1) Create one artificial time series composed of repeating signal (like sum of sinus signals) with noise and analyze repeating patterns (frequencies) in the signal. You can apply autocorrelation function or FFT of other method if you know.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import correlate
```

Creating an artificial time series signal

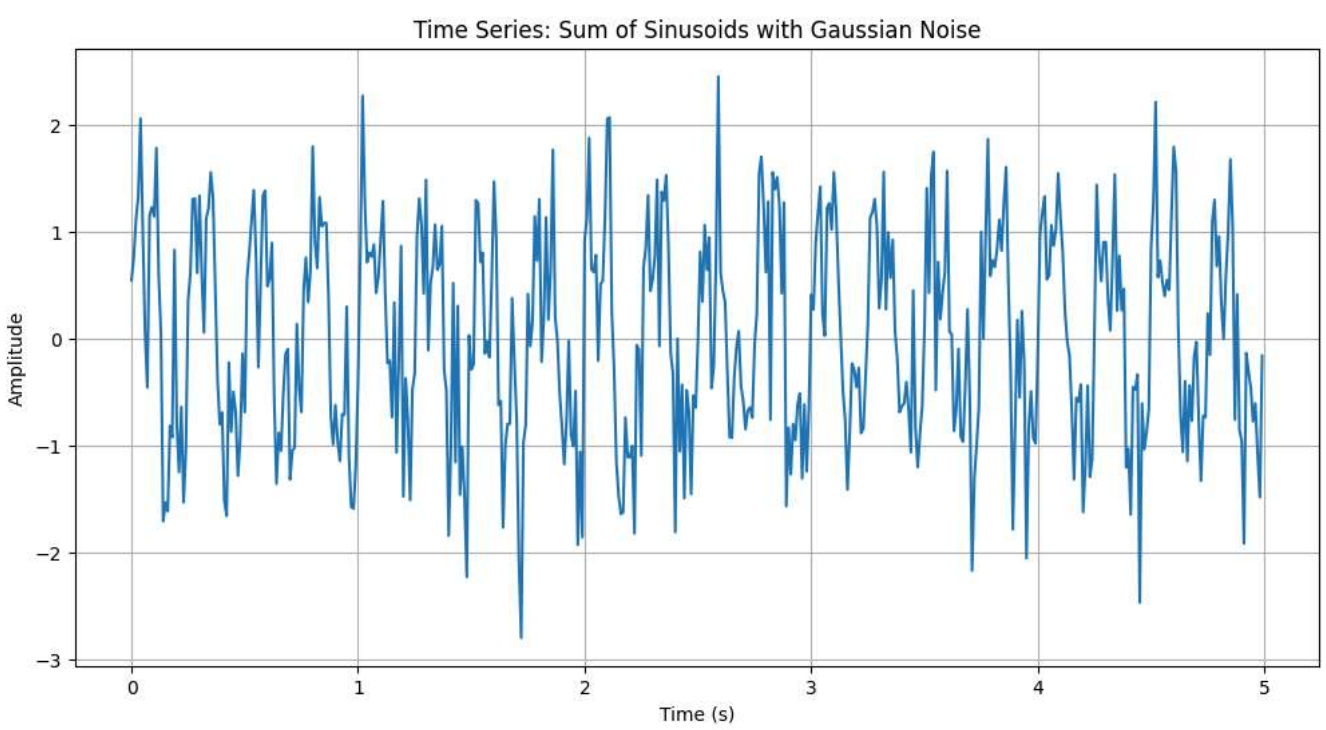
```
A1 , A2 = 1 , 0.6  #Amplitudes
f1 , f2 = 4 , 12   # Frequencies in Hz
sampling_rate = 100 # sampling rate in Hz
duration = 5.0

# Time array
t = np.linspace(0, duration, int(sampling_rate*duration), endpoint = False)

# Generate the signal (sum of two sinusoids)
sin_wave = A1*np.sin(2*np.pi*f1*t) + A2*np.sin(2*np.pi*f2*t)

# Adding noise(Gaussian)
noise = np.random.normal(0,0.5,sin_wave.shape)
noise_signal = sin_wave + noise

plt.figure(figsize=(12,6))
plt.plot(t,noise_signal,label="Signal with noise")
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title("Time Series: Sum of Sinusoids with Gaussian Noise")
plt.grid(True)
plt.show()
```



The generated time series $s(t)$ s(t) can be written as:

$s(t) = A1 \sin(2\pi f1 t) + A2 \sin(2\pi f2 t) + \text{noise}(t)$

Where:

$A1$and $A2$are the amplitudes of the sine waves. $f1$and $f2$are the frequencies of the sine waves. noise(t) is a random noise component added to simulate real-world randomness

Gaussian white noise is used, which means the noise has a mean of zero and a certain standard deviation.

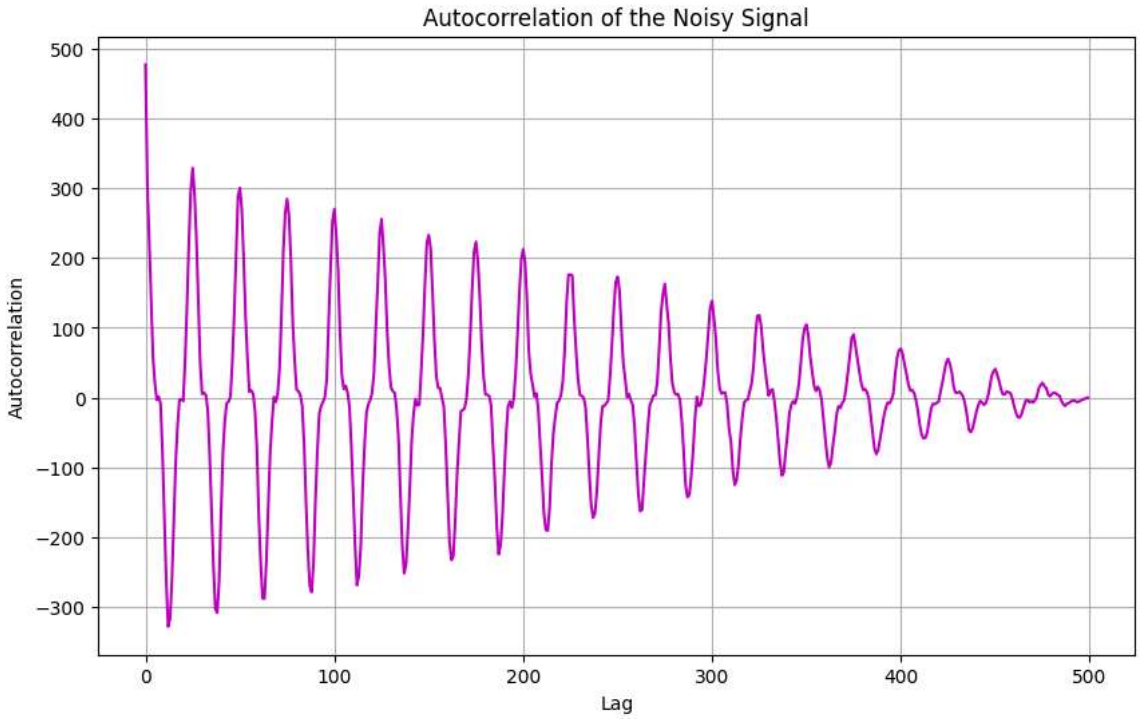
Mathematically, this could be represented as:

noise(t)~N(0,σ 2)

```
autocorr = correlate(noise_signal, noise_signal, mode='full')
lags = np.arange(-len(noise_signal)+1, len(noise_signal))

positive_lags = lags[len(lags) // 2:]
autocorr_pos = autocorr[len(lags) // 2:]

plt.figure(figsize=(10, 6))
plt.plot(positive_lags, autocorr_pos, label="Autocorrelation", color='m')
plt.title("Autocorrelation of the Noisy Signal")
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.grid(True)
plt.show()
```



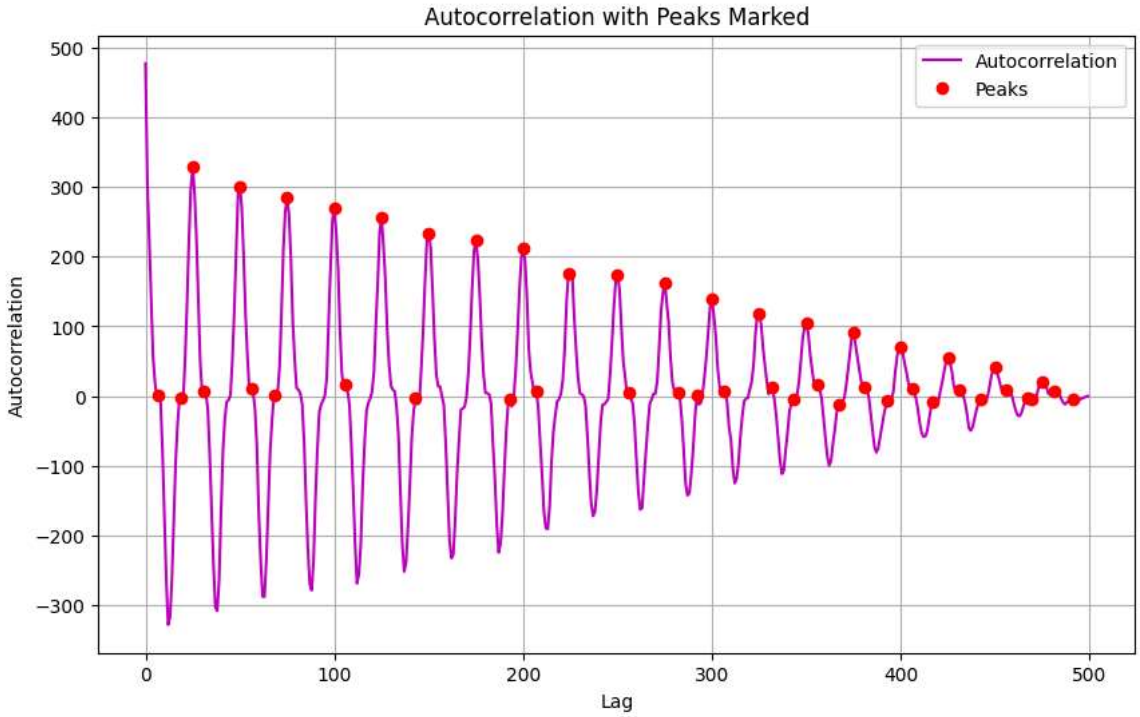

```
from scipy.signal import find_peaks

# Find the peaks of the autocorrelation
peaks, _ = find_peaks(autocorr_pos)

# Calculate the time intervals (lags) between peaks
lag_intervals = np.diff(positive_lags[peaks])

# Plot the autocorrelation with marked peaks
plt.figure(figsize=(10, 6))
plt.plot(positive_lags, autocorr_pos, label="Autocorrelation", color='m')
plt.plot(positive_lags[peaks], autocorr_pos[peaks], 'ro', label="Peaks")
plt.title("Autocorrelation with Peaks Marked")
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.grid(True)
plt.legend()
plt.show()

print("Lag intervals between peaks:", lag_intervals)
```




```
Lag intervals between peaks: [12  6  6 19  6 12  7 25  6 19 18  7 25 18  7  7 17 26  6 19  7 10  8  6
 19  7 11  7  6 11  8  6 12  7  6 11  8  6 11  8  6 11  2  6  6 10]
```

The autocorrelation plot reveals clear periodic patterns in the signal, with recurring peaks at regular intervals. These peaks correspond to the periods of the underlying sinusoidal components. The spacing between the peaks aligns with the periodicity of the two main frequencies:

1) The smaller peaks correspond to the higher frequency (15 Hz).

2) The larger, more spaced-out peaks correspond to the lower frequency (5 Hz).

This confirms the presence of both sinusoidal components in the noisy signal, even in the time domain.

Stationarity Check

To check for stationarity, we look for the decay rate of the autocorrelation function. If the autocorrelation decays quickly to zero, the signal is stationary. Slow decay suggests non-stationarity

```
threshold = 0.1

if np.all(np.abs(autocorr_pos[:10]) < 0.1):
    print("Stationarity: The time series appears stationary.")
else:
    print("Stationarity: The time series might be non-stationary due to slow decay.")
```

```
Stationarity: The time series might be non-stationary (slow decay).
```

Detecting Trends

To identify trends, we can look at the autocorrelation function's behavior at higher lags. If the values remain high for many lags, it suggests a long-term trend or persistent behavior.

```
if np.any(autocorr_pos[:100] > 0.2):
    print("Trend: Possible long-term trend detected.")
else:
    print("Trend: No significant trend detected.")
```

```
Trend: Possible long-term trend detected.
```

White Noise Check

White noise has no significant autocorrelation at non-zero lags, so we can check if the autocorrelation function remains close to zero for all lags beyond the first one.

```
if np.all(np.abs(autocorr_pos[1:]) < threshold):
    print("White Noise: The time series behaves like white noise.")
else:
    print("White Noise: The time series is not white noise (significant autocorrelation present).")
```

```
White Noise: The time series is not white noise (significant autocorrelation present).
```

Estimating Memory or Lag Dependencies

We can estimate how many lags the signal has by finding the significant autocorrelation lags.

```
1
2 significant_lags = positive_lags[autocorr_pos > 0.2]
3 print(f"Significant lags detected: {significant_lags}")
4
5 if len(significant_lags) > 0:
6     print(f"\nThe signal has memory up to {significant_lags[-1]} lags.")
7 else:
8     print("n No significant memory in the time series.")
9
```

```
Significant lags detected: [  0   1   2   3   4   5   7  21  22  23  24  25  26  27  28  29  30  31
  32  45  46  47  48  49  50  51  52  53  54  55  56  57  68  70  71  72
  73  74  75  76  77  78  79  80  81  82  95  96  97  98  99 100 101 102
 103 104 105 106 107 120 121 122 123 124 125 126 127 128 129 130 131 132
 146 147 148 149 150 151 152 153 154 155 156 171 172 173 174 175 176 177
 178 179 180 181 182 196 197 198 199 200 201 202 203 204 205 206 207 220
 221 222 223 224 225 226 227 228 229 230 231 246 247 248 249 250 251 252
 253 254 255 256 269 270 271 272 273 274 275 276 277 278 279 280 281 282
 292 295 296 297 298 299 300 301 302 303 304 305 306 307 320 321 322 323
 324 325 326 327 328 329 330 331 332 345 346 347 348 349 350 351 352 353
 354 355 356 357 371 372 373 374 375 376 377 378 379 380 381 382 396 397
 398 399 400 401 402 403 404 405 406 407 421 422 423 424 425 426 427 428
 429 430 431 432 433 446 447 448 449 450 451 452 453 454 455 456 457 458
 472 473 474 475 476 477 478 479 480 481 482 483 484]

The signal has memory up to 484 lags.
```

The number of significant lags will give us an idea of the "memory" of the signal.

Estimated Frequencies

```
1 frequencies = sampling_rate / lag_intervals
2 print("Estimated frequencies from autocorrelation:", frequencies)
```

```
Estimated frequencies from autocorrelation: [ 8.33333333 16.66666667 16.66666667  5.26315789 16.66666667  8.33333333
 14.28571429  4.         16.66666667  5.26315789  5.55555556 14.28571429
  4.          5.55555556 14.28571429 14.28571429  5.88235294  3.84615385
 16.66666667  5.26315789 14.28571429 10.         12.5        16.66666667
  5.26315789 14.28571429  9.09090909 14.28571429 16.66666667  9.09090909
 12.5        16.66666667  8.33333333 14.28571429 16.66666667  9.09090909
 12.5        16.66666667  9.09090909 12.5        16.66666667  9.09090909
 50.         16.66666667 16.66666667 10.        ]
```

These are the frequencies detected from the autocorrelation peaks. Ideally, we expected to see frequencies close to 4 Hz and 12 Hz, which is the known frequencies of the original signal. However, noise in the data and other factors led to some variation.

The autocorrelation analysis provided meaningful insights into the periodicity, trend and stationarity of the noisy time series. While some discrepancies in the frequency estimates arose due to the noise, significant patterns were detected, demonstrating that autocorrelation is effective at identifying structure in a time series.

2) Common Methods for Analyzing Repeating Patterns in Time Series

Autocorrelation

Autocorrelation measures how correlated a time series is with its lagged version at different time intervals. It helps to identify repeating patterns by showing periodic spikes at regular intervals.

Application: It is widely used for detecting seasonality or cyclic patterns in a time series. For example, in economic data, autocorrelation can reveal repeating yearly or quarterly business cycles.

Feature Extraction: Peaks in autocorrelation can be converted into features, such as lag values (time intervals) and amplitude (strength of correlation), which can inform a model of periodicity in the data.

Fast Fourier Transform (FFT)

Overview: FFT decomposes a time series into its constituent frequencies, transforming the data from the time domain to the frequency domain. It is particularly useful for identifying periodic patterns and their corresponding frequencies.

Application: FFT is used in signal processing, vibration analysis, and electrocardiogram (ECG) monitoring. It helps uncover hidden periodicities that might not be apparent in the time domain.

Feature Extraction: The dominant frequencies, amplitudes, and phases can serve as input features. These features are crucial in systems that rely on periodic behavior, such as predicting machinary failure in industrial IoT applications.

Wavelet Transform

Overview: Unlike FFT, which assumes a stationary signal, wavelet transform allows analysis of both frequency and time localization. It decomposes the signal using wavelets, enabling detection of patterns at different scales.

Application: Wavelet transform is useful for non-stationary time series (e.g., financial data, seismic data) where the nature of the repeating pattern can change over time.

Feature Extraction: The resulting wavelet coefficients can serve as features representing changes in frequency over time, which are useful in applications like earthquake prediction and fault detection in power systems.

Seasonal Decomposition of Time Series (STL)

Overview: STL is a method for decomposing a time series into three components: trend, seasonal, and residual. This is particularly useful for handling seasonal patterns and long-term trends in the data.

Application: Commonly applied in forecasting, STL can be used in retail demand forecasting, climate data analysis, and energy consumption modeling.

Feature Extraction: The extracted seasonality, trend, and residuals components can be input into machine learning models to forecast future values. For example, separating out seasonality from retail sales data can lead to more accurate demand forecasting.

Self-Similarity (Hurst Exponent)

Overview: The Hurst exponent is used to detect long-term memory and self-similarity in time series. It can identify whether a time series is a random walk, mean-reverting, or trending.

Application: The Hurst exponent is widely used in finance to detect asset price trends and in hydrology to predict river flow patterns.

Feature Extraction: The value of the Hurst exponent can itself be a feature in models predicting long-term behavior, particularly in fields where long-range dependence is critical (e.g., stock market predictions).

Sources referred for this jupyter notebook are -

https://www.scicoding.com/practical-guide-to-autocorrelation/

https://stackoverflow.com/questions/14058340/adding-noise-to-a-signal-in-python

https://brainly.com/question/37253676

https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/

https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Electronics/AC_Electrical_Circuit_Analysis%3A_A_Practical_Approach_(Fiore)/01%3A_Fundamentals/1.2%3A_Sinusoidal_Waveforms

https://towardsdatascience.com/understanding-autocorrelation-in-time-series-analysis-322ad52f2199

https://www.influxdata.com/blog/autocorrelation-in-time-series-data/

https://bookdown.org/rdpeng/timeseriesbook/autocorrelation.html

Utilized the above mentioned sources and ChatGPT to educate myself about Autocorrelation and other methods used for analysing repeating patterns in time series analysis and the relevant code required to perform the basic analysis.

```
1 Start coding or generate with AI.
```