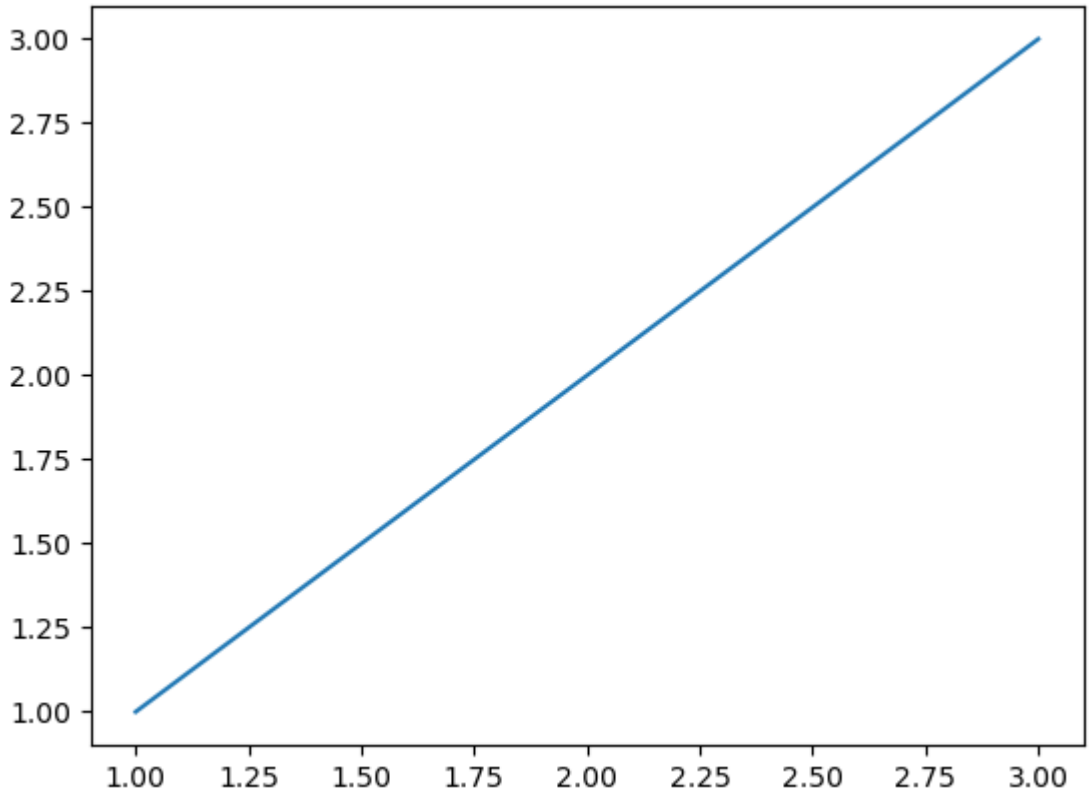
Working with Matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is a cross-platform, data visualization and graphical plotting library (histograms, scatter plots, bar charts, etc) for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

In [1]:
```
#importing library
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
plt.plot([1,2,3],[1,2,3]) #Ploting to our canvas
plt.show() # Showing what we plotted
```

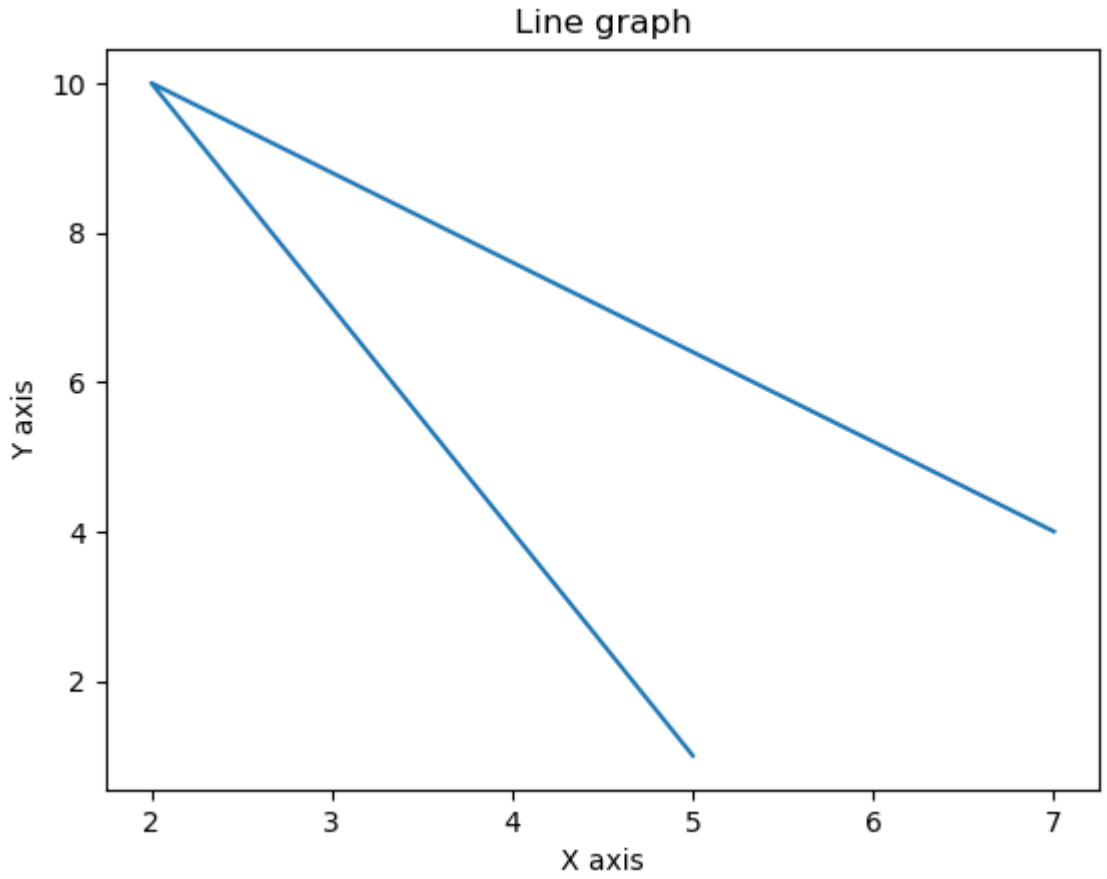


Let us now see how we can add a title, as well as the x-axis and y-axis names, using the title(), xlabel(), and ylabel() methods, respectively.

In [4]:
```
x = [5, 2, 7]
```

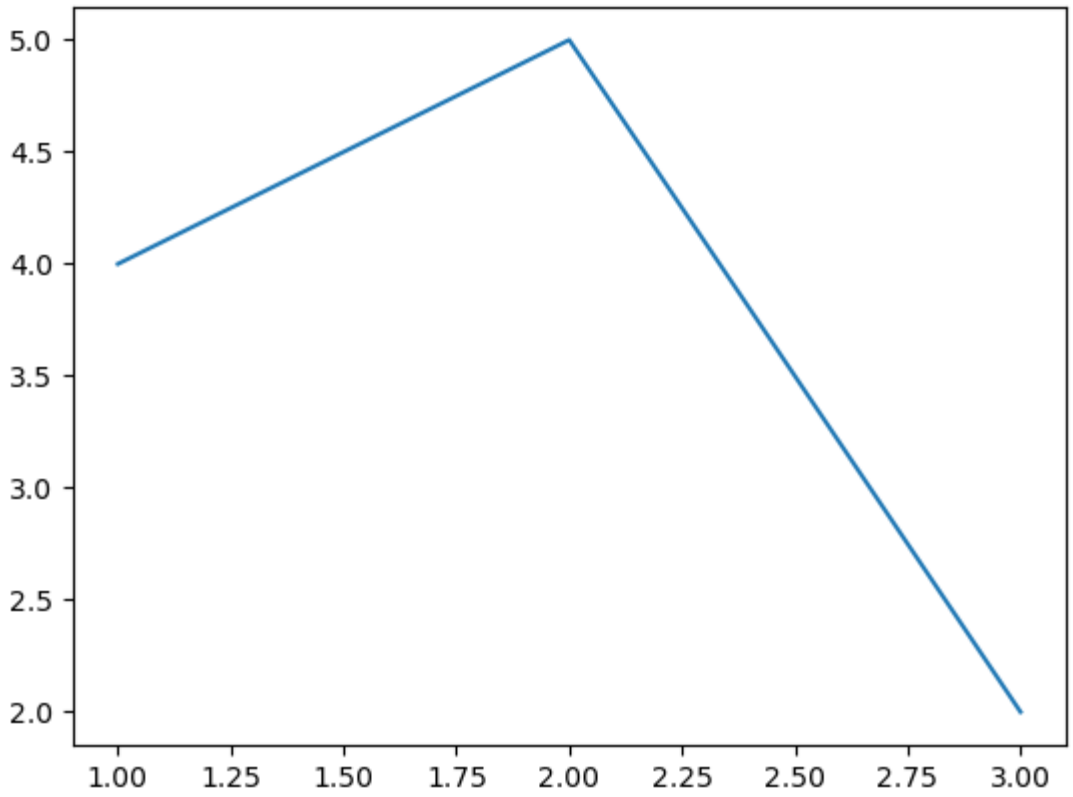
In [5]:
```
y = [1, 10, 4]
```

In [6]:
```
plt.plot(x, y)  # Ploting a graph
plt.title('Line graph') # Assigning title to the graph
plt.ylabel('Y axis')  #Labeling Y axis
plt.xlabel('X axis')  #Labeling X axis
plt.show()  # Diplay result
```



Users can also specify the size of the figure using the figure() method. Additionally, users can pass values as tuples, which make up the length of rows and columns to the argument figsize.

In [7]:
```
plt.plot([1,2,3],[4,5,2])
plt.figure(figsize=(100,120))
plt.show()
```

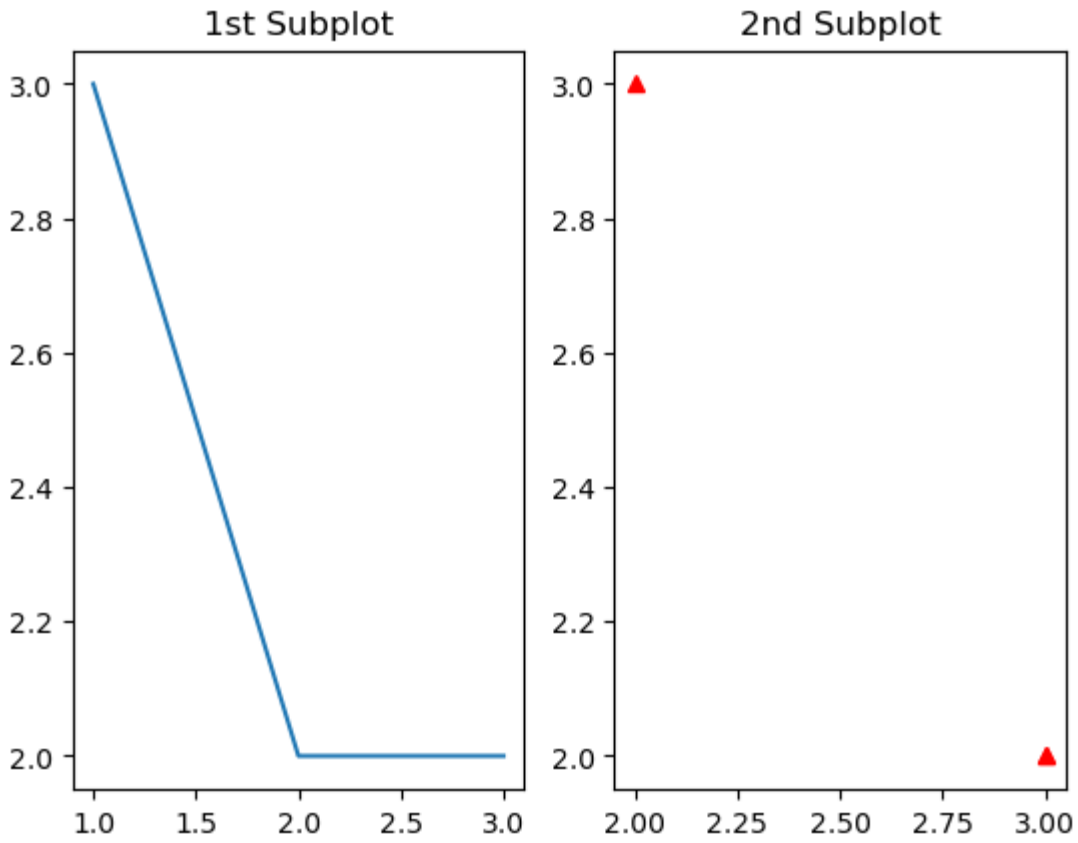



```
<Figure size 10000x12000 with 0 Axes>
```

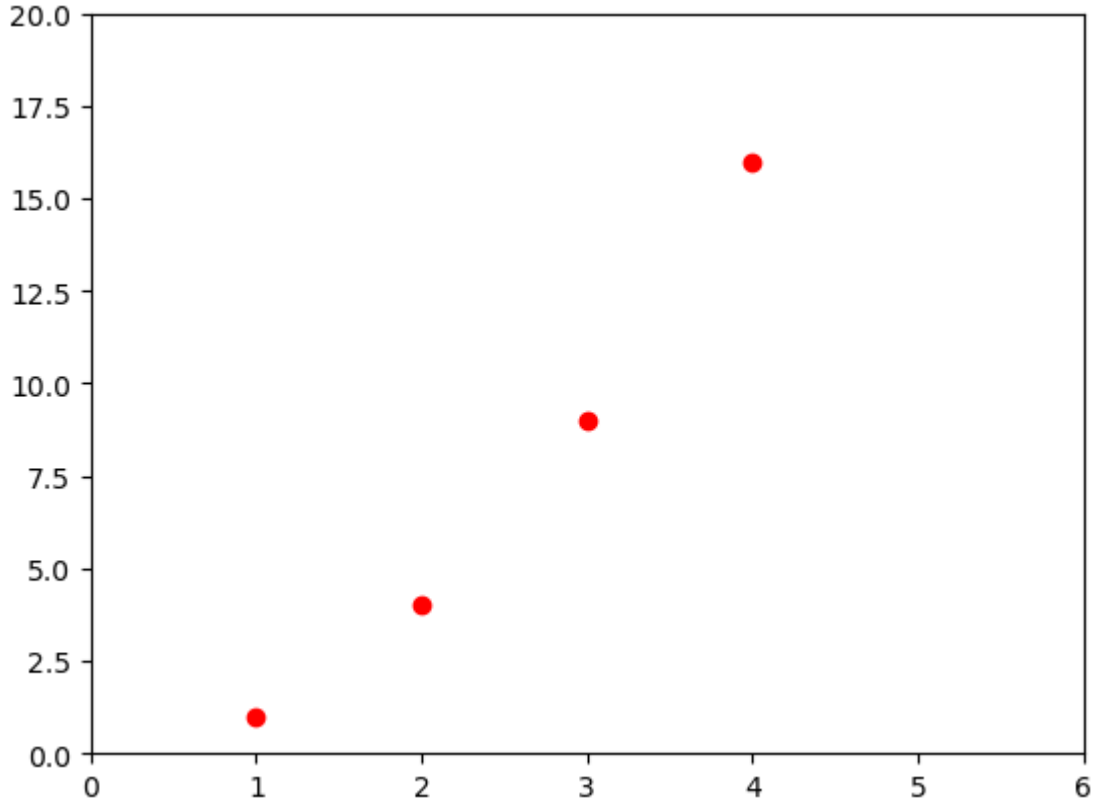
Matplotlib Subplots

You can use the subplot() method to add more than one plot in a figure. Syntax: plt.subplots(nrows, ncols, index) The three-integer arguments specify the number of rows and columns and the index of the subplot grid.

In [8]:
```
plt.subplot(1,2,1)
plt.plot([1,2,3],[3,2,2])
plt.title("1st Subplot")

plt.subplot(1,2,2)
plt.plot([2,3,3],[3,2,2], "r^")
plt.title("2nd Subplot")
plt.show()
```



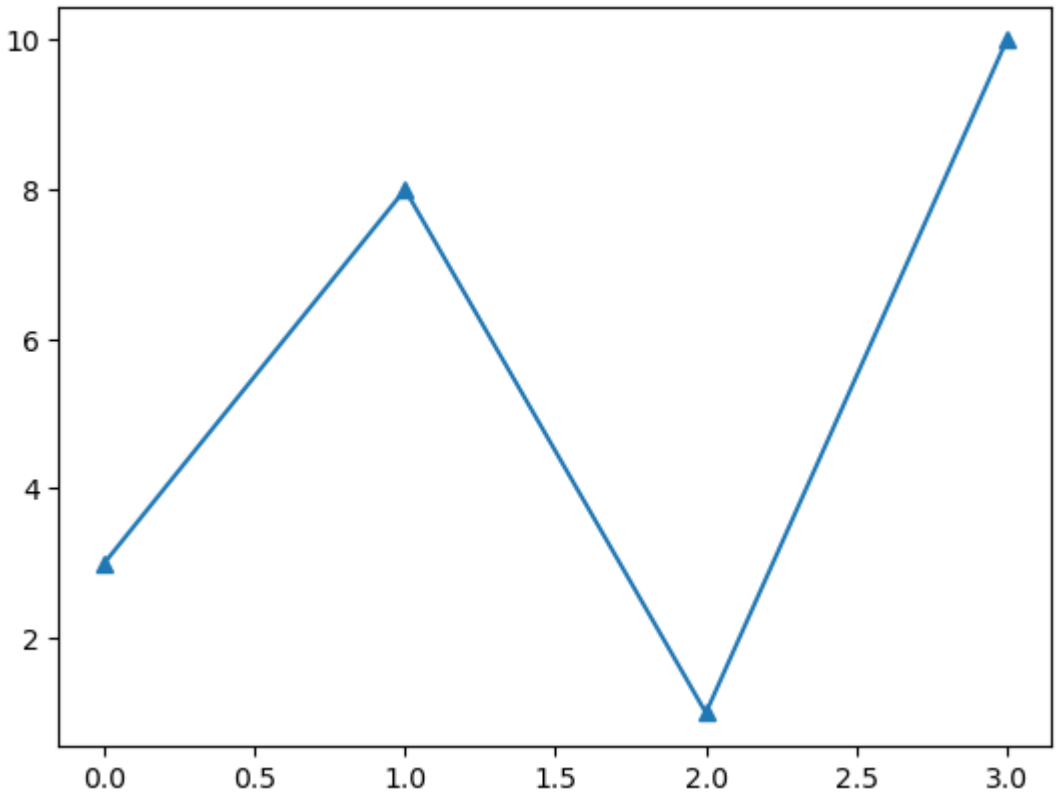
In [9]:
```
plt.plot([1, 2, 3, 4,5], [1, 4, 9, 16,25], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



'b'----> Using for the blue marker with default shape. 'ro'----> Red circle '-g'----> Green solid line '--'----> A

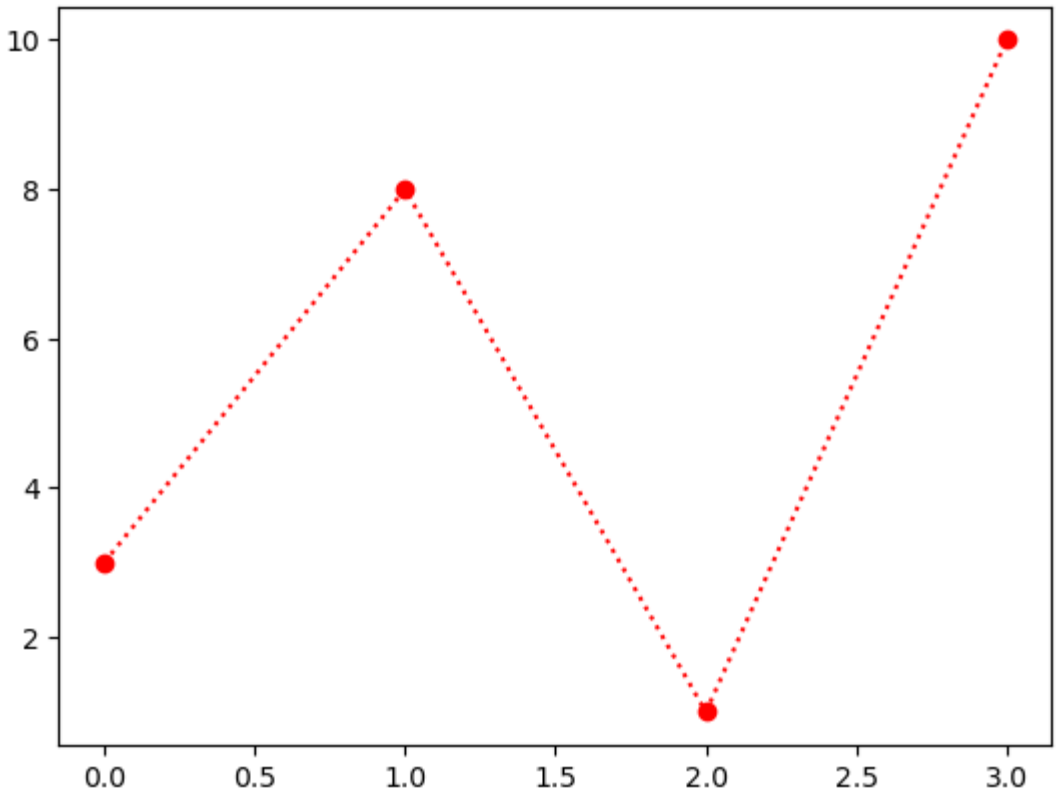
dashed line with the default color '^k:'----> Black triangle up markers connected by a dotted line
The matplotlib supports the following color abbreviation: Character Color 'b'------> Blue 'g'------> Green 'r'------> Red 'c'------> Cyan 'm'------> Magenta 'y'------> Yellow 'k'------> Black 'w'------> White
Markers You can use the keyword argument marker to emphasize each point with a specified marker:

In [15]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = '^')
plt.show()
```



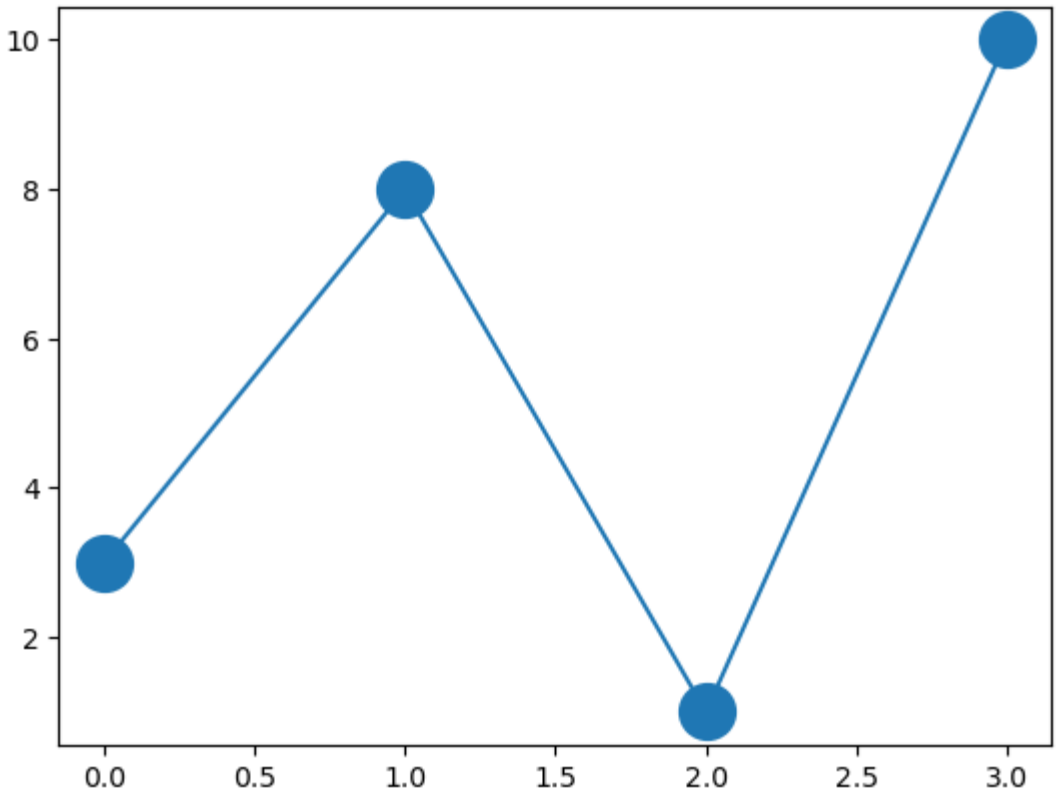
Marker Description 'o' Circle '*' Star '.' Point ',' Pixel 'x' X 'X' X (filled) '+' Plus 'P' Plus (filled) 's' Square 'D' Diamond 'd' Diamond (thin) 'p' Pentagon 'H' Hexagon 'h' Hexagon 'v' Triangle Down '^' Triangle Up '<' Triangle Left '>' Triangle Right '1' Tri Down '2' Tri Up '3' Tri Left '4' Tri Right '|' Vline '_' Hline
Format Strings fmt You can also use the shortcut string notation parameter to specify the marker. This parameter is also called fmt, and is written with this syntax: marker|line|color

In [16]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'o:r')
plt.show()
```



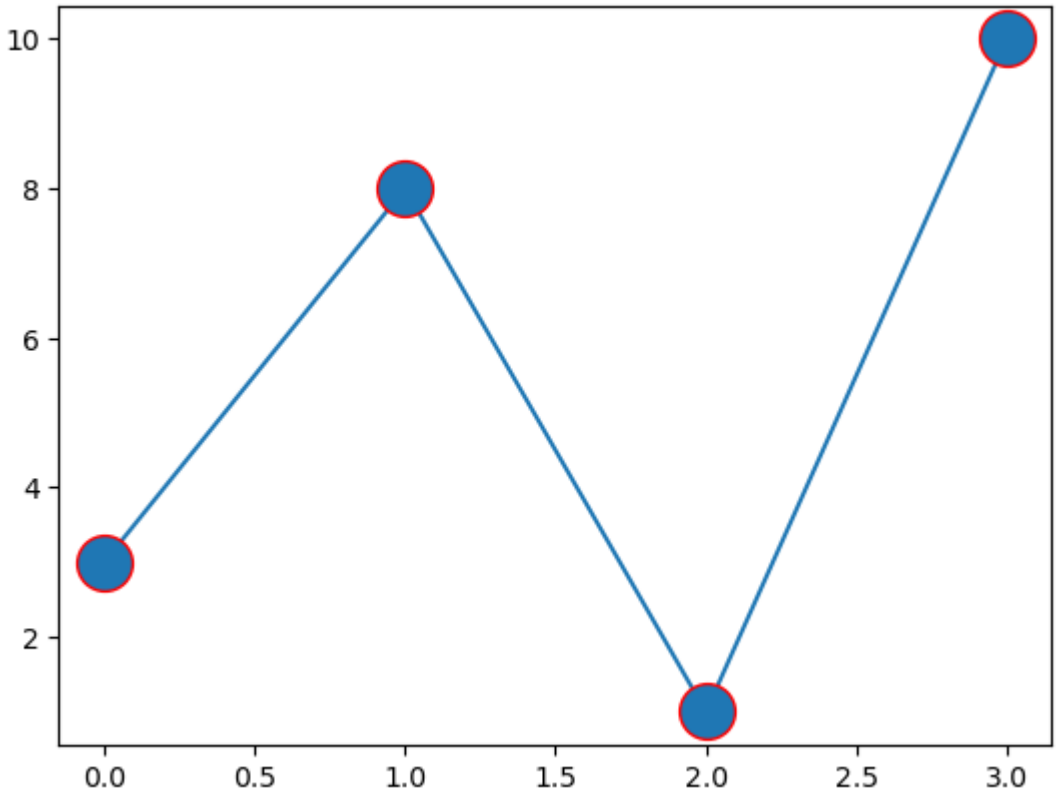
Line Syntax Description '-' Solid line ':' Dotted line '--' Dashed line '-.' Dashed/dotted line
Marker Size You can use the keyword argument markersize or the shorter version, ms to set the size of the markers:

In [17]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20)
plt.show()
```



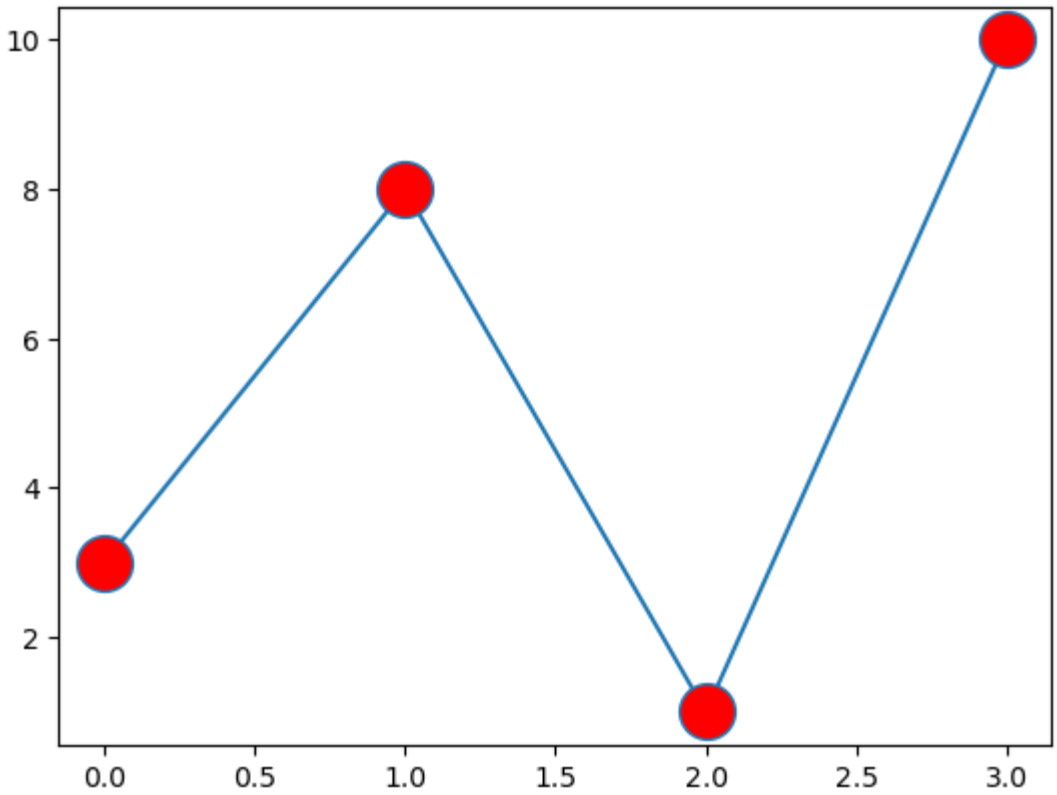
Marker Color You can use the keyword argument markeredgecolor or the shorter mec to set the color of the edge of the markers:

In [19]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')
plt.show()
```



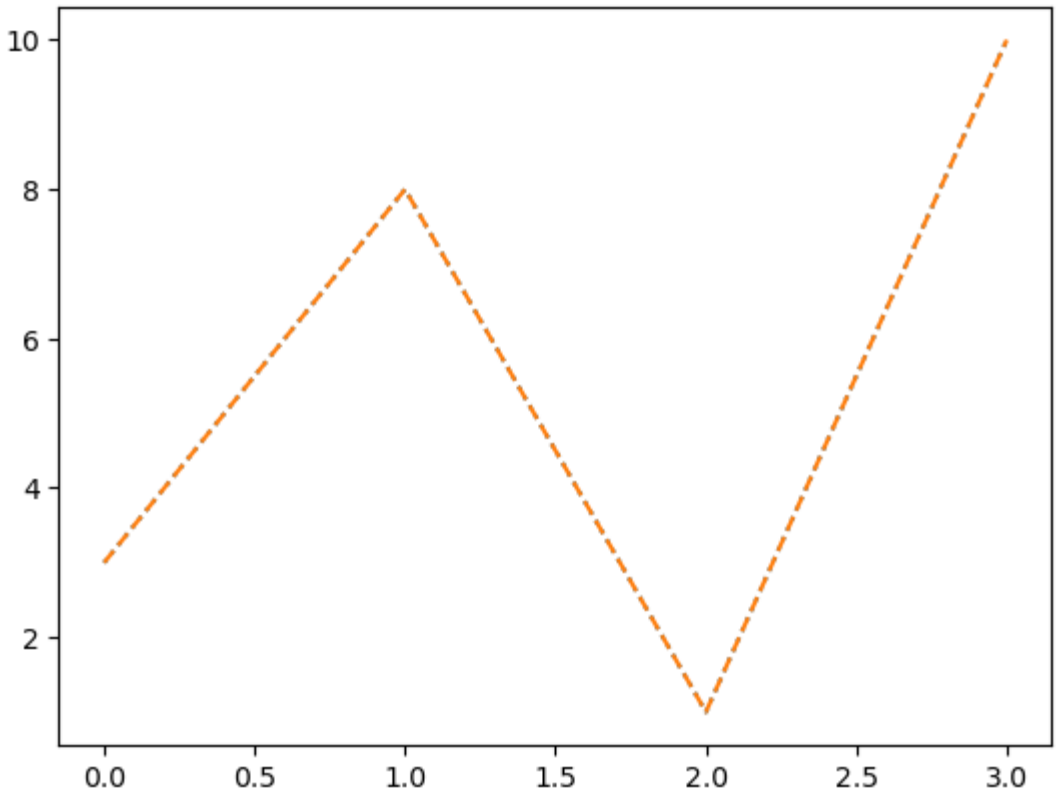
You can use the keyword argument markerfacecolor or the shorter mfc to set the color inside the edge of the markers:

In [20]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'r')
plt.show()
```



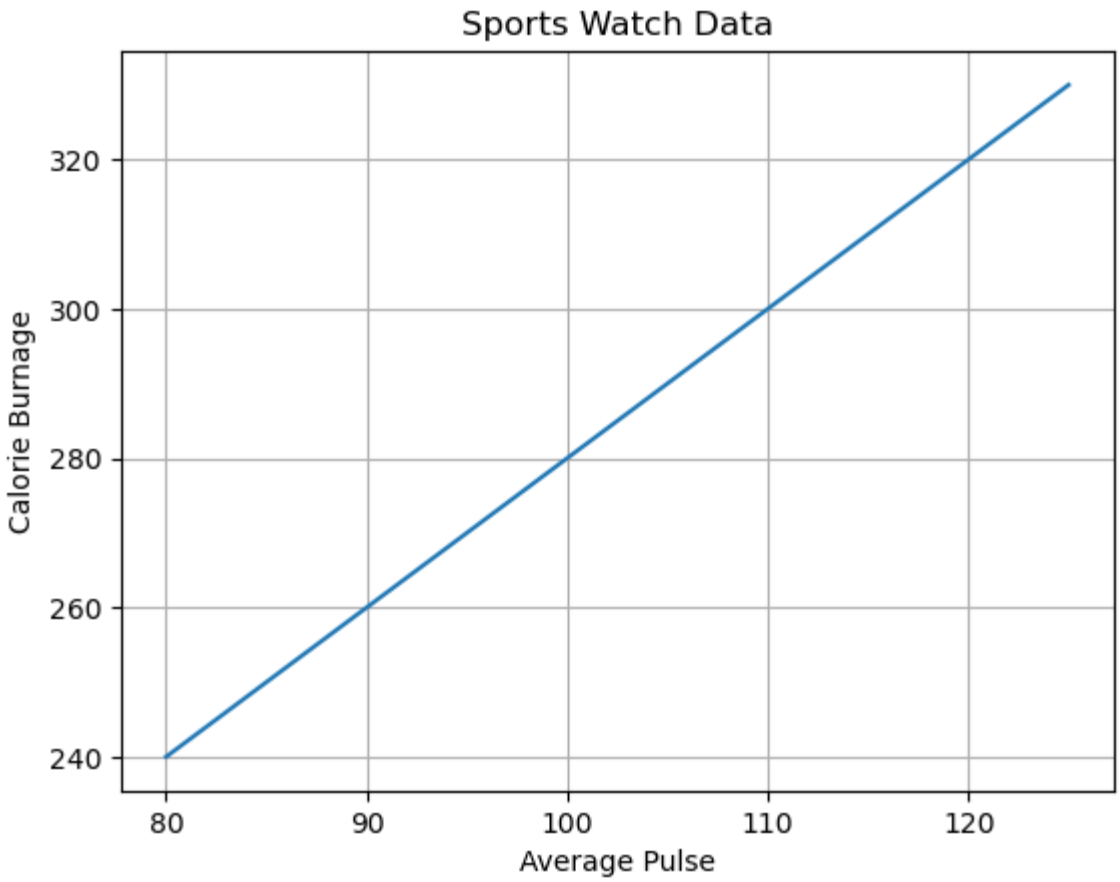
Linestyle You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line: Shorter Syntax The line style can be written in a shorter syntax: linestyle can be written as ls. dotted can be written as :. dashed can be written as --.

In [22]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.plot(ypoints, linestyle = 'dashed')
plt.show()
```



Matplotlib Adding Grid Lines

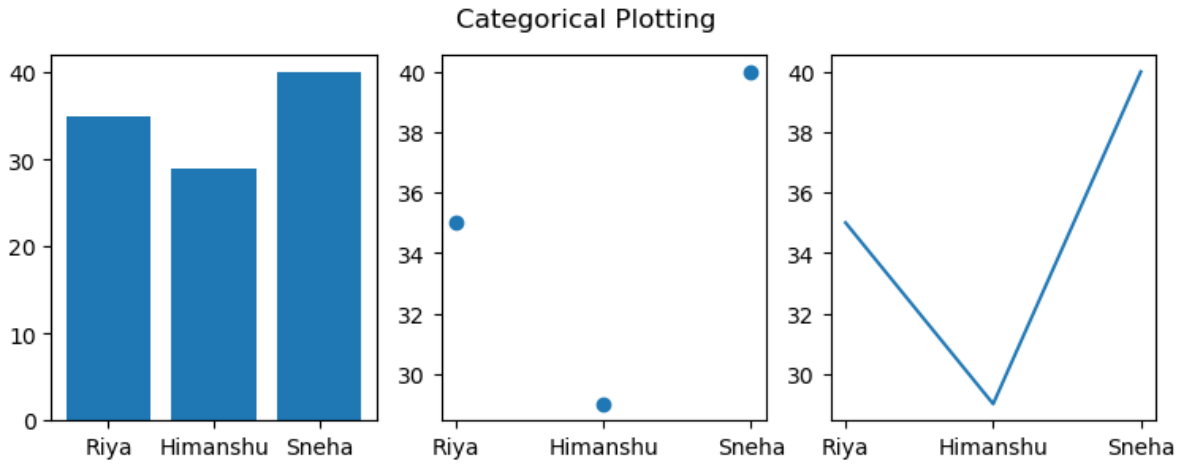
In [3]:
```
# you can use the grid() function to add grid lines to the plot.
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```



Plotting with categorical variables

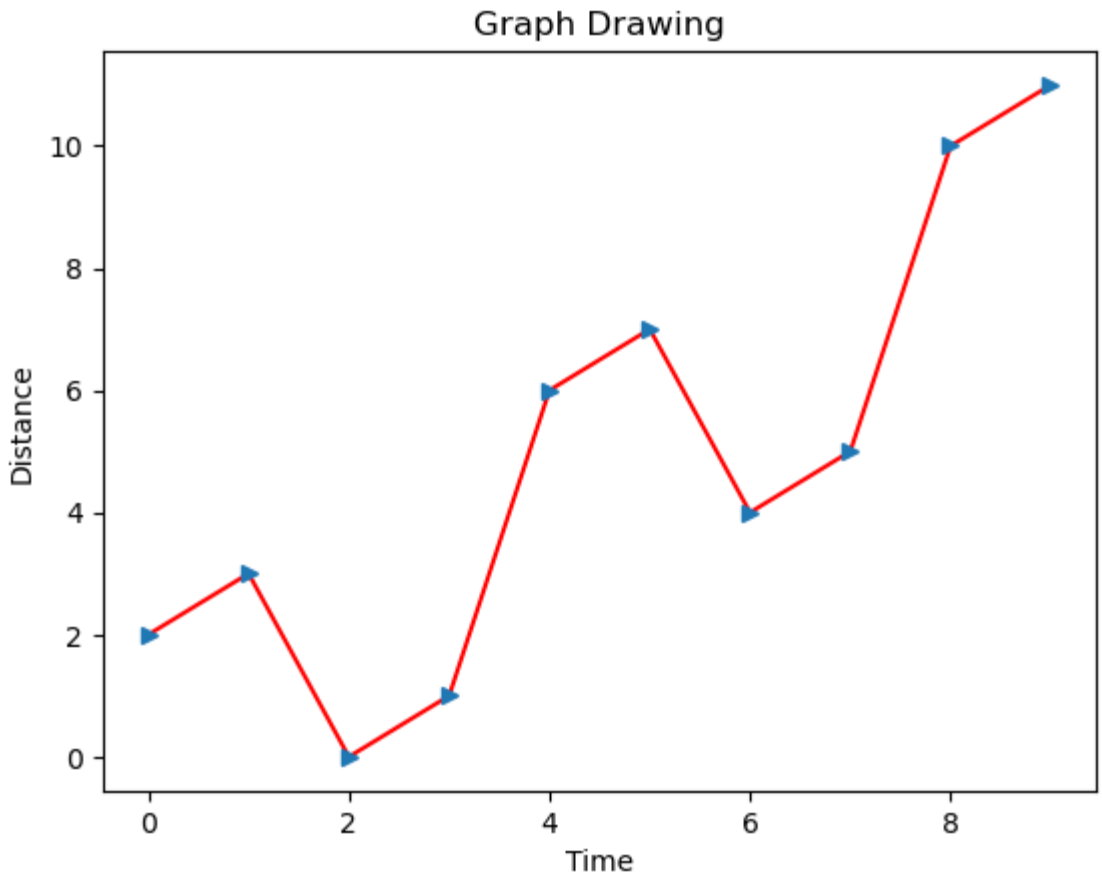
In [36]:
```
# Matplotlib allows us to pass categorical variables directly to many plotting func
```

In [37]:
```
Name=["Riya", "Himanshu", "Sneha"]
Age=[35,29,40]

plt.figure(figsize=(9,3))
plt.subplot(131)
plt.bar(Name, Age)
plt.subplot(132)
plt.scatter(Name, Age)
plt.subplot(133)
plt.plot(Name, Age)
plt.suptitle('Categorical Plotting')
plt.show()
```



Saving the chart file

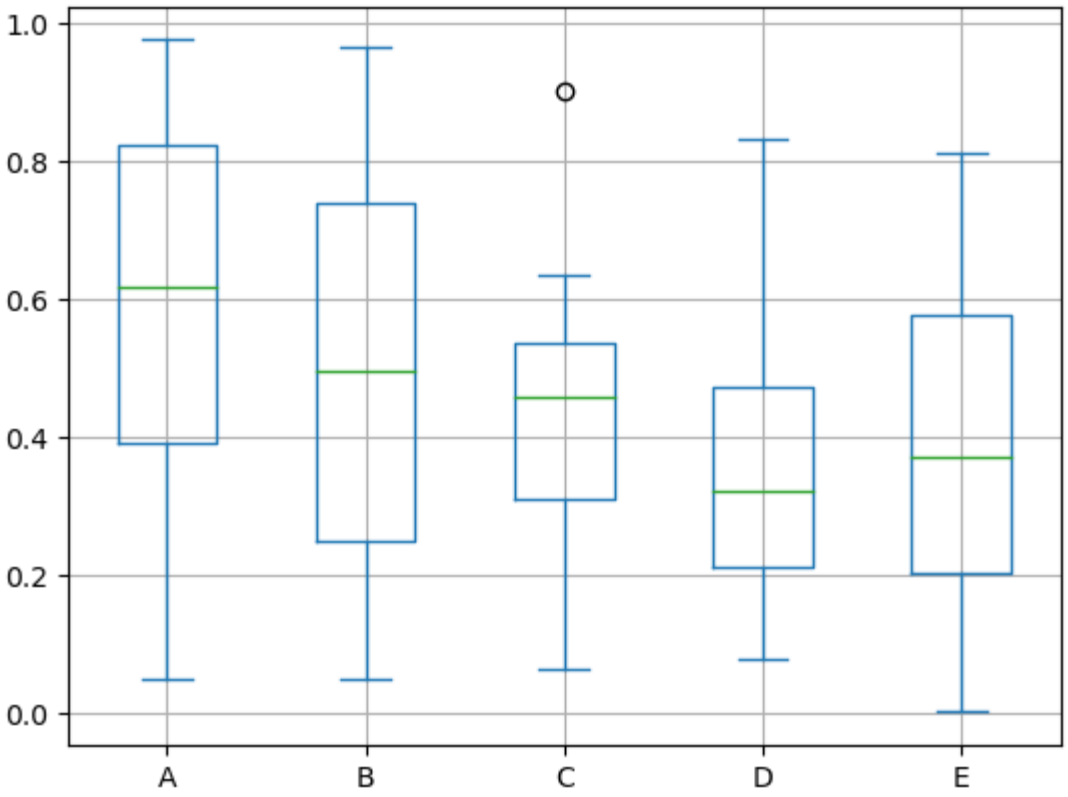
In [3]:
```
x = np.arange(0,10)
y = x ^ 2
#Labeling the Axes and Title
plt.title("Graph Drawing")
plt.xlabel("Time")
plt.ylabel("Distance")

# Formatting the line colors
plt.plot(x,y,'r')

# Formatting the line type
plt.plot(x,y,'>')

# save in pdf formats
plt.savefig('timevsdist.pdf', format='pdf')
```



Boxplots are a measure of how well distributed the data in a data set is. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them. Drawing a Box Plot Boxplot can be drawn calling Series.box.plot() and DataFrame.box.plot(), or DataFrame.boxplot() to visualize the distribution of values within each column. For instance, here is a boxplot representing five trials of 10 observations of a uniform random variable on [0,1).

In [8]:
```
import pandas as pd
df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])
df.plot.box(grid='True')
```

Out[8]:
```
<Axes: >
```



In [ ]: