Python-SciPy Library

Import Data

In [2]:

```
import pandas as pd
#importing data
data = pd.read_excel('somecars1.xlsx')
#print the dataset
data
print(data)
```

```
     mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  carb
0   21.0    6  160.0  110  3.90  2.620  16.46   0   1     4     4
1   21.0    6  160.0  110  3.90  2.875  17.02   0   1     4     4
2   22.8    4  108.0   93  3.85  2.320  18.61   1   1     4     1
3   21.4    6  258.0  110  3.08  3.215  19.44   1   0     3     1
4   18.7    8  360.0  175  3.15  3.440  17.02   0   0     3     2
5   14.3    8  360.0  245  3.21  3.570  15.84   0   0     3     4
6   24.4    4  146.7   62  3.69  3.190  20.00   1   0     4     2
7   22.8    4  140.8   95  3.92  3.150  22.90   1   0     4     2
8   19.2    6  167.6  123  3.92  3.440  18.30   1   0     4     4
9   17.8    6  167.6  123  3.92  3.440  18.90   1   0     4     4
10  16.4    8  275.8  180  3.07  4.070  17.40   0   0     3     3
11  17.3    8  275.8  180  3.07  3.730  17.60   0   0     3     3
12  15.2    8  275.8  180  3.07  3.780  18.00   0   0     3     3
13  10.4    8  472.0  205  2.93  5.250  17.98   0   0     3     4
14  10.4    8  460.0  215  3.00  5.424  17.82   0   0     3     4
15  14.7    8  440.0  230  3.23  5.345  17.42   0   0     3     4
16  30.4    4   75.7   52  4.93  1.615  18.52   1   1     4     2
17  33.9    4   71.1   65  4.22  1.835  19.90   1   1     4     1
18  21.5    4  120.1   97  3.70  2.465  20.01   1   0     3     1
19  15.5    8  318.0  150  2.76  3.520  16.87   0   0     3     2
20  15.2    8  304.0  150  3.15  3.435  17.30   0   0     3     2
21  13.3    8  350.0  245  3.73  3.840  15.41   0   0     3     4
22  19.2    8  400.0  175  3.08  3.845  17.05   0   0     3     2
23  26.0    4  120.3   91  4.43  2.140  16.70   0   1     5     2
24  30.4    4   95.1  113  3.77  1.513  16.90   1   1     5     2
25  15.8    8  351.0  264  4.22  3.170  14.50   0   1     5     4
26  19.7    6  145.0  175  3.62  2.770  15.50   0   1     5     6
27  15.0    8  301.0  335  3.54  3.570  14.60   0   1     5     8
28  21.4    4  121.0  109  4.11  2.780  18.60   1   1     4     2
```

In [ ]:

```

```

scipy.cluster

Clustering algorithms are useful in information theory, target detection, communications, compression, and other areas. The vq module only supports vector quantization and the k-means algorithms

To divide a dataset into k clusters.

In [3]:

```
#import libraries
import pandas  as pd
from  scipy.cluster.vq import kmeans, vq
#importing data
data = pd.read_excel('somecars1.xlsx')
#find out centroids with the help of kmeans functions
#k, number of clusters required
centroid, _ = kmeans(data,3)
#_ It's just a variable name, and it's conventional in python to use _ for throwaway variab
#It just indicates that _ isn't actually used.

#find out the cluster index for each record with vector quantization function
#vq(data,centroid)
idx, _ = vq(data,centroid)
#print the cluster index array
print("idx:",idx)
#below there are 3 clusters (0,1,2)
#centroid, _ = kmeans(data,3) we can change the value of 3 to 2
#print (data)
```

```
idx: [1 1 1 0 2 2 1 1 1 1 0 0 0 2 2 2 1 1 1 0 0 2 2 1 1 2 1 2 1]
```

In [4]:

```
#also print the centroids for each of the 11 columns and there are K=3 values.
centroid
```

Out[4]:

```
array([[  1.68333333e+01,   7.66666667e+00,   2.84566667e+02,
          1.58333333e+02,   3.03333333e+00,   3.62500000e+00,
          1.77683333e+01,   1.66666667e-01,   0.00000000e+00,
          3.00000000e+00,   2.33333333e+00],
       [  2.37357143e+01,   4.71428571e+00,   1.28500000e+02,
          1.01285714e+02,   3.99142857e+00,   2.58235714e+00,
          1.84514286e+01,   7.14285714e-01,   6.42857143e-01,
          4.14285714e+00,   2.64285714e+00],
       [  1.46444444e+01,   8.00000000e+00,   3.88222222e+02,
          2.32111111e+02,   3.34333333e+00,   4.16155556e+00,
          1.64044444e+01,   0.00000000e+00,   2.22222222e-01,
          3.44444444e+00,   4.00000000e+00]])
```

```
Now perform data whitening
```

whiten. Normalize a group of observations on a per feature basis. Before running k-means, it is beneficial to rescale each feature dimension of the observation set with whitening. Each feature is divided by its standard deviation across all observations to give it unit variance.

In [7]:

```
#import libraries
import pandas  as pd
from  scipy.cluster.vq import kmeans, vq, whiten
#importing data
data = pd.read_excel('somecars1.xlsx')
#whiten data
data = whiten(data)
print('whiten data')
print(data)
#find out centroids with the help of kmeans functions
#k, number of clusters required
centroid, _ = kmeans(data,3)
#find out the cluster index for each record with vector quantization function
#vq(data,centroid)
idx, _ = vq(data,centroid)
#print the cluster index array
idx
```

```
whiten data
[[  3.75298379   3.43361612   1.31900894   1.64580862   7.57136329
    2.72567145   9.40878577   0.           2.06094026   5.35068101
    2.57460122]
 [  3.75298379   3.43361612   1.31900894   1.64580862   7.57136329
    2.99095627   9.72889026   0.           2.06094026   5.35068101
    2.57460122]
 [  4.07466812   2.28907741   0.89033103   1.39145638   7.47429453
    2.41357167  10.63775839   2.06094026   2.06094026   5.35068101
    0.6436503 ]
 [  3.8244692    3.43361612   2.12690191   1.64580862   5.97943563
    3.34466936  11.11219898   2.06094026   0.           4.01301075
    0.6436503 ]
 [  3.34194271   4.57815482   2.9677701    2.6183319    6.11533189
    3.5787442    9.72889026   0.           0.           4.01301075
    1.28730061]
 [  2.55560325   4.57815482   2.9677701    3.66566466   6.2318144
    3.71398744   9.05438436   0.           0.           4.01301075
    2.57460122]
 [  4.36060974   2.28907741   1.20936632   0.92763759   7.1636745
    3.31866104  11.43230348   2.06094026   0.           5.35068101
    1.28730061]
 [  4.07466812   2.28907741   1.16072786   1.42138017   7.6101908
    3.27704774  13.08998749   2.06094026   0.           5.35068101
    1.28730061]
 [  3.43129947   3.43361612   1.38166186   1.84031328   7.6101908
    3.5787442   10.46055769   2.06094026   0.           5.35068101
    2.57460122]
 [  3.18110055   3.43361612   1.38166186   1.84031328   7.6101908
    3.5787442   10.80352679   2.06094026   0.           5.35068101
    2.57460122]
 [  2.93090163   4.57815482   2.27364165   2.69314138   5.96002187
    4.23415375   9.94610403   0.           0.           4.01301075
    1.93095091]
 [  3.09174379   4.57815482   2.27364165   2.69314138   5.96002187
    3.88044066  10.06042706   0.           0.           4.01301075
    1.93095091]
 [  2.71644541   4.57815482   2.27364165   2.69314138   5.96002187
    3.93245729  10.28907313   0.           0.           4.01301075
    1.93095091]
```

```
[  1.85862054   4.57815482   3.89107636   3.0671888    5.68822935
   5.46174623  10.27764083   0.           0.           4.01301075
   2.57460122]
[  1.85862054   4.57815482   3.79215069   3.21680776   5.82412561
   5.64276411  10.1861824    0.           0.           4.01301075
   2.57460122]
[  2.62708865   4.57815482   3.62727457   3.44123621   6.27064191
   5.56057783   9.95753633   0.           0.           4.01301075
   2.57460122]
[  5.43289082   2.28907741   0.6240561    0.77801862   9.57097975
   1.68013717  10.58631302   2.06094026   2.06094026   5.35068101
   1.28730061]
[  6.05838812   2.28907741   0.5861346    0.97252328   8.19260336
   1.90901035  11.37514196   2.06094026   2.06094026   5.35068101
   0.6436503 ]
[  3.84234055   2.28907741   0.99008108   1.45130397   7.18308825
   2.5644199   11.43801963   2.06094026   0.           4.01301075
   0.6436503 ]
[  2.77005947   4.57815482   2.62153026   2.24428449   5.35819556
   3.66197081   9.64314799   0.           0.           4.01301075
   1.28730061]
[  2.71644541   4.57815482   2.50611698   2.24428449   6.11533189
   3.57354254   9.88894251   0.           0.           4.01301075
   1.28730061]
[  2.37688974   4.57815482   2.88533205   3.66566466   7.24132951
   3.99487724   8.80858983   0.           0.           4.01301075
   2.57460122]
[  3.43129947   4.57815482   3.29752234   2.6183319    5.97943563
   4.00007891   9.74603872   0.           0.           4.01301075
   1.28730061]
[  4.64655136   2.28907741   0.99172984   1.36153259   8.60029215
   2.2263118    9.54597341   0.           2.06094026   6.68835126
   1.28730061]
[  5.43289082   2.28907741   0.78398594   1.69069431   7.31898452
   1.57402325   9.66029644   2.06094026   2.06094026   6.68835126
   1.28730061]
[  2.82367352   4.57815482   2.89357585   3.9499407    8.19260336
   3.29785439   8.28842002   0.           2.06094026   6.68835126
   2.57460122]
[  3.52065622   3.43361612   1.19535185   2.6183319    7.02777824
   2.88172135   8.8600352    0.           2.06094026   6.68835126
   3.86190183]
[  2.68070271   4.57815482   2.48138556   5.01223535   6.87246822
   3.71398744   8.34558154   0.           2.06094026   6.68835126
   5.14920243]
[  3.8244692    2.28907741   0.99750051   1.63084673   7.97905208
   2.89212467  10.63204224   2.06094026   2.06094026   5.35068101
   1.28730061]]
```

Out[7]:

```
array([2, 2, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       2, 0, 2, 2, 2, 0])
```

In [6]:

```
#also print the centroids
centroid
```

Out[6]:

```
array([[  3.5295919 ,   3.62437257,   1.70001016,   2.70560963,
          7.63931142,   2.97275045,   9.02961437,   0.        ,
          2.06094026,   6.24246117,   3.00370142],
       [  2.68963838,   4.57815482,   2.94812237,   2.90510158,
          6.05870845,   4.26961175,   9.79891312,   0.        ,
          0.        ,   4.01301075,   1.98458844],
       [  4.3216177 ,   2.60122433,   1.10294628,   1.41729966,
          7.60842591,   2.73919578,  11.02074056,   2.06094026,
          0.93679103,   5.22907462,   1.28730061]])
```

scipy.stats

find the mean and standard deviation

In [ ]:

```

```

%matplotlib is a magic function in IPython. ... %matplotlib inline sets the backend of matplotlib to the 'inline' backend: With this backend, the output of plotting commands is displayed inline within frontends like the Jupyter notebook, directly below the code cell that produced it.

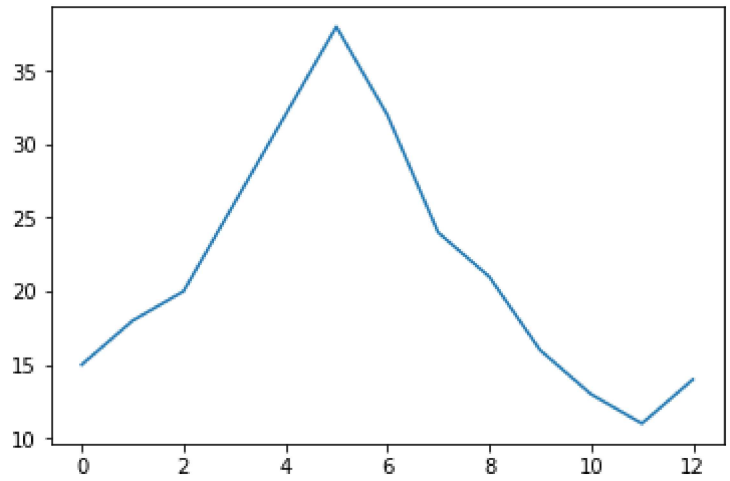
In [17]:

```
#import numpy

import numpy as np
#create the marks array
coffee = np.array([15,18,20,26,32,38,32,24,21,16,13,11,14])
print(coffee.mean(), coffee.std())
#let us see the data distribution by plotting it

import matplotlib.pyplot as pyplot
%matplotlib inline

pyplot.plot(range(13),coffee)
#plt.plot(range(13),coffee)
```

```
21.5384615385 8.06335857316
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x1bd0fd401d0>]
```




```
Find the z-score
```

Z-scores are a way to compare results to a “normal” population. Results from tests or surveys have thousands of possible results and units; those results can often seem meaningless. For example, knowing that someone’s weight is 150 pounds might be good information, but if you want to compare it to the “average” person’s weight, looking at a vast table of data can be overwhelming (especially if some weights are recorded in kilograms). A z-score can tell you where that person’s weight is compared to the average population’s mean weight.

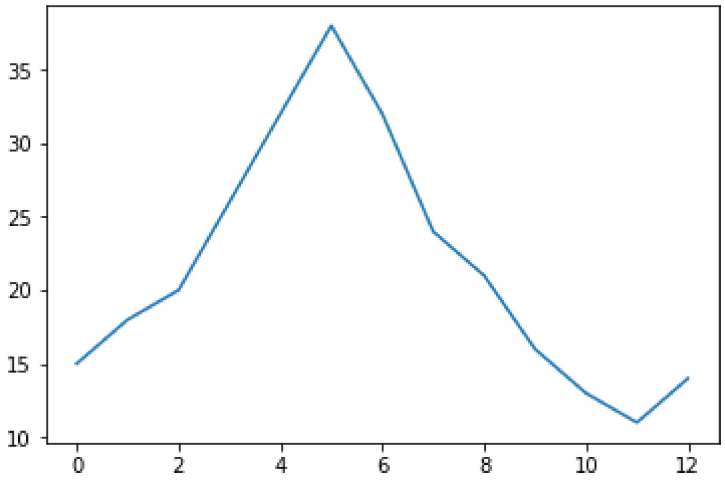
In [19]:

```
#import numpy
import numpy as np
#create the marks array
coffee = np.array([15,18,20,26,32,38,32,24,21,16,13,11,14])
from scipy import stats
#find the zscore
print(stats.zscore(coffee))
print(coffee.mean(), coffee.std())
#let us see the data distribution by plotting it
import matplotlib.pyplot as pyplot
pyplot.plot(range(13),coffee)
#plt.plot(range(13),coffee)
```

```
[-0.81088562 -0.43883222 -0.19079662  0.55331019  1.297417    2.0415238
  1.297417    0.30527459 -0.06677882 -0.68686782 -1.05892123 -1.30695683
 -0.93490342]
21.5384615385 8.06335857316
```

Out[19]:

```
[<matplotlib.lines.Line2D at 0x1bd10b1d5f8>]
```



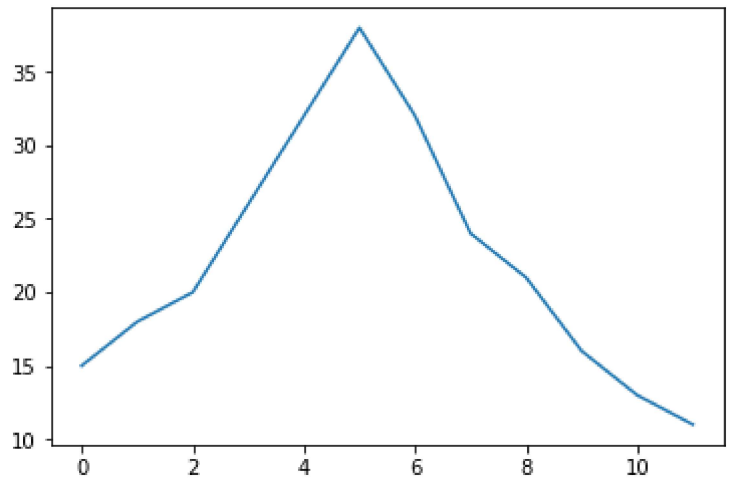
In [21]:

```
#import numpy
import numpy as np
#create the marks array
coffee = np.array([15,18,20,26,32,38,32,24,21,16,13,11])
#import scipy stats
from scipy import stats
#find the zscore
print(coffee.mean(), coffee.std())
#let us see the data distribution by plotting it
import matplotlib.pyplot as pyplot
pyplot.plot(range(12),coffee)
```

```
22.1666666667 8.0811852816
```

Out[21]:

```
[<matplotlib.lines.Line2D at 0x1bd104c1358>]
```



In [22]:

```
#import numpy
import numpy as np
from scipy import stats
#create the numpy array consisting of frequency of people going to gym and frequency of smo
obs = np.array([[7,1,3],[87,18,84],[12,3,4],[9,1,7]])
#since we are lookingfor only p values, ignore the rest
_,p,_,_ = stats.chi2_contingency(obs)
#print p
p
```

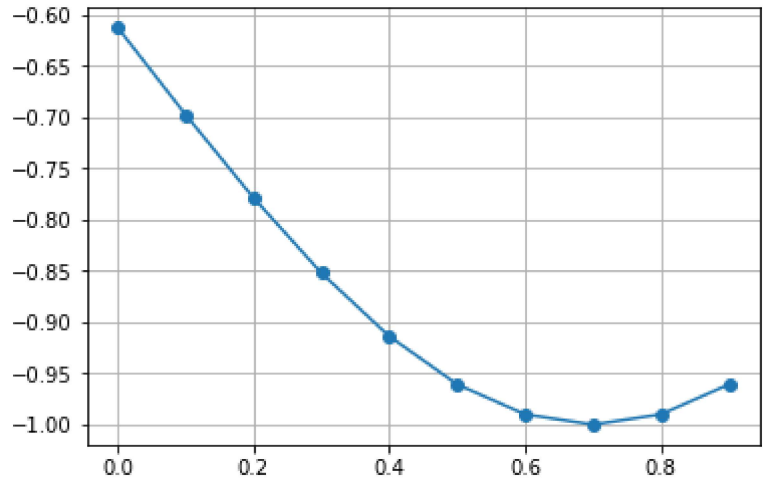
Out[22]:

```
0.48284216946545633
```

scipy.optimize

In [23]:

```
#generate one function and plot with matplotlib
#import matplotlib
import matplotlib.pyplot as plt
#import numpy
import numpy as np
x= np.arange(0.0,1.0,0.1)
#create function
def f(x):
    return -np.exp(-(x-0.7)**2)
#plot function
plt.plot(x,f(x),'o-')
plt.grid()
```



In [24]:

```
#find at which x value we get the minimum function
from scipy import optimize
#generating the function
import numpy as np
def f(x):
    return -np.exp(-(x-0.7)**2)
#find the minimum of the function
result = optimize.minimize_scalar(f)
#now find the corresponding x value
x_min = result.x
#print the x value
x_min
```

Out[24]:

```
0.6999999997839409
```

scipy.integrate

In [25]:

```
#import scipy integrate
import scipy.integrate as intg
#create one function to find the integration
def integrad(x):
    return x**2
#apply quad() function, get only the answer, ignore rest
ans,_ = intg.quad(integrad,0,1)
#print ans
ans
```

Out[25]:

```
0.33333333333333337
```

scipy.linalg

Determinant of a square matrix

In [26]:

```
#import scipy linalg package
from scipy import linalg
#import numpy to the square matrix
import numpy as np
data = np.array([[1,2,3],[3,4,5],[5,6,7]])
#find determinant
linalg.det(data)
```

Out[26]:

```
-1.1842378929335004e-15
```

Inverse of a square matrix

In [27]:

```
#import scipy linalg package
from scipy import linalg
#import numpy to the square matrix
import numpy as np
data = np.array([[1,2,3],[3,4,5],[5,6,7]])
#find determinant
linalg.inv(data)
```

Out[27]:

```
array([[ -1.18515780e+15,   2.37031559e+15,  -1.18515780e+15],
       [  2.37031559e+15,  -4.74063119e+15,   2.37031559e+15],
       [ -1.18515780e+15,   2.37031559e+15,  -1.18515780e+15]])
```

Eigen values of a square matrix

In [28]:

```
#import scipy linalg package
from scipy import linalg
#import numpy to the square matrix
import numpy as np
data = np.array([[1,2,3],[3,4,5],[5,6,7]])
#find determinant
linalg.eigvals(data)
```

Out[28]:

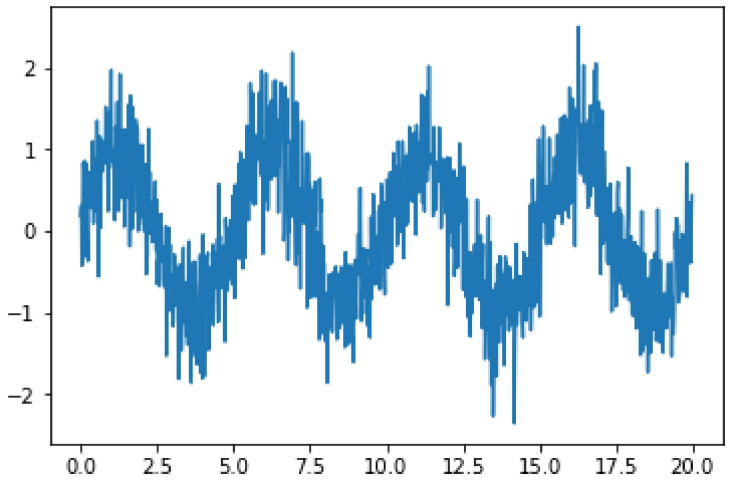
```
array([  1.29282032e+01+0.j,  -9.28203230e-01+0.j,   6.16237757e-16+0.j])
```

scipy.fftpack

Create one noisy signal

In [29]:

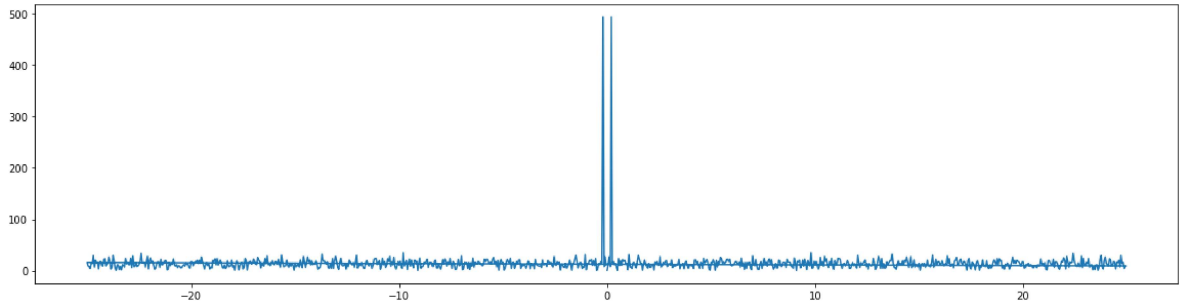
```
#craete noisy signal
import matplotlib.pyplot as plt
import numpy as np
#create a signal with time_step=0.02
time_step = 0.02
period = 5
time_vec= np.arange(0,20, time_step)
sig = np.sin(2*np.pi/period*time_vec)+ 0.5*np.random.randn(time_vec.size)
plt.plot(time_vec,sig)
plt.show()
```



Apply fft

In [30]:

```
from scipy import fftpack
#Since we didnt not know the signal frequency, we only knew the sampling time step of the s
#The function fftfreq() returns the FFT sample frequency points.
sample_freq = fftpack.fftfreq(sig.size, d = time_step)
#now apply the fft() in the signal to find the discrete fourier transform
sig_fft = fftpack.fft(sig)
#Calculate the absolute value element-wise
power = np.abs(sig_fft)
plt.figure(figsize=(20,5))
#plot the absolute values of each sample_freq
plt.plot(sample_freq, power)
plt.show()
#here at sample_freq = 0.2 and -0.2 we have absolute values of 5.15943859e+02 = 515.9438593
#print(sample_freq)
#print(power)
```

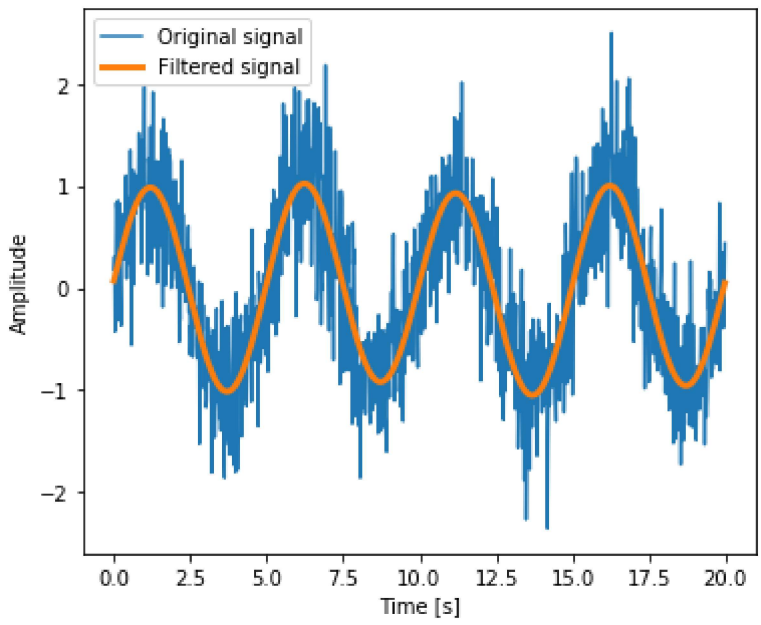


Apply inverse fft to get the filtered signal

In [31]:

```
#Filter out the sample frequencies that are greater than 0 with numpy.where(condition)
pos_mask = np.where(sample_freq > 0)
#Apply the fiter on smaple_freq and store the +ve sample_freq on freqs
freqs = sample_freq[pos_mask]

#print(power[pos_mask].argmax())
#Find the peak frequency, here we focus on only the positive frequencies
peak_freq = freqs[power[pos_mask].argmax()]
#now get an array copy of the signal where we already applied fft.
high_freq_fft = sig_fft.copy()
#assign the ones greater than peak freq as 0 in order to remove the noise
high_freq_fft[np.abs(sample_freq) > peak_freq] = 0

#print(high_freq_fft)
#Now apply inverese fft on the new high_freq_fft this will be the filtered signal
filtered_sig = fftpack.ifft(high_freq_fft)
#plot
plt.figure(figsize=(6, 5))
#now plot the original signal for reference
plt.plot(time_vec, sig, label='Original signal')
#now plot the filtered signal
plt.plot(time_vec, filtered_sig, linewidth=3, label='Filtered signal')
#add label, legend
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend(loc='best')
#show
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:531: Comple
xWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

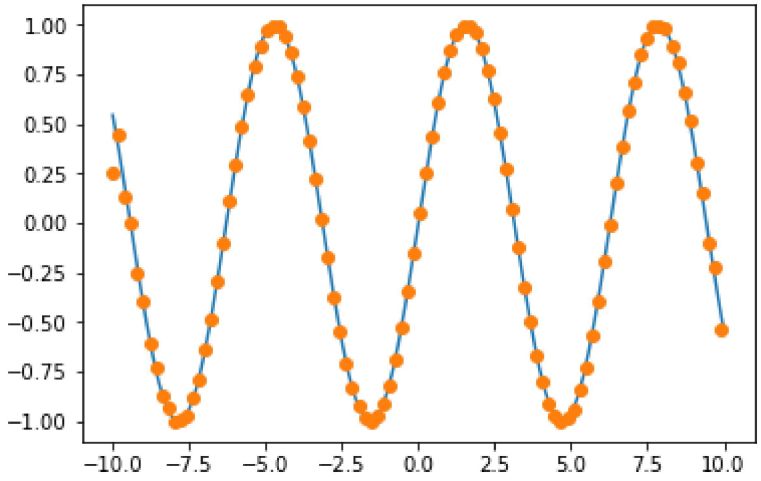


scipy.signal

Resampling

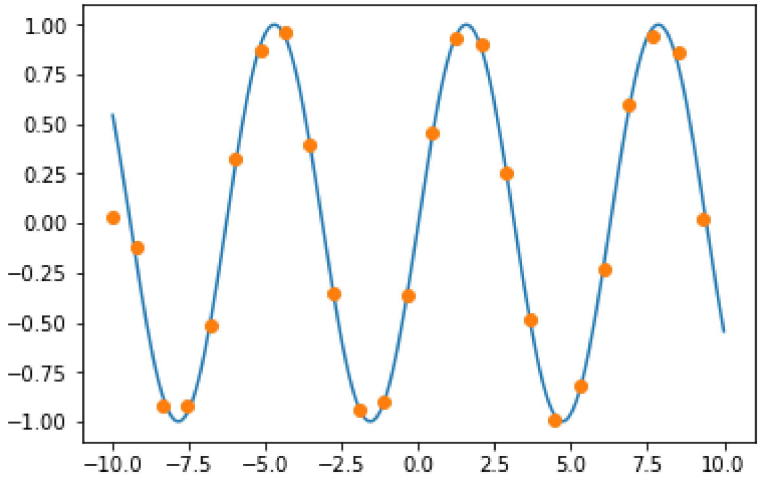
In [34]:

```
#scipy.signal uses FFT to resample a 1D signal.
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
#Now let us create a signal with 200 data point
t = np.linspace(-10, 10, 200) #Defining Time Interval
y = np.sin(t)
x_resampled=signal.resample(y, 100) #Number of required samples is 100
plt.plot(t, y)
#for x axis slice t into 2 step size
plt.plot(t[::2], x_resampled, 'o')
plt.show()
```



In [33]:

```
import numpy as np
t = np.linspace(-10, 10, 200)
x = np.sin(t)
from scipy import signal
x_resampled = signal.resample(x, 25) # Number of required samples is 25
plt.plot(t, x)
plt.plot(t[::8], x_resampled, 'o')
plt.show()
```



In [ ]:

```
```