Complete python code for denoising the data signal with Fast Fourier Transform Algorithm

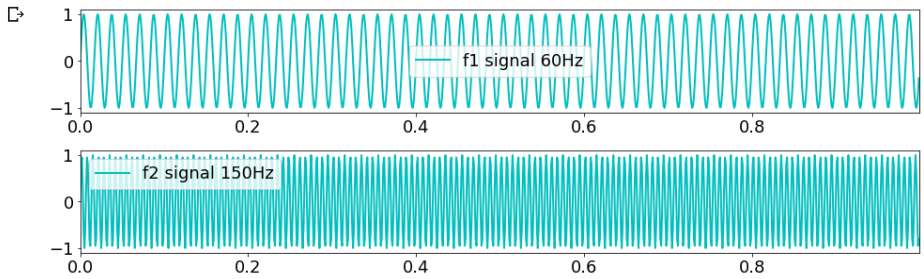
Import essential libraries and setup

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

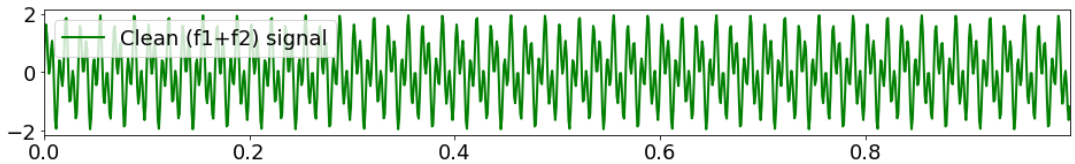
```
plt.rcParams['figure.figsize'] = [16, 12] #define size of the plot to be drawn.
plt.rcParams.update({'font.size': 18}) # update rc (runtime configuration) from default to any
```

```
# Creating simple signals with two frequencies 50Hz an 120Hz
dt = 0.001 # frame difference 0.001
t = np.arange(0,1,dt)
f1 = np.sin(2*np.pi*60*t) #first signal f1 50Hz
f2 = np.sin(2*np.pi*150*t) # second singal f2 120Hz
```

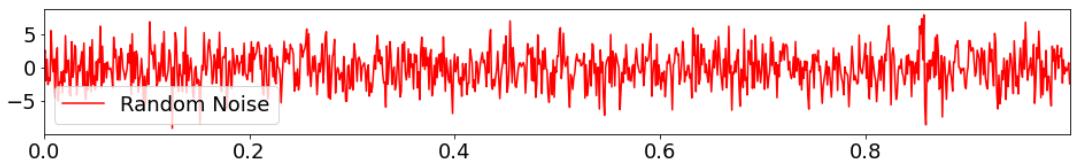
```
#plot signals f1 and f2
plt.rcParams['figure.figsize'] = [16, 2]
plt.plot(t,f1,color='c',LineWidth=2,label='f1 signal 60Hz')
plt.xlim(t[0],t[-1])
plt.legend()
plt.savefig('signal_60hz.png')
plt.show()

plt.rcParams['figure.figsize'] = [16, 2]
plt.plot(t,f2,color='c',LineWidth=2,label='f2 signal 150Hz')
plt.xlim(t[0],t[-1])
plt.legend()
plt.savefig('signal_150hz.png')
plt.show()
```




```
f_clean= f1 + f2 # Sum of 2 frequencies
#plot clean sum of two signals f1 and f2
plt.rcParams['figure.figsize'] = [16, 2]
plt.plot(t,f_clean,color='g',LineWidth=2,label='Clean (f1+f2) signal')
plt.xlim(t[0],t[-1])
plt.legend()
plt.savefig('sum_signals.png')
plt.show()
```



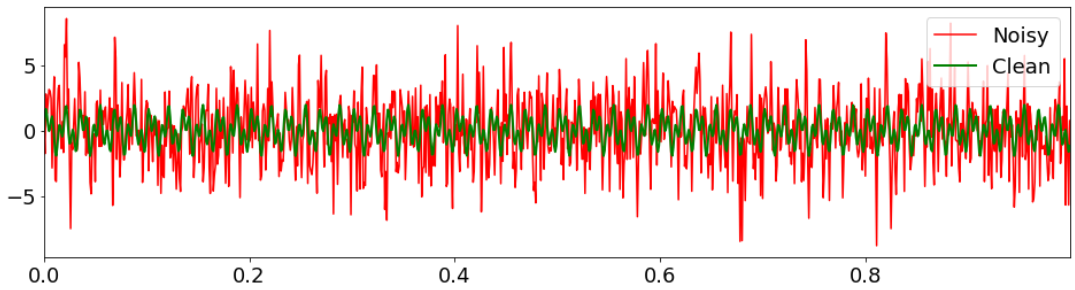

```
#add some random noise to the signal.
f = f_clean + 2.5*np.random.randn(len(t))
#plot Noise signal
plt.rcParams['figure.figsize'] = [16, 2]
plt.plot(t,f,color='r',LineWidth=1.5,label='Random Noise')
plt.xlim(t[0],t[-1])
plt.legend()
plt.savefig('with_noise.png')
plt.show()
```




```
#Plot data signal with noisy signal
plt.plot(t,f,color='r',LineWidth=1.5,label='Noisy')
plt.plot(t,f_clean,color='g',LineWidth=2,label='Clean')
plt.xlim(t[0],t[-1])
```

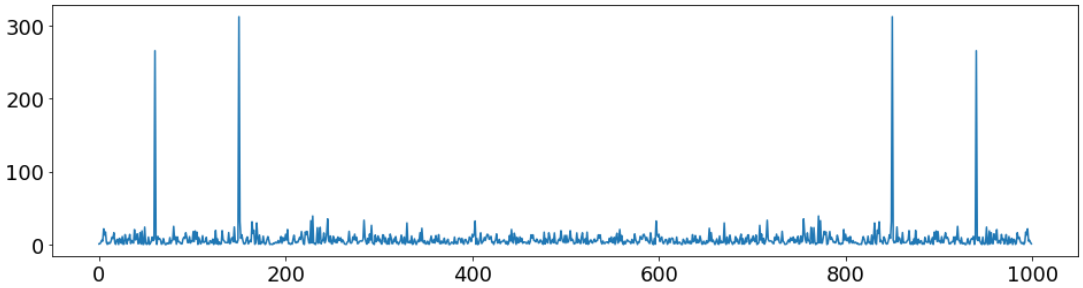
```
plt.savefig('original_noisy.png')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f7b277b5c10>
```




```
#converting time domain into frequency domian.
#Compute the Fast Fourier Transform (FFT)

n = len(t)
fhat = np.fft.fft(f,n)                     # Compute the FFT
PSD = fhat * np.conj(fhat) / n             # Power spectrum density (power per freq)
freq = (1/(dt*n)) * np.arange(n)           # Create x-axis of frequencies in Hz
L = np.arange(1,np.floor(n/2),dtype='int') # Only plot the first half of freqs

plt.plot(freq,PSD)
plt.savefig('time_to_freq.png')
plt.show()
```

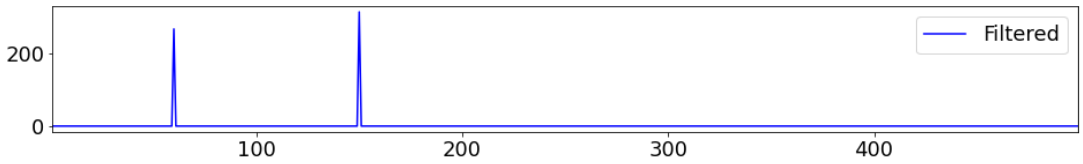
```
/usr/local/lib/python3.8/dist-packages/matplotlib/cbook/__init__.py:1317: ComplexWarning: Casting complex values to real discards the imagina
  return np.asarray(x, float)
```




```
## Use the PSD (power spectral density) to filter out noise
# removing noise in frequency domian.

indices = PSD > 120       # Find all freqs with large power
PSDclean = PSD * indices  # Zero out all others
fhat = indices * fhat     # Zero out small Fourier coeffs. in Y
ffilt = np.fft.ifft(fhat) # Inverse FFT for filtered time signal
```

```
#plot removing the noise by applying threshold of PSD>100 in frequecy domain
plt.rcParams['figure.figsize'] = [16, 2]
plt.plot(freq[L],PSDclean[L],color='b',LineWidth=1.5,label='Filtered')
plt.xlim(freq[L[0]],freq[L[-1]])
plt.legend()
plt.savefig('denoise_freq.png')
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/matplotlib/cbook/__init__.py:1317: ComplexWarning: Casting complex values to real discards the imagina
  return np.asarray(x, float)
```




```
# Plot of original signal and filtered signal
plt.rcParams['figure.figsize'] = [16, 2]
plt.plot(t,f_clean,color='g',LineWidth=1.5,label='Clean -Original')
plt.plot(t,ffilt,color='b',LineWidth=2,label='Filtered -recovered')
plt.xlim(t[0],t[-1])
plt.legend()
plt.savefig('fft_infft_signals.png')
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/matplotlib/cbook/__init__.py:1317: ComplexWarning: Casting complex values to real discards the imagina
  return np.asarray(x, float)
```