# Matplolib

official website

- We can understand things in better way
- to understand the information in easy manner

**3file**

**1 -- 100 records representing 100 student details 2 -- 2 persons 3 -- multiple persons salary increment**

**how many ways to store our data?**

- files, databases -- txt,csv, excel,html,table,tsv ... cloud, sql, no sql, mongobd.. strings
- photos/videos -- pixel
- Graphical representation -- 2D, 1D -- Very easy

In [1]:

```
import matplotlib.pyplot as plt  # scripting layer
```

**pypplot, styles, mpl_toolkits .. etc**

In [28]:

```
import matplotlib
matplotlib.__version__
```

Out[28]:

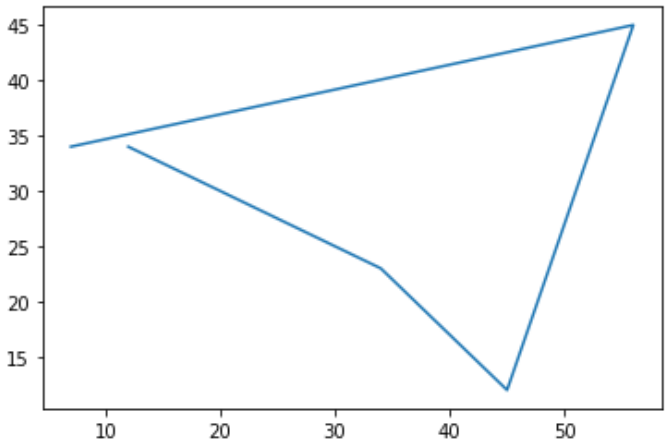
```
'3.2.2'
```

# 1. Line Plot

- Linear plot -- 2D data
- plt.plot()
- x values, y values
- application : Stock markets, weather monitoring systems.. which is used at mant points, which is changed with respective time

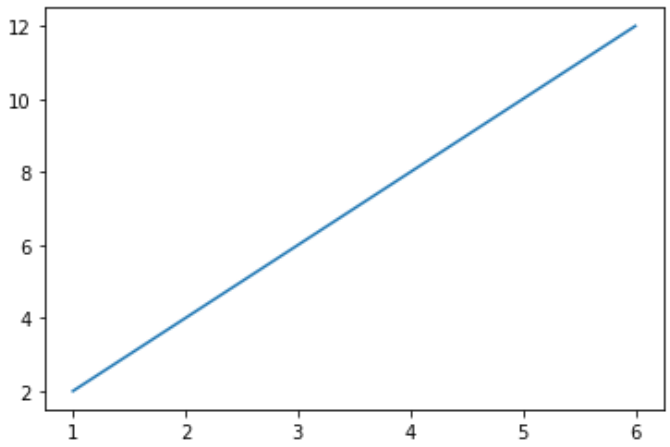
In [2]:

```
x_vals = [12,34,45,56,7]
y_vals = [34,23,12,45,34]
plt.plot(x_vals,y_vals)
plt.show()
```



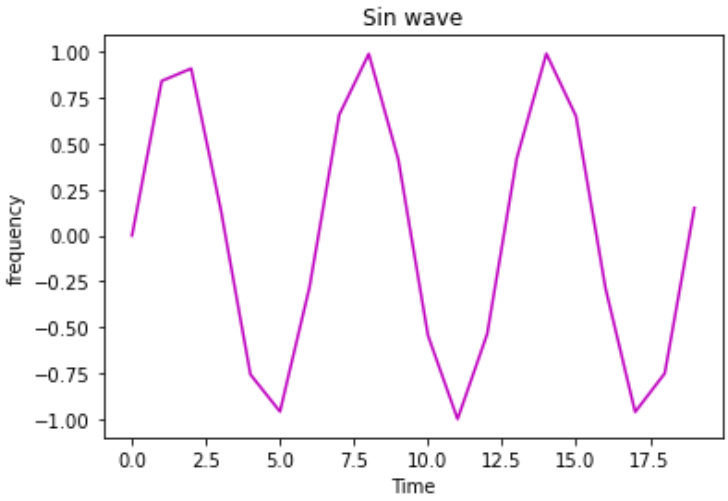
In [7]:

```
import numpy as np
x = np.array([1,2,3,4,5,6])
y = x*2
plt.plot(x,y)
plt.show()
```




In [19]:

```
x = np.arange(20)
y = np.sin(x)
plt.plot(x,y,"m")
plt.title("Sin wave")
plt.xlabel("Time")
plt.ylabel("frequency")
plt.show()
```




In [15]:

```
help(plt.plot)
```

```
Help on function plot in module matplotlib.pyplot:

plot(*args, scalex=True, scaley=True, data=None, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], *, data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, *y*.

    The optional parameter *fmt* is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the *Notes* section below.
```

```
>>> plot(x, y)        # plot x and y using default line style and color
>>> plot(x, y, 'bo')  # plot x and y using blue circle markers
>>> plot(y)           # plot y using x as index array 0..N-1
>>> plot(y, 'r+')     # ditto, but with red plusses

You can use `.Line2D` properties as keyword arguments for more
control on the appearance. Line properties and *fmt* can be mixed.
The following two calls yield identical results:

>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
...      linewidth=2, markersize=12)

When conflicting with *fmt*, keyword arguments take precedence.


**Plotting labelled data**

There's a convenient way for plotting objects with labelled data (i.e.
data that can be accessed by index ``obj['y']``). Instead of giving
the data in *x* and *y*, you can provide the object in the *data*
parameter and just give the labels for *x* and *y*::

>>> plot('xlabel', 'ylabel', data=obj)

All indexable objects are supported. This could e.g. be a `dict`, a
`pandas.DataFame` or a structured numpy array.


**Plotting multiple sets of data**

There are various ways to plot multiple sets of data.

- The most straight forward way is just to call `plot` multiple times.
  Example:

  >>> plot(x1, y1, 'bo')
  >>> plot(x2, y2, 'go')

- Alternatively, if your data is already a 2d array, you can pass it
  directly to *x*, *y*. A separate data set will be drawn for every
  column.

  Example: an array ``a`` where the first column represents the *x*
  values and the other columns are the *y* columns::

  >>> plot(a[0], a[1:])

- The third way is to specify multiple sets of *[x]*, *y*, *[fmt]*
  groups::

  >>> plot(x1, y1, 'g^', x2, y2, 'g-')

  In this case, any additional keyword argument applies to all
  datasets. Also this syntax cannot be combined with the *data*
  parameter.

By default, each line is assigned a different style specified by a
'style cycle'. The *fmt* and line property parameters are only
necessary if you want explicit deviations from these defaults.
Alternatively, you can also change the style cycle using
:rc:`axes.prop_cycle`.


Parameters
----------
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    *x* values are optional and default to `range(len(y))`.

    Commonly, these parameters are 1D arrays.
```

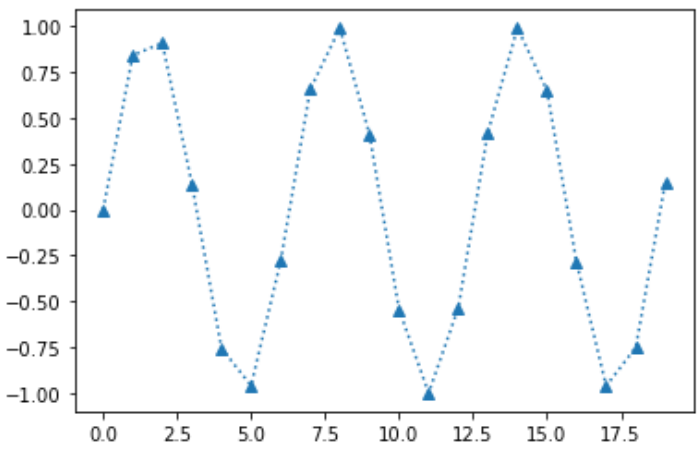
```
        They can also be scalars, or two-dimensional (in that case, the
        columns represent separate data sets).

        These arguments cannot be passed as keywords.

    fmt : str, optional
        A format string, e.g. 'ro' for red circles. See the *Notes*
        section for a full description of the format strings.

        Format strings are just an abbreviation for quickly setting
        basic line properties. All of these and more can also be
        controlled by keyword arguments.

        This argument cannot be passed as keyword.

    data : indexable object, optional
        An object with labelled data. If given, provide the label names to
        plot in *x* and *y*.

        .. note::
            Technically there's a slight ambiguity in calls where the
            second label is a valid *fmt*. `plot('n', 'o', data=obj)`
            could be `plt(x, y)` or `plt(y, fmt)`. In such cases,
            the former interpretation is chosen, but a warning is issued.
            You may suppress the warning by adding an empty format string
            `plot('n', 'o', '', data=obj)`.

    Other Parameters
    ----------------
    scalex, scaley : bool, optional, default: True
        These parameters determined if the view limits are adapted to
        the data limits. The values are passed on to `autoscale_view`.

    **kwargs : `.Line2D` properties, optional
        *kwargs* are used to specify properties like a line label (for
        auto legends), linewidth, antialiasing, marker face color.
        Example::

        >>> plot([1, 2, 3], [1, 2, 3], 'go-', label='line 1', linewidth=2)
        >>> plot([1, 2, 3], [1, 4, 9], 'rs', label='line 2')

        If you make multiple lines with one plot command, the kwargs
        apply to all those lines.

        Here is a list of available `.Line2D` properties:

        Properties:
        agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi valu
e, and returns a (m, n, 3) array
        alpha: float or None
        animated: bool
        antialiased or aa: bool
        clip_box: `.Bbox`
        clip_on: bool
        clip_path: Patch or (Path, Transform) or None
        color or c: color
        contains: callable
        dash_capstyle: {'butt', 'round', 'projecting'}
        dash_joinstyle: {'miter', 'round', 'bevel'}
        dashes: sequence of floats (on/off ink in points) or (None, None)
        data: (2, N) array or two 1D arrays
        drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, de
fault: 'default'
        figure: `.Figure`
        fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
        gid: str
        in_layout: bool
        label: object
        linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
        linewidth or lw: float
        marker: marker style
        markeredgecolor or mec: color
```

```
        markeredgecolor or mec: color
        markeredgewidth or mew: float
        markerfacecolor or mfc: color
        markerfacecoloralt or mfcalt: color
        markersize or ms: float
        markevery: None or int or (int, int) or slice or List[int] or float or (float, fl
oat)
        path_effects: `.AbstractPathEffect`
        picker: float or callable[[Artist, Event], Tuple[bool, dict]]
        pickradius: float
        rasterized: bool or None
        sketch_params: (scale: float, length: float, randomness: float)
        snap: bool or None
        solid_capstyle: {'butt', 'round', 'projecting'}
        solid_joinstyle: {'miter', 'round', 'bevel'}
        transform: `matplotlib.transforms.Transform`
        url: str
        visible: bool
        xdata: 1D array
        ydata: 1D array
        zorder: float

    Returns
    -------
    lines
        A list of `.Line2D` objects representing the plotted data.

    See Also
    --------
    scatter : XY scatter plot with markers of varying size and/or color (
        sometimes also called bubble chart).

    Notes
    -----
    **Format Strings**

    A format string consists of a part for color, marker and line::

        fmt = '[marker][line][color]'

    Each of them is optional. If not provided, the value from the style
    cycle is used. Exception: If ``line`` is given, but no ``marker``,
    the data will be a line without markers.

    Other combinations such as ``[color][marker][line]`` are also
    supported, but note that their parsing may be ambiguous.

    **Markers**

    =============    ===============================
    character        description
    =============    ===============================
    ``'.'``          point marker
    ``','``          pixel marker
    ``'o'``          circle marker
    ``'v'``          triangle_down marker
    ``'^'``          triangle_up marker
    ``'<'``          triangle_left marker
    ``'>'``          triangle_right marker
    ``'1'``          tri_down marker
    ``'2'``          tri_up marker
    ``'3'``          tri_left marker
    ``'4'``          tri_right marker
    ``'s'``          square marker
    ``'p'``          pentagon marker
    ``'*'``          star marker
    ``'h'``          hexagon1 marker
    ``'H'``          hexagon2 marker
    ``'+'``          plus marker
    ``'x'``          x marker
    ``'D'``          diamond marker
    ``'d'``          thin_diamond marker
    ``'|'``          vline marker
```

```
    '|'             vline marker
    '_'             hline marker
    =============   ===============================

    **Line Styles**

    =============   ===============================
    character       description
    =============   ===============================
    '-'             solid line style
    '--'            dashed line style
    '-.'            dash-dot line style
    ':'             dotted line style
    =============   ===============================

    Example format strings::

        'b'    # blue markers with default shape
        'or'   # red circles
        '-g'   # green solid line
        '--'   # dashed line with default color
        '^k:'  # black triangle_up markers connected by a dotted line

    **Colors**

    The supported color abbreviations are the single letter codes

    =============   ===============================
    character       color
    =============   ===============================
    'b'             blue
    'g'             green
    'r'             red
    'c'             cyan
    'm'             magenta
    'y'             yellow
    'k'             black
    'w'             white
    =============   ===============================

    and the 'CN' colors that index into the default property cycle.

    If the color is the only part of the format string, you can
    additionally use any  `matplotlib.colors` spec, e.g. full names
    ('green') or hex strings ('#008000').
```

In [23]:

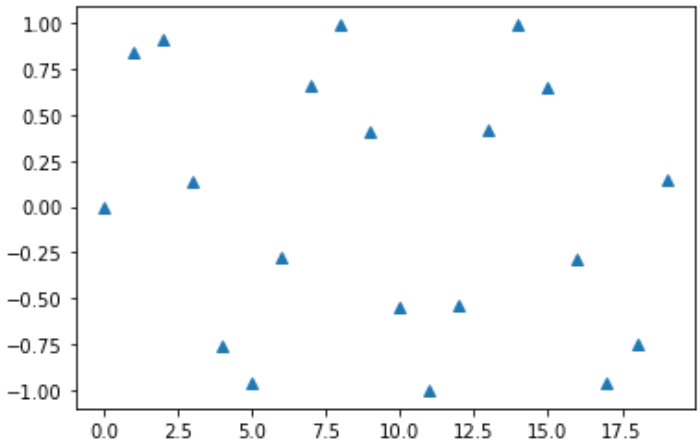
```
plt.plot(x,y,marker = "^",linestyle = ':')
# markers -- D, d , P, o , ^, <, >
# linestyle -- -, --, -., :, " "
plt.show()
```




In [24]:

```
plt.plot(x,y,marker = "^",linestyle = ':')
```

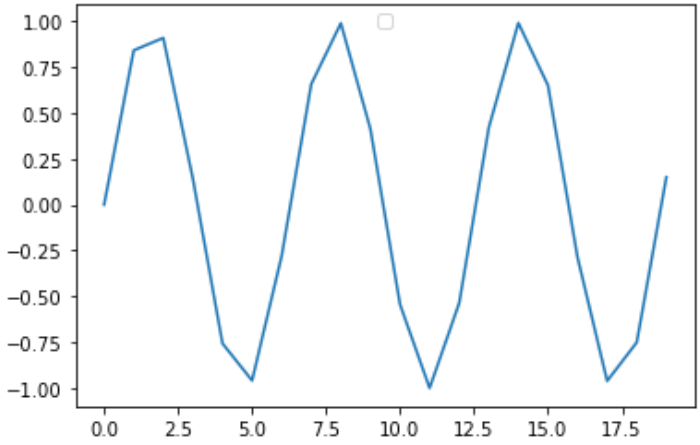
```
plt.plot(x,y,marker = "^",linestyle = "")
# markers -- D, d , P, o , ^, <, >
# linestyle -- -, --, -., :, " "
plt.show()
```




In [32]:

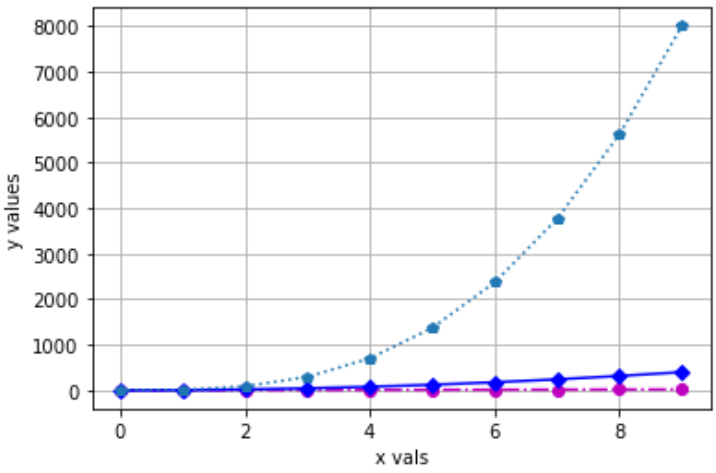
```
plt.plot(x,y)
plt.legend(loc = "upper center")
plt.show()
```

```
No handles with labels found to put in legend.
```




In [46]:

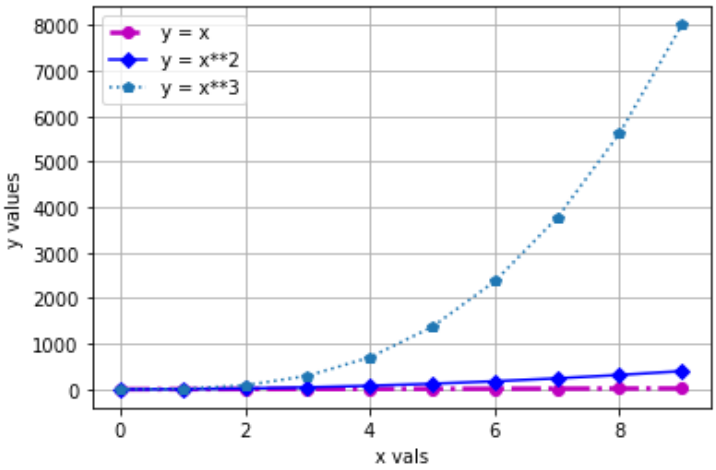
```
x = np.linspace(0,20,10)
plt.plot(x, "m", marker = "o", linestyle = "-.")
plt.plot(x**2, "b", marker = "D", linestyle = "-")
plt.plot(x**3, "*", marker = "p", linestyle = ":")
plt.grid()
plt.xlabel("x vals")
plt.ylabel("y values")
plt.show()
```




In [53]:

In [52]:

```
x = np.linspace(0,20,10)
plt.plot(x, "m", marker = "o", linestyle = "-.", linewidth = 2.5)
plt.plot(x**2, "b", marker = "D", linestyle = "-")
plt.plot(x**3,"*", marker = "p", linestyle = ":")
plt.grid()
plt.xlabel("x vals")
plt.ylabel("y values")
plt.legend(["y = x", "y = x**2", 'y = x**3'])
plt.show()
```

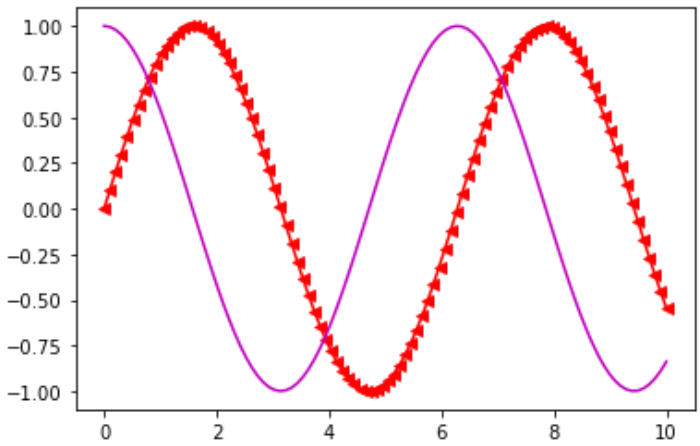



In [58]:

```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x), color = "r", marker = "<")
plt.plot(x,np.cos(x),color = "m")

plt.plot()
```

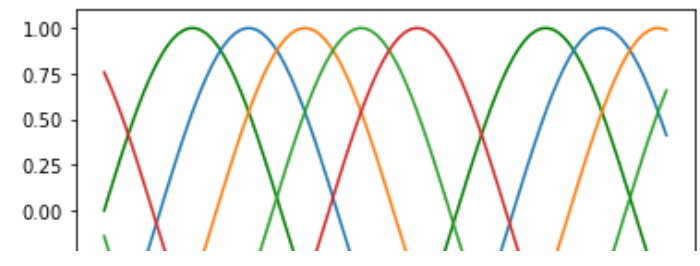
Out[58]:

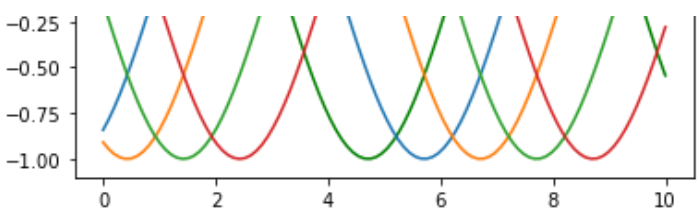
```
[]
```




In [61]:

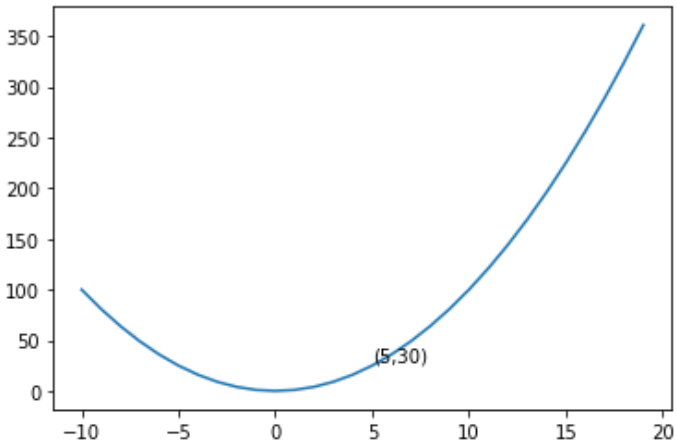
```
plt.plot(x, np.sin(x-0), "g")
plt.plot(x, np.sin(x-1))
plt.plot(x, np.sin(x-2))
plt.plot(x, np.sin(x-3))
plt.plot(x, np.sin(x-4))
plt.show()
```






In [64]:

```
x  = np.arange(-10,20)
y = x**2
plt.plot(x,y)
plt.text(5,30,"(5,30)")
plt.show()
```

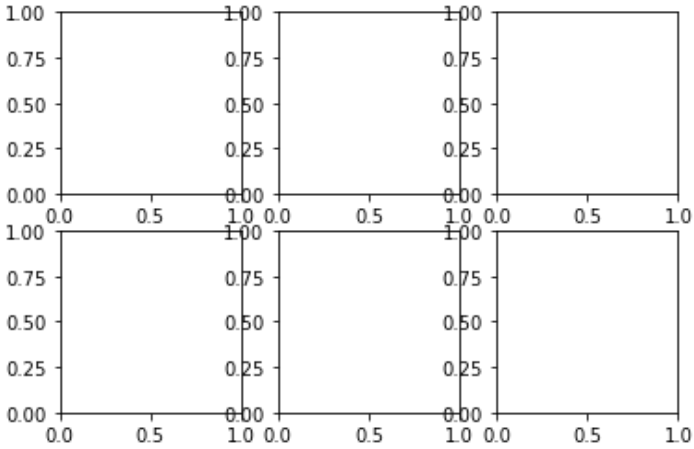



# Subplots in plot

- **used to plot multiple plots at a time**
- **subplots( no. of rows, no. of columns)**

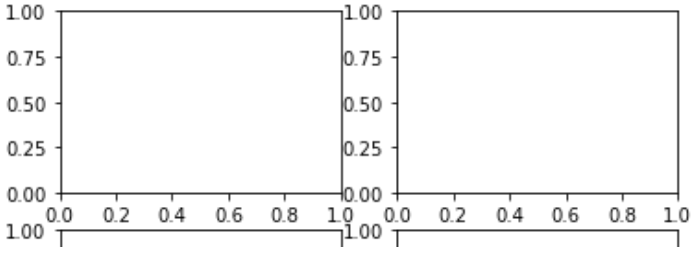
In [66]:

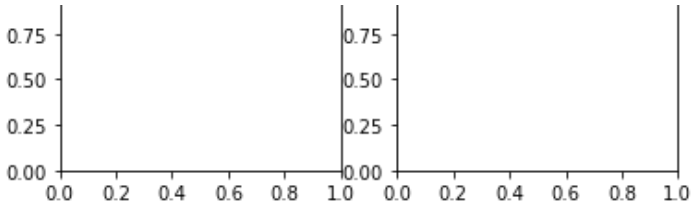
```
plt.subplots(2,3)
plt.show()
```




In [67]:

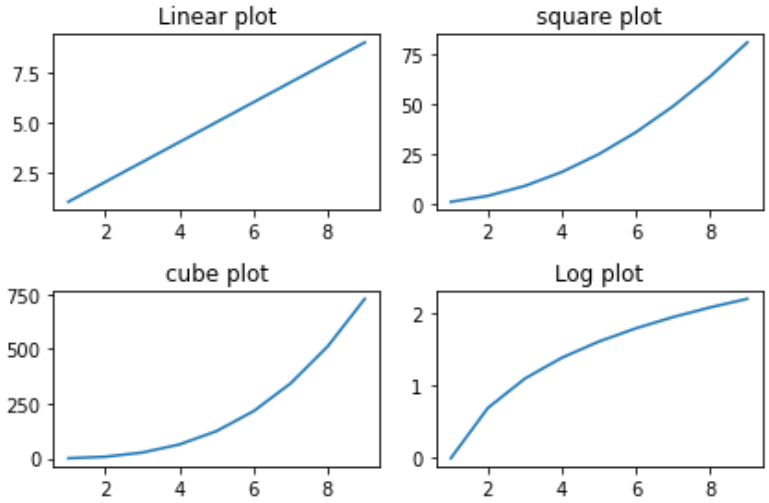
```
fig, data = plt.subplots(2,2)
```






In [77]:

```
fig, data = plt.subplots(2,2)
x = np.arange(1,10)
data[0][0].plot(x,x)
data[0][0].set_title("Linear plot")
data[0][1].plot(x,x**2)
data[0][1].set_title("square plot")
data[1][0].plot(x,x**3)
data[1][0].set_title("cube plot")
data[1][1].plot(x,np.log(x))
data[1][1].set_title("Log plot")
plt.tight_layout()
# print(fig)
# print(data)
```

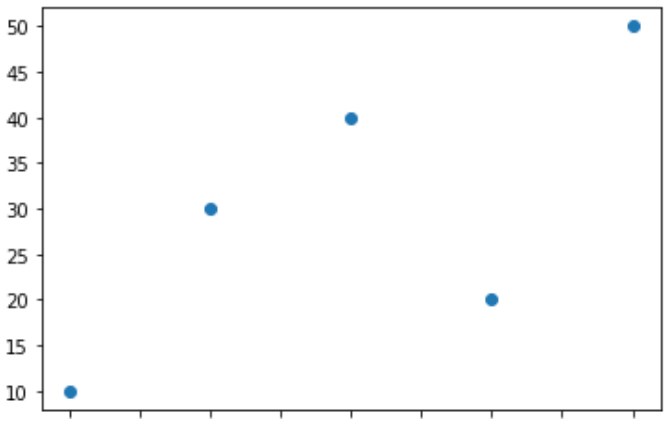



# 2. Scatter plot

- **Applications : Medical, Clustering..**
- **used to identify probability**
- **data points are represented by dots**
- **easy find outliers**

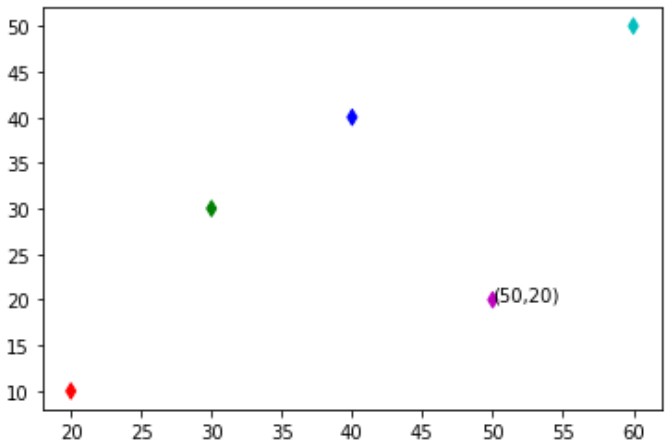
In [78]:

```
x = [20,30,40,50,60]
y = [10,30,40,20,50]
plt.scatter(x,y)
plt.show()
```



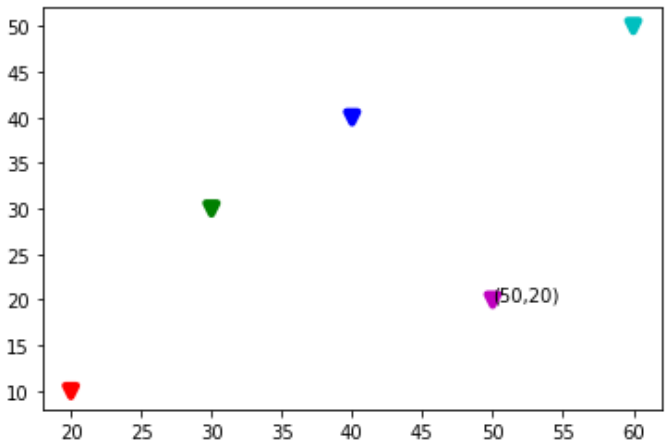

In [82]:

```
x = [20,30,40,50,60]
y = [10,30,40,20,50]
plt.scatter(x,y, color = ["r","g","b","m","c"],marker = "d")
plt.text(50,20,"(50,20)")
plt.show()
```



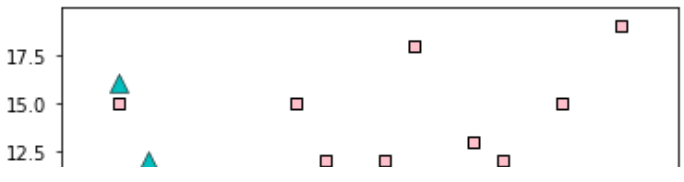

In [86]:

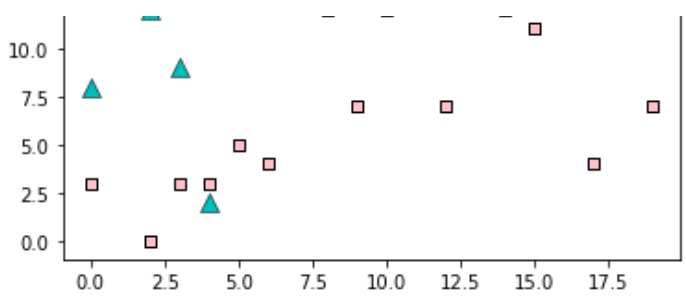
```
x = [20,30,40,50,60]
y = [10,30,40,20,50]
plt.scatter(x,y, color = ["r","g","b","m","c"],marker = "v", linewidth = 4)
plt.text(50,20,"(50,20)")
plt.show()
```




In [107]:

```
# plotting different data in single graph
# d ataset1
x = np.arange(20)
y = np.random.randint(0,20,20)

# dataset 2
x1 = np.arange(0,5)
y1 = np.random.randint(0,20,5)

plt.scatter(x,y, color = "pink", linewidth = 1, marker = "s", edgecolors="black")

plt.scatter(x1,y1,color = "c", linewidth = 0.5, marker = "^", s = 100, edgecolors="black
")
plt.show()
```






In [104]:

```
np.random.randint(0,20,10)
```

Out[104]:

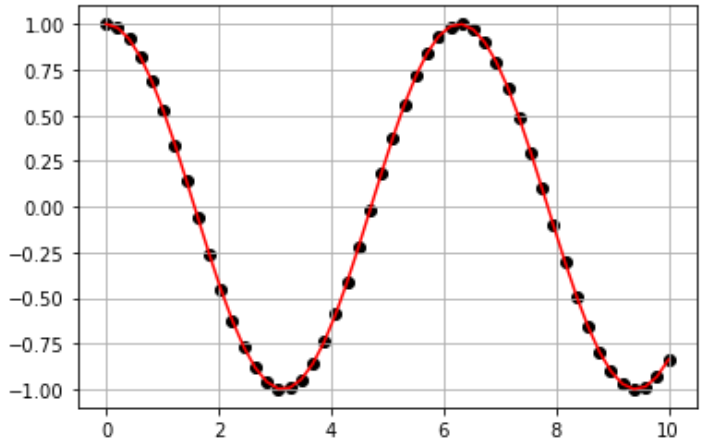
```
array([ 4,  6, 11,  1,  0,  0, 13, 12, 16,  4])
```

In [119]:

```
x = np.linspace(0,10,50)
y = np.cos(x)
```

In [121]:

```
# line & scatter plot in a single graph
plt.scatter(x,y, color = "black")
plt.plot(x,y, "r")
plt.grid()
plt.show()
```




In [122]:

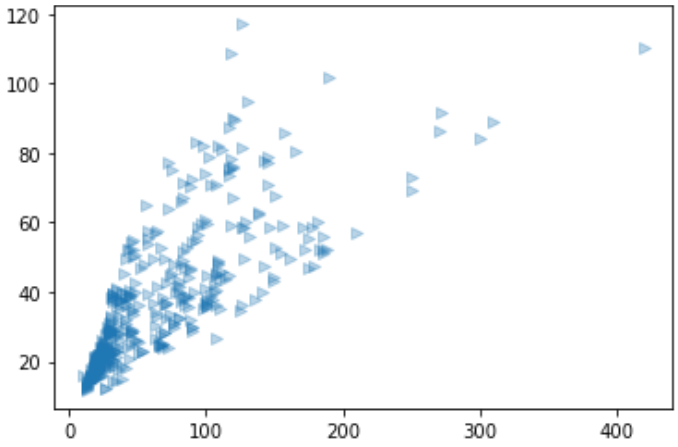
```
import pandas as pd
birds = pd.read_csv("https://raw.githubusercontent.com/APSSDC-Data-Analysis/DataAnalysis-Batch-7/main/Datasets/birds.csv")
birds.head()
```

Out[122]:

| | id | huml | humw | ulnal | ulnaw | feml | femw | tibl | tibw | tarl | tarw | type |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 80.78 | 6.68 | 72.01 | 4.88 | 41.81 | 3.70 | 5.50 | 4.03 | 38.70 | 3.84 | SW |
| **1** | 1 | 88.91 | 6.63 | 80.53 | 5.59 | 47.04 | 4.30 | 80.22 | 4.51 | 41.50 | 4.01 | SW |
| **2** | 2 | 79.97 | 6.37 | 69.26 | 5.28 | 43.07 | 3.90 | 75.35 | 4.04 | 38.31 | 3.34 | SW |
| **3** | 3 | 77.65 | 5.70 | 65.76 | 4.77 | 40.04 | 3.52 | 69.17 | 3.40 | 35.78 | 3.41 | SW |
| **4** | 4 | 62.80 | 4.84 | 52.09 | 3.73 | 33.95 | 2.72 | 56.27 | 2.96 | 31.88 | 3.13 | SW |

In [127]:

```
plt.scatter(birds["huml"], birds["feml"],marker = ">", alpha = 0.3)
plt.show()
```

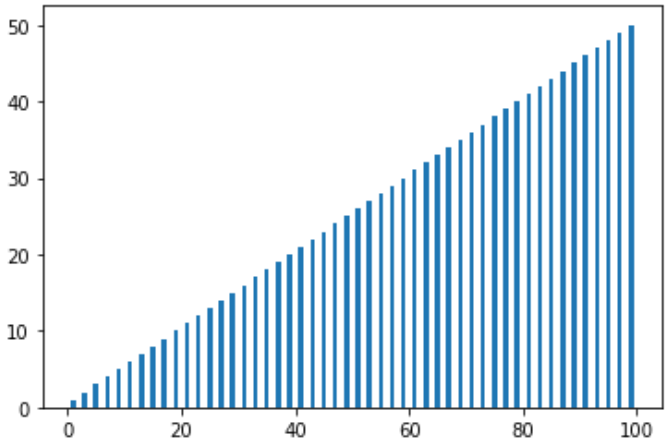



# 3. Bar Graphs

- **mainly depends on heights**
- **for both numeric data & categorial data**
- **Applications: representing populations in each & every year,**

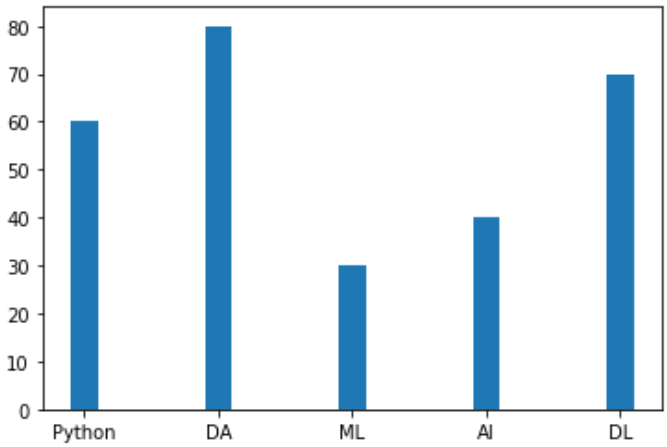
In [133]:

```
x = np.arange(1,100,2)
y = np.linspace(1, 50, len(x))
plt.bar(x,y)
plt.show()
```



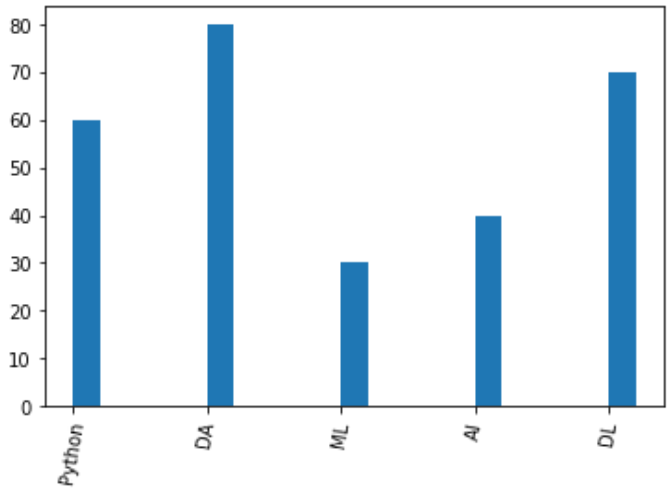

In [137]:

```
sub = ["Python","DA","ML","AI","DL"]
y = [60,80,30,40,70]  # height of bars
plt.bar(sub,y, width = 0.2)
plt.show()
```



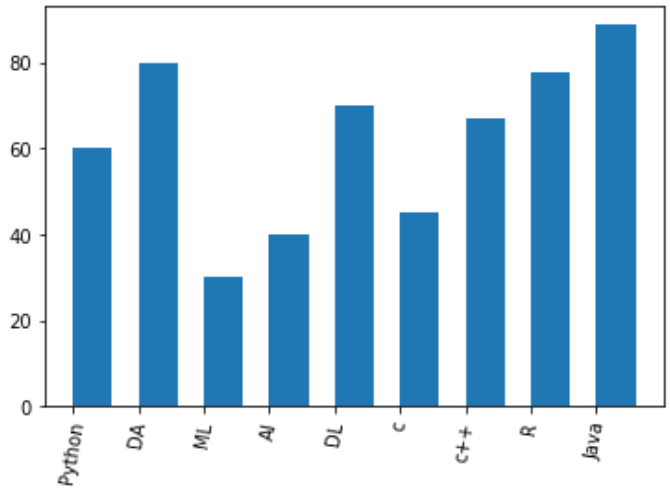

In [140]:

```
sub = ["Python","DA","ML","AI","DL"]
y = [60,80,30,40,70]  # height of bars
plt.bar(sub,y, width = 0.2, align= "edge")
plt.xticks(rotation = 80)
plt.show()
```



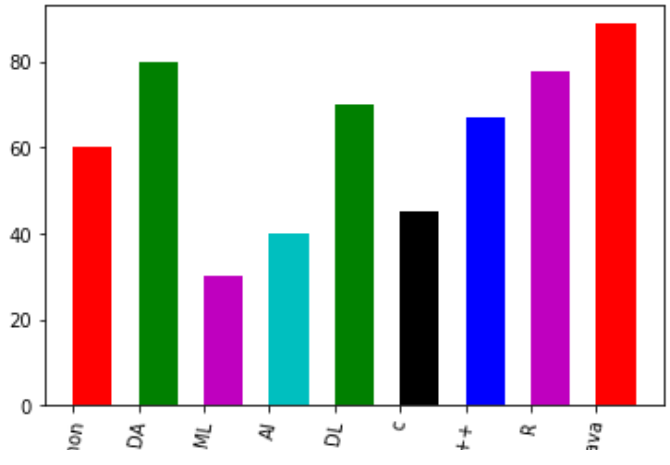

In [146]:

```
sub = ["Python","DA","ML","AI","DL", "c","c++","R","Java"]
y = [60,80,30,40,70, 45,67,78,89]  # height of bars
plt.bar(sub,y, width = 0.6, align= "edge")
plt.xticks(rotation = 80)
plt.show()
```




In [149]:

```
sub = ["Python","DA","ML","AI","DL", "c","c++","R","Java"]
y = [60,80,30,40,70, 45,67,78,89]  # height of bars
plt.bar(sub,y, width = 0.6, align= "edge", color = ["r","g","m","c","g","black","b","m","r"])
plt.xticks(rotation = 80)
plt.show()
```

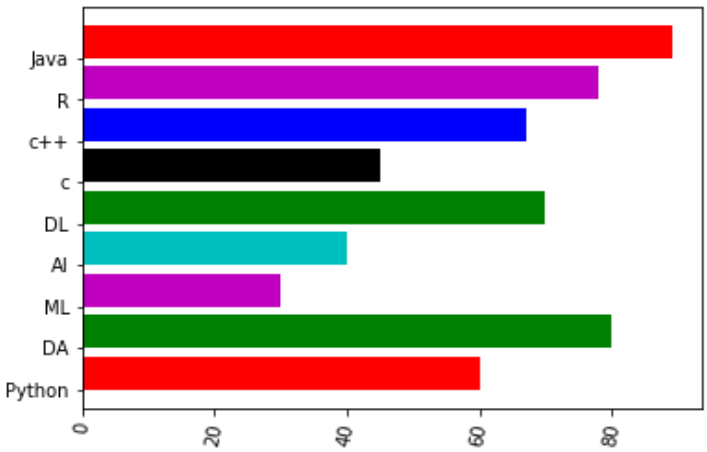


In [ ]:

```
help(plt.bar)
```

In [151]:

```
sub = ["Python","DA","ML","AI","DL", "c","c++","R","Java"]
y = [60,80,30,40,70, 45,67,78,89]  # height of bars
plt.barh(sub,y, align= "edge", color = ["r","g","m","c","g","black","b","m","r"])
plt.xticks(rotation = 80)
plt.show()
```




# Task

- **Display multiple lines in Subplots**
  - **Subplot1**
    - **Let x = np.linspace(1,50,100)**
    - **line1 : Take x on X-axis & Take sin of x on Y-axis**
    - **line2 : Take x on X-axis & Take cos of x on Y-axis**
  - **Subplot2**
    - **line1 : Take x on both X and y axis**
    - **line2 : Take x on X-axis and x^2 on y-axis**
    - **line3 : Take x on X-axis and x^3 on Y-axis**

In [ ]:

```

```