

Homework -07

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Section 1: Every waveform can be represented as a combination of Sin Waves:

Let's us create other waveforms from sin wave combinations

f_s is the sampling frequency, while f is a base frequency for the signal content. We create a signal that contains components at a couple of multiples of this base frequency. Note the amplitudes here since we will be trying to extract those correctly from the FFT later.

In []:

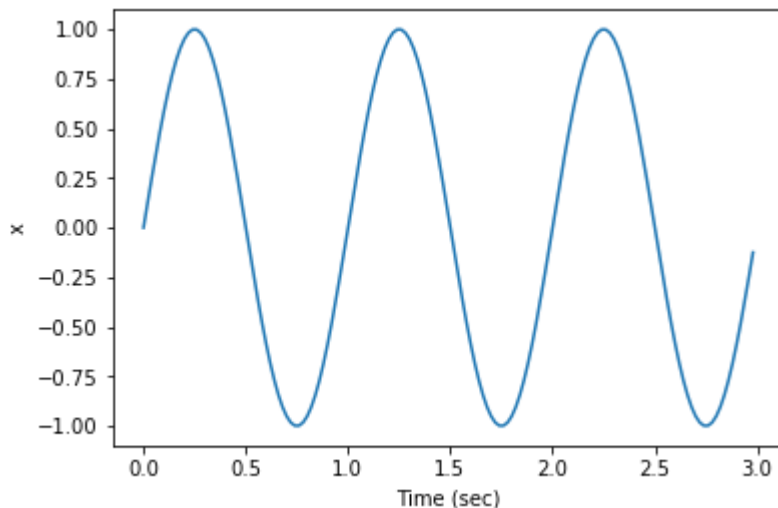
In [4]:

```
# Our pure sin waveform

f_s = 50.0 # Hz
f = 1.0 # Hz
time = np.arange(0.0, 3.0, 1/f_s) # sample 50 samples each second so for 3 second.
x1 = np.sin(2 * np.pi * f * time)
print('Signal size: ',x1.size)

plt.plot(time, x1)
plt.xlabel("Time (sec)")
plt.ylabel("x")
plt.show()
```

Signal size: 150



```
In [5]: # Add more sine waves to the earlier wave with different amplitude and phase
plt.figure(figsize=(10,10))

plt.subplot(221)
plt.plot(time, x1,c='blue')
plt.xlabel("Time (sec)")
plt.ylabel("x")
plt.title('x1')

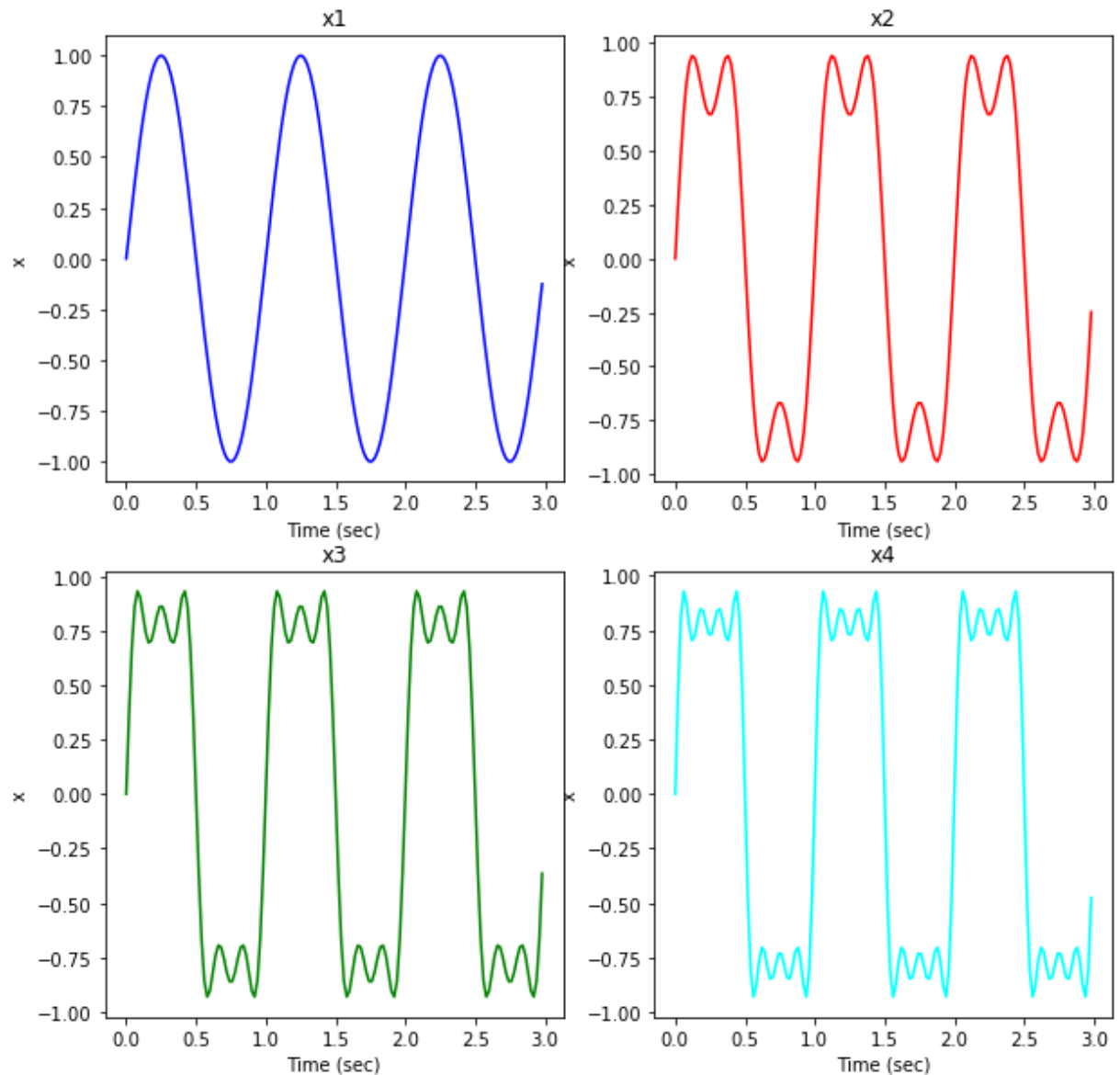
# Add sin wave of different amplitude and phase to wave x1

x2= x1 + np.sin(3*2 * np.pi * f * time)/3# this has 2 frequencies
plt.subplot(222)
plt.plot(time, x2,c='red')
plt.xlabel("Time (sec)")
plt.ylabel("x")
plt.title('x2')

plt.subplot(223)
x3= x2 + np.sin(5*2 * np.pi * f * time)/5 #this has 3 frequencies - 5
plt.plot(time, x3,c='green')
plt.xlabel("Time (sec)")
plt.ylabel("x")
plt.title('x3')

plt.subplot(224)
x4= x3 + np.sin(7*2 * np.pi * f * time)/7 # this has 4 frequencies - 10
plt.plot(time, x4,c='aqua')
plt.xlabel("Time (sec)")
plt.ylabel("x")
plt.title('x4')
plt.suptitle('OBSERVE THE TRANSFORMATION OF SIN WAVES INTO FLAT PEAK WAVES')
plt.show()
```

OBSERVE THE TRANSFORMATION OF SIN WAVES INTO FLAT PEAK WAVES



Let us use FFT transformation on the signals to convert them to frequency domain, *this will give us information about the composition of our wave signals*. I can use a frequency filter on my signal to retrieve a portion of it. Lets us convert x4 signal to FFT and retrieve frequency plot for frequencies greater than 2 Hz.

```
In [6]: def get_fft(x,f_s=50):
    ...
    Function to convert raw timeseries signal to Fourier domain
    x:raw_signal
    f_s=sampling frequency if f_s=50 means sample 50 points per second from the t

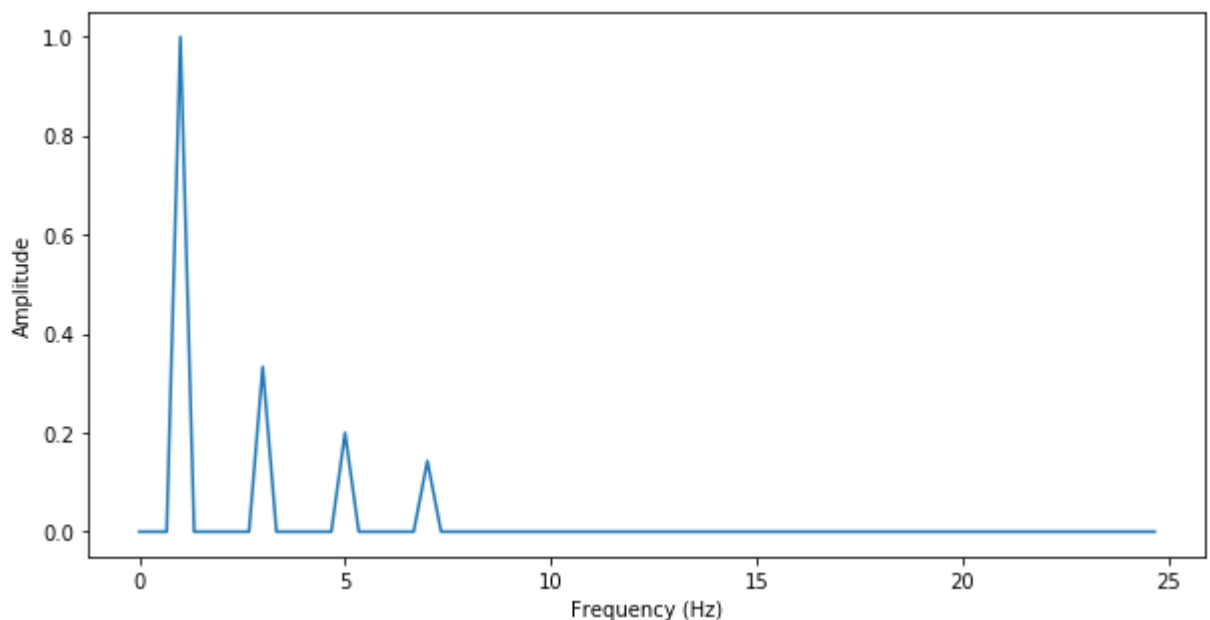
    plots a folded and scaled frequency plot of the signal
    and returns the frequency spectrum dataframe of the signal
    ...

    n = len(x)
    fft_x = np.fft.fft(x) # gives the complex representation of each point

    # Find all the frequencies in the FFT
    freqs = np.fft.fftfreq(n, d=1/f_s)

    # Plot the folded and scaled frequency plot
    half_n = int(np.ceil(n/2.0))
    fft_x_half = (2.0 / n) * fft_x[:half_n]
    fft_x_half=np.abs(fft_x_half)
    freq_half = freqs[:half_n]
    plt.figure(figsize=(10,5))
    plt.plot(freq_half, fft_x_half)
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude")
    plt.show()
    spectrum = pd.DataFrame({'frequency (Hz)': freq_half,'magnitude': fft_x_half,
    return spectrum
```

```
In [7]: # plot a frequency spectrum of x4 signal
s=get_fft(x4,f_s=50)
```



```
In [8]: # Lets filter some frequency ranges from the signal
s_filtered=s[s['frequency (Hz)']>4]
s_filtered.plot(x='frequency (Hz)',figsize=(10,5))
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1824de06940>
```

TODO

Q1: TODO: Now plot the frequency spectrum of signals x1, x2 ,x3 created above, what do you observe? How many frequency peaks can you see in each signal. Does it correspond to the number of sin waves that were used to create the signals, enlist the number of sine waves used in creation of each signal and the frequency peaks that you can see in their plot.

Observe x3 plot: Why are amplitudes different for each frequency? What is the ratio of amplitudes of different frequencies?

In [9]: *#The amplitudes of each frequency are relative to how much we divide them by when
 #So in the first case it was 1:1/3:1/5:1/10.
 #In this case it is 1:1/3:1/5 or 1 : 0.3333 : 0.1 if we look inside s*
 s

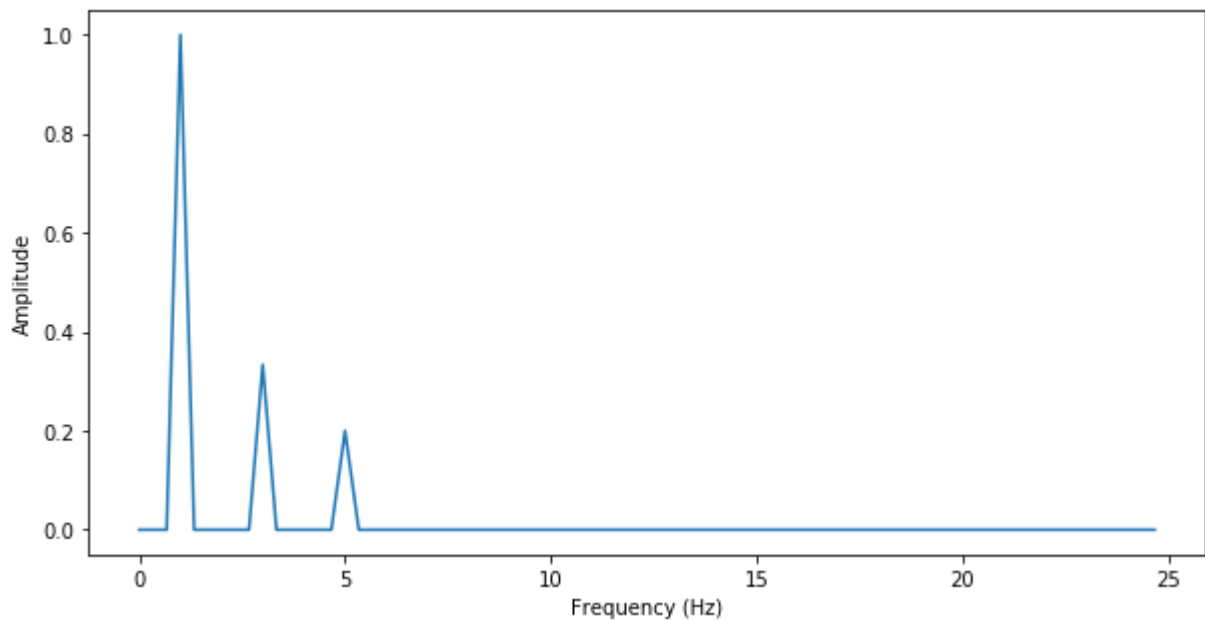
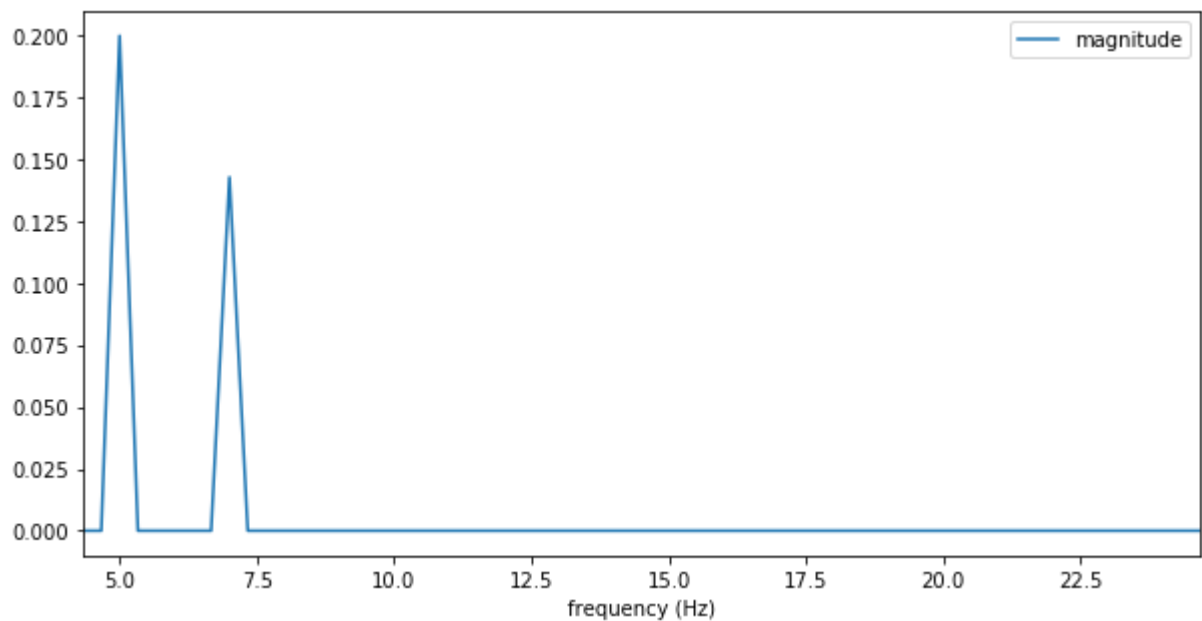
Out[9]:

	frequency (Hz)	magnitude
0	0.000000	1.198533e-17
1	0.333333	4.708312e-16
2	0.666667	1.693711e-16
3	1.000000	1.000000e+00
4	1.333333	2.532933e-16
5	1.666667	2.175634e-16
6	2.000000	8.562421e-17
7	2.333333	3.622784e-17
8	2.666667	1.605641e-16
9	3.000000	3.333333e-01
10	3.333333	4.453292e-17
11	3.666667	2.852648e-16
12	4.000000	1.687330e-16
13	4.333333	1.762855e-16
14	4.666667	1.024952e-16
15	5.000000	2.000000e-01
16	5.333333	3.043032e-17
17	5.666667	4.721993e-17
18	6.000000	1.094215e-16
19	6.333333	9.186276e-17
20	6.666667	4.559675e-17
21	7.000000	1.428571e-01
22	7.333333	2.067820e-16
23	7.666667	1.157871e-16
24	8.000000	1.308576e-16
25	8.333333	7.217396e-17
26	8.666667	8.540974e-17
27	9.000000	3.280439e-16
28	9.333333	1.806352e-16
29	9.666667	1.149617e-16
...
45	15.000000	3.640578e-16

	frequency (Hz)	magnitude
46	15.333333	5.918332e-17
47	15.666667	1.483689e-16
48	16.000000	2.518113e-17
49	16.333333	8.177367e-17
50	16.666667	1.387662e-16
51	17.000000	1.189139e-16
52	17.333333	1.646223e-16
53	17.666667	1.441729e-16
54	18.000000	7.511443e-17
55	18.333333	1.579188e-17
56	18.666667	1.895656e-16
57	19.000000	4.026080e-16
58	19.333333	4.482345e-17
59	19.666667	6.820695e-17
60	20.000000	1.317559e-16
61	20.333333	8.304112e-17
62	20.666667	3.973239e-17
63	21.000000	1.583243e-16
64	21.333333	6.677440e-17
65	21.666667	4.949965e-17
66	22.000000	3.107260e-17
67	22.333333	1.460695e-16
68	22.666667	1.578467e-16
69	23.000000	1.186047e-16
70	23.333333	1.588979e-16
71	23.666667	1.896099e-16
72	24.000000	9.710802e-17
73	24.333333	1.031463e-16
74	24.666667	1.179605e-16

75 rows × 2 columns

```
In [10]: # plot a frequency spectrum of x3 signal
s=get_fft(x3,f_s=50)
print(s[30:44])
```

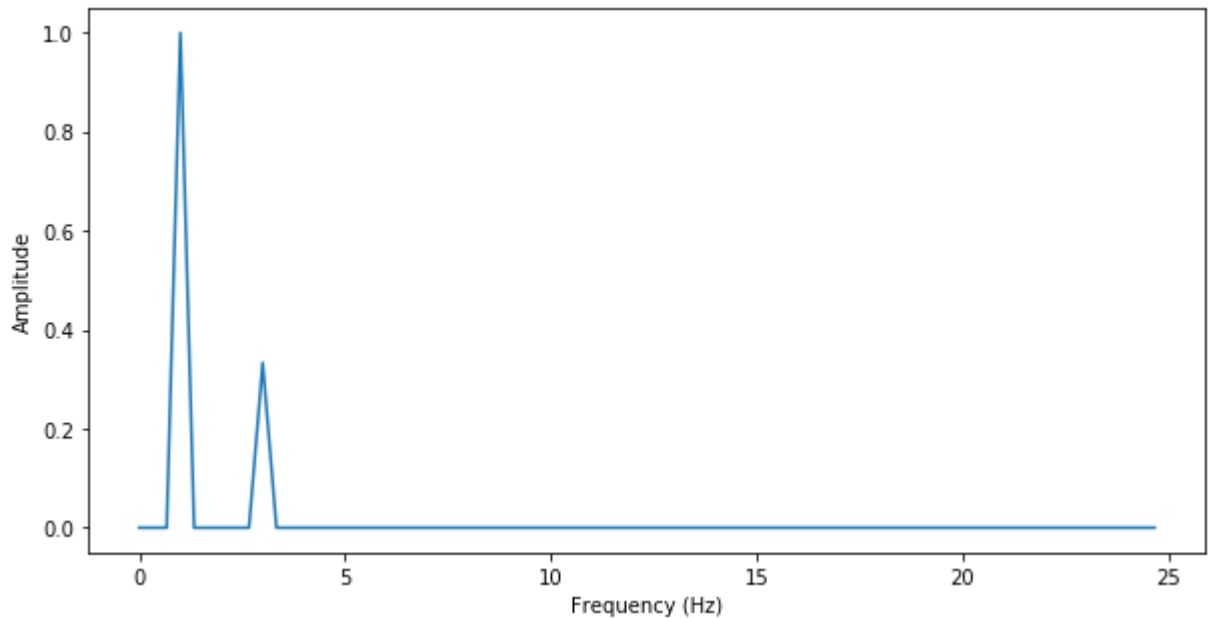


	frequency (Hz)	magnitude
30	10.000000	1.041679e-16
31	10.333333	1.132031e-16
32	10.666667	1.432593e-16
33	11.000000	1.230410e-16
34	11.333333	1.305870e-16
35	11.666667	9.809328e-17
36	12.000000	8.808944e-17
37	12.333333	7.270848e-17
38	12.666667	1.372254e-16
39	13.000000	1.309158e-16
40	13.333333	7.481132e-17
41	13.666667	1.342337e-16

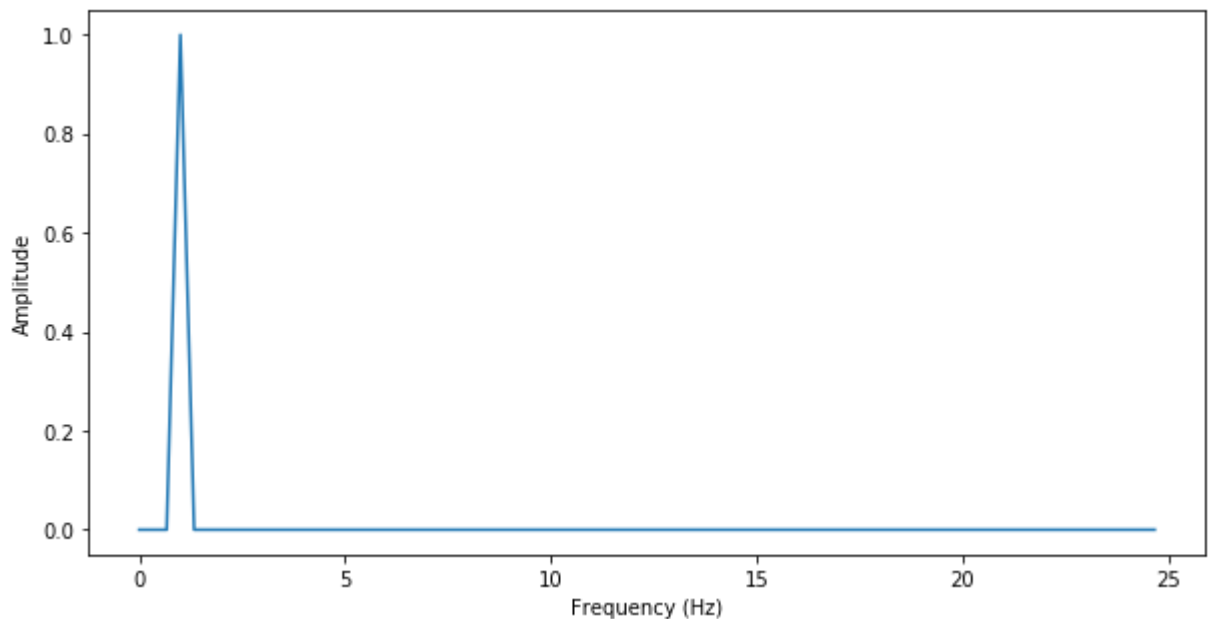
42	14.000000	7.411444e-17
43	14.333333	1.645896e-16



```
In [11]: # plot a frequency spectrum of x2 signal  
s=get_fft(x2,f_s=50)
```



```
In [12]: # plot a frequency spectrum of x1 signal  
s=get_fft(x1,f_s=50)
```



Q2: TODO Read the data in the file eeg.csv. It has EEG (Electroencephalogram that measures brainwaves) voltages of a person recorded over a constant timespan in two situations first when he was relaxing and 2nd when he was solving maths problems.

Convert this time series EEG into a fourier transform using a sampling frequency of 150.

Filter noise from brain signals assuming average range of brain waves frequencies is between 0.20-40Hz. Remove all other frequencies from the signal and plot the new fft.

```
In [13]: import json
import pandas as pd
eeg_file=pd.read_csv('eeg.csv')
eeg_file['raw_values']=eeg_file['raw_values'].map(json.loads)
```

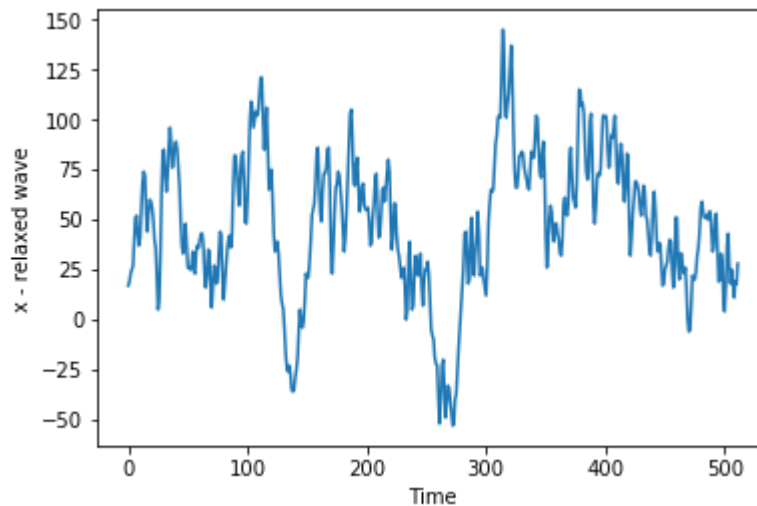
```
In [14]: relax = np.array(eeg_file['raw_values'][0])
math = np.array(eeg_file['raw_values'][1])
```

```
In [15]: # Signal for relaxed brainwaves

time = np.arange(0.0, 512.0, 1)
print('Relax size: ',relax.size)

plt.plot(time, relax)
plt.xlabel("Time")
plt.ylabel("x - relaxed wave")
plt.show()
```

Relax size: 512

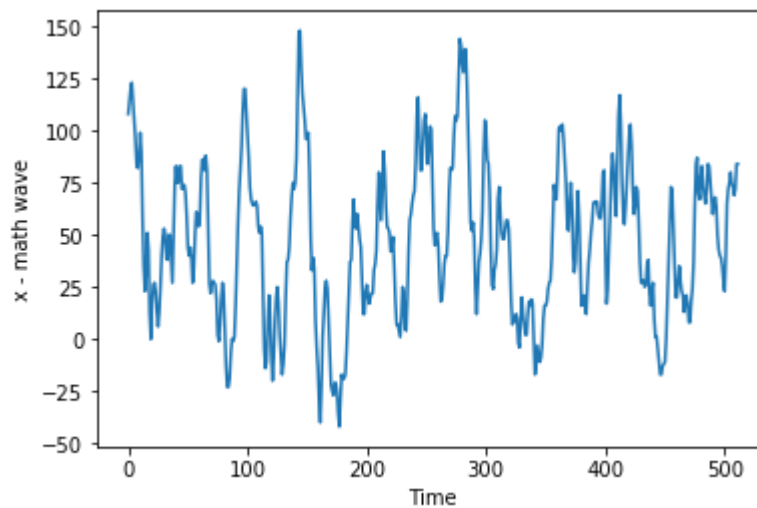


```
In [16]: # Signal for math brainwaves

time = np.arange(0.0, 512.0, 1)
print('Math size: ', math.size)

plt.plot(time, math)
plt.xlabel("Time")
plt.ylabel("x - math wave")
plt.show()
```

Math size: 512



```
In [17]: def get_fft_eeg(x, f_s=150):
    ...
    Function to convert raw timeseries signal to Fourier domain
    x:raw_signal
    f_s=sampling frequency if f_s=50 means sample 50 points per second from the t

    plots a folded and scaled frequency plot of the signal
    and returns the frequency spectrum dataframe of the signal
    ...

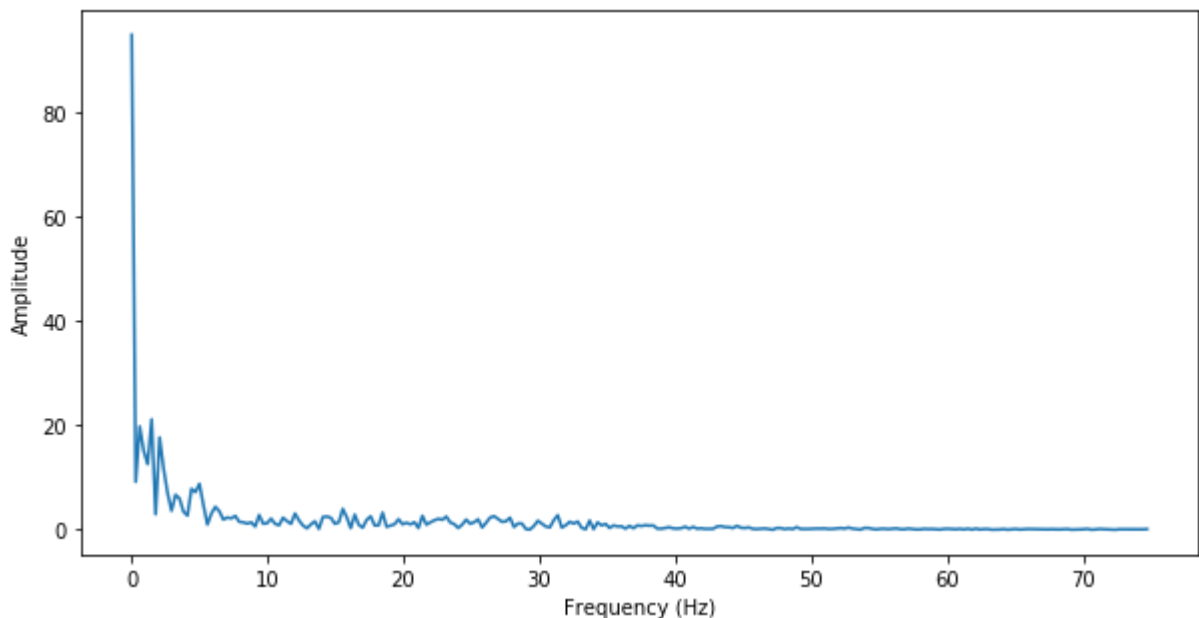
    n = len(x)
    fft_x = np.fft.fft(x) # gives the complex representation of each point

    # Find all the frequencies in the FFT
    freqs = np.fft.fftfreq(n, d=1/f_s)

    # Plot the folded and scaled frequency plot
    half_n = int(np.ceil(n/2.0))
    fft_x_half = (2.0 / n) * fft_x[:half_n]
    fft_x_half=np.abs(fft_x_half)
    freq_half = freqs[:half_n]
    plt.figure(figsize=(10,5))
    plt.plot(freq_half, fft_x_half)
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude")
    plt.show()
    spectrum = pd.DataFrame({'frequency (Hz)': freq_half,'magnitude': fft_x_half,
    return spectrum
```

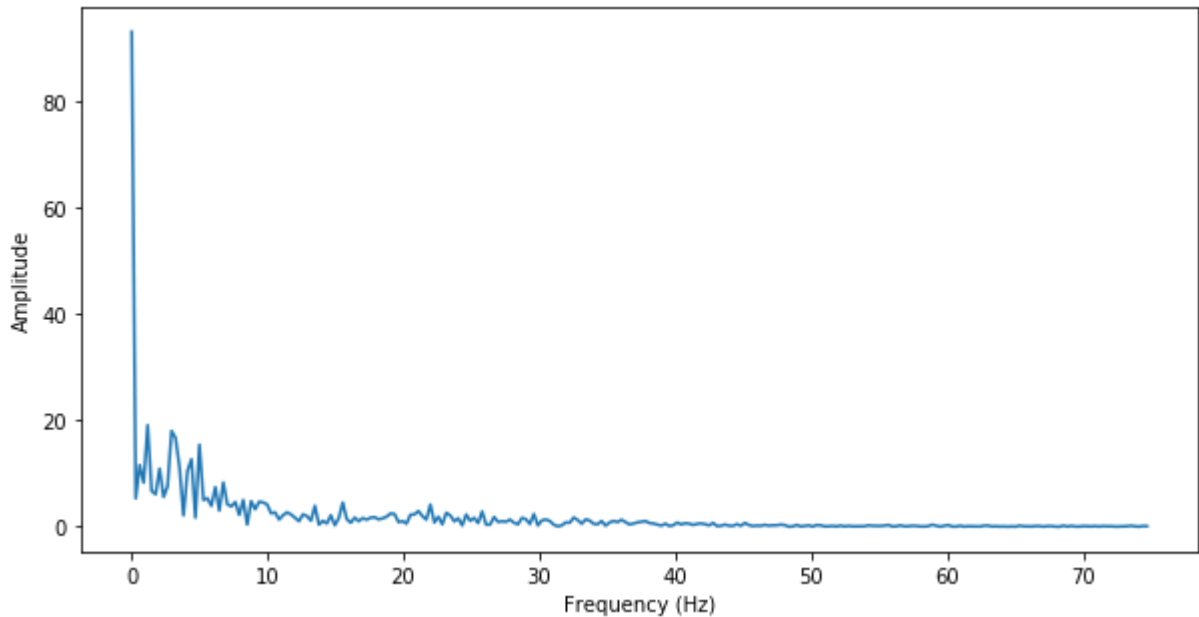
```
In [18]: # plot a frequency spectrum of relaxed brainwaves signal

r = get_fft(relax, f_s = 150)
```



In [19]: *# plot a frequency spectrum of relaxed brainwaves signal*

```
m = get_fft(math, f_s = 150)
```



In [20]: *# filter relaxed wave signal to exclude any signal outside the frequency range of 0.2 to 40 Hz*

```
r_filtered = r[(r['frequency (Hz)'] > 0.2) & (r['frequency (Hz)'] < 40)]
r_filtered.plot(x = 'frequency (Hz)', figsize=(10,5))
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1824f74cbe0>

In [21]: *# filter math wave signal to exclude any signal outside the frequency range of 0.2 to 40 Hz*

```
r_filtered = r[(r['frequency (Hz)'] > 0.2) & (r['frequency (Hz)'] < 40)]
r_filtered.plot(x = 'frequency (Hz)', figsize=(10,5))
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1824f79cb38>

Section 2: Fourier Operation on Images

The Fourier Transform of an image transforms it from a spatial domain to a frequency domain. In the Fourier domain of image, each point represents a particular frequency contained in the spatial domain image.

FFT of images is an image processing tool used in tasks such as image analysis, image filtering, image reconstruction and image compression.

In the frequency domain an image is decomposed into its sinusoidal components, it is easy to examine frequencies of the image and get idea about the geometric structure in the spatial domain.

In [22]: *# You can use this to experiment and understand the working of filters*

```
#img_test=np.array(range(0,49)).reshape(7,7)
```

```
In [23]: #!pip install opencv-python
```

2.1 Simple High Pass Image filter

As discussed in the lecture High pass filter acts like a Edge Enhancer, it lets through the high frequencies and eliminates the low frequencies in our image. High frequencies represent the details in our image.

This function (`high_pass_filter_image`) below, implements a simple high pass filter. You can

We will use opencv package to read and save images
Install opencv: on the terminal: `pip install opencv-python`
Or
on the ipython notebook run: `!pip install opencv-python`

```

In [24]: # Low pass and high pass filter in image show output
import cv2
img = cv2.imread('lena.jpg',0)

def high_pass_filter_image(img,filter_size=5):
    '''img= image pixel array -2d
    filter_size=number of frequencies you want to eliminate in your image starting
    filter_size number of frequencies will be removed from both negative and positive
    The function does the following:
    1. Converts spatial image pixels to FFT
    2. Centres the FFT with 0 frequency at the array centre and abs(frequency) in
    3. It then identifies the centre of the FFT of the image.
    4. It does a high pass filter on the centred FFT by setting all points around
        the centre, within the range of the filter size =0.

    5. It then converts the FFT back into spatial domain and returns the filtered
    # fft to convert the image to freq domain
    f = np.fft.fft2(img)

    # shift the center
    f_shift = np.fft.fftshift(f)

    rows, columns = img.shape
    centre_row,centre_col = rows//2 , columns//2

    # Frequencies that appear in the transform
    frequencies=np.fft.fftfreq(img.shape[0]*img.shape[1]).reshape(img.shape[0],img.shape[1])

    frequencies2 = np.fft.fftshift((frequencies))
    print('Maximum frequency=',np.abs(frequencies).max())
    print('Minimum frequency=',np.abs(frequencies).min())

    # High Pass Filter (HPF)-- remove the low frequencies
    assert filter_size<=len(img)

    # set all values in the filter range to 0
    f_shift[centre_row-filter_size:centre_row+filter_size,centre_col-filter_size:centre_col+filter_size]=0

    # recentre shift back (we shifted the center before)
    f_revert = np.fft.ifftshift(f_shift)

    # inverse fft to get the image back
    img_back = np.fft.ifft2(f_revert)

    img_back = np.abs(img_back)

    plt.subplot(121),plt.imshow(img)

```

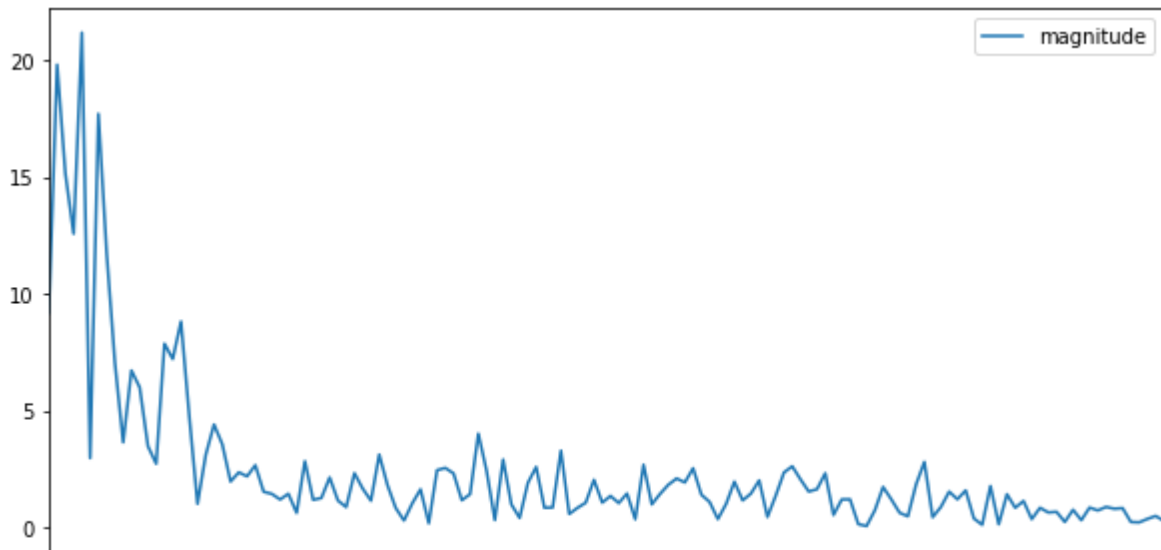
```
plt.title('Input Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122), plt.imshow(img_back)
plt.title('Filtered Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

In [25]: `high_pass_filter_image(img, 20)`

Maximum frequency= 0.5
Minimum frequency= 0.0

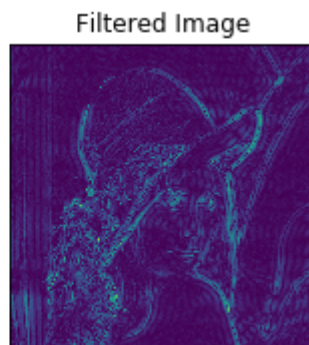


TODO :::

Q3: Do a high pass filter on lena.jpg using filter size of 20,100,200,250,256. What do you observe wrt to increase in size of the filter and filtered image.

In [26]: `high_pass_filter_image(img, 20)`

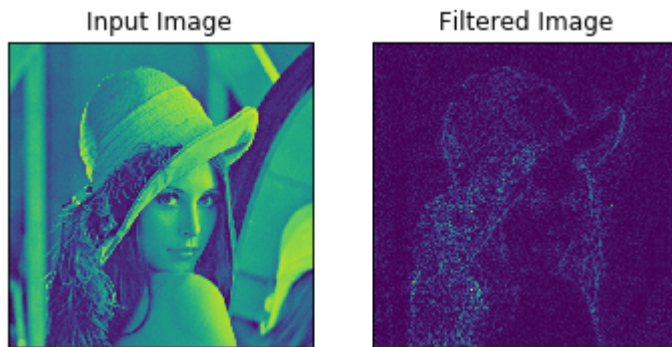
Maximum frequency= 0.5
Minimum frequency= 0.0




```
In [27]: high_pass_filter_image(img,100)
```

Maximum frequency= 0.5

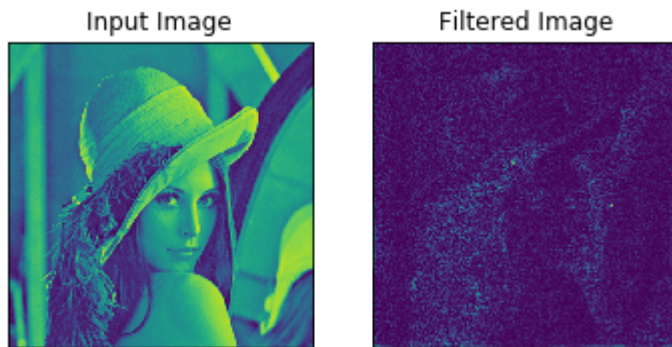
Minimum frequency= 0.0



```
In [28]: high_pass_filter_image(img,200)
```

Maximum frequency= 0.5

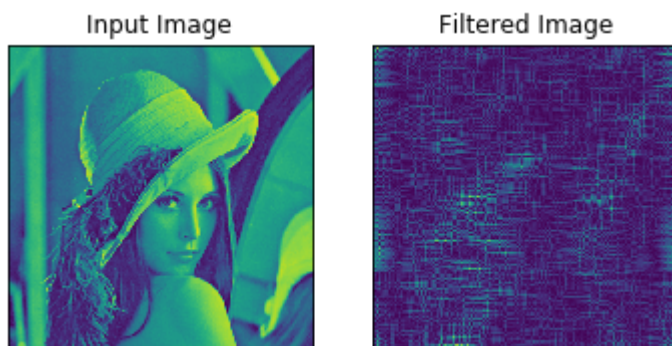
Minimum frequency= 0.0



```
In [29]: high_pass_filter_image(img,250)
```

Maximum frequency= 0.5

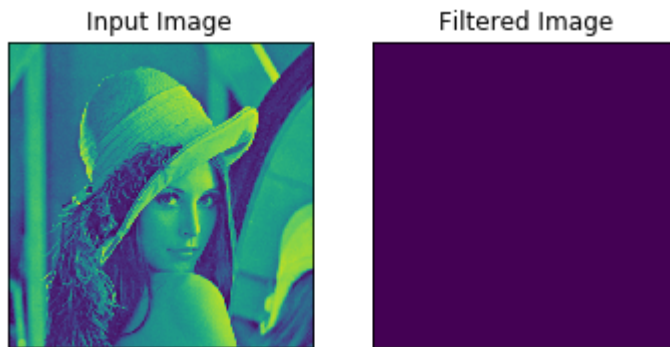
Minimum frequency= 0.0



```
In [30]: high_pass_filter_image(img,256)
```

Maximum frequency= 0.5

Minimum frequency= 0.0



```
In [31]: As the filter size increases, the sharpness of the edges
```

File "<ipython-input-31-841a273043f1>", line 1

As the filter size increases, the sharpness of the edges

^

SyntaxError: invalid syntax

TODO :::

2.2 Simple Low Pass Image filter:

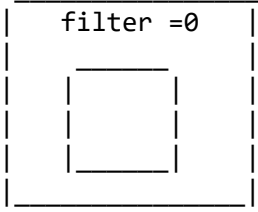
Low pass filter is the inverse of high pass filter it eliminates points with high frequencies generating an overall shading in the image.

Q4: Implement a simple low pass filter by setting points with frequencies greater than certain threshold frequency to 0. Plot original image and a low pass filtered image for filter size [220,230,250] i.e set these number of highest absolute frequencies equal to 0.(Note in a shifted FFT high frequencies are towards the periphery).See figure shown in the next cell.

Hint: You have to follow all the steps i.n high pass filter except that here instead of setting the central FFT points in the centred FFT filter square to 0, the filter sets peripheral high frequencies to 0.

```

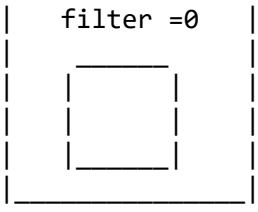
In [32]: """
Low Pass Image filter:



"""
print()

# Low pass and high pass filter in image show output
import cv2
img = cv2.imread('lena.jpg',0)

def low_pass_filter_image(img,filter_size=5):
    '''img= image pixel array -2d
    filter_size=number of frequencies you want to eliminate in your image starting
    filter_size number of frequencies will be removed from both negative and positive
    The function does the following:
    1. Converts spatial image pixels to FFT
    2. Centres the FFT with 0 frequency at the array centre and abs(frequency) in
    3. It then identifies the centre of the FFT of the image.
    4. It does a high pass filter on the centred FFT by setting all points around
    the centre, within the range of the filter size =0.



    5. It then converts the FFT back into spatial domain and returns the filtered
    # fft to convert the image to freq domain
    f = np.fft.fft2(img)

    # shift the center
    f_shift = np.fft.fftshift(f)

    rows, columns = img.shape
    centre_row,centre_col = rows//2 , columns//2

    # Frequencies that appear in the transform
    frequencies=np.fft.fftfreq(img.shape[0]*img.shape[1]).reshape(img.shape[0],img.shape[1])

    frequencies2 = np.fft.fftshift((frequencies))
    print('Maximum frequency=',np.abs(frequencies).max())
    print('Minimum frequency=',np.abs(frequencies).min())

    # High Pass Filter (HPF)-- remove the low frequencies
    assert filter_size<=len(img)

```

```

# set all values in the filter ange to 0
#f_shift[centre_row-filter_size:centre_row+filter_size,centre_col-filter_size
f_shift[0:filter_size,0:filter_size] = 0
f_shift[rows-filter_size : rows,columns-filter_size : columns] = 0

# recentre shift back (we shifted the center before)
f_revert = np.fft.ifftshift(f_shift)

# inverse fft to get the image back
img_back = np.fft.ifft2(f_revert)

img_back = np.abs(img_back)

plt.subplot(121),plt.imshow(img)
plt.title('Input Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(img_back)
plt.title('Filtered Image'), plt.xticks([]), plt.yticks([])

plt.show()

```

In [33]: low_pass_filter_image(img,220)

Maximum frequency= 0.5

Minimum frequency= 0.0

Input Image



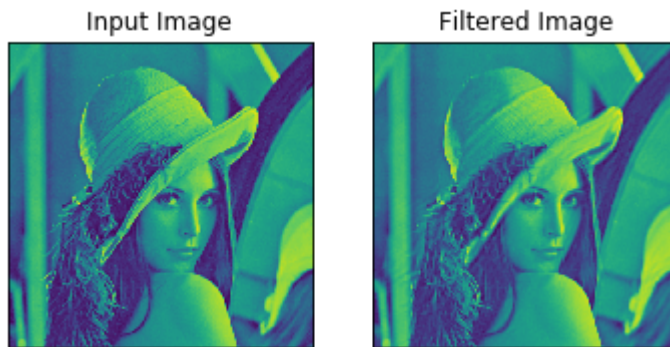
Filtered Image



```
In [34]: low_pass_filter_image(img,230)
```

Maximum frequency= 0.5

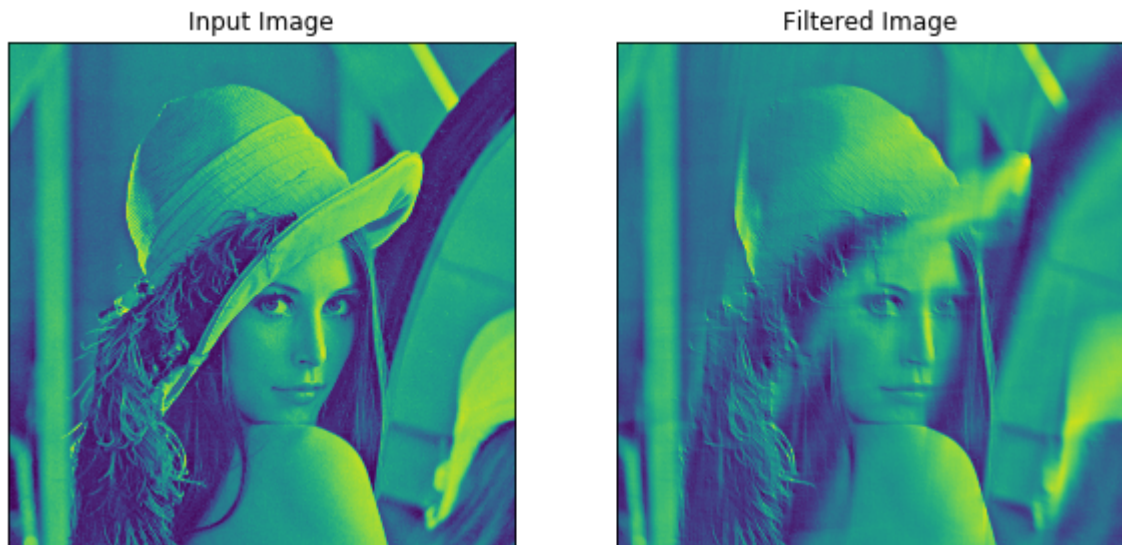
Minimum frequency= 0.0



```
In [52]: low_pass_filter_image(img,250)
```

Maximum frequency= 0.5

Minimum frequency= 0.0



```
In [ ]: #The high pass filter Lets only the high frequencies, which make up the details of
#The sharpness of the edges is inversely relevant to the filter size. So as the f
#When the filter size hits the max of 256 there are no more details in the filtered
```

Section 3: Regularization

Ref: https://github.com/ikhlaqsidhu/data-x/blob/master/05c-theory-tools-regularization-ml-rg1tm/regularization_notebook_v2.ipynb (https://github.com/ikhlaqsidhu/data-x/blob/master/05c-theory-tools-regularization-ml-rg1tm/regularization_notebook_v2.ipynb)

For this question you should use the code snippets used in the above referenced notebook.

Q1: Below you will see that a polynomial regression model of degree 20 and no regularisation has been fitted on the data. Observe that the model completely overfits the data.

Your task is to use a ridge regularised polynomial regression of degree 20 on the same data and report the best alpha value explaining your reason behind choosing it as the best.

Q2:Next, use lasso regularised polynomial regression of degree 20 on the same data and identify the non-zero weighted features of your best model.

In [36]: *#Importing libraries. The same will be used throughout the article.*

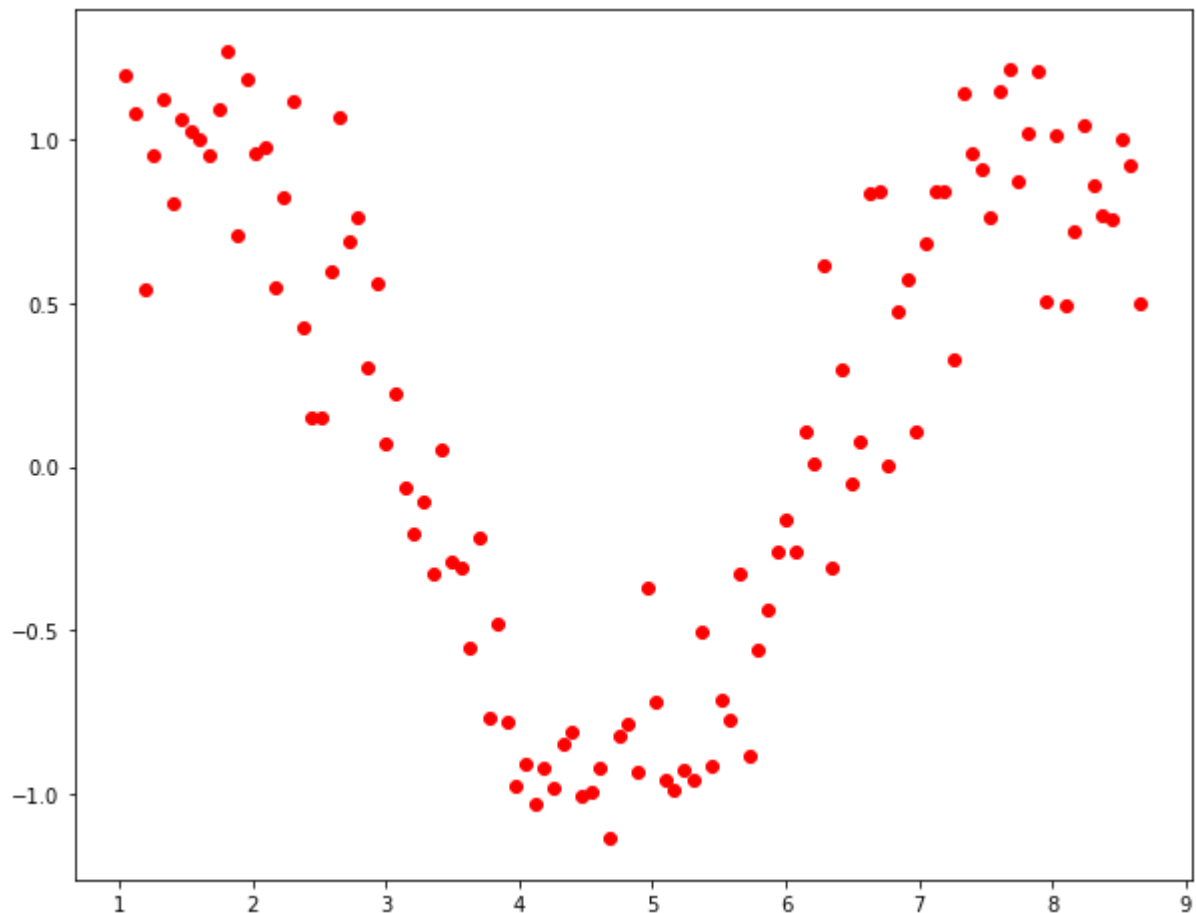
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 10, 8
from sklearn.linear_model import Ridge

#Import Linear Regression model from scikit-Learn.
from sklearn.linear_model import LinearRegression

import warnings
warnings.filterwarnings('ignore')

# read file
data=pd.read_csv('data_modelling.csv')
data.head()
plt.scatter(data['x'],data['y'],color='red')
```

Out[36]: <matplotlib.collections.PathCollection at 0x182500a5da0>



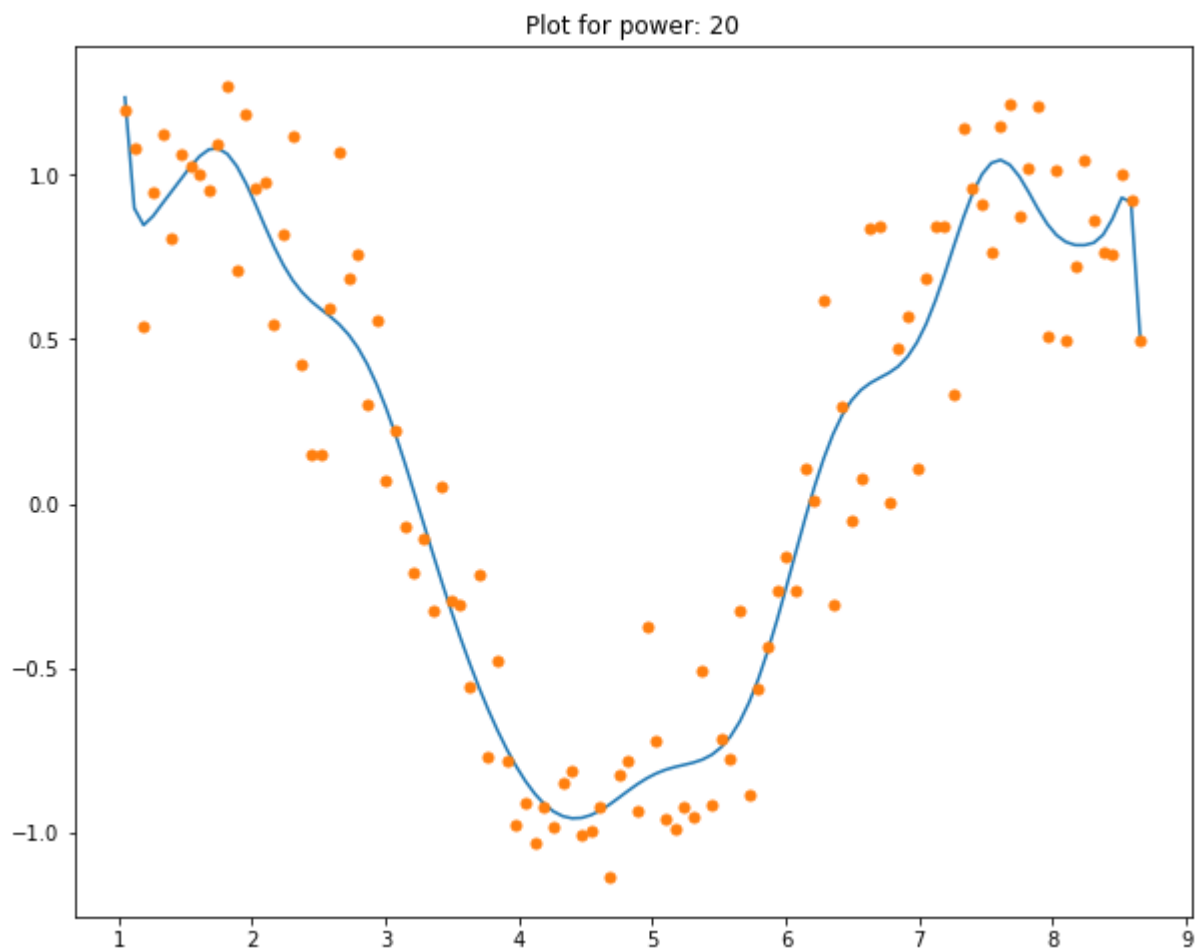
```
In [37]: def linear_regression(label,predictors, power):  
         #initialize predictors:  
  
         # add powers of feature  
  
         #Fit the model  
         linreg = LinearRegression(normalize=True)  
         linreg.fit(predictors,label)  
         y_pred = linreg.predict(predictors)  
  
         #Check if a plot is to be made for the entered power  
  
         plt.plot(data['x'],y_pred)  
         plt.plot(data['x'],label,'.',ms=10)  
         plt.title('Plot for power: %d'%power)  
  
         #Return the result in pre-defined format  
         rss = sum((y_pred-label)**2)  
         ret = [rss]  
         ret.extend([linreg.intercept_])  
         ret.extend(linreg.coef_)  
         return ret
```



```
In [38]: # add powers of x to the dataframe

for i in range(2,20+1): #power of 1 is already there
    colname = 'x_%d'%i    #new var will be x_power
    data[colname] = data['x']**i

predictors=data.loc[:, data.columns != 'y']
label=data['y']
result=linear_regression(label,predictors, power=20)
```



```
In [39]: from sklearn.linear_model import Ridge
def ridge_regression(data, predictors, alpha, models_to_plot={}):
    #Fit the model
    ridgereg = Ridge(alpha=alpha,normalize=True)
    ridgereg.fit(data[predictors],data['y'])
    y_pred = ridgereg.predict(data[predictors])

    #Check if a plot is to be made for the entered alpha
    if alpha in models_to_plot:
        plt.subplot(models_to_plot[alpha])
        plt.tight_layout()
        plt.plot(data['x'],y_pred)
        plt.plot(data['x'],data['y'],'.')
        plt.title('Plot for alpha: %.3g'%alpha)

    #Return the result in pre-defined format
    rss = sum((y_pred-data['y'])**2)
    ret = [rss]
    ret.extend([ridgereg.intercept_])
    ret.extend(ridgereg.coef_)
    return ret
```

```
In [40]: #Add 20 powers to the data
for i in range(2,20+1): #power of 1 is already there
    colname = 'x_%d'%i      #new var will be x_power
    data[colname] = data['x']**i
```

```

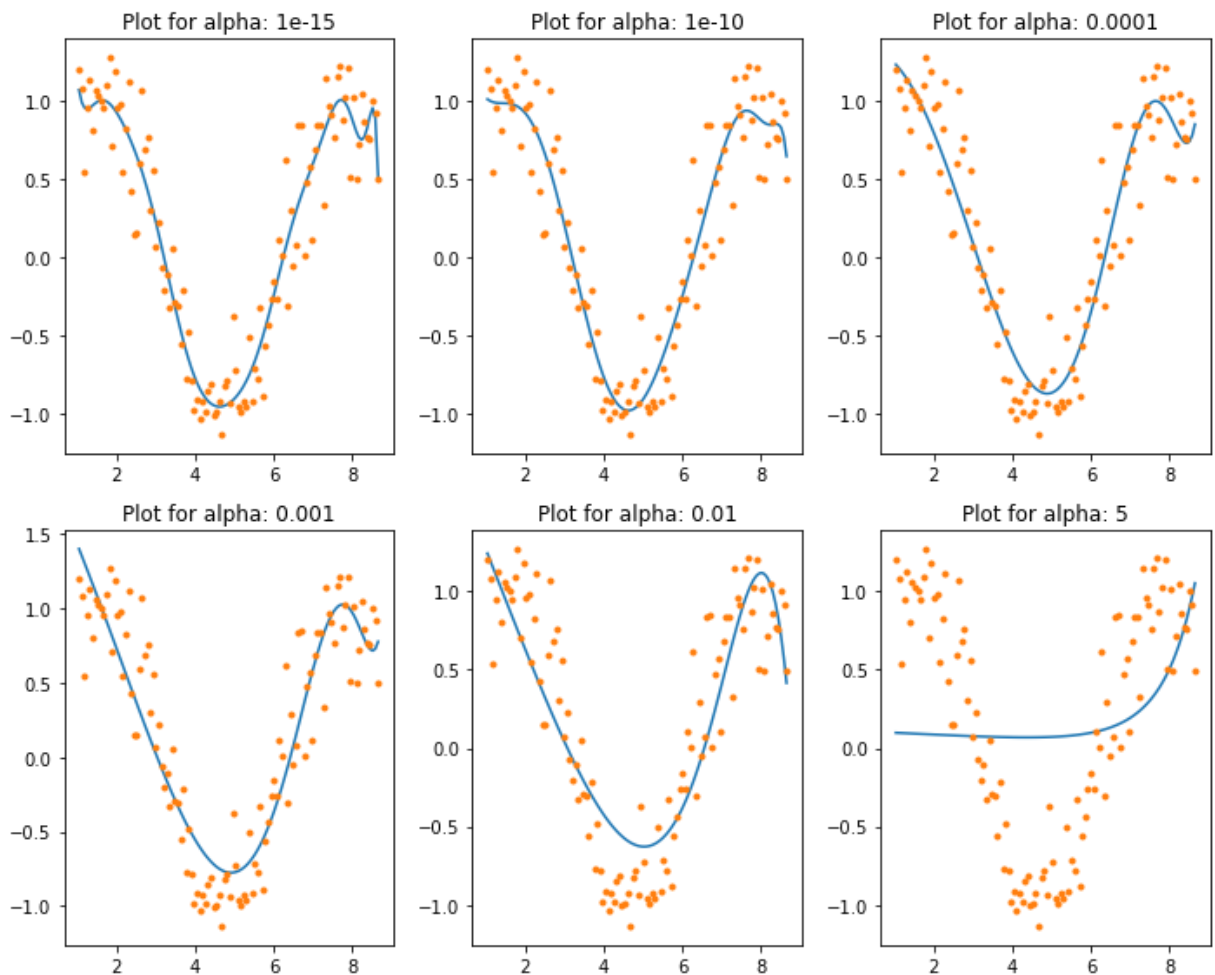
In [41]: #Initialize predictors to be set of 15 powers of x
predictors=['x']
predictors.extend(['x_%d'%i for i in range(2,20+1)]) ### 16

#Set the different values of alpha to be tested
alpha_ridge = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 5, 10, 20]

#Initialize the dataframe for storing coefficients.
col = ['rss','intercept'] + ['coef_x_%d'%i for i in range(1,20+1)] #### 16
ind = ['alpha_%.2g'%alpha_ridge[i] for i in range(0,10)]
coef_matrix_ridge = pd.DataFrame(index=ind, columns=col)

models_to_plot = {1e-15:231, 1e-10:232, 1e-4:233, 1e-3:234, 1e-2:235, 5:236}
for i in range(10):
    coef_matrix_ridge.iloc[i,] = ridge_regression(data, predictors, alpha_ridge[i]

```

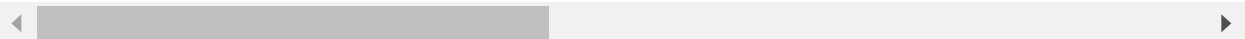


```
In [42]: #Set the display format to be scientific for ease of analysis
pd.options.display.float_format = '{:,.2g}'.format
coef_matrix_ridge
```

Out[42]:

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7
alpha_1e-15	5.7	51	-1.7e+02	2.5e+02	-1.9e+02	90	-24	3.3	-0.013
alpha_1e-10	6	2.5	-3.6	2.9	-0.9	0.059	0.0093	0.00029	-0.00011
alpha_1e-08	6.1	0.75	-0.26	0.8	-0.4	0.037	0.0048	-0.00013	-6.5e-05
alpha_0.0001	7.4	1.3	0.18	-0.22	0.00094	0.0023	0.00028	1.9e-05	2.7e-07
alpha_0.001	8.9	2.1	-0.57	-0.054	0.0022	0.00088	0.00012	1.1e-05	6.6e-07
alpha_0.01	12	1.9	-0.63	-0.012	0.0025	0.00048	5.6e-05	5.2e-06	3.9e-07
alpha_1	45	0.26	-0.059	-0.002	7.8e-05	2.9e-05	4.6e-06	5.9e-07	6.8e-08
alpha_5	53	0.11	-0.012	-0.00023	4.6e-05	1e-05	1.6e-06	2e-07	2.4e-08
alpha_10	54	0.1	-0.0048	2.6e-06	3.9e-05	7.3e-06	1e-06	1.3e-07	1.6e-08
alpha_20	56	0.12	-0.0017	8e-05	2.9e-05	5e-06	6.9e-07	8.7e-08	1.1e-08

10 rows × 22 columns



```
In [53]: # count how many of the coefficients are zero (in the full coefficient matrix)
coef_matrix_ridge.apply(lambda x: sum(x.values==0),axis=1)
```

```
Out[53]: alpha_1e-15      0
alpha_1e-10      0
alpha_1e-08      0
alpha_0.0001     0
alpha_0.001      0
alpha_0.01       0
alpha_1          0
alpha_5          0
alpha_10         0
alpha_20         0
dtype: int64
```

```
In [ ]: #The alpha of 0.0001 seems to fit the data best of all.
#It fits a smooth line through all the data points without overfitting, while also
```



```
In [44]: ### TO-DO CHOOSE BEST ALPHA EXPLANATION
```

```
In [45]: from sklearn.linear_model import Lasso
def lasso_regression(data, predictors, alpha, models_to_plot={}):
    #Fit the model
    lassoreg = Lasso(alpha=alpha, normalize=True, max_iter=1e5)
    lassoreg.fit(data[predictors], data['y'])
    y_pred = lassoreg.predict(data[predictors])

    #Check if a plot is to be made for the entered alpha
    if alpha in models_to_plot:
        plt.subplot(models_to_plot[alpha])
        plt.tight_layout()
        plt.plot(data['x'], y_pred)
        plt.plot(data['x'], data['y'], '.')
        plt.title('Plot for alpha: %.3g'%alpha)

    #Return the result in pre-defined format
    rss = sum((y_pred - data['y'])**2)
    ret = [rss]
    ret.extend([lassoreg.intercept_])
    ret.extend(lassoreg.coef_)
    return ret
```

```
In [46]: #Add 20 powers to the data
for i in range(2, 20+1): #power of 1 is already there
    colname = 'x_%d'%i #new var will be x_power
    data[colname] = data['x']**i
```

```

In [47]: #Initialize predictors to all 15 powers of x
predictors=['x']
predictors.extend(['x_%d'%i for i in range(2,20+1)])

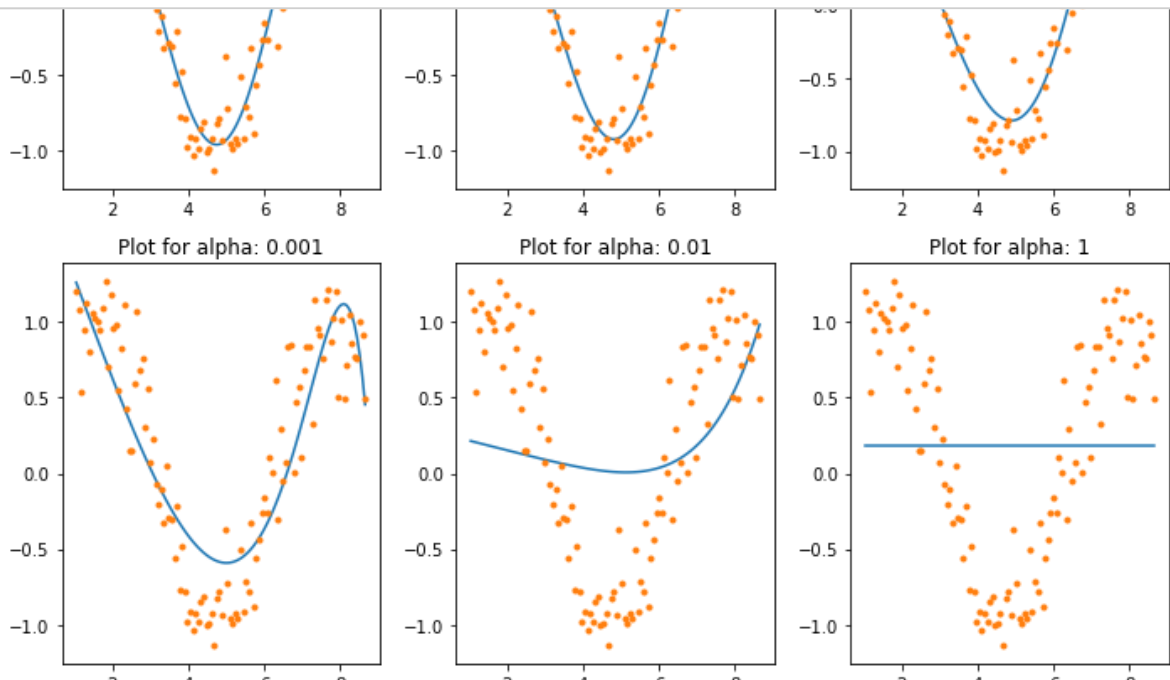
#Define the alpha values to test
alpha_lasso = [1e-15, 1e-10, 1e-8, 1e-5,1e-4, 1e-3,1e-2, 1, 5, 10]

#Initialize the dataframe to store coefficients
col = ['rss','intercept'] + ['coef_x_%d'%i for i in range(1,20+1)]
ind = ['alpha_%.2g'%alpha_lasso[i] for i in range(0,10)]
coef_matrix_lasso = pd.DataFrame(index=ind, columns=col)

#Define the models to plot
models_to_plot = {1e-10:231, 1e-5:232,1e-4:233, 1e-3:234, 1e-2:235, 1:236}

#Iterate over the 10 alpha values:
for i in range(10):
    coef_matrix_lasso.iloc[i,] = lasso_regression(data, predictors, alpha_lasso[i]

```

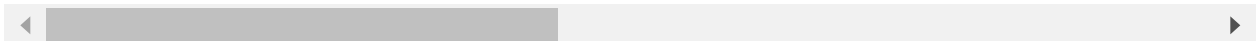


```
In [48]: #Set the display format to be scientific for ease of analysis
pd.options.display.float_format = '{:,.2g}'.format
coef_matrix_lasso
```

Out[48]:

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7
alpha_1e-15	6.2	-0.86	2.7	-1.1	0.07	0.0075	0.00022	-2.7e-05	-5.1e-06
alpha_1e-10	6.2	-0.86	2.7	-1.1	0.07	0.0075	0.00022	-2.7e-05	-5.1e-06
alpha_1e-08	6.2	-0.86	2.7	-1.1	0.07	0.0075	0.00022	-2.7e-05	-5.1e-06
alpha_1e-05	6.5	0.12	1.5	-0.6	0.012	0.009	0	-0	-9.2e-07
alpha_0.0001	8.5	1.8	-0.34	-0.11	0	0.0023	0.00014	0	0
alpha_0.001	13	2	-0.69	0	0	0.0014	0	0	0
alpha_0.01	46	0.28	-0.064	-0	0	0	0	3e-06	0
alpha_1	62	0.18	0	0	0	0	0	0	0
alpha_5	62	0.18	0	0	0	0	0	0	0
alpha_10	62	0.18	0	0	0	0	0	0	0

10 rows × 22 columns



```
In [49]: coef_matrix_lasso.apply(lambda x: sum(x.values==0),axis=1)
```

```
Out[49]: alpha_1e-15      0
alpha_1e-10      0
alpha_1e-08      0
alpha_1e-05      9
alpha_0.0001     14
alpha_0.001      16
alpha_0.01       18
alpha_1          20
alpha_5          20
alpha_10         20
dtype: int64
```

```
In [51]: row = coef_matrix_lasso.iloc[4]
df = pd.DataFrame(row)
df.loc[df['alpha_0.0001'] != 0]
```

Out[51]:

	alpha_0.0001
rss	8.5
intercept	1.8
coef_x_1	-0.34
coef_x_2	-0.11
coef_x_4	0.0023
coef_x_5	0.00014
coef_x_11	-7.1e-10
coef_x_20	8.8e-19

In []: