plt.title("Avg Household Income")
plt.xlabel("Income")
plt.ylabel("Num of Persons")

plt.hist(inc,
         bins= 8, # means that the range will be divided into 10 equal-width bins
         range=(0, 28),
         density= True,
         cumulative= True, # Default: False, True will reverse the graph
         histtype= 'barstacked', # 'bar', 'barstacked', 'step', 'stepfilled' also ava
         alpha= 1,
         align= 'mid', # 'left', 'mid', 'right' also avail
         orientation= 'horizontal', # Default: Vertical
         rwidth= 0.8,
         edgecolor= 'r',
         color= 'g',
         stacked= False)
plt.grid()
```



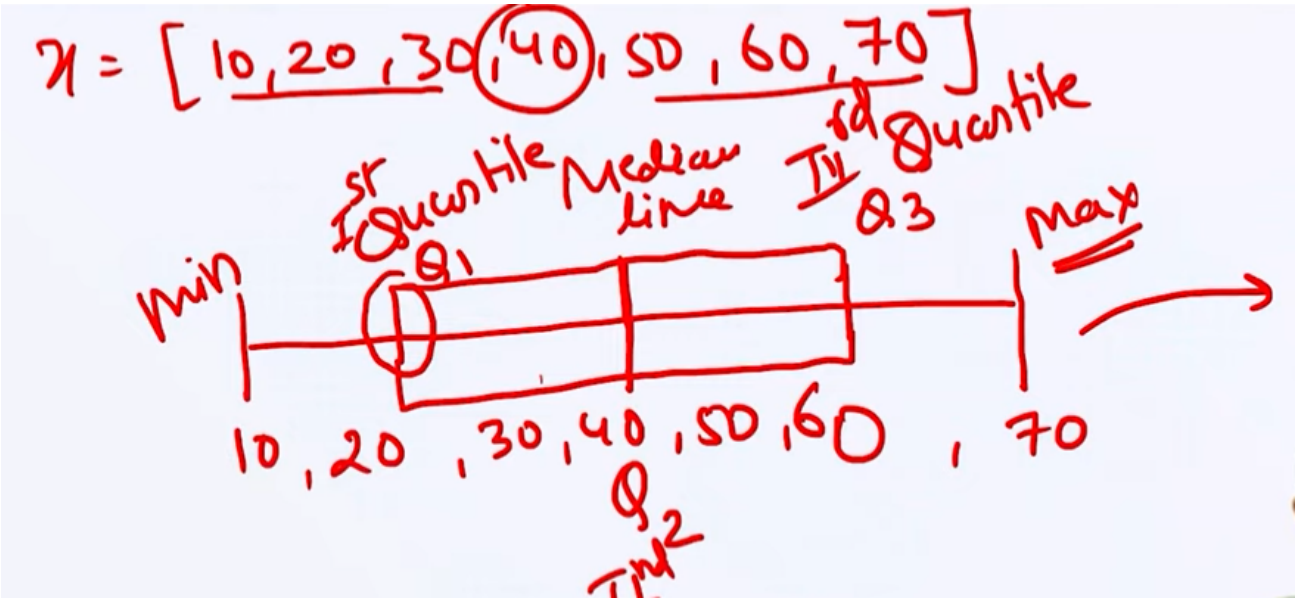
In [48]:
```
# Using the parameters of contour
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values

plt.hist(x, label = 'Male')
plt.hist(y, label = 'Female')

plt.axvline(30, color= 'r', label = 'Gender Line') # Line inside a chart
plt.legend()
```

Out[48]: <matplotlib.legend.Legend at 0x7931c361ae60>



Box plot:

plt.boxplot(X)

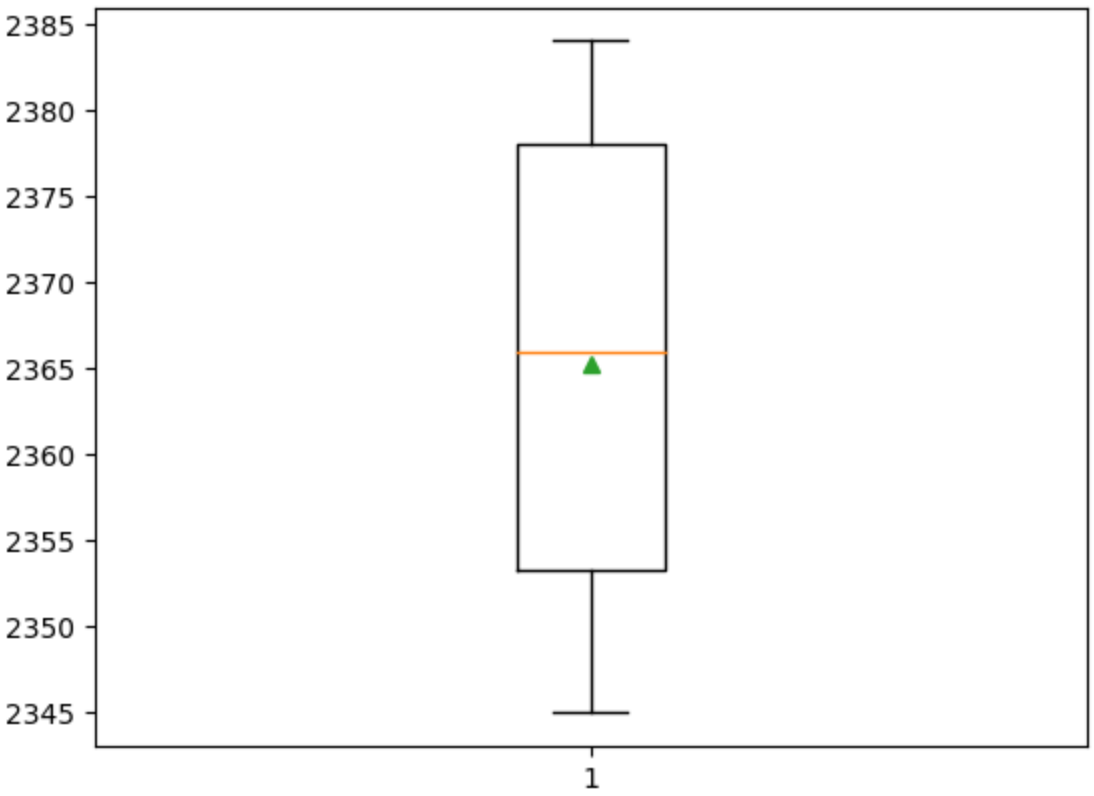


The `plt.box()` function in `matplotlib.pyplot` has the following parameters:

1. **data**: array or sequence of array to be plotted
2. **notch**: optional parameter accepts boolean values
3. **vert**: optional parameter accepts boolean values false and true for horizontal and vertical plot respectively
4. **bootstrap**: optional parameter accepts int specifies intervals around notched boxplots
5. **usermedians**: optional parameter accepts array or sequence of array dimension compatible with data
6. **positions**: optional parameter accepts array and sets the position of boxes
7. **widths**: optional parameter accepts array and sets the width of boxes
8. **patch_artist**: optional parameter having boolean values
9. **labels**: sequence of strings sets label for each dataset
10. **meanline**: optional having boolean value try to render meanline as full width of box
11. **order**: optional parameter sets the order of the boxplot
12. **showmeans**: It will show the mean position
13. **sym**: To change the outliers color
14. **boxprops**: To change the box color
15. **capprops**: To change the line color
16. **whiskerprops**: To change the whisker color

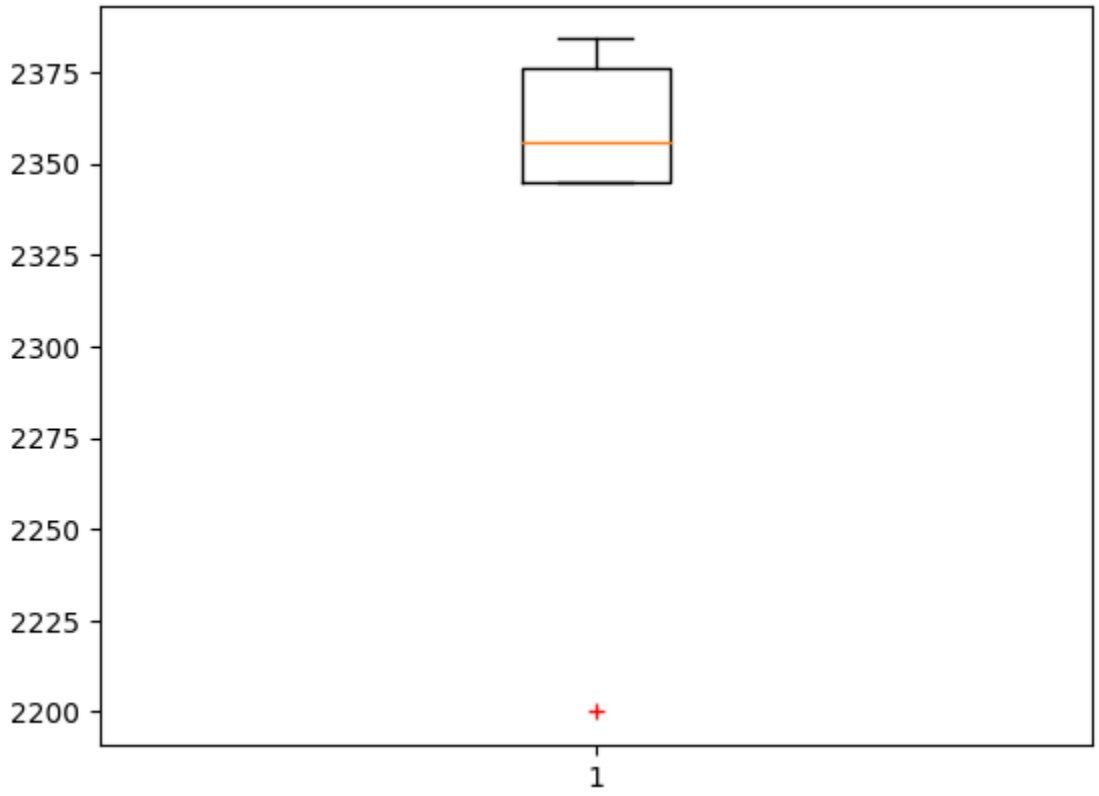
In [49]:
```
# Printing the Reliance share price through box plot
ril = [2345, 2356, 2376, 2384]

plt.boxplot(ril, showmeans = True)
```

Out[49]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7931c135a890>,
  <matplotlib.lines.Line2D at 0x7931c1359060>],
 'caps': [<matplotlib.lines.Line2D at 0x7931c135b4c0>,
  <matplotlib.lines.Line2D at 0x7931c135af50>],
 'boxes': [<matplotlib.lines.Line2D at 0x7931c135a7a0>],
 'medians': [<matplotlib.lines.Line2D at 0x7931c1359ed0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7931c1359fc0>],
 'means': [<matplotlib.lines.Line2D at 0x7931c1359900>]}
```

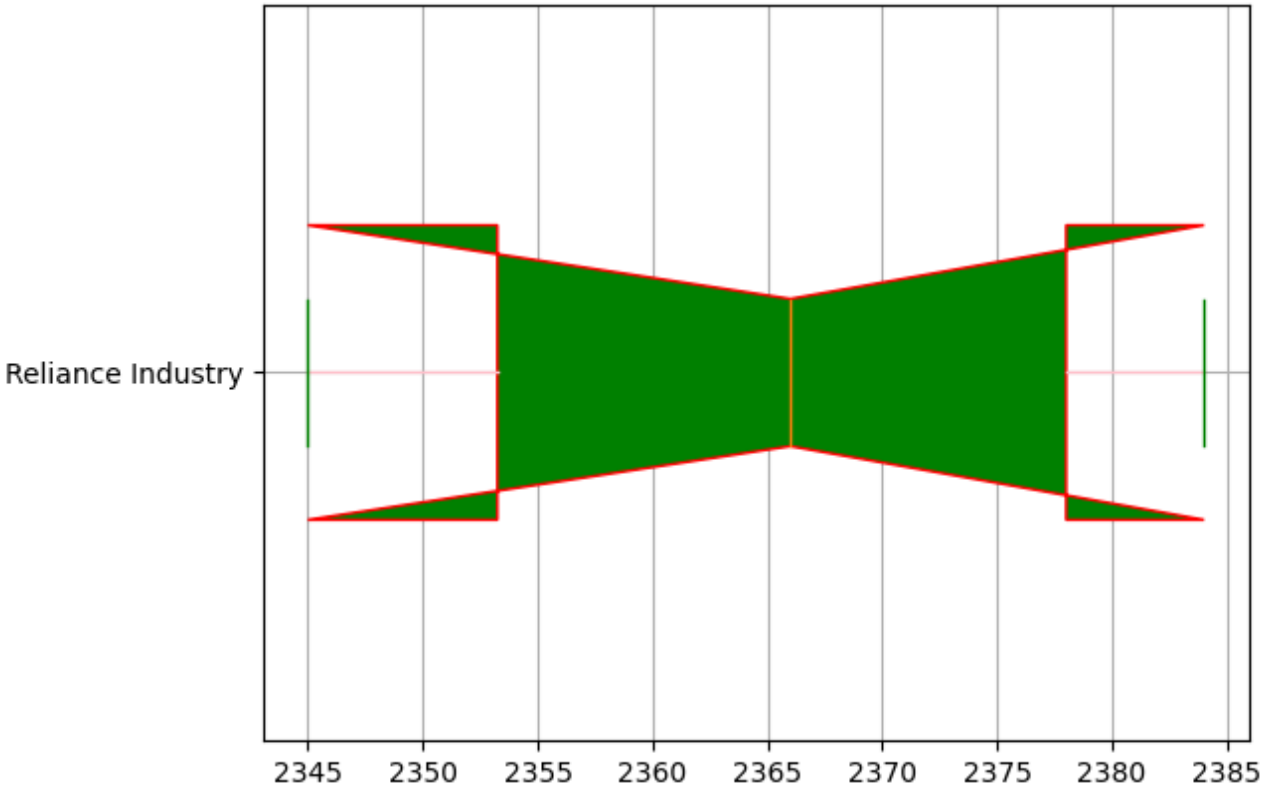


In [50]:
```
# Printing the Reliance share price with a outlier
ril = [2345, 2356, 2376, 2384, 2200]

plt.boxplot(ril,
            sym='r+') # To change the color of outlier
```

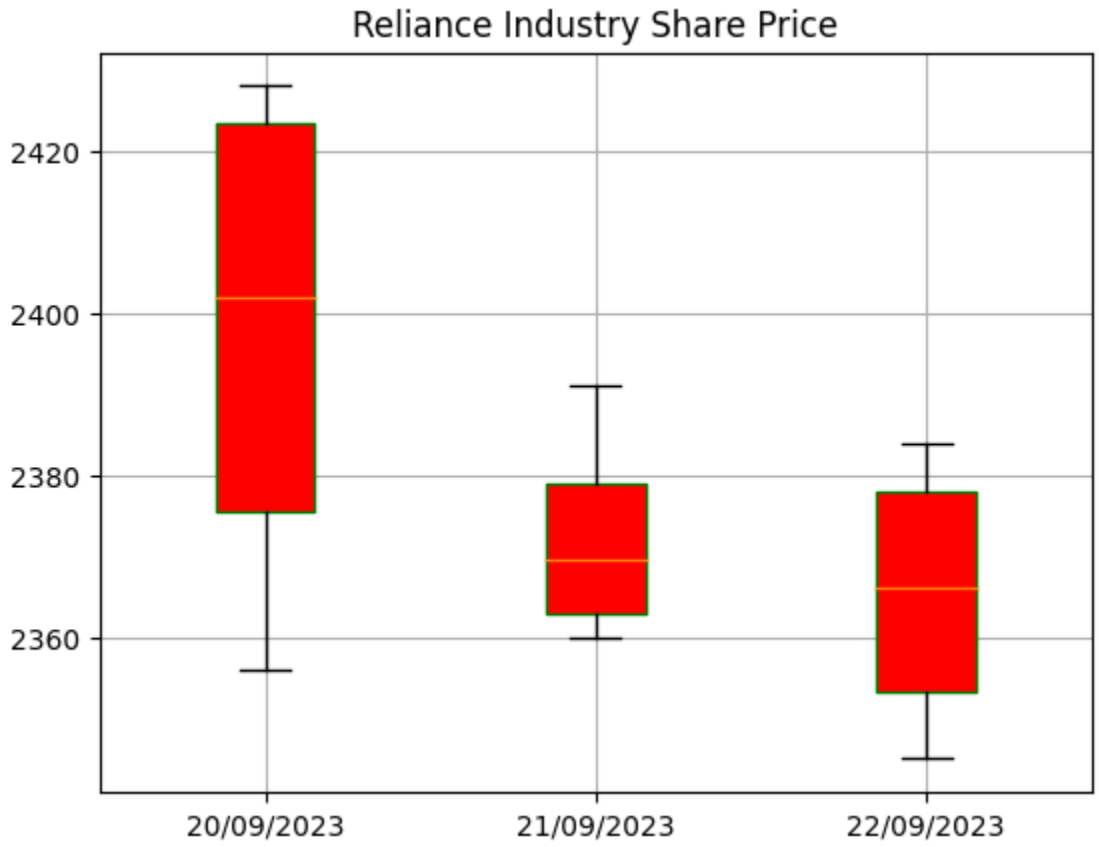
Out[50]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7931c13ae740>,
  <matplotlib.lines.Line2D at 0x7931c13ae080>],
 'caps': [<matplotlib.lines.Line2D at 0x7931c13ada50>,
  <matplotlib.lines.Line2D at 0x7931c13ac790>],
 'boxes': [<matplotlib.lines.Line2D at 0x7931c13aea40>],
 'medians': [<matplotlib.lines.Line2D at 0x7931c13adb10>],
 'fliers': [<matplotlib.lines.Line2D at 0x7931c13ad540>],
 'means': []}
```



In [51]:
```
# Using the parameters
ril = [2345, 2356, 2376, 2384]

plt.boxplot(ril,
            notch = True,
            vert = False, # Default vertical is True
            widths = 0.4, # Default: 0.2
            bootstrap = 1000, # confidence intervals around the median for notched bo
            patch_artist = True, # Default: False
            labels = ["Reliance Industry"],
            boxprops = dict(color = 'r', facecolor = 'g'), # Facecolor will change th
            capprops = dict(color = 'g'),
            whiskerprops = dict(color = 'pink')
            )
plt.grid()
```



In [52]:
```
# Plotting of multiple box plot

day_1 = [2356, 2382, 2422, 2428]
day_2 = [2360, 2364, 2375, 2391]
day_3 = [2345, 2356, 2376, 2384]

ril = [day_1, day_2, day_3]

plt.title("Reliance Industry Share Price")
plt.boxplot(ril,
            patch_artist = True,
            labels = ['20/09/2023', '21/09/2023', '22/09/2023'],
            boxprops = dict(color = 'g', facecolor = 'r'))
plt.grid()
```

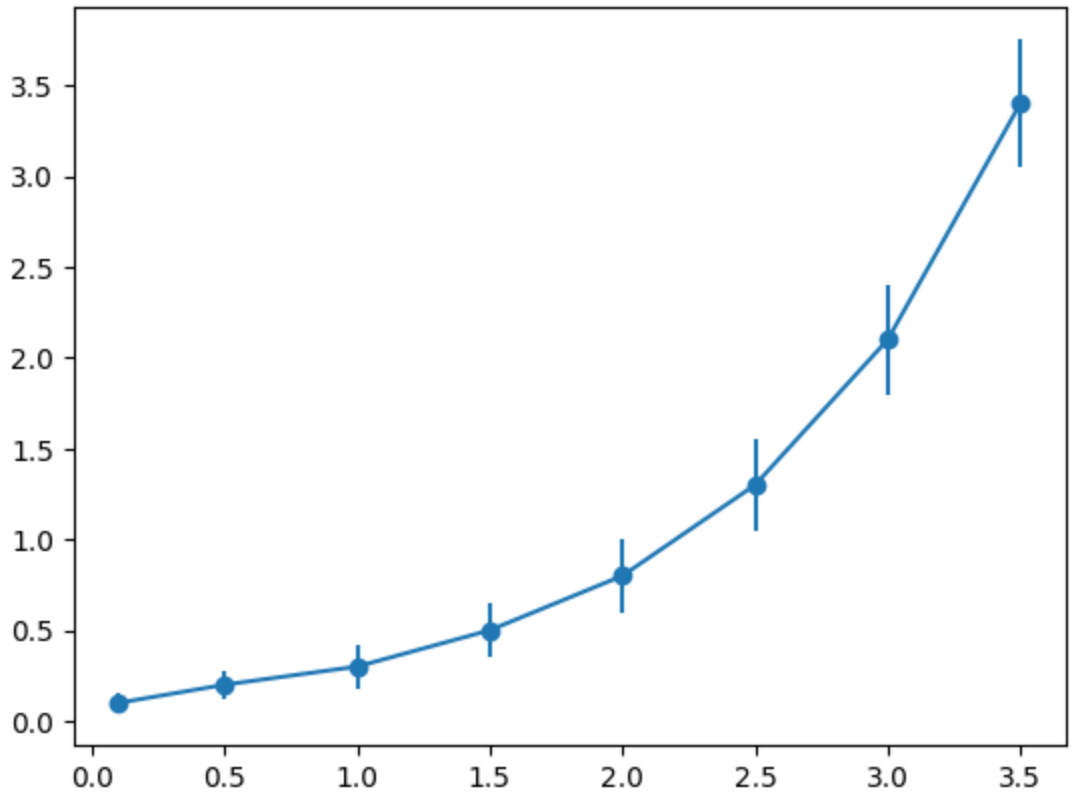


Error bar plot:

plt.errorbar(x, y, yerr, xerr)

In [53]:
```
# Create data
x_data = [0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5]
y_data = [0.1, 0.2, 0.3, 0.5, 0.8, 1.3, 2.1, 3.4]
error = [0.05, 0.08, 0.12, 0.15, 0.20, 0.25, 0.30, 0.35]

# Create error bars
plt.errorbar(x_data, y_data, yerr=error, fmt='-o')
```

Out[53]: <ErrorbarContainer object of 3 artists>

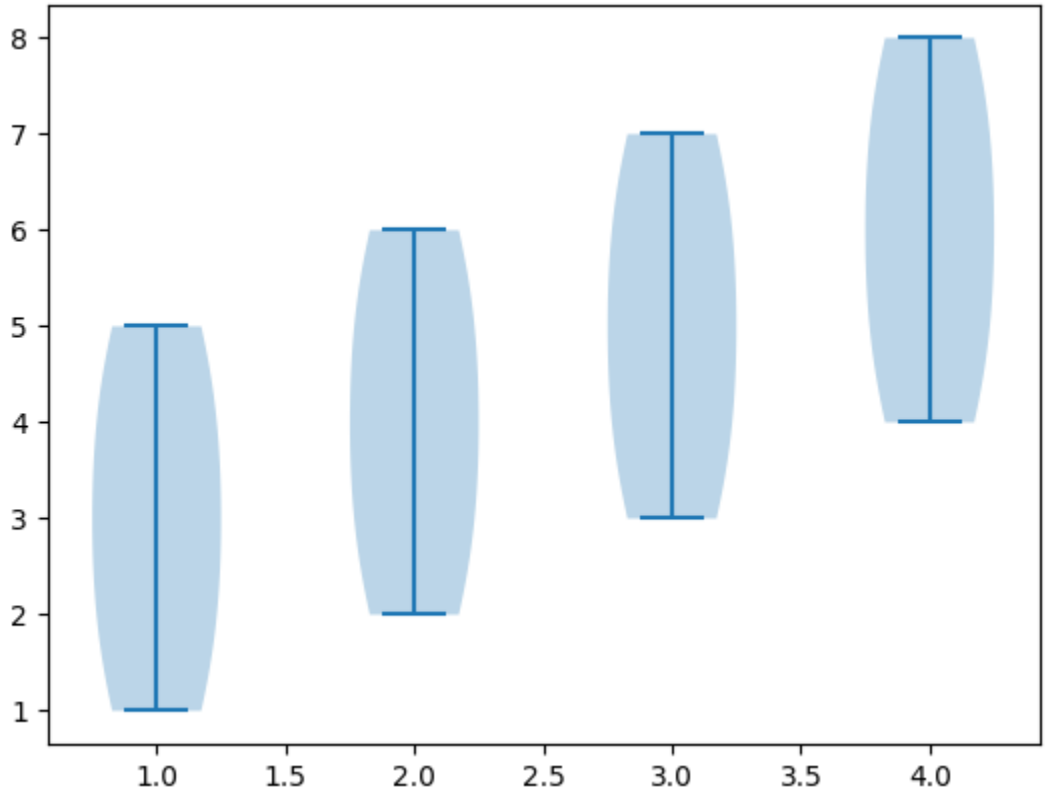


Violin Plot:

plt.violinplot(D)

In [54]:
```
# Create data
data = [[1, 2, 3, 4, 5], [2, 3, 4, 5, 6], [3, 4, 5, 6, 7], [4, 5, 6, 7, 8]]

# Create violin plot
plt.violinplot(data)
```

Out[54]:
```
{'bodies': [<matplotlib.collections.PolyCollection at 0x7931c1625db0>,
  <matplotlib.collections.PolyCollection at 0x7931c0fa2710>,
  <matplotlib.collections.PolyCollection at 0x7931c0fa2a70>,
  <matplotlib.collections.PolyCollection at 0x7931c0fa2dd0>],
 'cmaxes': <matplotlib.collections.LineCollection at 0x7931c1168220>,
 'cmins': <matplotlib.collections.LineCollection at 0x7931c0fa3430>,
 'cbars': <matplotlib.collections.LineCollection at 0x7931c0fa3820>}
```

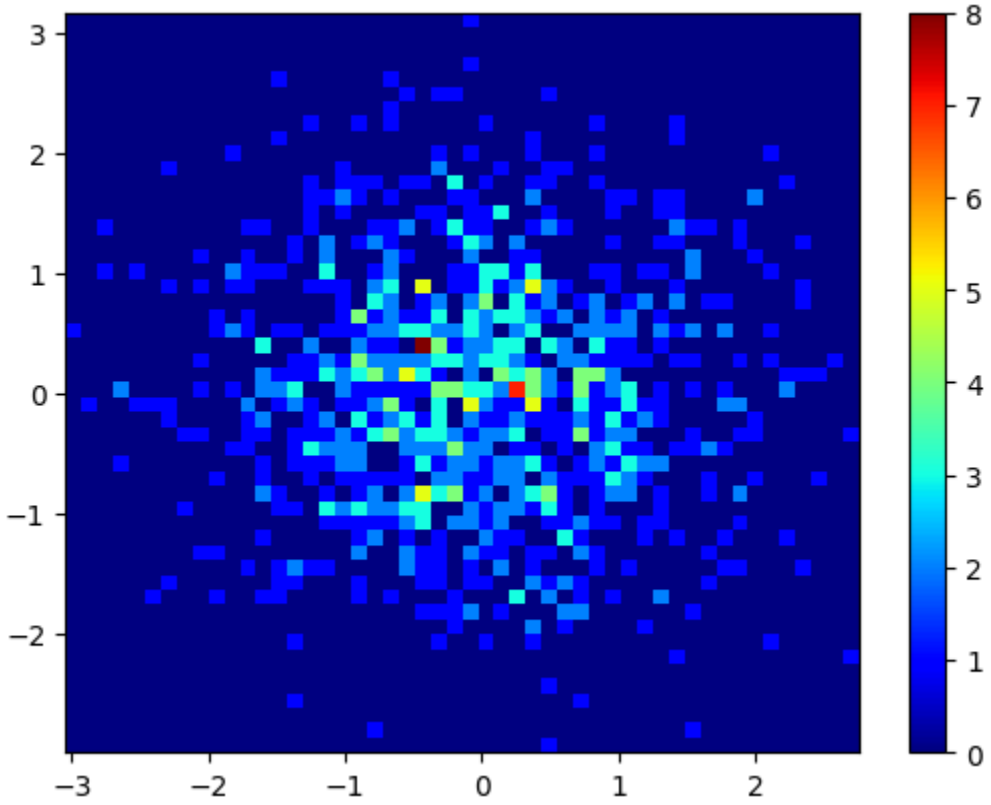


Hist 2D:

plt.hist2D(x, y)

In [55]:
```
# Create data
np.random.seed(0)
x = np.random.normal(0, 1, 1000)
y = np.random.normal(0, 1, 1000)

# Create 2D histogram
plt.hist2d(x, y, bins=(50, 50), cmap=plt.cm.jet)

plt.colorbar()
```

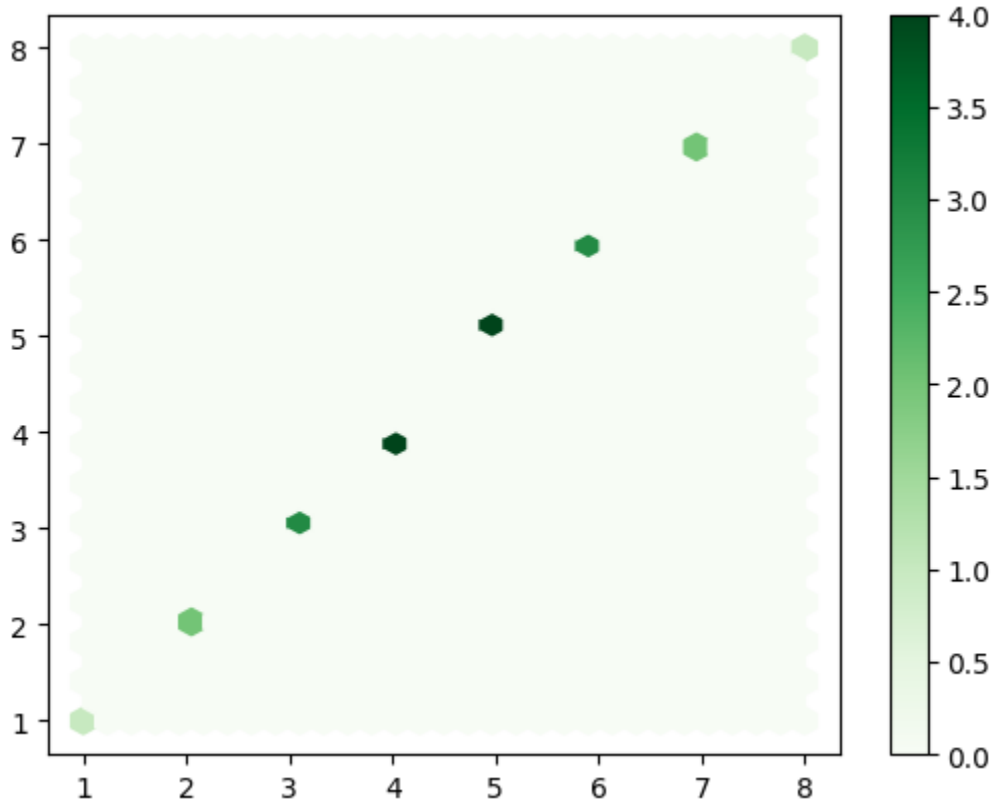
Out[55]: <matplotlib.colorbar.Colorbar at 0x7931c102b460>



Hexbin:

plt.hexbin(x, y)

In [56]:
```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8]
y = [1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8]

# Create a hexbin plot
plt.hexbin(x, y, gridsize=30, cmap='Greens')

# Add a colorbar for reference
plt.colorbar()

# Show the plot
plt.show()
```



Pie Plot

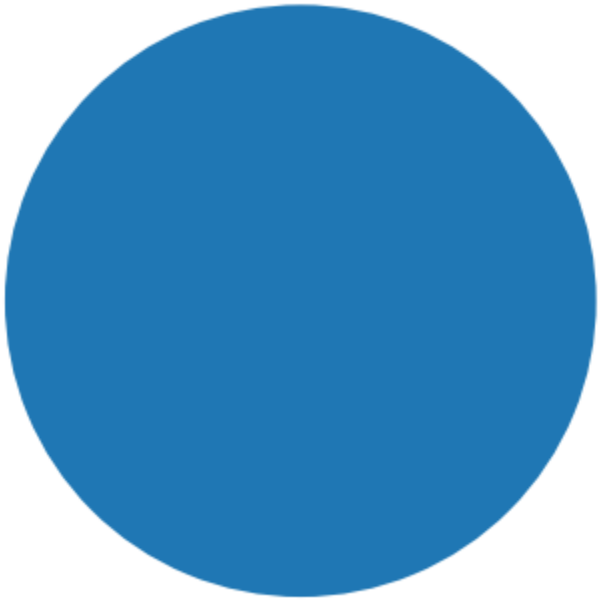
Syntax: plt.pie(x), where x can be list or tuple

The `plt.pie()` function in `matplotlib.pyplot` has the following parameters:

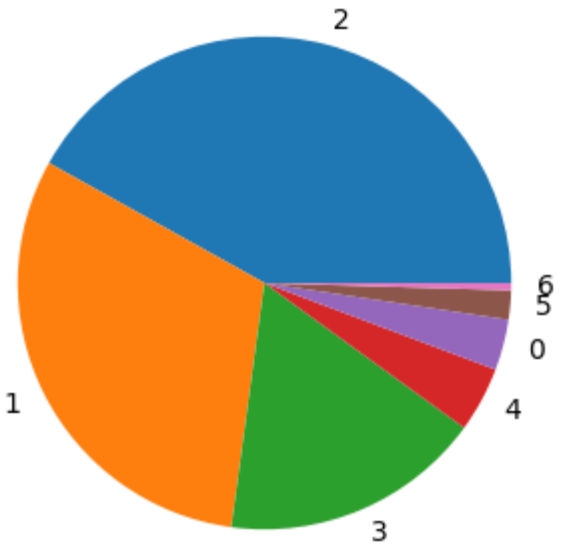
1. **x**: 1D array-like. The wedge sizes.
2. **explode**: array-like, default: None. Specifies the fraction of the radius with which to offset each wedge.
3. **labels**: list, default: None. A sequence of strings providing the labels for each wedge.
4. **colors**: color or array-like of color, default: None. A sequence of colors through which the pie chart will cycle.
5. **autopct**: None or str or callable, default: None. If not None, autopct is a string or function used to label the wedges with their numeric value.
6. **pctdistance**: float, default: 0.6. The relative distance along the radius at which the text generated by autopct is drawn.
7. **labeldistance**: float or None, default: 1.1. The relative distance along the radius at which the labels are drawn.
8. **shadow**: bool or dict, default: False. If bool, whether to draw a shadow beneath the pie.

9. **startangle**: float, default: 0 degrees. The angle by which the start of the pie is rotated, counterclockwise from the x-axis.
10. **radius**: float, default: 1. The radius of the pie.
11. **counterclock**: bool, default: True. Specify fractions direction, clockwise or counterclockwise.
12. **wedgeprops**: dict, default: None. Dict of arguments passed to each patches.Wedge of the pie.
13. **textprops**: dict, default: None. Dict of arguments to pass to the text objects.
14. **center**: (float, float), default: (0, 0). The coordinates of the center of the chart.
15. **frame**: bool, default: False. Plot Axes frame with the chart if true.
16. **rotatelabels**: bool, default: False.
17. **hatch**: str or list, default: None. Hatching pattern applied to all pie wedges or sequence of patterns through which the chart will cycle.
18. **normalize**: bool, default: True .
19. **data**: array-like, optional.

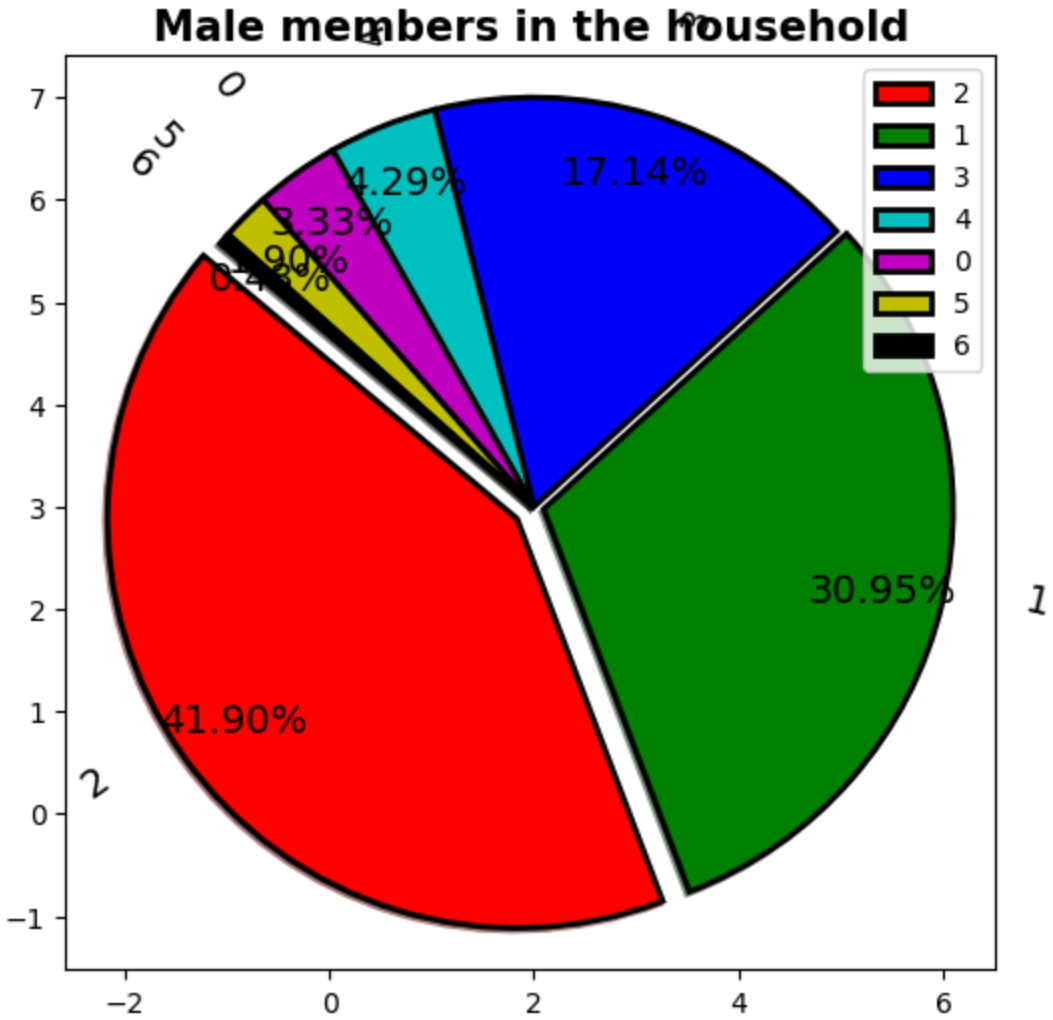
In [57]:
```
plt.pie([1])
plt.show()
```



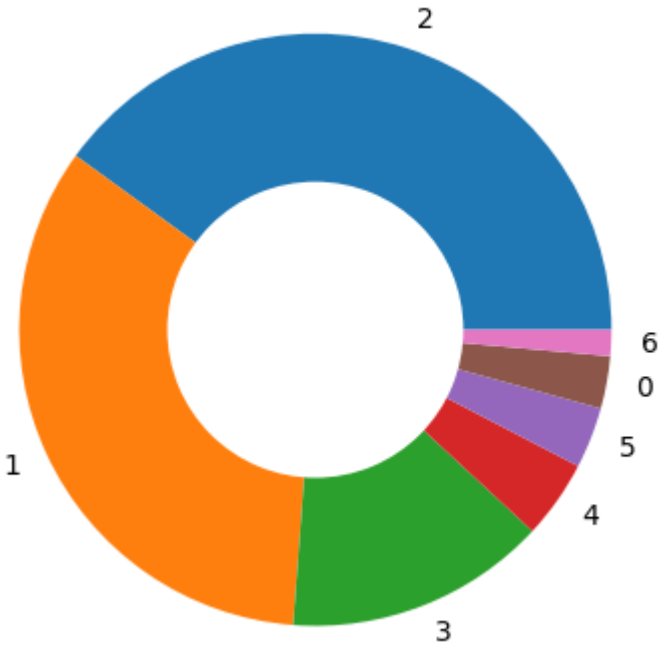
In [58]:
```
# Number of  memberss in household showing through pie chart and percentage
plt.figure(1, figsize=(4,4))

label=file.MALE_MEMBERS.value_counts().index # Use value_counts to get the frequency
count=file.MALE_MEMBERS.value_counts().values # Use value_counts to get the frequency

plt.pie(count, labels=label)
plt.show()
```



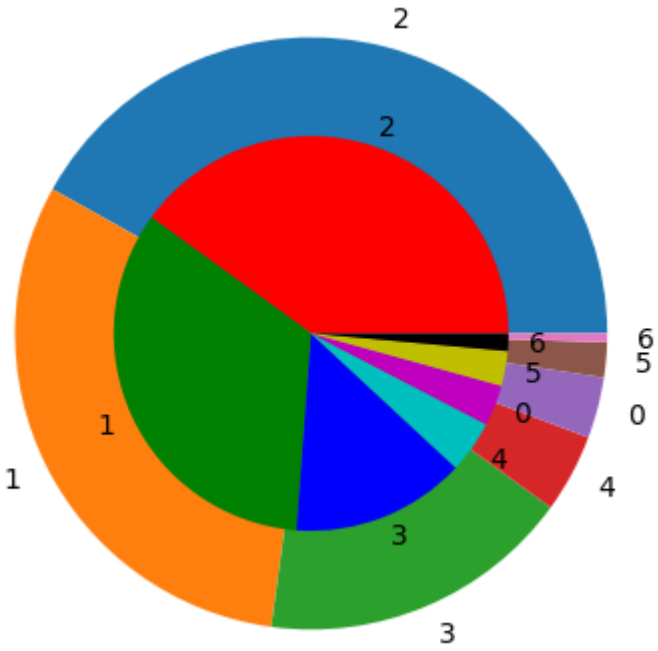
In [59]:
```
# Using the parameters inside plt.pie() graph
plt.figure(1, figsize=(6,6))

label=file.MALE_MEMBERS.value_counts().index
count=file.MALE_MEMBERS.value_counts().values

plt.title("Male members in the household", fontsize = 15, fontweight ="bold")

ex =[0.2, 0.1, 0, 0, 0, 0, 0]
co = ['r', 'g', 'b', 'c', 'm', 'y', 'k']

plt.pie(count,
        explode= ex,
        labels= label,
        colors= co,
        autopct= '%0.2f%%',
        pctdistance= 0.85,
        labeldistance= 1.2,
        shadow= True,
        startangle= 140,
        radius= 4, # By default it's 1
        counterclock= True,
        wedgeprops={'linewidth': 2, 'edgecolor': 'black'}, # This is a dictionary of
        textprops={'fontsize': 14}, # This is a dictionary of arguments. For example,
        center=(2, 3), # The default value is (0, 0). For example, center = (2, 3) se
        frame = True,
        rotatelabels = True)

plt.legend(loc = 'upper right')
plt.show()
```



In [60]:
```
# Number of  memberss in household showing through donout chart and percentage
# We can also create donout chart, through plt.gca().add_artist
plt.figure(1, figsize=(4,4))

label=file.FEMALE_MEMBERS.value_counts().index # Use value_counts to get the frequenc
count=file.FEMALE_MEMBERS.value_counts().values # Use value_counts to get the frequen

plt.pie(count, labels=label, radius = 1.2)
plt.pie([1], colors = 'w', radius = 0.6)
plt.show()
```



In [61]:
```
# Creating multiple pie graph in a chart
plt.figure(1, figsize=(4,4))

label= file.MALE_MEMBERS.value_counts().index
male= file.MALE_MEMBERS.value_counts().values
female= file.FEMALE_MEMBERS.value_counts().values

co = ['r', 'g', 'b', 'c', 'm', 'y', 'k']

plt.pie(male, labels=label, radius = 1.2)
plt.pie(female, labels=label, radius = 0.8, colors = co)
plt.show()
```



3. Gridded data

Display an image on the axes:

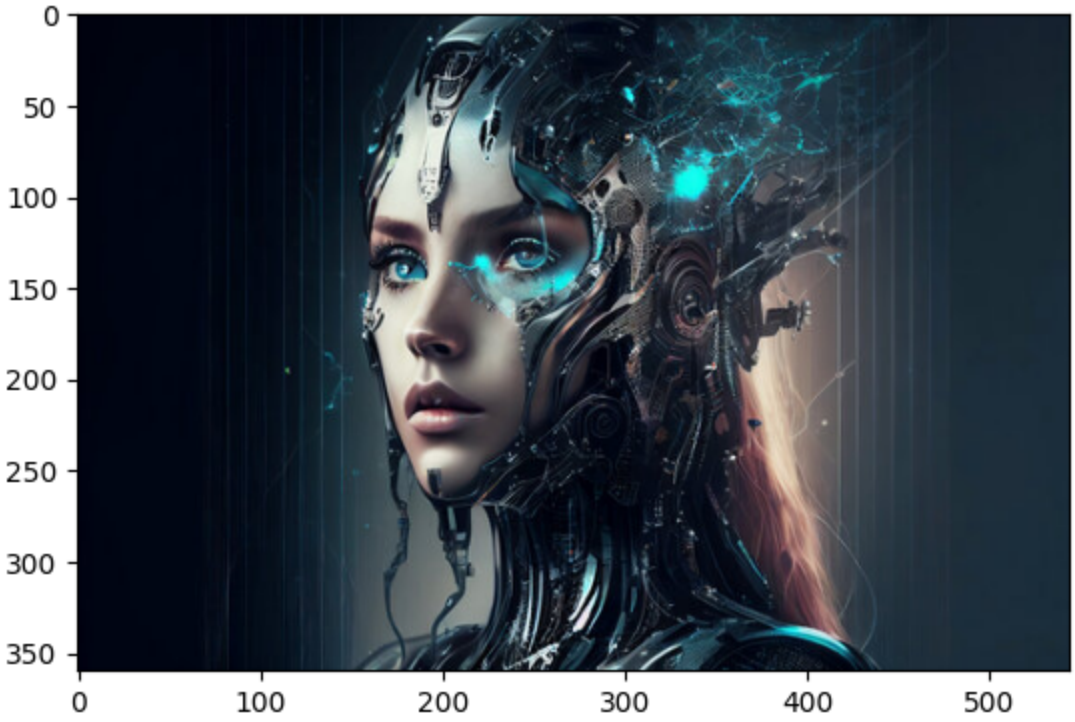
plt.imshow(Z)

The `plt.imshow()` function in `matplotlib.pyplot` has the following parameters:

1. **X**: This parameter is the data of the image.
2. **cmap**: This parameter is a colormap instance or registered colormap name.
3. **norm**: This parameter is the Normalize instance scales the data values to the canonical colormap range [0, 1] for mapping to colors
4. **vmin, vmax**: These parameter are optional in nature and they are colorbar range.
5. **alpha**: This parameter is a intensity of the color.
6. **aspect**: This parameter is used to controls the aspect ratio of the axes.
7. **interpolation**: This parameter is the interpolation method which used to display an image.
8. **interpolation_stage**: controls the stage at which interpolation is performed. The default value is "data"
9. **origin**: This parameter is used to place the [0, 0] index of the array in the upper left or lower left corner of the axes.
10. **resample**: This parameter is the method which is used for resembling.
11. **extent**: This parameter is the bounding box in data coordinates.
12. **filternorm**: This parameter is used for the antigrain image resize filter.
13. **filterrad**: This parameter is the filter radius for filters that have a radius parameter.
14. **url**: This parameter sets the url of the created AxesImage.

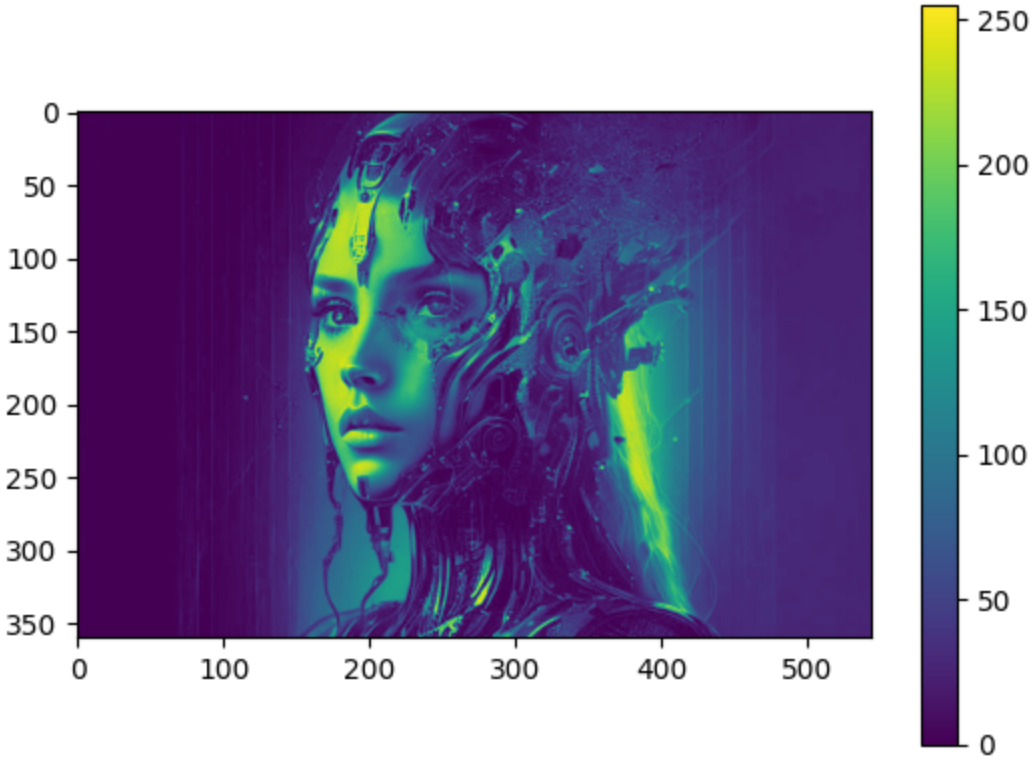
In [62]:
```
# Printing an image
robo = plt.imread('/content/drive/MyDrive/ML and DL DataSets/3.1_Robo_img.jpg') # imr
plt.imshow(robo) # to remove the axis, we can use: plt.axis('off')

print(type(robo)) # type of the image
print(robo.shape) # shape of the image
```

```
<class 'numpy.ndarray'>
(360, 544, 3)
```

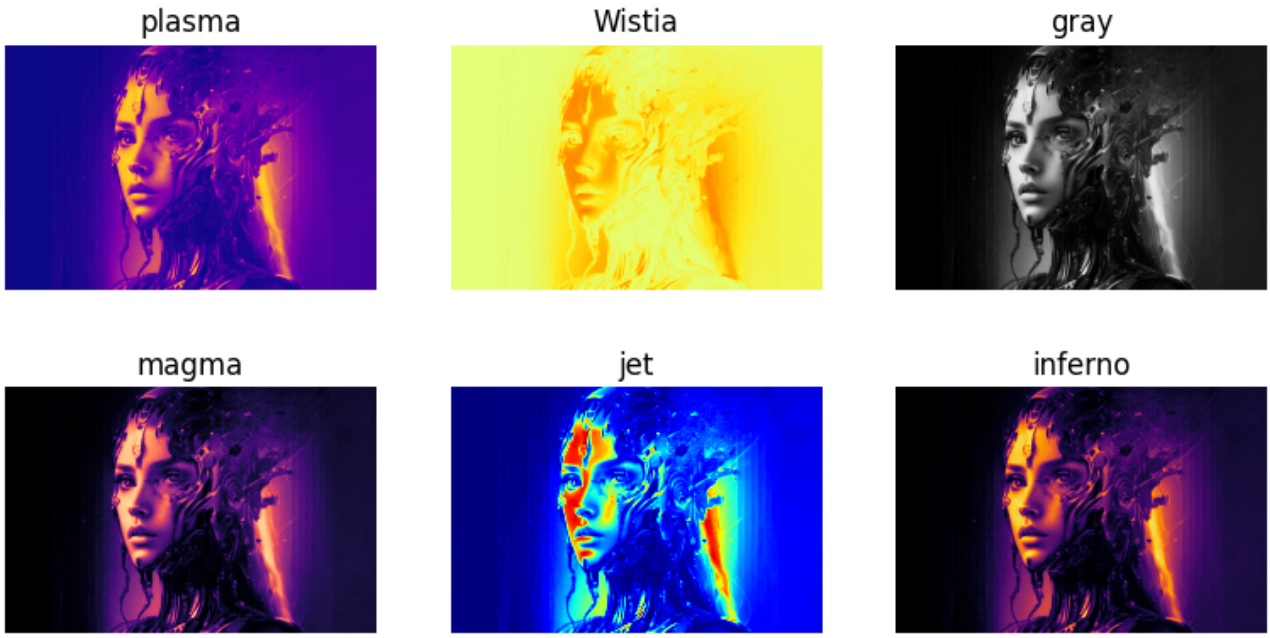


In [63]:
```
# Printing a cmap image
# In cmapscale, we can only print 2D image, so first we have to convert it into 2D
gray_robo = robo[:, :, 0]

plt.imshow(gray_robo)
plt.colorbar()
plt.rcParams['figure.figsize'] # Getting default image size
```

Out[63]: [6.4, 4.8]



In [64]:
```
# Showing image in different colors
color_robo = plt.imread('/content/drive/MyDrive/ML and DL DataSets/3.1_Robo_img.jpg')
robo_2D = color_robo[:, :, 0] # Converting into 2D array, to use it in cmap

fig, axes = plt.subplots(nrows = 2, ncols = 3, figsize = plt.figaspect(1/2))  # Figas
map = ['plasma', 'Wistia', 'gray', 'magma', 'jet', 'inferno']

for i, cmap in zip(axes.flat, map):
  i.imshow(robo_2D, cmap = cmap)
  i.set(title = cmap)
  i.axis('off')

plt.show()
```

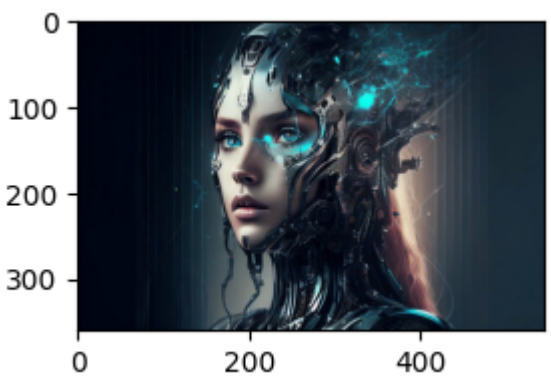


In [65]:
```
# Matplotlib imshow size
# Default size is plt.rcParams['figure.figsize'] = [6.4, 4.8] inches

''' 1. plt.figure(figsize = (8, 6)), 8 inches wide and 6 inches tall
    2. plt.figure(figsize = plt.figaspect(2)), twice as tall as it's wide '''
```

Out[65]:
```
" 1. plt.figure(figsize = (8, 6)), 8 inches wide and 6 inches tall\n    2. plt.figur
e(figsize = plt.figaspect(2)), twice as tall as it's wide "
```

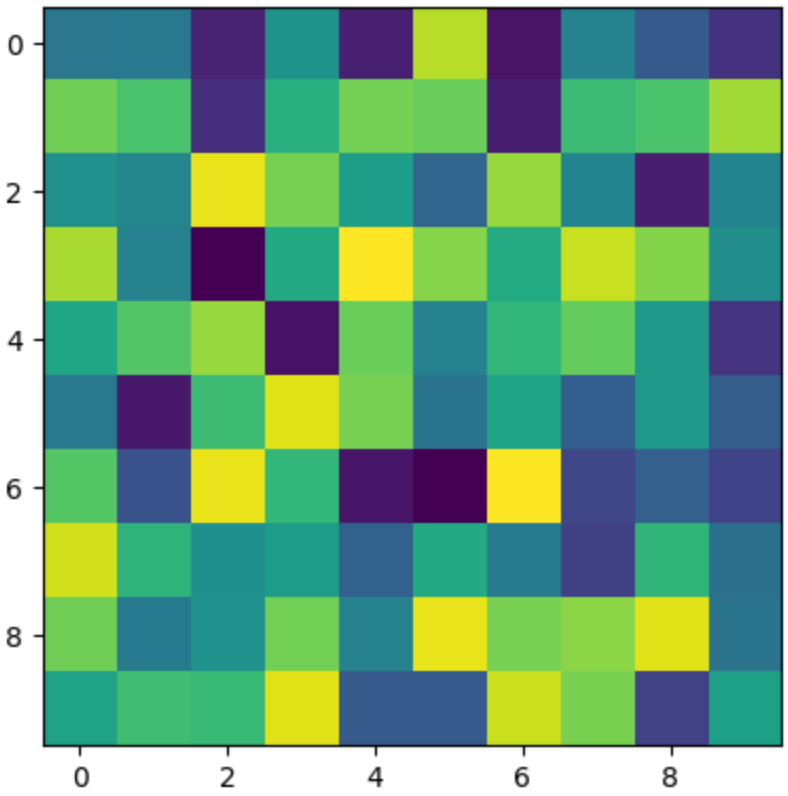
In [66]:
```
# Changing the size
plt.figure(figsize = (4, 2))

plt.imshow(robo) # to remove the axis, we can use: plt.axis('off')
```

Out[66]:
```
<matplotlib.image.AxesImage at 0x7931c0d24430>
```

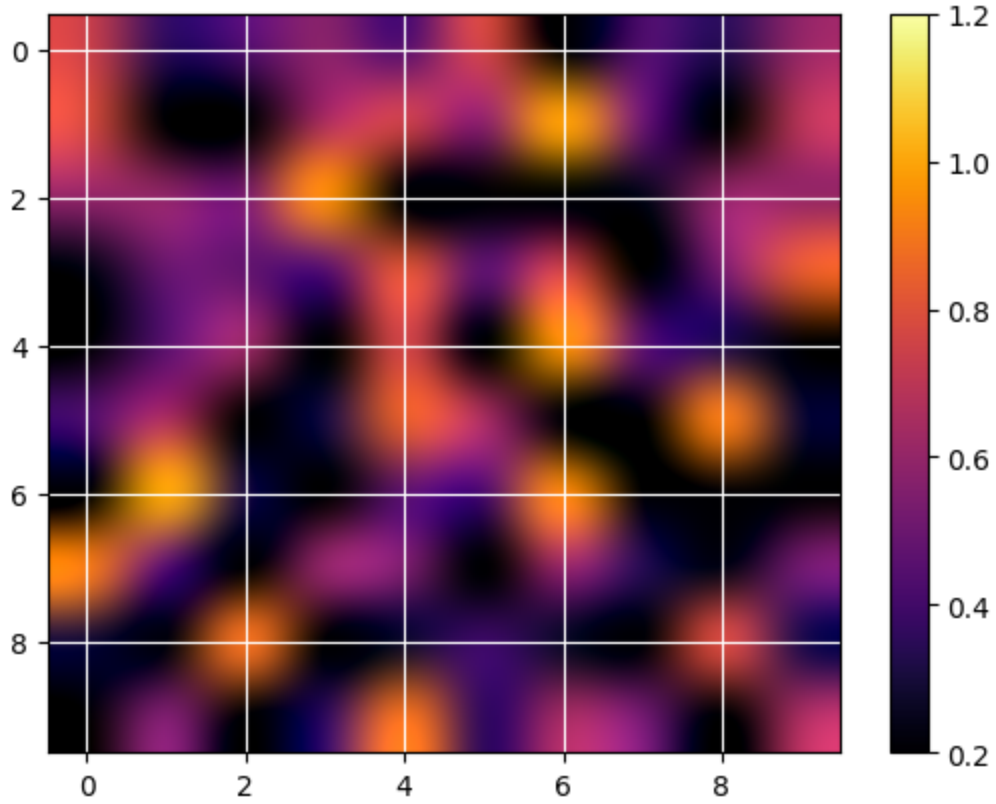


In [67]:
```
# Create a 2D array with random values
Z = np.random.random((10,10))

# Display the array
plt.imshow(Z)
```

Out[67]: <matplotlib.image.AxesImage at 0x7931c0cbfc40>



In [68]:
```
# Using the parameters of imshow
Z = np.random.random((10,10))

# Display the array
plt.imshow(Z,
           cmap = 'inferno',
           norm = 'linear', # "linear", "log", "symlog", "logit", etc are avail
           aspect = 'auto',
           interpolation = 'blackman', # Default: 'antialiased'. Supports many interp
           interpolation_stage = 'rgba', # Default: 'data'
           alpha = 1.0,
           vmin = 0.2,
           vmax = 1.2,
           origin = 'upper', # upper and lower. Default: upper
           extent = None,
           filternorm = True, # Default: True
           filterrad = 3.2, # Default 4.0
           resample = True) # Default: True
plt.colorbar()
plt.grid(color = 'white')
```



Create a pseudocolor plot of a 2-D array:

plt.pcolormesh([X, Y, Z]) / plt.pcolormesh(Z)

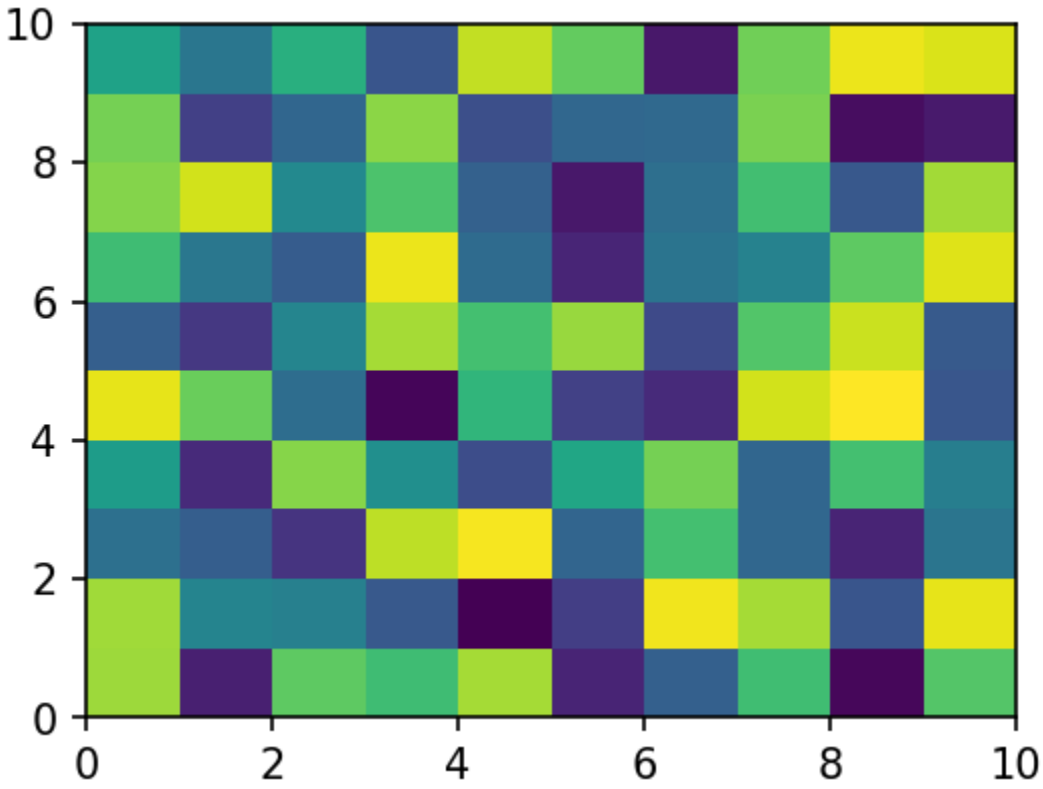
The `plt.pcolormesh()` function in `matplotlib.pyplot` has the following parameters:

1. **C**: This parameter contains the values in 2D array which are to be color-mapped.
2. **X, Y**: These parameter are the coordinates of the quadrilateral corners.
3. **cmap**: This parameter is a colormap instance or registered colormap name.
4. **norm**: This parameter is the Normalize instance scales the data values to the canonical colormap range [0, 1] for mapping to colors
5. **vmin, vmax**: These parameter are optional in nature and they are colorbar range.
6. **alpha**: This parameter is a intensity of the color.
7. **snap**: This parameter is used to snap the mesh to pixel boundaries.

8. **edgecolors**: This parameter is the color of the edges. {‘none’, None, ‘face’, color, color sequence}
9. **shading**: This parameter is the fill style. It can be flat or gouraud.

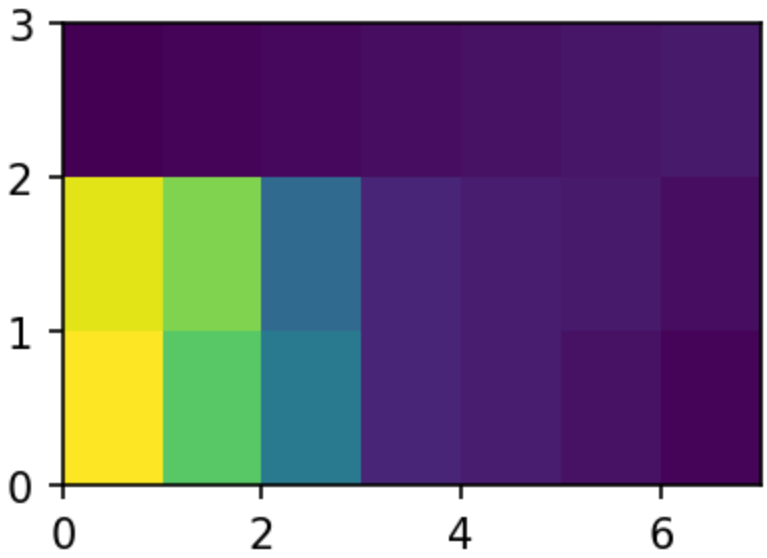
In [69]:
```
fig = plt.figure(figsize=(4, 3), dpi=150)
# Create a 2D array with random values
Z = np.random.random((10,10))

# Display the array
plt.pcolormesh(Z)
```

Out[69]:
```
<matplotlib.collections.QuadMesh at 0x7931c0f18070>
```



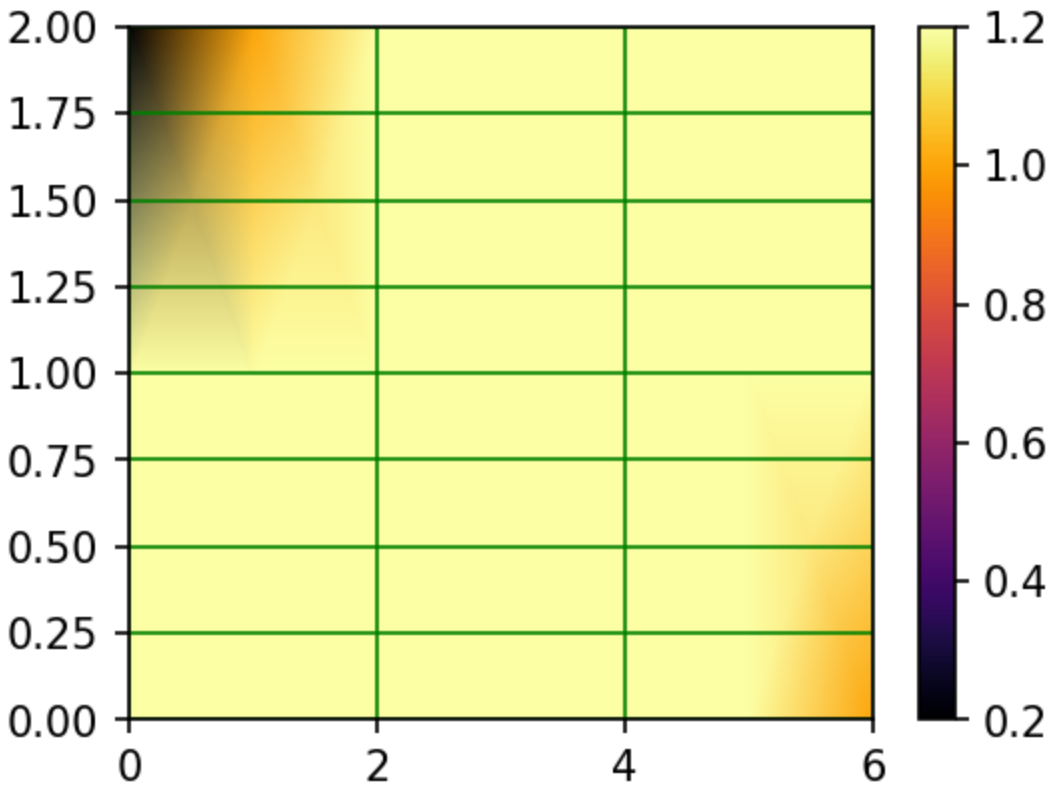
In [70]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Viewing male and female in Colormesh plot
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

plt.pcolormesh([x, y, num])
```

Out[70]: <matplotlib.collections.QuadMesh at 0x7931c0ef7e80>



In [71]:
```
fig = plt.figure(figsize=(4, 3), dpi=150)

# Using the parameters of Colormesh Plot
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

plt.pcolormesh([x, y, num],
            cmap = 'inferno',
            norm = 'linear', # "linear", "log", "symlog", "logit", etc are avail
            alpha = 1.0,
            vmin = 0.2,
            vmax = 1.2,
            snap = True, # True means the mesh elements will be snapped to pixel bound
            edgecolor = 'black',
            shading='gouraud') # Default: 'flat'

plt.grid(color = 'g')
plt.colorbar()
```

Out[71]:
```
<matplotlib.colorbar.Colorbar at 0x7931c16015a0>
```



Contour plot:

plt.contour([X, Y, Z]) / plt.contour(Z)

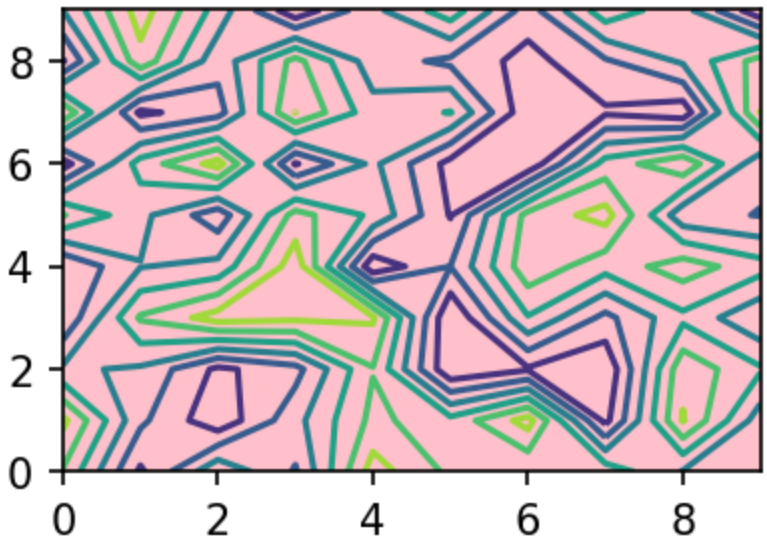
The `plt.contour()` function in `matplotlib.pyplot` has the following parameters:

1. **X, Y**: These are array-like and optional. They represent the coordinates of the values in Z. X and Y must both be 2D with the same shape as Z (e.g., created via numpy.meshgrid).
2. **Z**: This is an array-like parameter that represents the height values over which the contour is drawn. Color-mapping is controlled by cmap, norm, vmin, and vmax.
3. **levels**: This is an optional parameter that determines the number and positions of the contour lines/regions.
4. **corner_mask**: This is a boolean parameter that enables/disables corner masking, which only has an effect if Z is a masked array.

5. **colors**: This is an optional parameter that represents the colors of the levels, i.e., the lines for contour.
6. **alpha**: This is an optional parameter that represents the alpha blending value, between 0 (transparent) and 1 (opaque).
7. **vmin, vmax**: These parameter are optional in nature and they are colorbar range.
8. **cmap**: This is an optional parameter that represents the Colormap instance or registered colormap name used to map scalar data to colors.
9. **linewidths**: This parameter is used to specify the line width of the contour lines.
10. **linestyles**: This parameter is used to specify the style of the contour lines.
11. **norm**: This is an optional parameter that represents the normalization method used to scale scalar data to the [0, 1] range before mapping to colors using cmap.

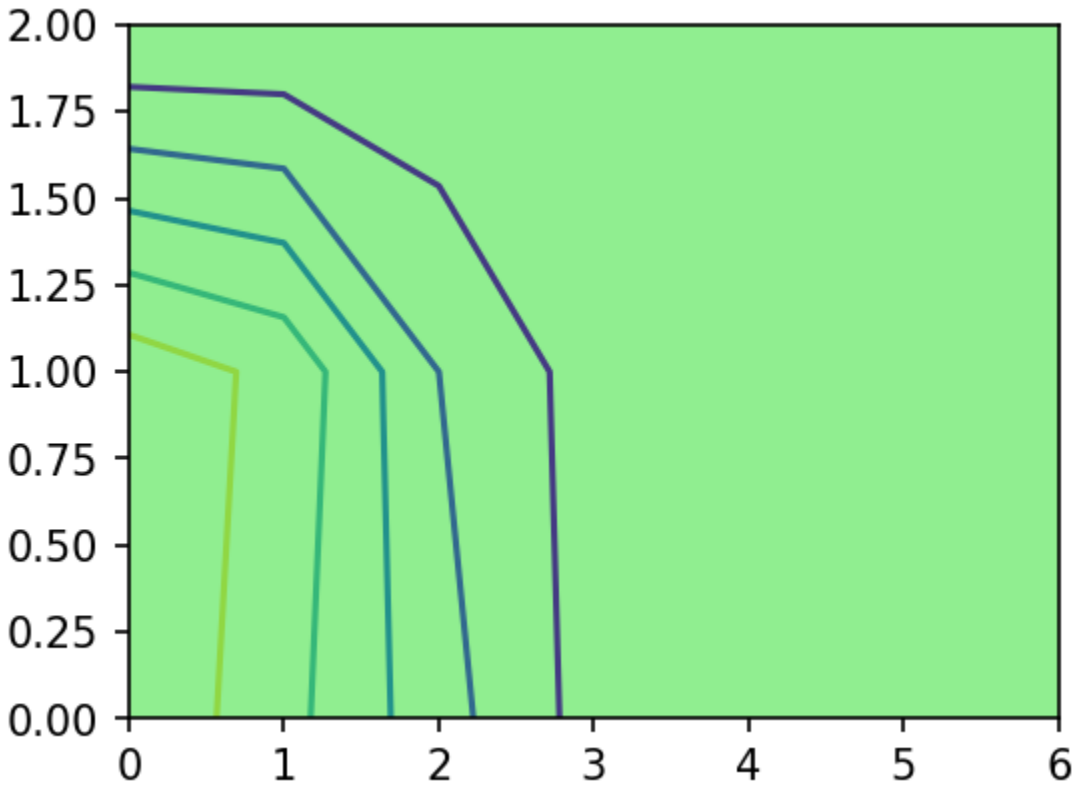
In [72]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

# Get the axes object.
ax = plt.gca()
# Call the `set_facecolor()` method on the axes object and pass in the desired backgr
ax.set_facecolor('pink') # Change the background color

# Create a 2D array with random values
Z = np.random.random((10,10))

# Display the array
plt.contour(Z)
```

Out[72]:
```
<matplotlib.contour.QuadContourSet at 0x7931c0ba7fa0>
```



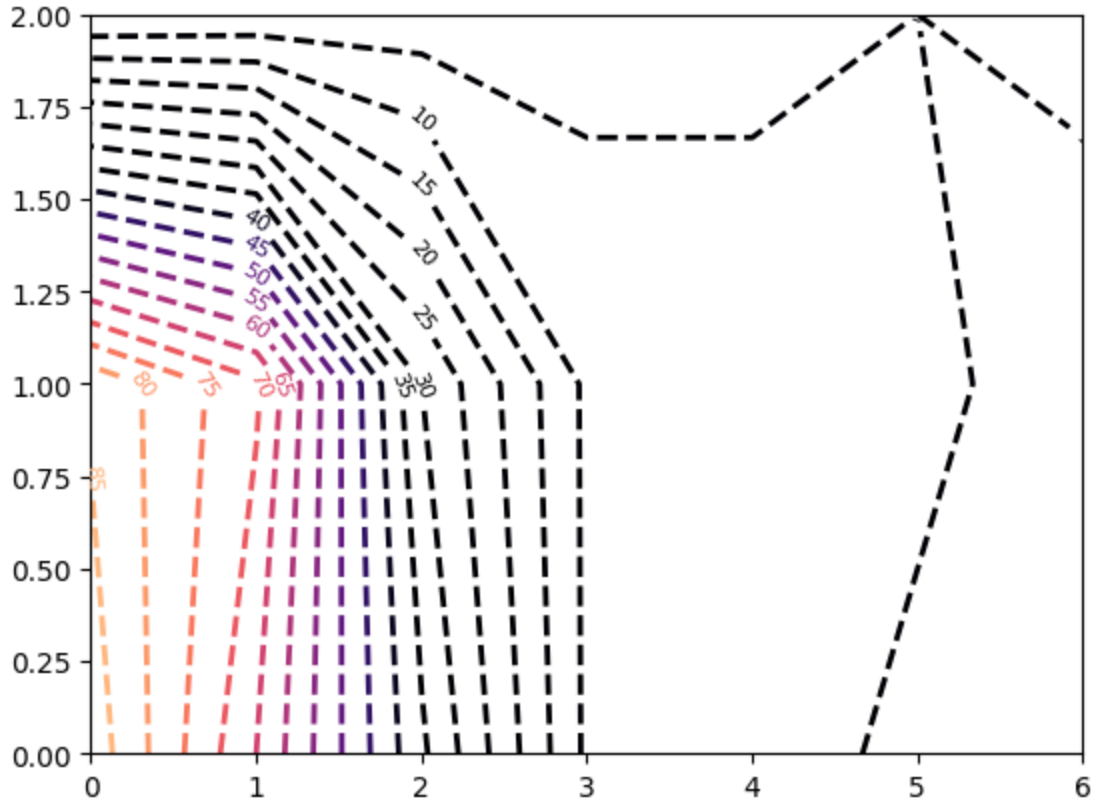
In [73]:
```
fig = plt.figure(figsize=(4, 3), dpi=150)

# Viewing male and female in Contour plot
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

ax = plt.gca()
ax.set_facecolor("lightgreen")

plt.contour([x, y, num])
```

Out[73]: <matplotlib.contour.QuadContourSet at 0x7931c0910d30>



In [74]:
```
# Using the parameters of contour
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

con = plt.contour([x, y, num],
                  levels = 20,
                  corner_mask= True, # Default: True, Enable/disable corner masking,
                  alpha = 1,
                  vmin = 38,
                  vmax = 98,
                  cmap = 'magma',
                  linewidths = 2, # Default: 1.5
                  linestyles = 'dashed', # 'solid', 'dashed', 'dashdot', 'dotted'
                  norm = 'log') # "linear", "log", "symlog", "logit", etc

plt.clabel(con, inline= True, fontsize= 8) # Adding label in contour plot
```

Out[74]:
```
<a list of 18 text.Text objects>
```



Filled contour plot:

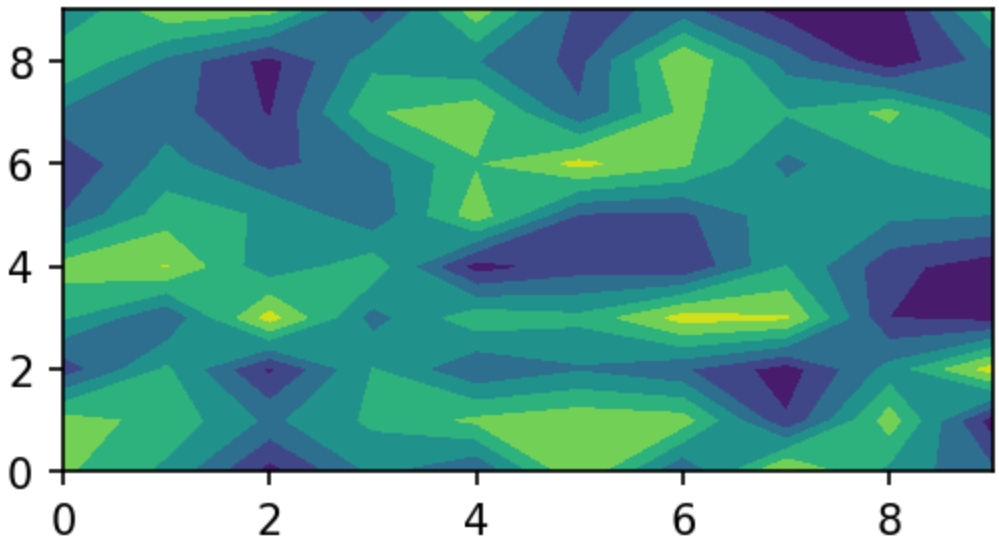
plt.contourf([X, Y, Z]) / plt.contourf(Z)

The `plt.contourf()` function in `matplotlib.pyplot` has the following parameters:

Parameter in `plt.contour()` and `plt.contourf()` are almost same, there is no major differance. Only, cntourf will help to fill the area.

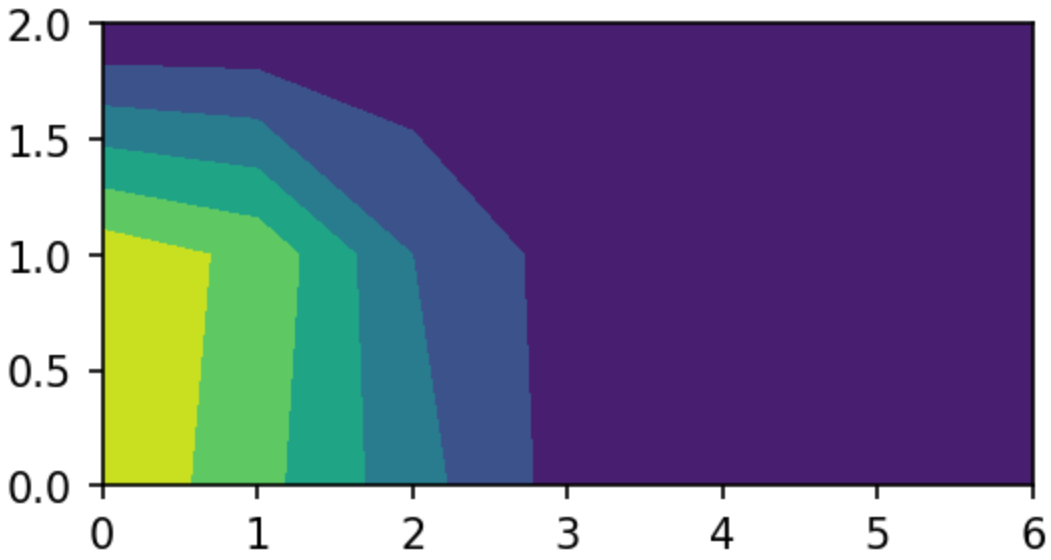
In [75]:
```
fig = plt.figure(figsize=(4, 2), dpi=150)

# Create a 2D array with random values
Z = np.random.random((10,10))

# Display the array
plt.contourf(Z)
```

Out[75]: <matplotlib.contour.QuadContourSet at 0x7931c08ae200>



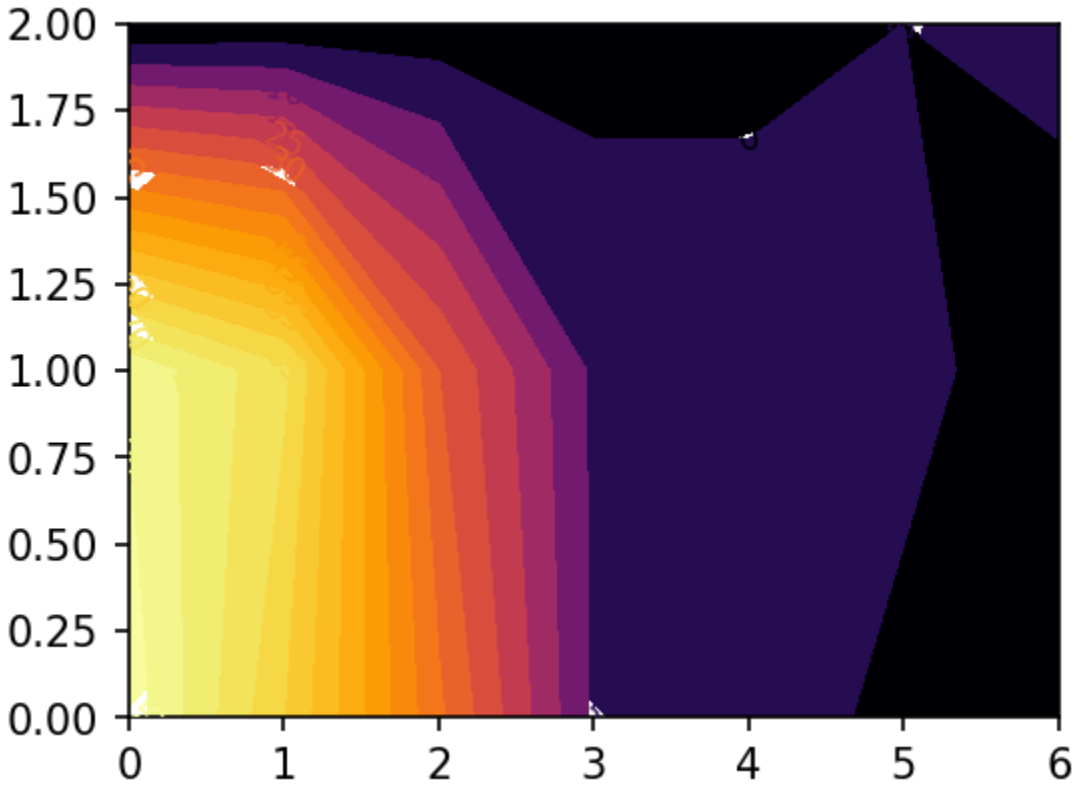
In [76]:
```
fig = plt.figure(figsize=(4, 2), dpi=150)

# Viewing male and female in Contour plot
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

plt.contourf([x, y, num])
```

Out[76]: <matplotlib.contour.QuadContourSet at 0x7931c0726380>



In [77]:
```
fig = plt.figure(figsize=(4, 3), dpi=150)

#Using the parameters of contour
x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

con = plt.contourf([x, y, num],
                   levels = 20,
                   corner_mask= True, # Default: True, Enable/disable corner masking,
                   alpha = 1,
                   cmap = 'inferno',
                   norm = 'log') # "linear", "log", "symlog", "logit", etc

plt.clabel(con, inline= True, fontsize= 8) # Adding label in contour plot
```

Out[77]: <a list of 19 text.Text objects>

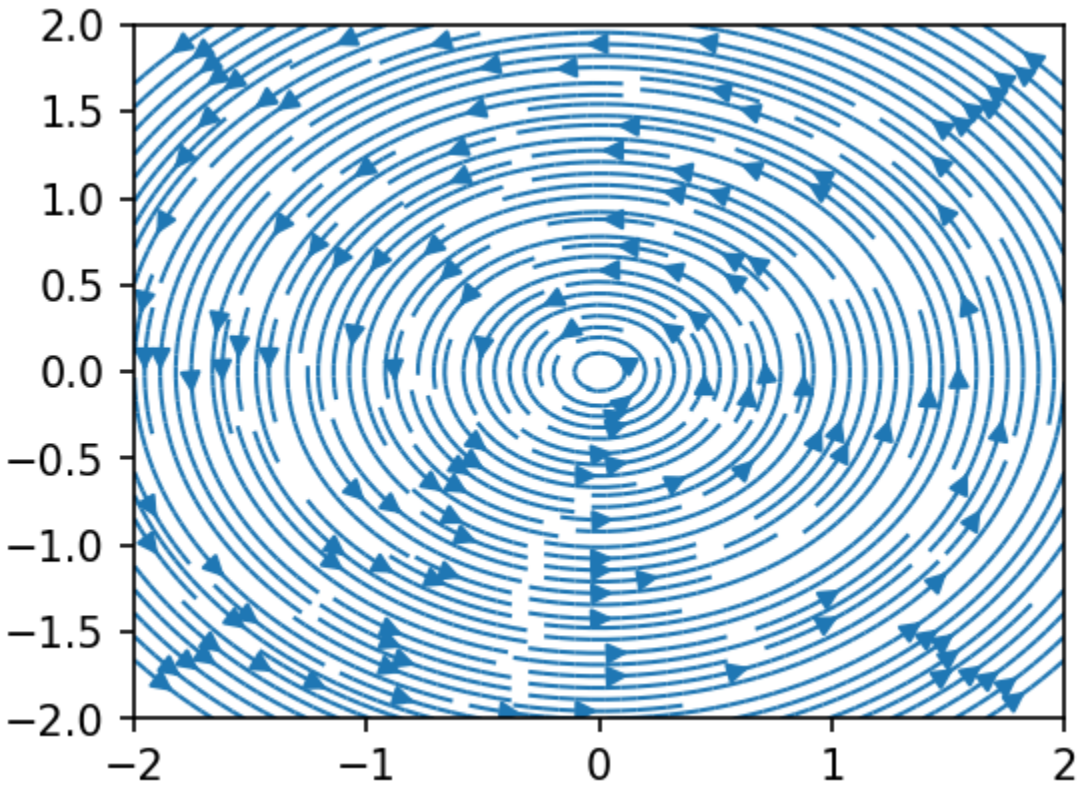


Streamplot:

plt.streamplot(X, Y, U, V)

In [78]:
```
fig = plt.figure(figsize=(4, 3), dpi=150)

# Create a grid of points
x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)

# Create a simple vector field: a vortex centered at the origin
U = -Y
V = X

# Create a stream plot
plt.streamplot(X, Y, U, V, density=2, linewidth=1, arrowsize=1, cmap='viridis')
```

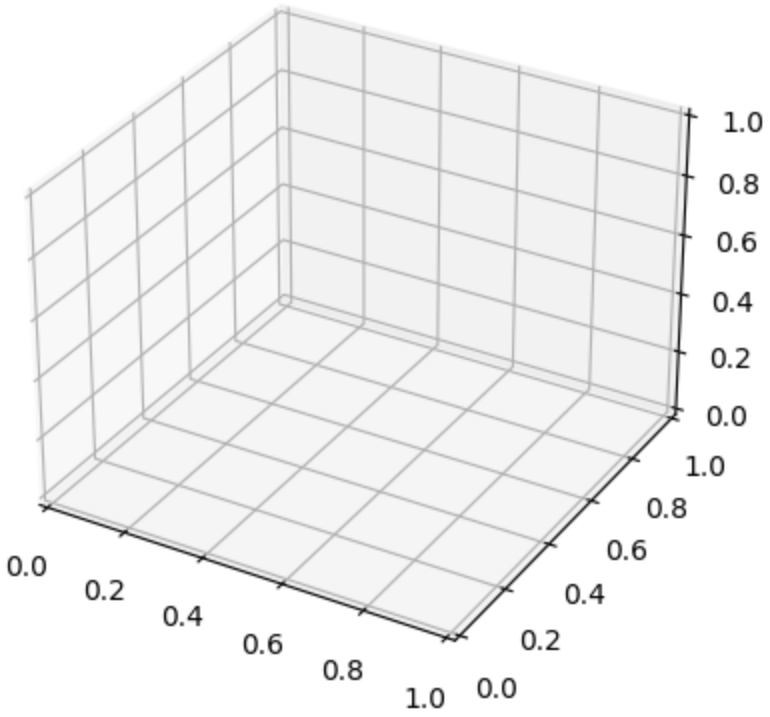
Out[78]:
```
<matplotlib.streamplot.StreamplotSet at 0x7931c040e530>
```



3D and volumetric data

In [79]:
```
ax = plt.axes(projection = '3d')

print("The dtype is: ",type(ax))
```

```
The dtype is:  <class 'mpl_toolkits.mplot3d.axes3d.Axes3D'>
```



Scatter 3D Plot:

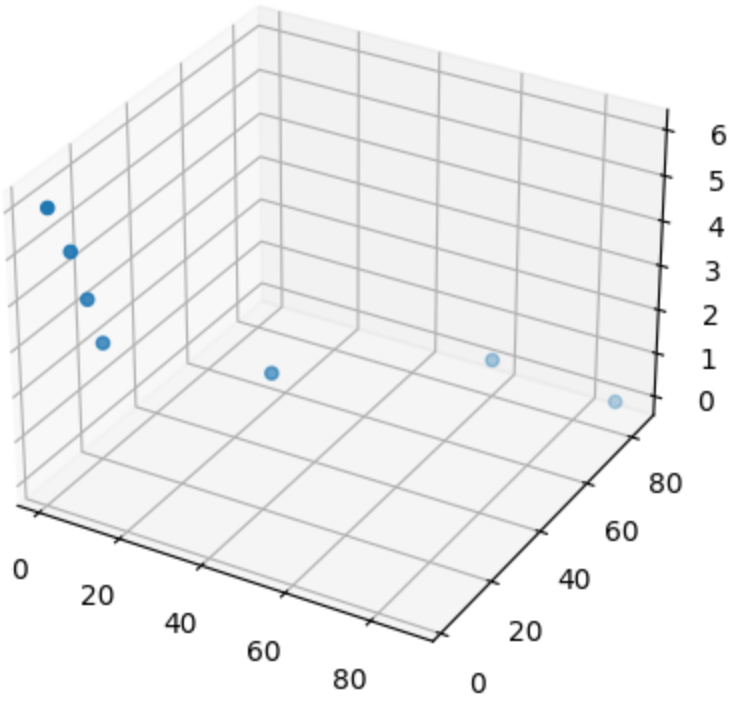
ax.scatter(x, y, z), Here only numerical value will support. X, Y and Z can be, list, tuple or array

The `ax.scatter()` function in matplotlib.pyplot has the following parameters:

1. **xs, ys, zs**: These are arrays or sequences of n numbers representing the data positions in the x, y, and z dimensions respectively.
2. **s**: This is the marker size in points**2. It can be a single value for all markers or a sequence of values for each marker individually.
3. **c**: This is the marker color. It can be a single color format string, a sequence of color specifications of length n, or a sequence of n numbers to be mapped to colors using the cmap and norm parameters.
4. **depthshade**: This is a boolean value that determines whether or not to shade the scatter markers to give the appearance of depth. Default is True.

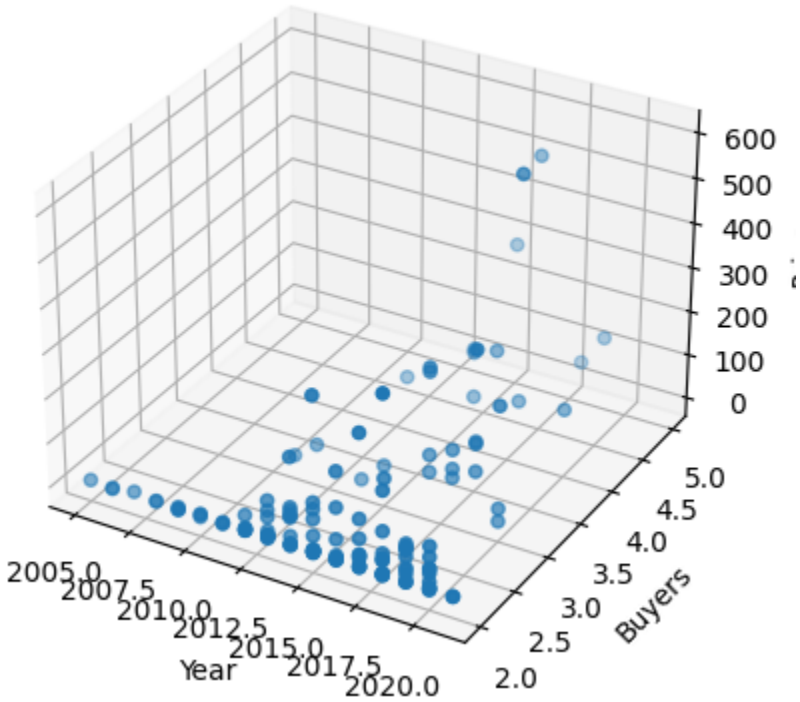
In [80]:
```
ax_eda = plt.axes(projection = '3d') # We have to specify this in every cell. Otherwi

x = file.MALE_MEMBERS.value_counts().values
y = file.FEMALE_MEMBERS.value_counts().values
num = [0, 1, 2, 3, 4, 5, 6]

ax_eda.scatter (x, y, num)
```

Out[80]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7931c0e554e0>

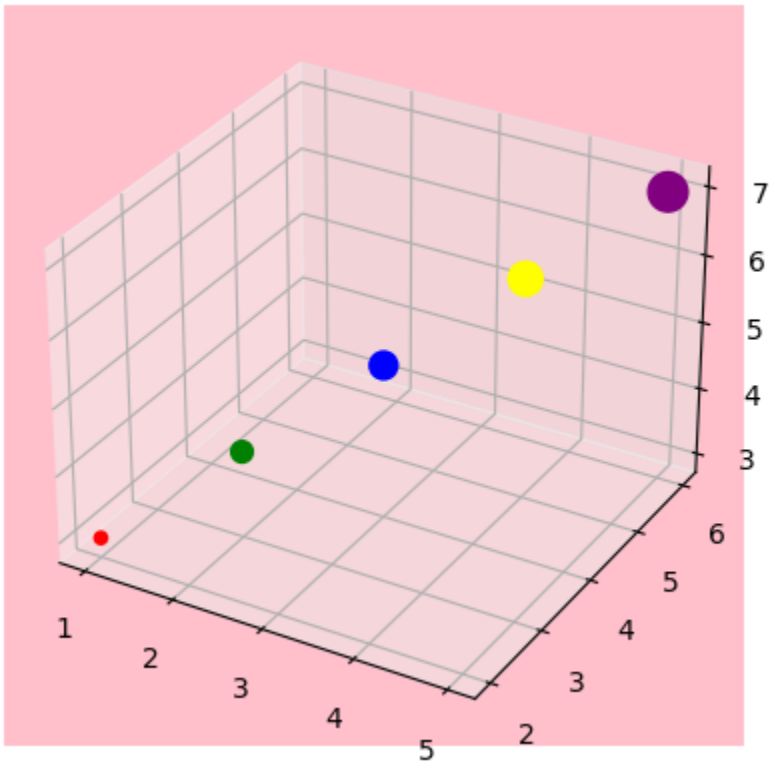


In [81]:
```
#3d plotting of Year and Buyers wrt Price
fig = plt.figure()
ax_car = fig.add_subplot(111 , projection = '3d') # 111 argument is a 3-digit integer
                                                  # the second the number of columns,

x = df_car[['Year']]
y = df_car[['Buyers']]
z = df_car[['Price (Lakhs)']]

# Giving the label
ax_car.set_xlabel ('Year')
ax_car.set_ylabel ('Buyers')
ax_car.set_zlabel ('Price')

# Scatter plotting of graph
ax_car.scatter (x,y,z)
```

Out[81]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7931c026e1a0>



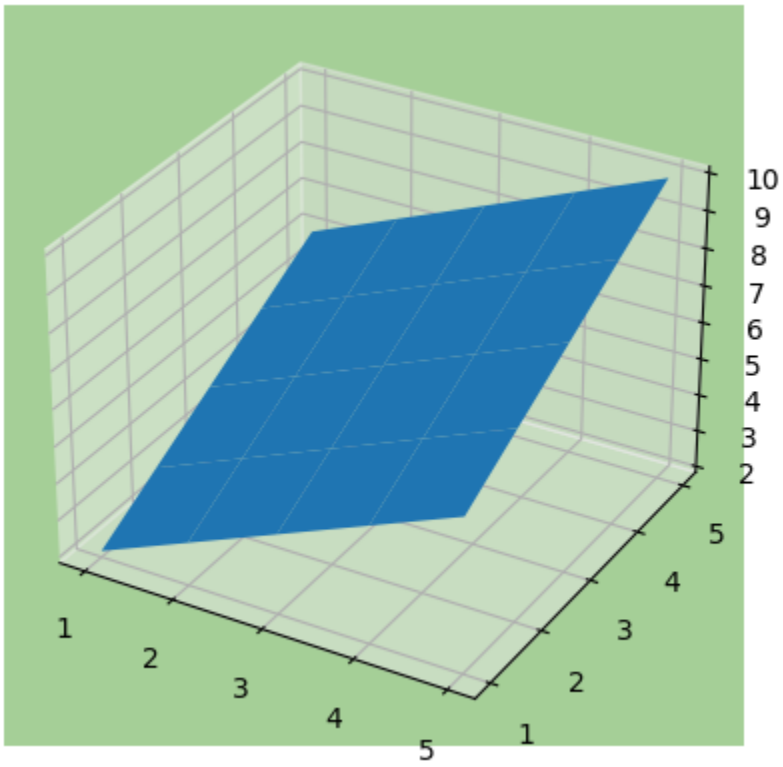
In [82]:
```
# Using the parameters of 3D scatter plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = [1, 2, 3, 4, 5]
y = [2, 3, 4, 5, 6]
z = [3, 4, 5, 6, 7]

ax.scatter(x, y, z,
           s = [20, 60, 100, 150, 200], # Size of the circle
           c = ['red', 'green', 'blue', 'yellow', 'purple'], # Color of the circle
           depthshade = False) # Default:True, it will control the opacity
ax.set_facecolor("pink")
```



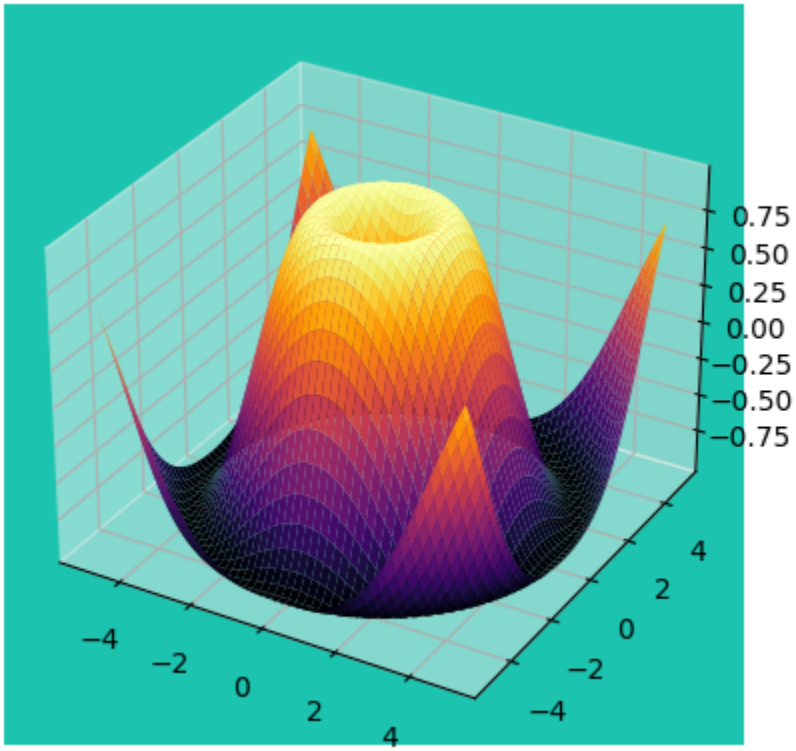
Plot Surface:

plot_surface(X, Y, Z)

In [83]:
```
# Create a figure
fig = plt.figure()
# Create a 3D plot
ax = fig.add_subplot(111, projection='3d')

x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 4, 5]
x, y = np.meshgrid(x, y) # We have to use this otherwise, it will throw error: 'list'
z = x+y

# Plot the data
ax.plot_surface(x, y, z)
ax.set_facecolor("#a5cf97") # For bckground color
```

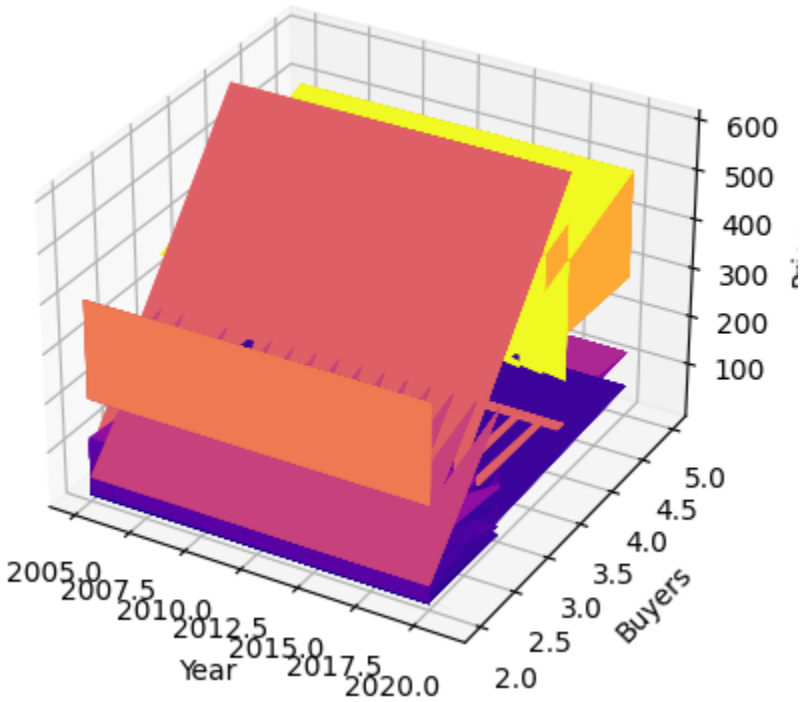


In [84]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x, y) # 'list' object has no attribute 'ndim' typically occurs whe
                         # instead. The .ndim attribute is used to get the number of
z = np.sin(np.sqrt(x**2 + y**2))

ax.plot_surface(x, y, z, cmap='inferno')
ax.set_facecolor("#1CC4AF")
```



In [85]:
```
#3d plotting of Year and Buyers wrt Price
fig = plt.figure()
ax_car = fig.add_subplot(111 , projection = '3d') # 111 argument is a 3-digit integer
                                                  # the second the number of columns,

x = df_car[['Year']]
y = df_car[['Buyers']]
x, y = np.meshgrid(x, y) # meshgrid function is used to create a rectangular grid out
z = df_car[['Price (Lakhs)']]

# Giving the label
ax_car.set_xlabel ('Year')
ax_car.set_ylabel ('Buyers')
ax_car.set_zlabel ('Price')

ax_car.plot_surface (x,y,z, cmap='plasma')
```

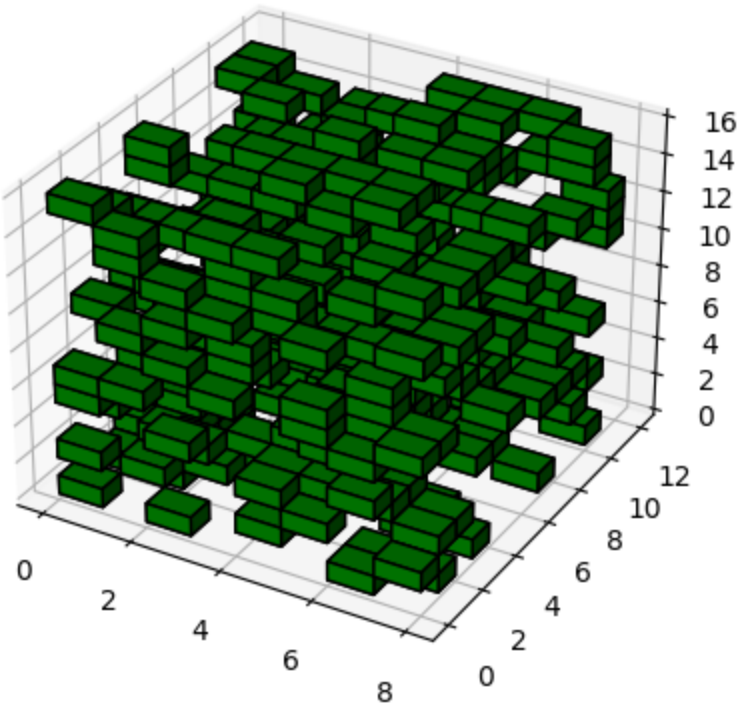
Out[85]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7931bfd315a0>



Voxels 3D:

voxels([x, y, z], field)

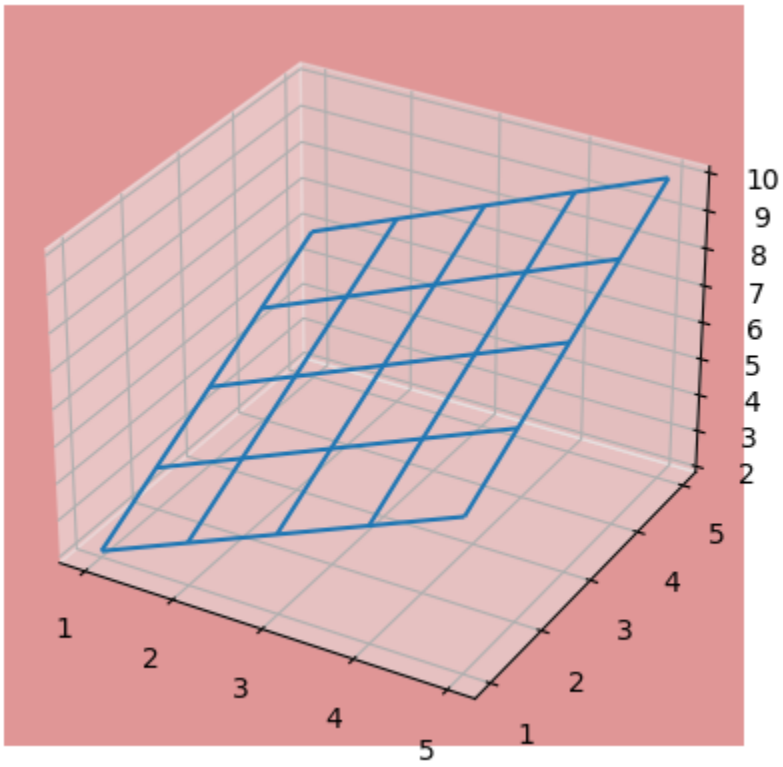
In [86]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# Random data points between 0 and 1
data = np.random.choice([0, 1], size=(8, 12, 16), p=[0.80, 0.20]) # Size: Output shap
                                                                  # p: The probabilit
# Plot the voxels
ax.voxels(data, edgecolor="k", facecolors='green')
plt.show()
```



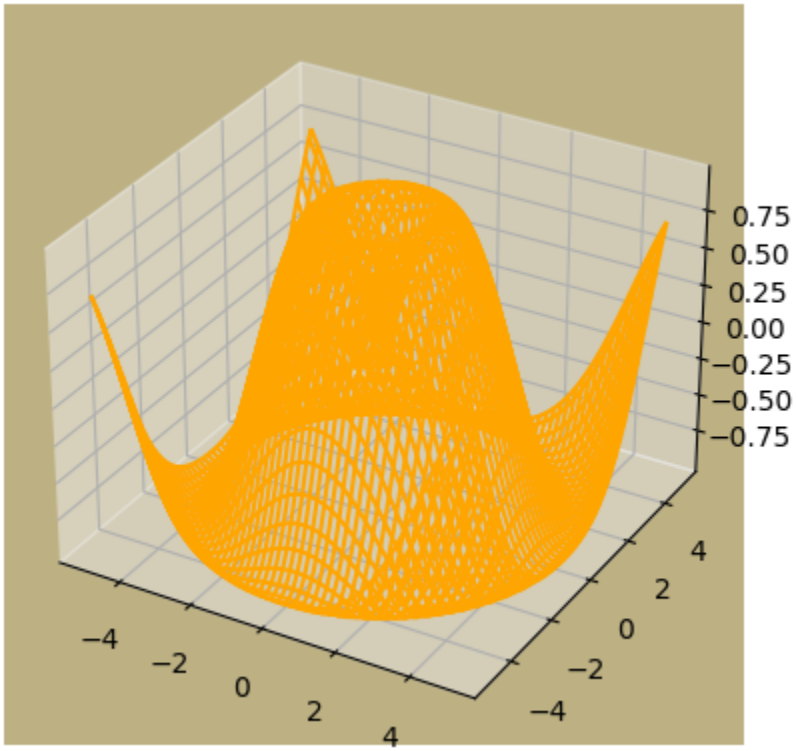
Plot Wireframe:

plot_wireframe(X, Y, Z)

In [87]:
```
# Create a figure
fig = plt.figure()
# Create a 3D plot
ax = fig.add_subplot(111, projection='3d')

x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 4, 5]
x, y = np.meshgrid(x, y) # We have to use this otherwise, it will throw error: 'list'
z = x+y

# Plot the data
ax.plot_wireframe(x, y, z)
ax.set_facecolor("#e09696")
```

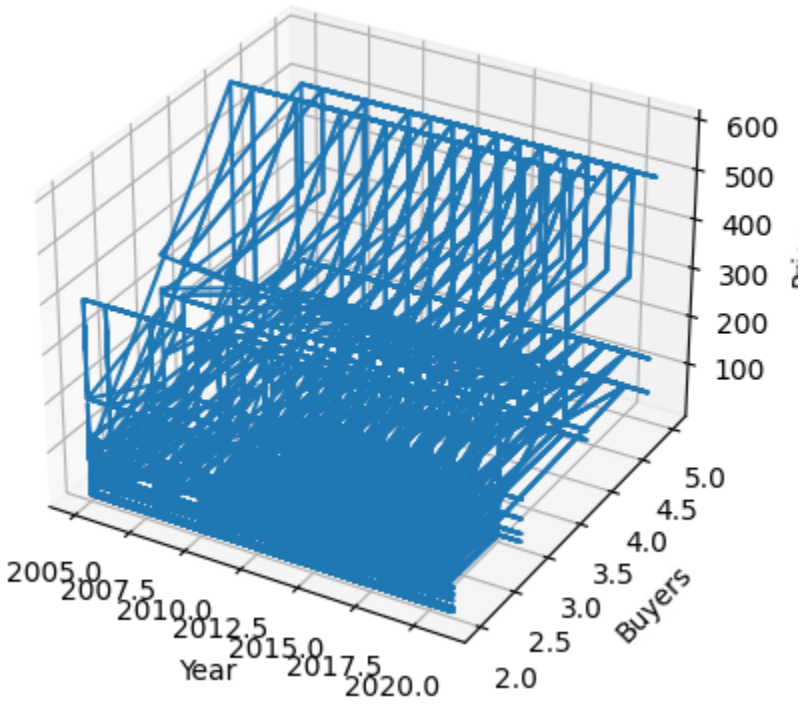


In [88]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x, y)
z = np.sin(np.sqrt(x**2 + y**2))

ax.plot_wireframe(x, y, z, color='orange')
ax.set_facecolor("#bdb184")
```



In [89]:
```
#3d plotting of Year and Buyers wrt Price
fig = plt.figure()
ax_car = fig.add_subplot(111 , projection = '3d')

x = df_car[['Year']]
y = df_car[['Buyers']]
x, y = np.meshgrid(x, y)
z = df_car[['Price (Lakhs)']]

# Giving the label
ax_car.set_xlabel ('Year')
ax_car.set_ylabel ('Buyers')
ax_car.set_zlabel ('Price')

ax_car.plot_wireframe(x,y,z)
```

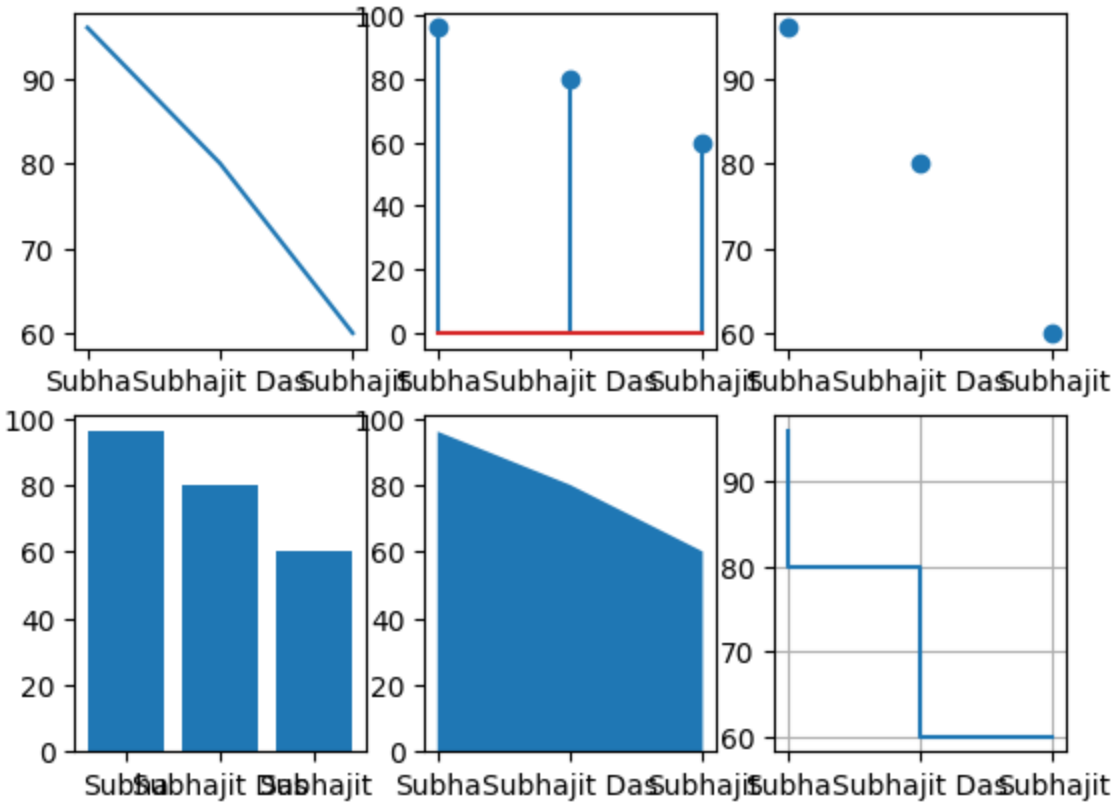
Out[89]: <mpl_toolkits.mplot3d.art3d.Line3DCollection at 0x7931c004b460>



Subplot

plt.subplot(nrows, ncols, index)

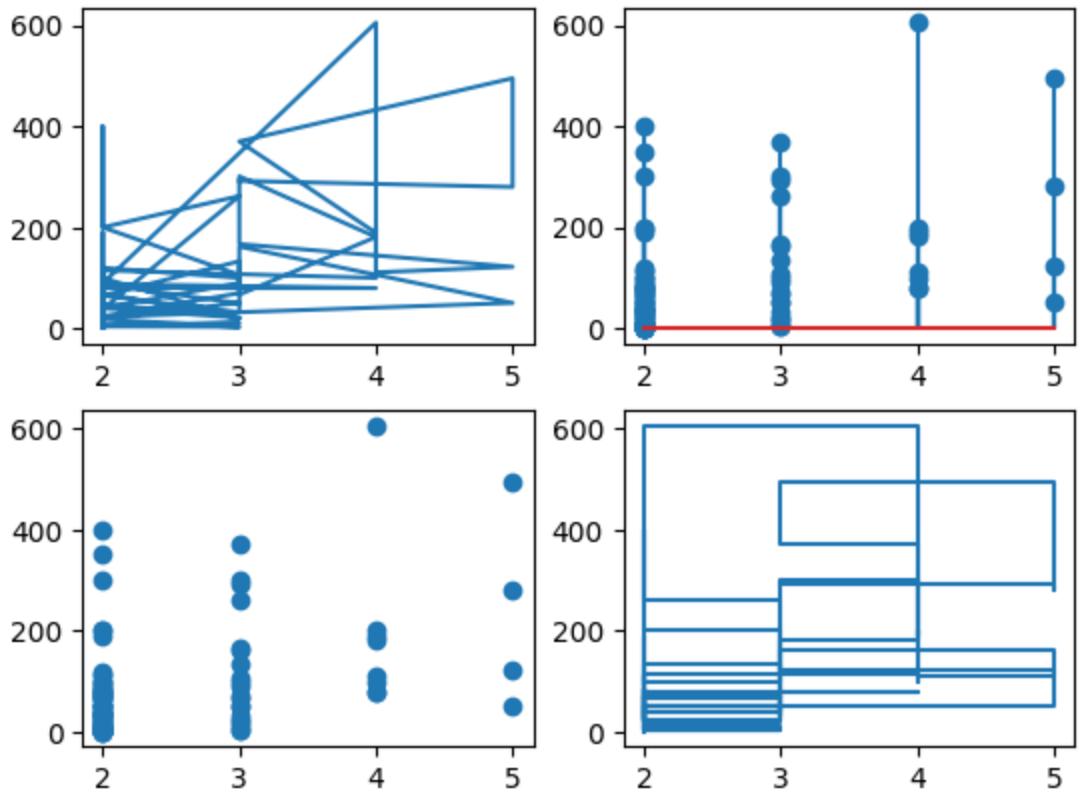
In [90]:
```
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

# Line Plot in subplot
plt.subplot(2, 3, 1)
plt.plot(name, marks)

# Stem Plot in subplot
plt.subplot(2, 3, 2)
plt.stem(name, marks)

# Scatter Plot in subplot
plt.subplot(2, 3, 3)
plt.scatter(name, marks)

# Bar Plot in subplot
plt.subplot(2, 3, 4)
plt.bar(name, marks)

# Stack Plot in subplot
plt.subplot(2, 3, 5)
plt.stackplot(name, marks)

# Step Plot in subplot
plt.subplot(2, 3, 6)
plt.step(name, marks)

plt.grid()
```



In [91]:
```
# Price of a car wrt no.of buyers
x = df_car[['Buyers']]
y = df_car[['Price (Lakhs)']]

# Line Plot in subplot
plt.subplot(2, 2, 1)
plt.plot(x, y)

# Stem Plot in subplot
plt.subplot(2, 2, 2)
plt.stem(x, y)

# Scatter Plot in subplot
plt.subplot(2, 2, 3)
plt.scatter(x, y)

# Step Plot in subplot
plt.subplot(2, 2, 4)
plt.step(x, y)
```

```
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning:
In a future version, DataFrame.min(axis=None) will return a scalar min over the enti
re DataFrame. To retain the old behavior, use 'frame.min(axis=0)' or just 'frame.min
()'
  return reduction(axis=axis, out=out, **passkwargs)
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning:
In a future version, DataFrame.max(axis=None) will return a scalar max over the enti
re DataFrame. To retain the old behavior, use 'frame.max(axis=0)' or just 'frame.max
()'
  return reduction(axis=axis, out=out, **passkwargs)
```

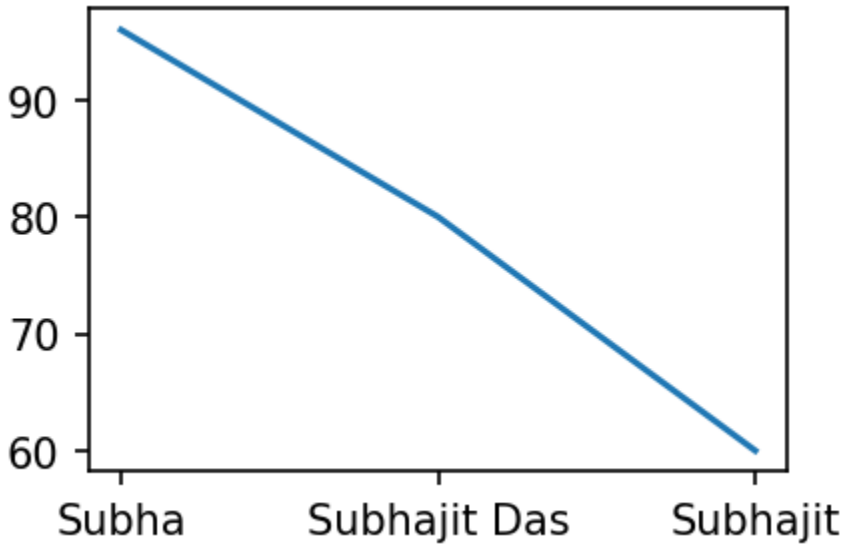
Out[91]: [<matplotlib.lines.Line2D at 0x7931bf09a290>]



Graph Saving

plt.savefig(file_name)

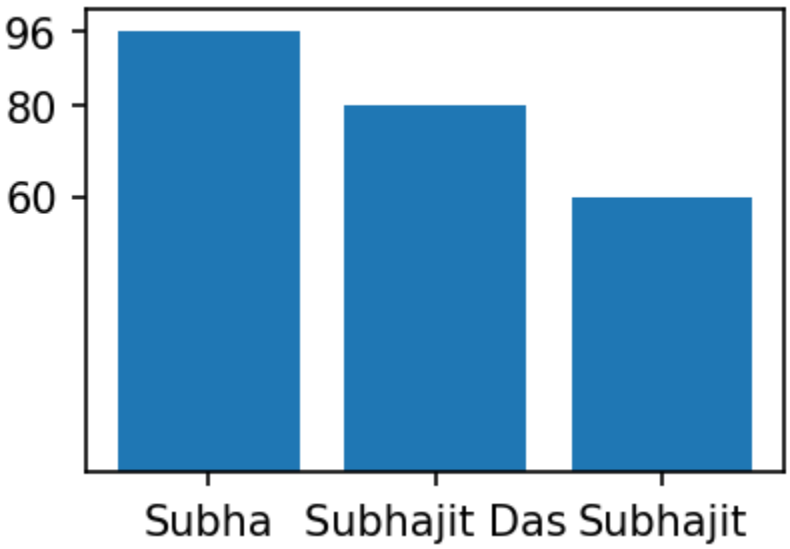
In [92]:
```
fig = plt.figure(figsize=(3,2), dpi=150)

# Can save 2d and 3d plot
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

plt.plot(name, marks)

plt.savefig("Sample.pdf", # It will save in pdf format, by default it's png
            dpi = 2000, # It will increase the resolution
            facecolor = 'g', # It will set a padding in the chart
            transparent = True,
            bbox_inches = 'tight') # It will reduce the padding size
```



Axis in Matplotlib

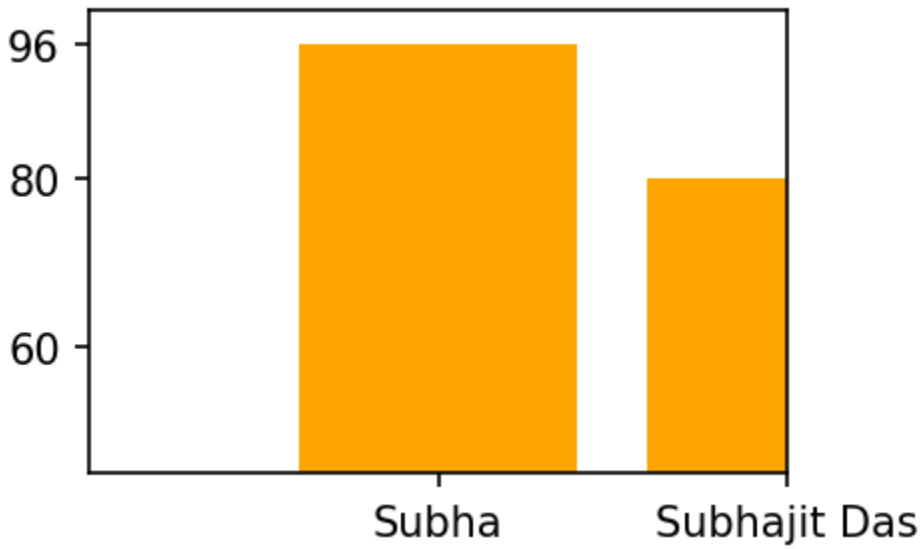
In [93]:
```
fig = plt.figure(figsize=(3,2), dpi=150)

# Adding labels in plot
name = (1, 2, 3)
marks = [96, 80, 60]

plt.xticks(name, labels = ['Subha', 'Subhajit Das', 'Subhajit'])
plt.yticks(marks)

plt.bar(name, marks)
```

Out[93]: <BarContainer object of 3 artists>



In [94]:
```
fig = plt.figure(figsize=(3,2), dpi=150)

# Specify limit in plot
name = (1, 2, 3)
marks = [96, 80, 60]

plt.xticks(name, labels = ['Subha', 'Subhajit Das', 'Subhajit'])
plt.yticks(marks)

plt.xlim(0,2)
plt.ylim(45,100)

plt.bar(name, marks, color = 'orange')
```

Out[94]:
```
<BarContainer object of 3 artists>
```



Text in Matplotlib

In [95]:
```
fig = plt.figure(figsize=(3, 2), dpi=150)

name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

plt.title("Student Marks", fontsize = 12)
plt.xlabel("Names", fontsize = 12)
plt.ylabel("Marks", fontsize = 12)

plt.text(0,88,"Calculas",   # row position, column position, text
         fontsize = 8,
         style = 'italic',
         bbox = {'facecolor' : 'green'})

plt.annotate("Deep Learning", xy=(1,76), # text, arrow position
             xytext = (0.5, 60), # text position
             arrowprops = dict(facecolor = 'black', shrink = 0.1)) # arrow color, arr

plt.scatter(name, marks, label = 'Marks')

plt.legend(loc = 1,
           facecolor = 'orange',
           edgecolor = 'red',
           framealpha = 0.8,
           shadow = True)
```

Out[95]: <matplotlib.legend.Legend at 0x7931be390b80>