Matplotlib

January 21, 2021

1 Module 0 - Matplotlib

A PDF version of this notebook is available at Module 0 - Matplotlib

We generally import the matplotlib.pyplot under the alias `plt`

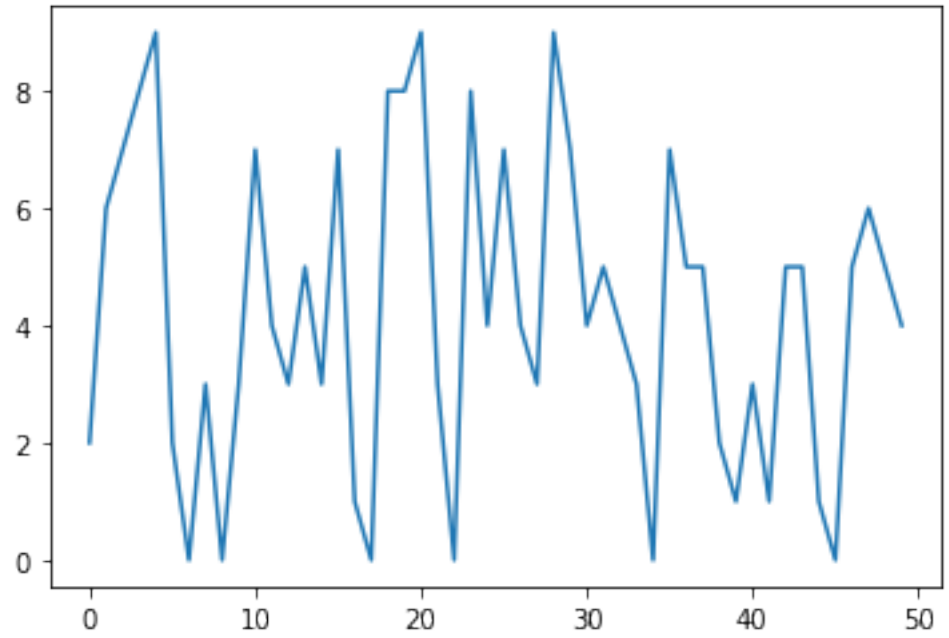
[1]:
```
import matplotlib.pyplot as plt
```

[2]:
```
## let's import numpy as well
import numpy as np
```

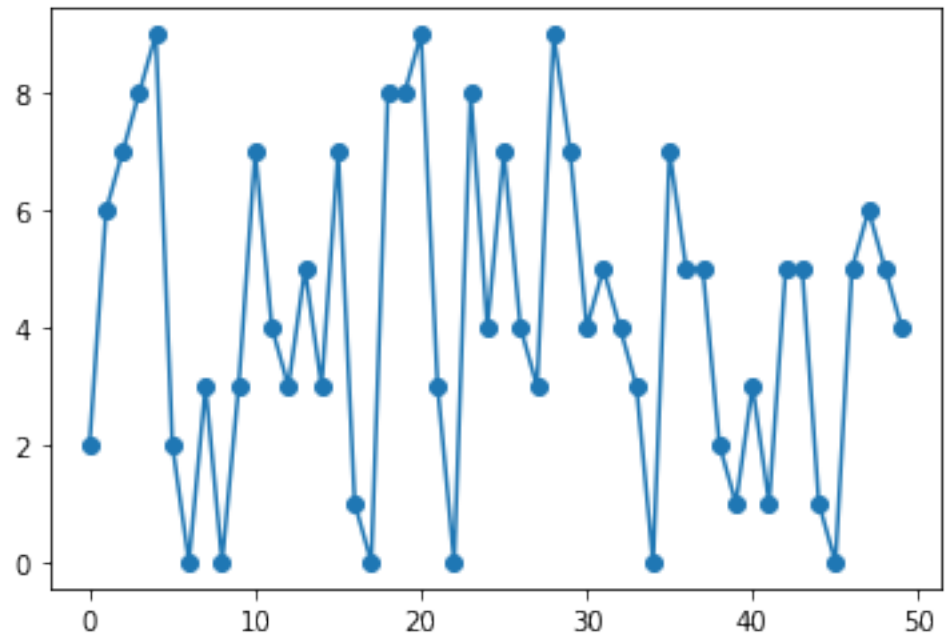
1.0.1 Line Plots

[25]:
```
## 50 random integers from 0 to 10
y = np.random.randint(0, 10, 50)
```

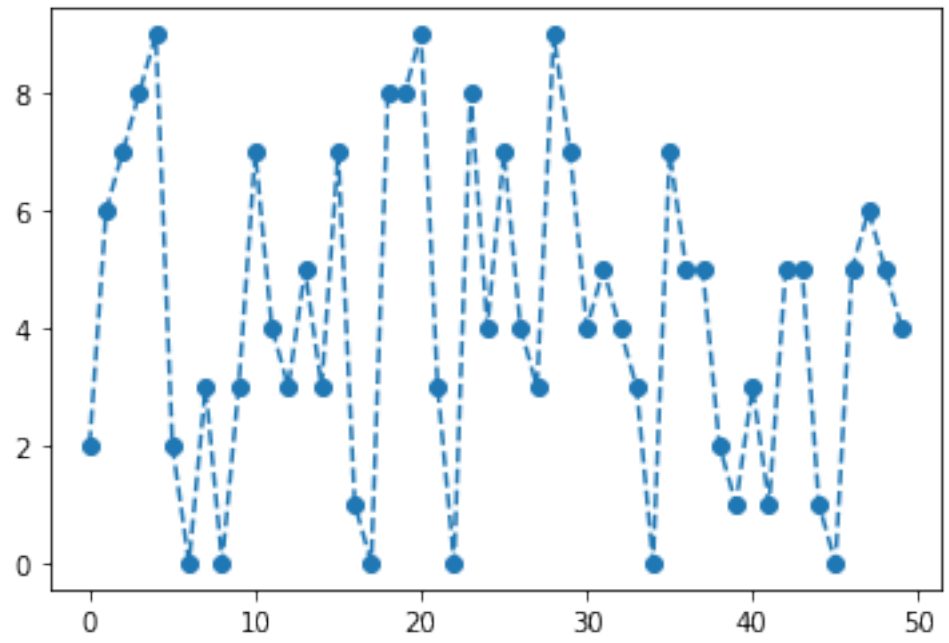
[26]:
```
plt.plot(y)
plt.show()  ## this is not completely necessary in Jupyter notebooks, but it␣
 ↪removes some background graph info
```



[27]:
```
## change markers
plt.plot(y, marker = 'o')
plt.show()
```

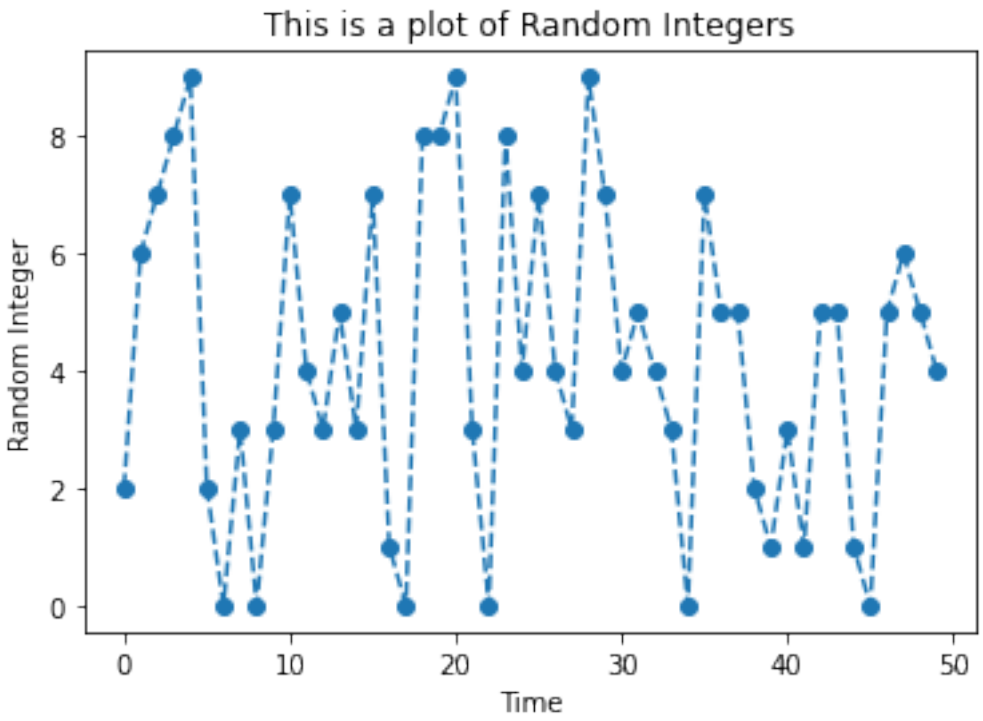


[28]:
```
## change markers and linestyle
plt.plot(y, marker = 'o', linestyle = '--')
plt.show()
```



You can add some other properties to the graphs. For example, you can modify the labels

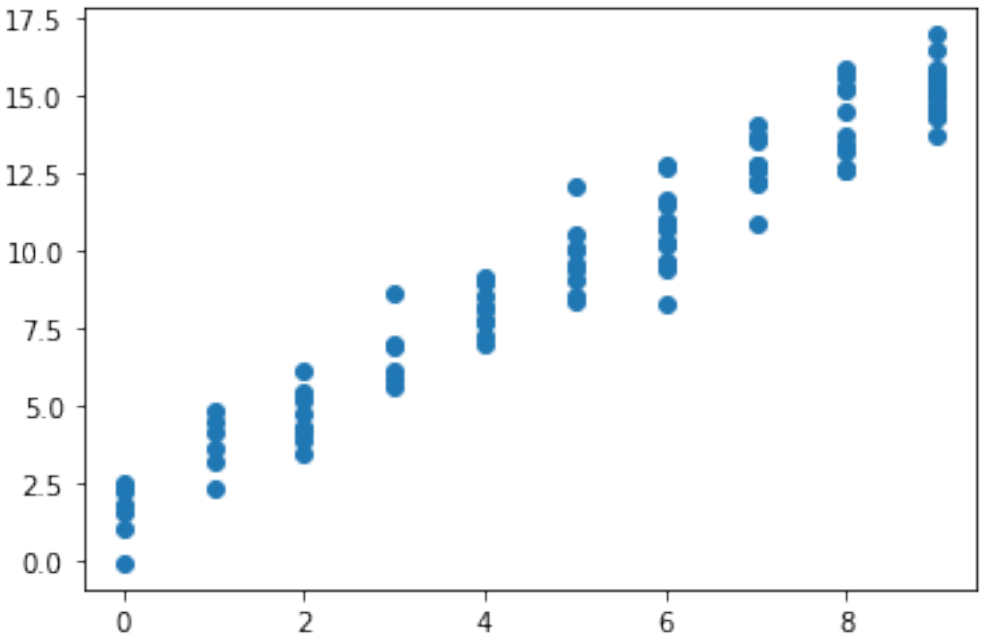
[29]:
```
plt.plot(y, marker = 'o', linestyle = '--')
plt.xlabel("Time")
plt.ylabel("Random Integer")
plt.title("This is a plot of Random Integers")
plt.show()
```



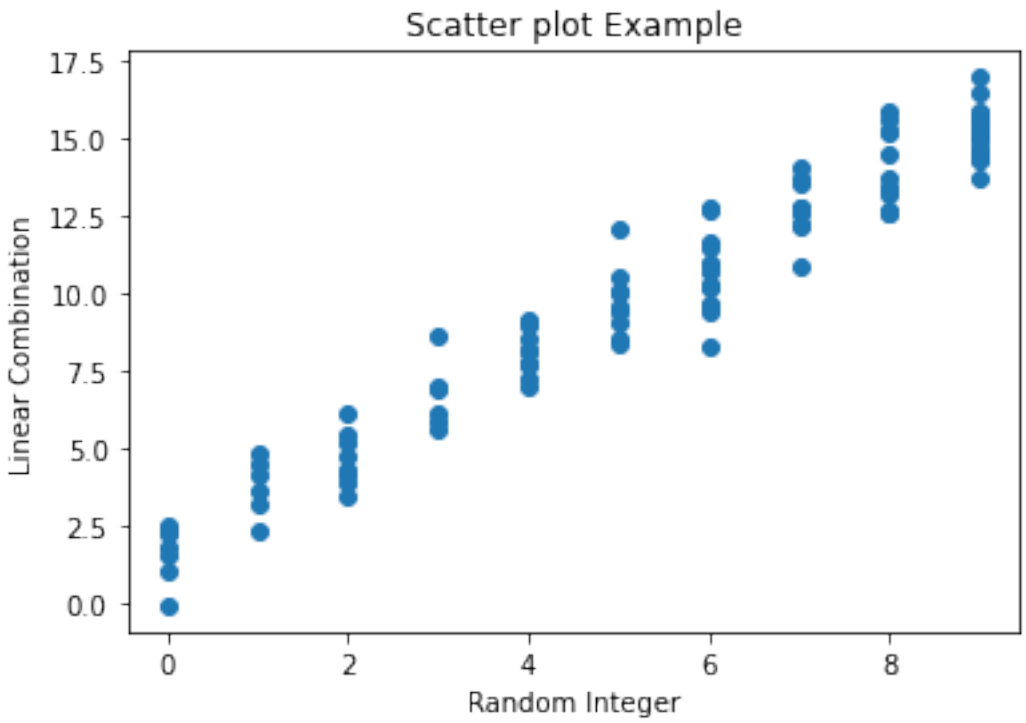
1.0.2 Scatter Plots

[30]:
```
## 50 random integers from 0 to 10
x = np.random.randint(0,10,100)
## let's create a linear combination of the x
## y-intercept 2
## slope 1.5
## some standard normal random error
y = 2 + 1.5*x + np.random.normal(0,1,100)
```

[34]:
```
## Scatter plot of x vs y
plt.scatter(x, y)
plt.show()
```

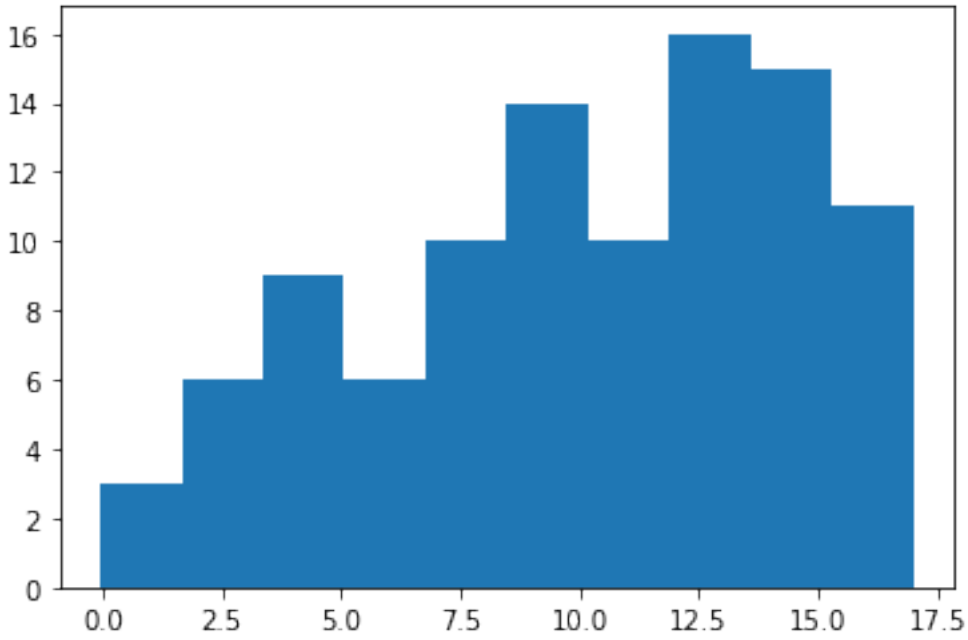


[35]:
```
## We can also add properties
plt.scatter(x, y)
plt.xlabel('Random Integer')
plt.ylabel("Linear Combination")
plt.title("Scatter plot Example")
plt.show()
```

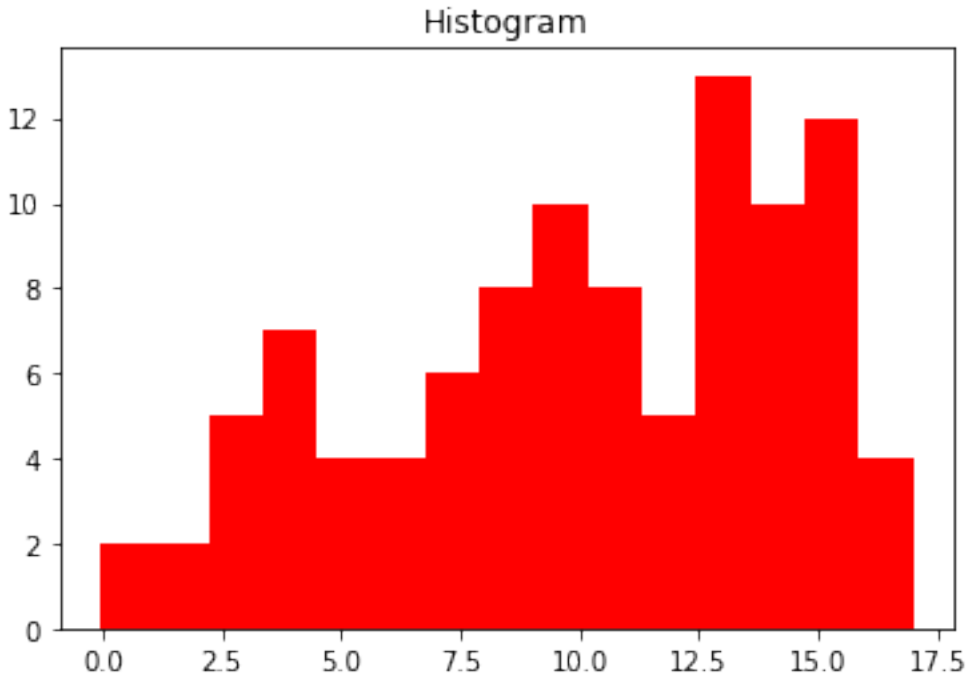


1.0.3 Histograms

[36]:
```
plt.hist(y)
plt.show()
```

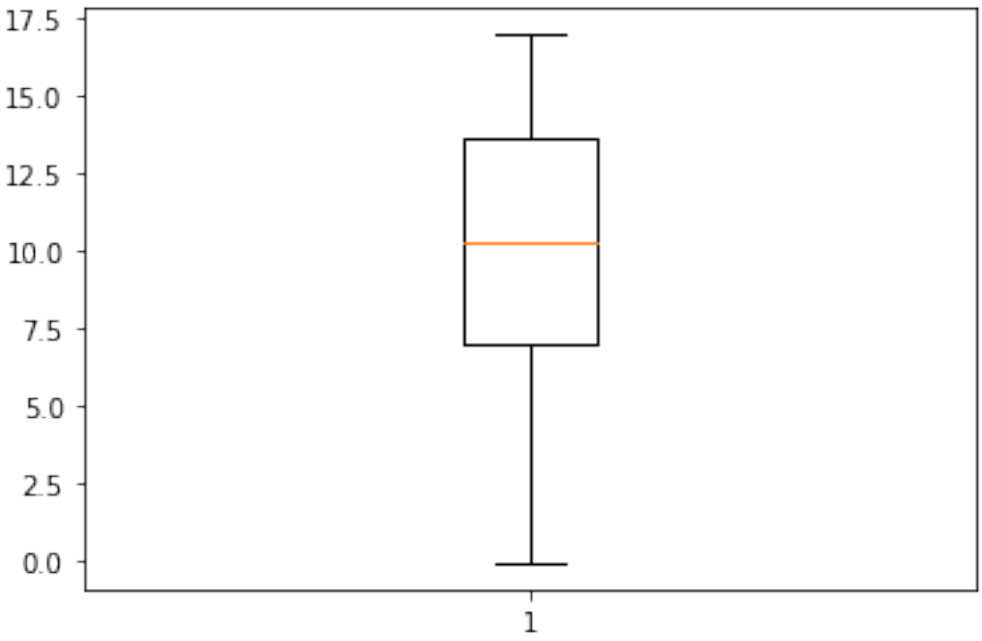


[41]:
```
plt.hist(y, bins = 15, color = 'r') ## change number of bins and color
plt.title("Histogram")
plt.show()
```

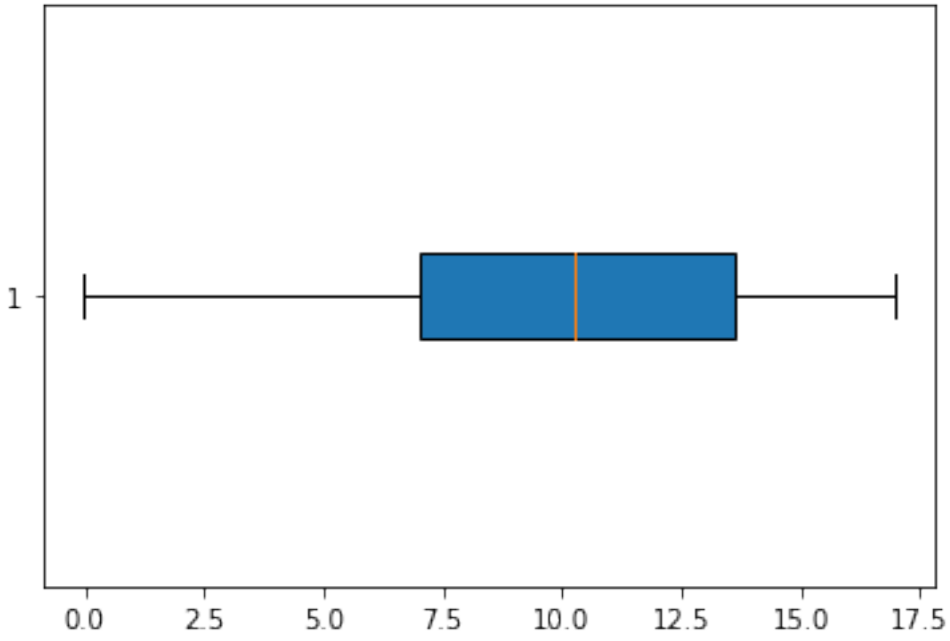


1.0.4 Box Plots

[44]:
```
plt.boxplot(y)
plt.show()
```



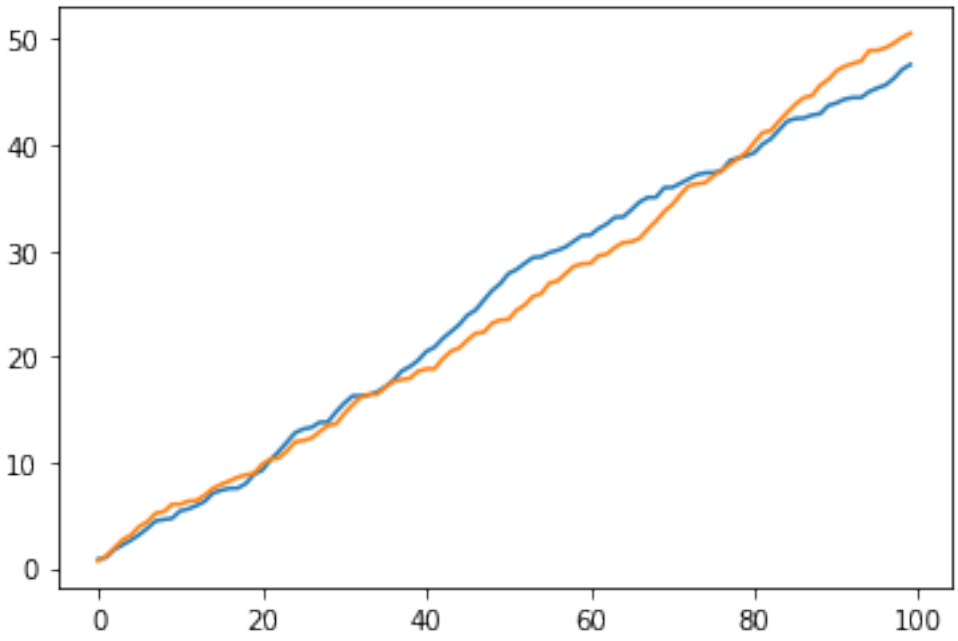
[46]:
```
plt.boxplot(y, vert = False, patch_artist=True) ## change options
plt.show()
```



1.0.5 Subplots

[51]:
```
x = np.random.random(100)
## let's create the cumulative sum of the variable
x = x.cumsum()

y = np.random.random(100)
## let's create the cumulative sum of the variable
y = y.cumsum()
```

[53]:
```
## you can use plt.plot twice to plot on top each other
plt.plot(x)
plt.plot(y)
plt.show()
```

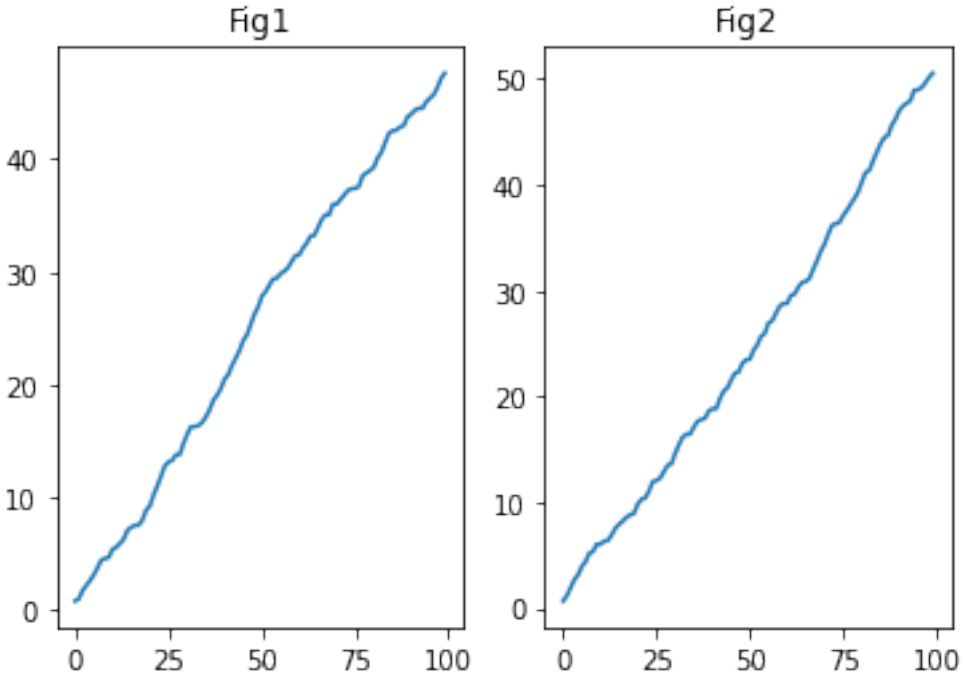


You can use the subplot function to produce subplots. The syntax is:

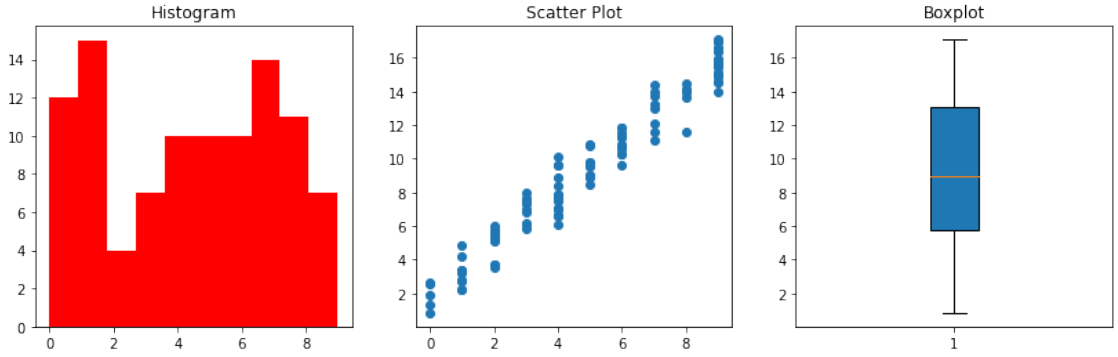
`plt.subplot(n_rows, n_cols, index_to_modify)`

[55]:
```
# 1 row, two cols
plt.subplot(1,2,1) # modify figure 1
plt.plot(x)
plt.title("Fig1")

plt.subplot(1,2,2) # modify figure 2
plt.plot(y)
plt.title("Fig2")
```

```
plt.show()
```



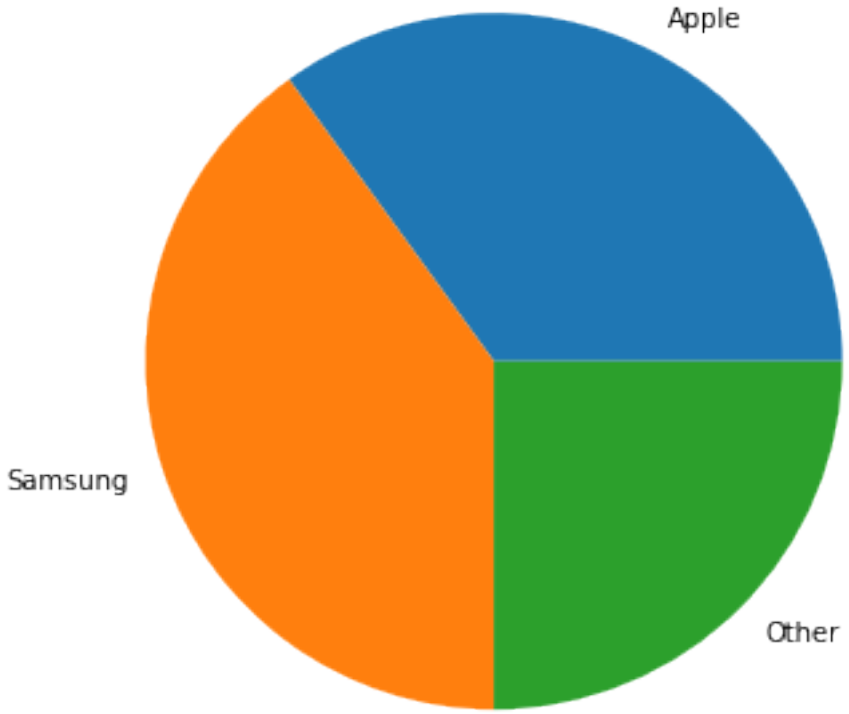
[69]:
```
## You can combine any type of plot
plt.figure(figsize=(14,4)) #change fig size

plt.subplot(1,3,1) # modify figure 1
plt.hist(x, color='r')
plt.title("Histogram")

plt.subplot(1,3,2) # modify figure 2
x = np.random.randint(0,10,100)
y = 2 + 1.5*x + np.random.normal(0,1,100)
plt.scatter(x, y)
plt.title("Scatter Plot")

plt.subplot(1,3,3) # modify figure 3
plt.boxplot(y, vert = True, patch_artist=True)
plt.title("Boxplot")

plt.show()
```



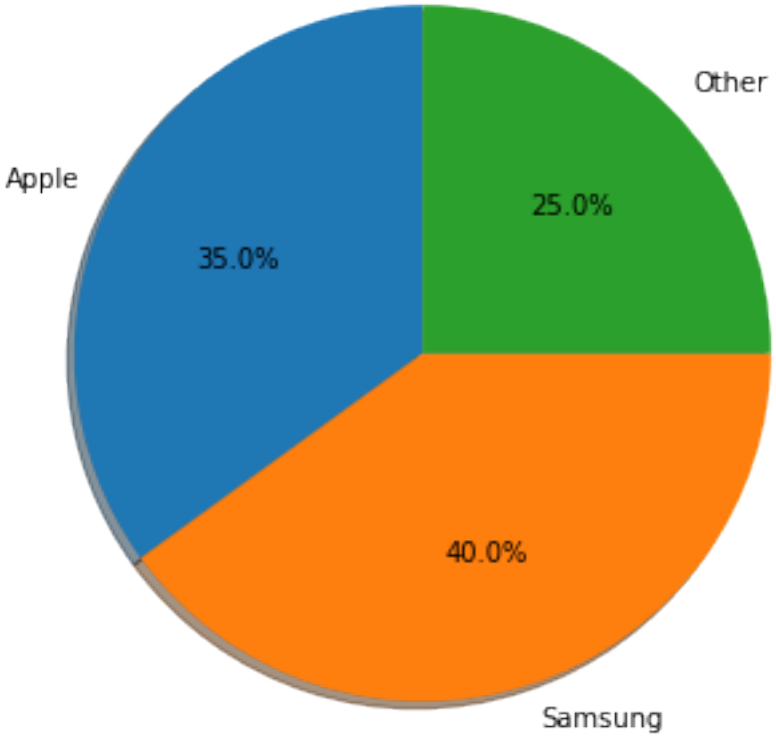
1.0.6 Pie Plots

[79]:
```
phones = ['Apple', 'Samsung', 'Other']
sizes = [35, 40, 25]
```

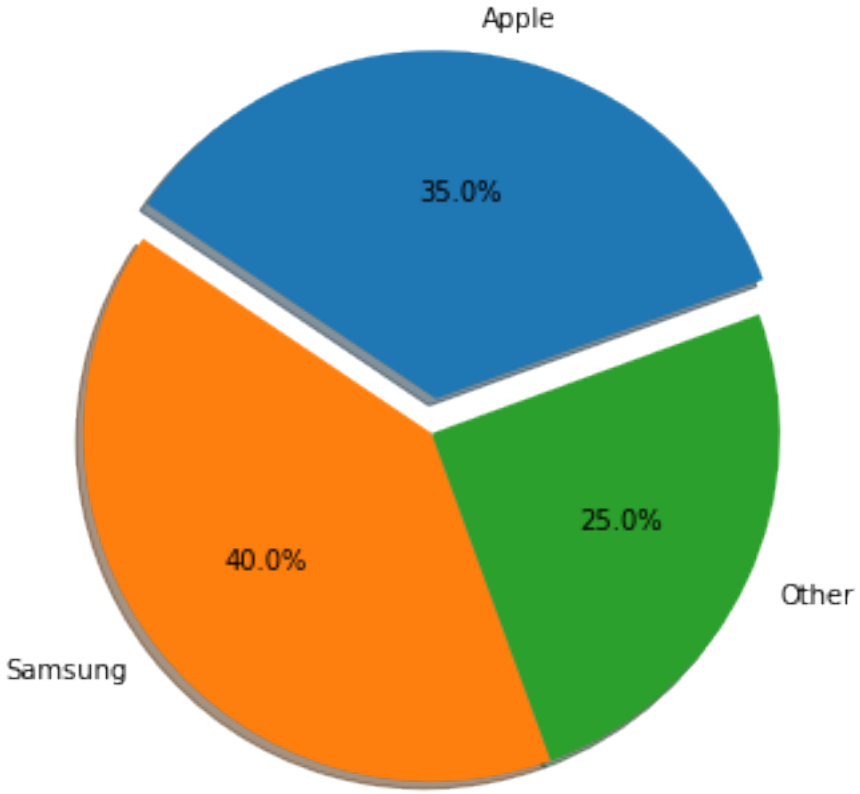
[82]:
```
plt.figure(figsize=(6,6)) #change fig size

plt.pie(sizes, labels = phones)
plt.show()
```



[84]:
```
plt.figure(figsize=(6,6)) #change fig size

## let's add some options to make it look better
plt.pie(sizes, labels = phones, shadow = True, autopct = '%1.1f%%',␣
↪startangle=90)
plt.show()
```



[87]:
```
plt.figure(figsize=(6,6)) #change fig size

## let's add explode values
exp_values = (0.1, 0, 0)

plt.pie(sizes, labels = phones, shadow = True,
        autopct = '%1.1f%%', startangle=20, explode = exp_values)
plt.show()
```


Apple
35.0%
25.0%
40.0%
Other
Samsung

[ ]: