In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import statsmodels as ss
import numpy as np
import matplotlib
import pandas as pd
```

Exercise #1.1

In this exercise, your task is two-fold:

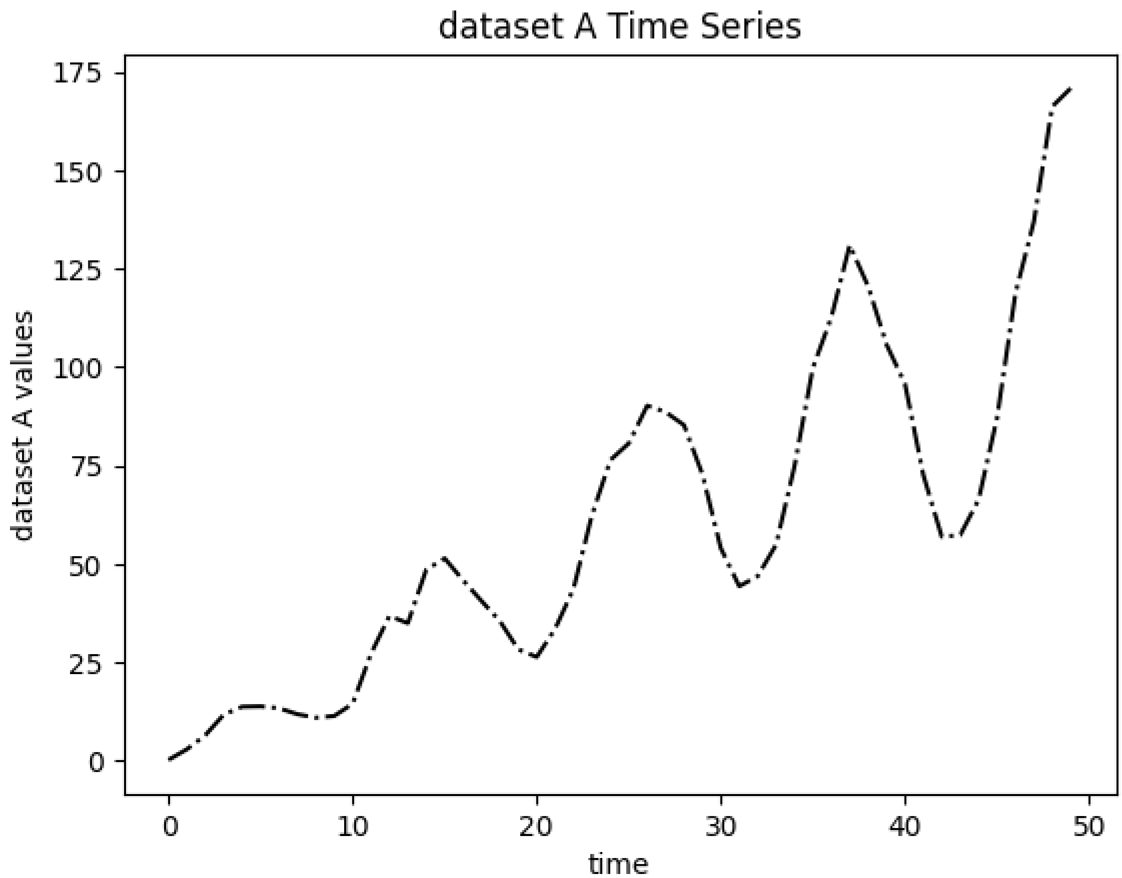
1. Determine if **Dataset_A** and **Dataset_B** are additive or multiplicative time series.
2. Determine the frequency of the seasonal component.

Set Path / Load Datasets

In [2]:
```python
# get data
path_to_file = "./" # Modify if data are in a different directory

time = np.arange(0, 50)
dataset_A = np.load(path_to_file + "dataset_A.npy")
dataset_B = np.load(path_to_file + "dataset_B.npy")
```

Plot Dataset_A

In [3]:
```python
# insert code here

plt.plot(time, dataset_A, 'k-.')
plt.title("dataset A Time Series")
plt.xlabel("time")
plt.ylabel("dataset A values");
```



Additive or Multiplicative?

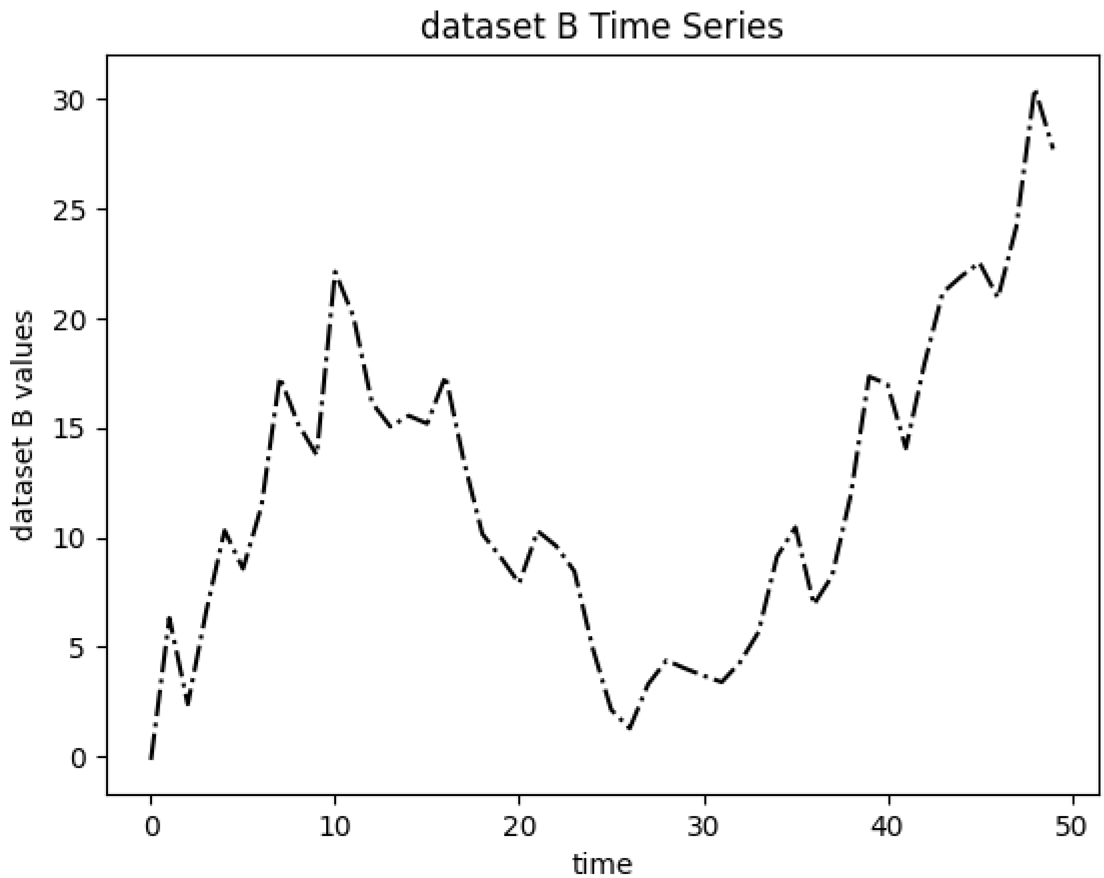
multiplicative

Frequency of Seasonal Component?

10

Plot Dataset_B

In [4]:
```
# insert code here

#plt.plot(dataset_B)

plt.plot(time, dataset_B, 'k-.')
plt.title("dataset B Time Series")
plt.xlabel("time")
plt.ylabel("dataset B values");
```



Additive or Multiplicative?

Additive

Frequency of Seasonal Component?

5

Exercise #1.2

In this exercise, your task is decompose **Dataset_A** and **Dataset_B**. You should first create a decomposition model in Python. Then you should plot the original series, the trend, seasonality, and residuals, in that order.

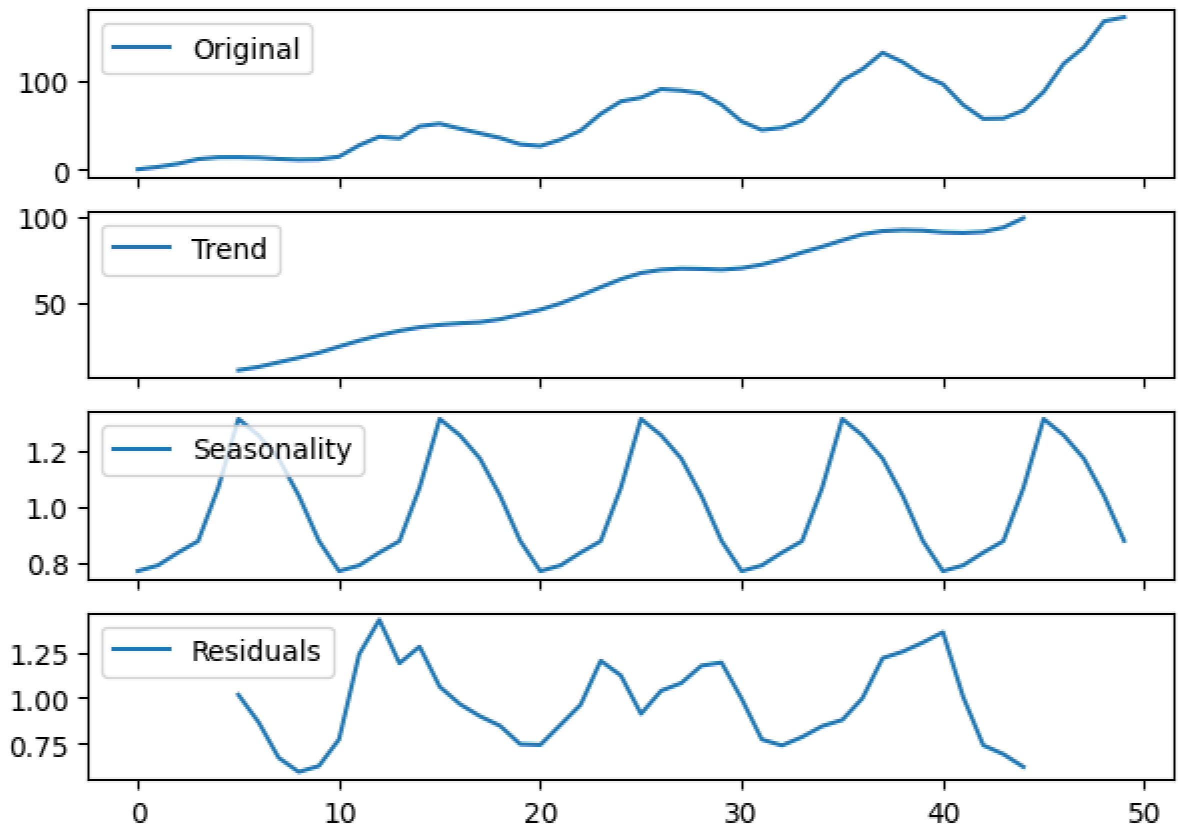
Decomposition Models

In [5]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.tsa.seasonal import seasonal_decompose



def ses_add(data, p):
    ss_decomposition = seasonal_decompose(x=data, model='additive', period=p)
    return ss_decomposition

def ses_mul(data, p):
```

```
    ss_decomposition = seasonal_decompose(x=data, model='multiplicative', period=p)
    return ss_decomposition
```

Dataset_A Plot

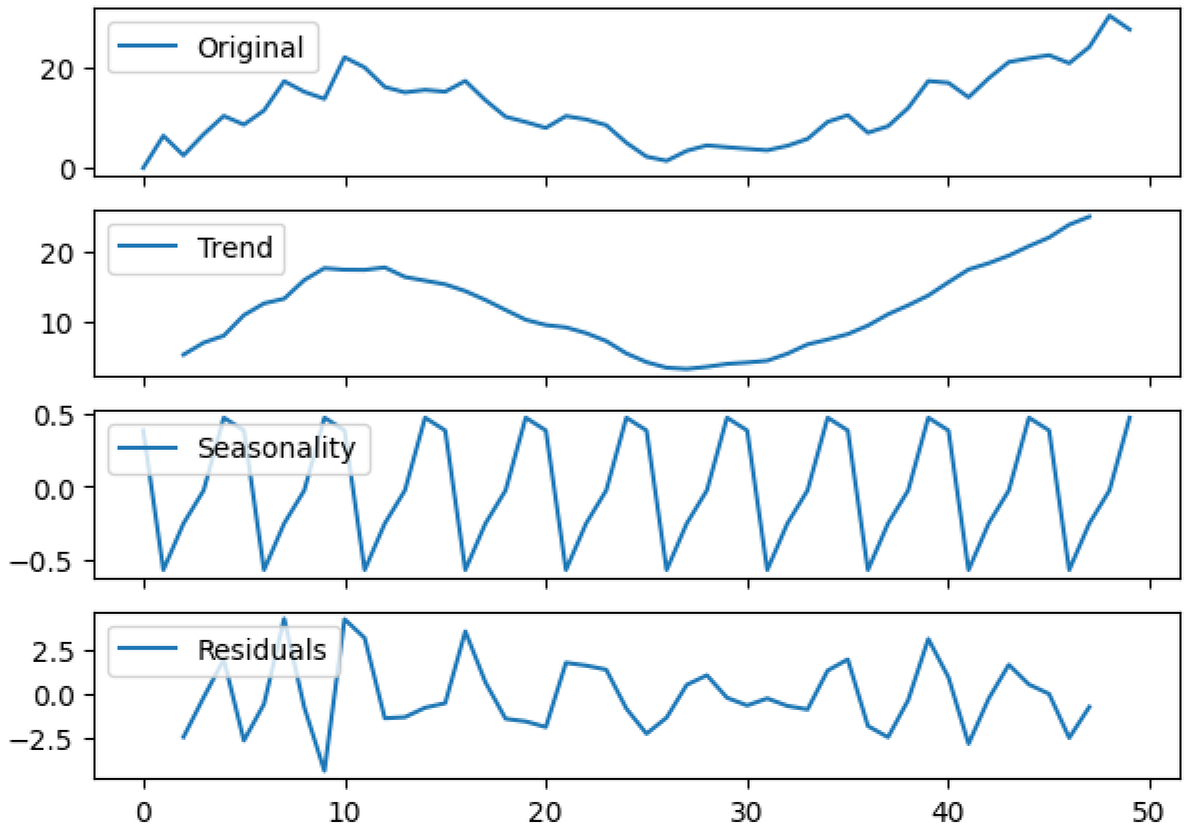
In [6]:
```
ss_decomposition = ses_mul(dataset_A, 10)
estimated_trend = ss_decomposition.trend
estimated_seasonal = ss_decomposition.seasonal
estimated_residual = ss_decomposition.resid

fig, axes = plt.subplots(4, 1, sharex=True, sharey=False)
fig.set_figheight(5)
fig.set_figwidth(7)

axes[0].plot(dataset_A, label='Original')
axes[0].legend(loc='upper left');

axes[1].plot(estimated_trend, label='Trend')
axes[1].legend(loc='upper left');

axes[2].plot(estimated_seasonal, label='Seasonality')
axes[2].legend(loc='upper left');

axes[3].plot(estimated_residual, label='Residuals')
axes[3].legend(loc='upper left');
```



Dataset_B Plot

In [7]:
```
ss_decomposition = ses_add(dataset_B, 5)
estimated_trend = ss_decomposition.trend
estimated_seasonal = ss_decomposition.seasonal
estimated_residual = ss_decomposition.resid

fig, axes = plt.subplots(4, 1, sharex=True, sharey=False)
fig.set_figheight(5)
```

```
fig.set_figwidth(7)

axes[0].plot(dataset_B, label='Original')
axes[0].legend(loc='upper left');

axes[1].plot(estimated_trend, label='Trend')
axes[1].legend(loc='upper left');

axes[2].plot(estimated_seasonal, label='Seasonality')
axes[2].legend(loc='upper left');

axes[3].plot(estimated_residual, label='Residuals')
axes[3].legend(loc='upper left');
```



Exercise #2.1

In this exercise, your task is to:

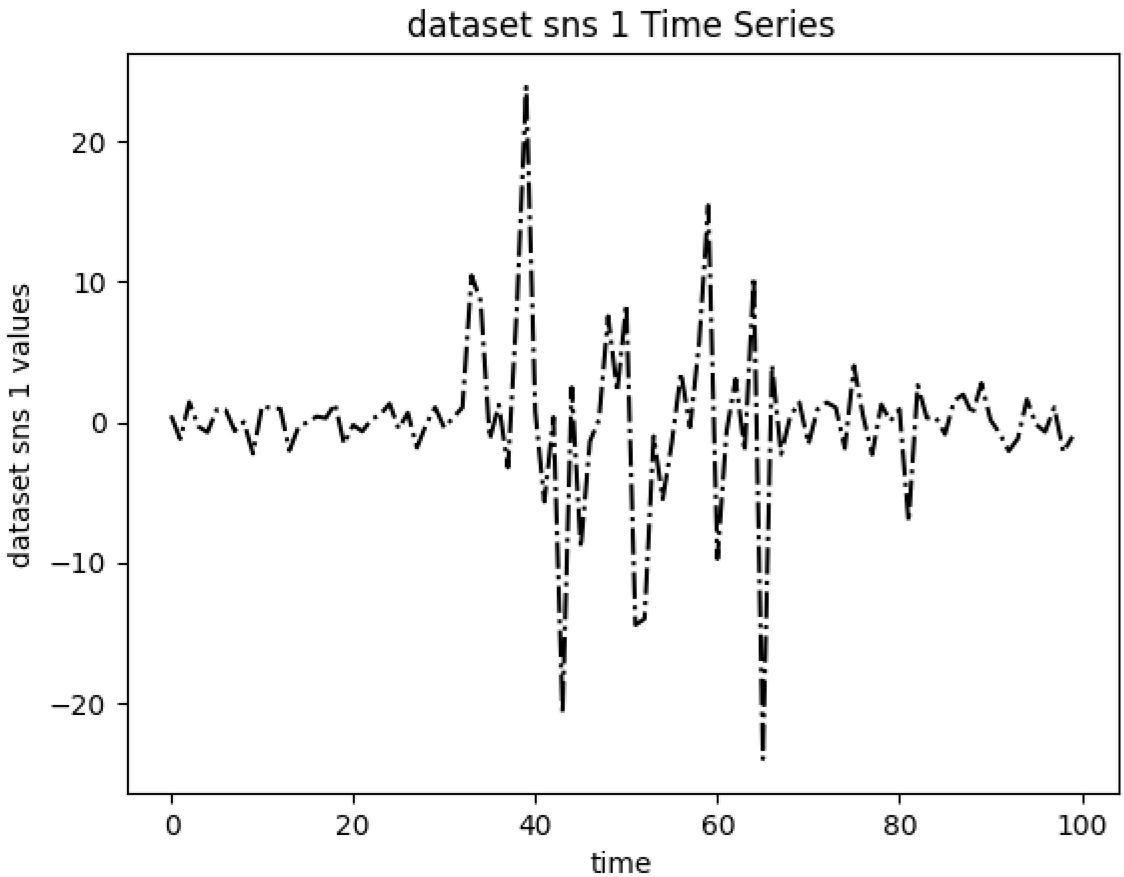
1. Create a time variable called **mytime** that is composed of the integers from 0 to 99 inclusive.
2. Read in **dataset_SNS_1.npy** and **dataset_SNS_2.npy** as **dataset_SNS_1** and **dataset_SNS_2**, respectively.
3. Plot each time series dataset.
4. Start thinking about whether each is stationary or nonstationary.

In [8]:
```
# create time variable

time = np.arange(0, 100)
```

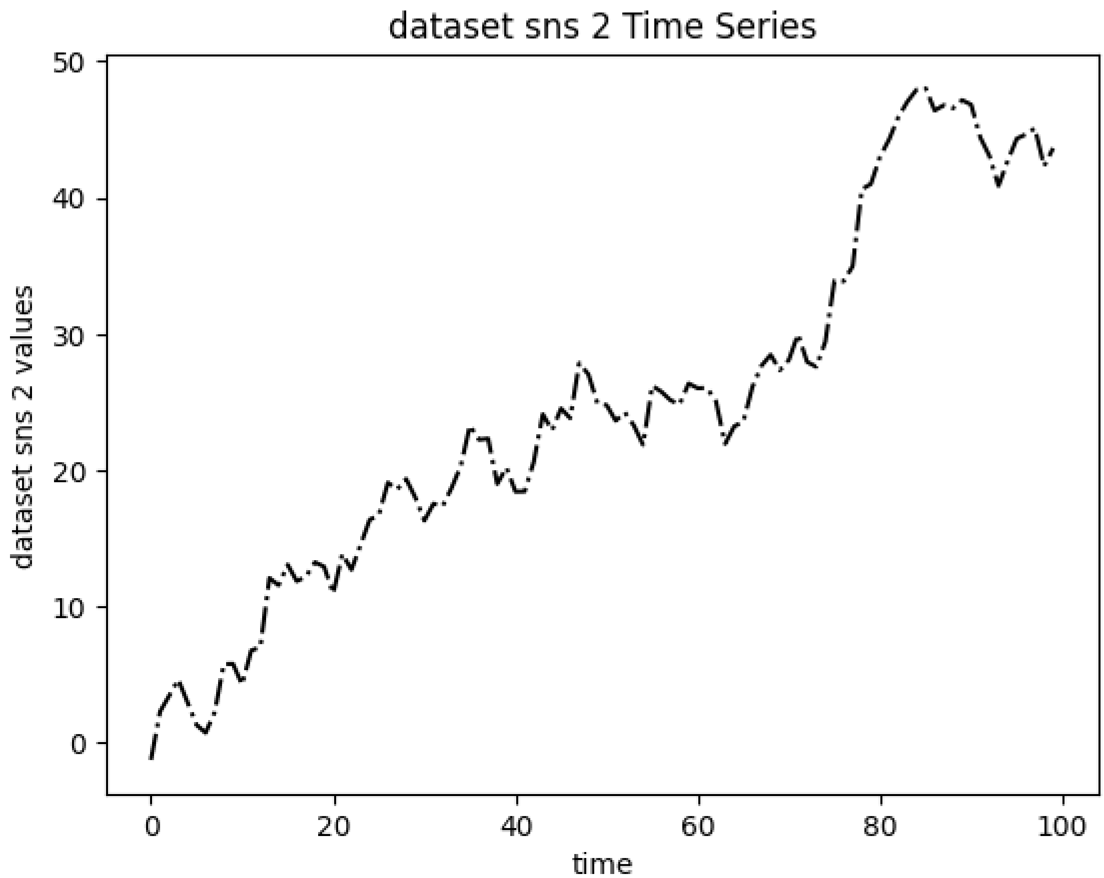
In [9]:
```
# get data
path_to_file = "./" # Modify if data are in a different directory
```

```
dataset_SNS_1 = np.load(path_to_file + "dataset_SNS_1.npy")
dataset_SNS_2 = np.load(path_to_file + "dataset_SNS_2.npy")
```

In [10]:
```
# plot dataset_SNS_1



plt.plot(time, dataset_SNS_1, 'k-.')
plt.title("dataset sns 1 Time Series")
plt.xlabel("time")
plt.ylabel("dataset sns 1 values");
```



In [11]:
```
# plot dataset_SNS_2

plt.plot(time, dataset_SNS_2, 'k-.')
plt.title("dataset sns 2 Time Series")
plt.xlabel("time")
plt.ylabel("dataset sns 2 values");
```



Your Preliminary Thoughts

Are both datasets stationary or is one stationary and one nonstationary or are both nonstationary?

dataset 2 appears most definitely to be nonstationary because of clear trend, while dataset 1 appears to have heteroscedacity. It also appears to be non stationary

---

Exercise #2.2

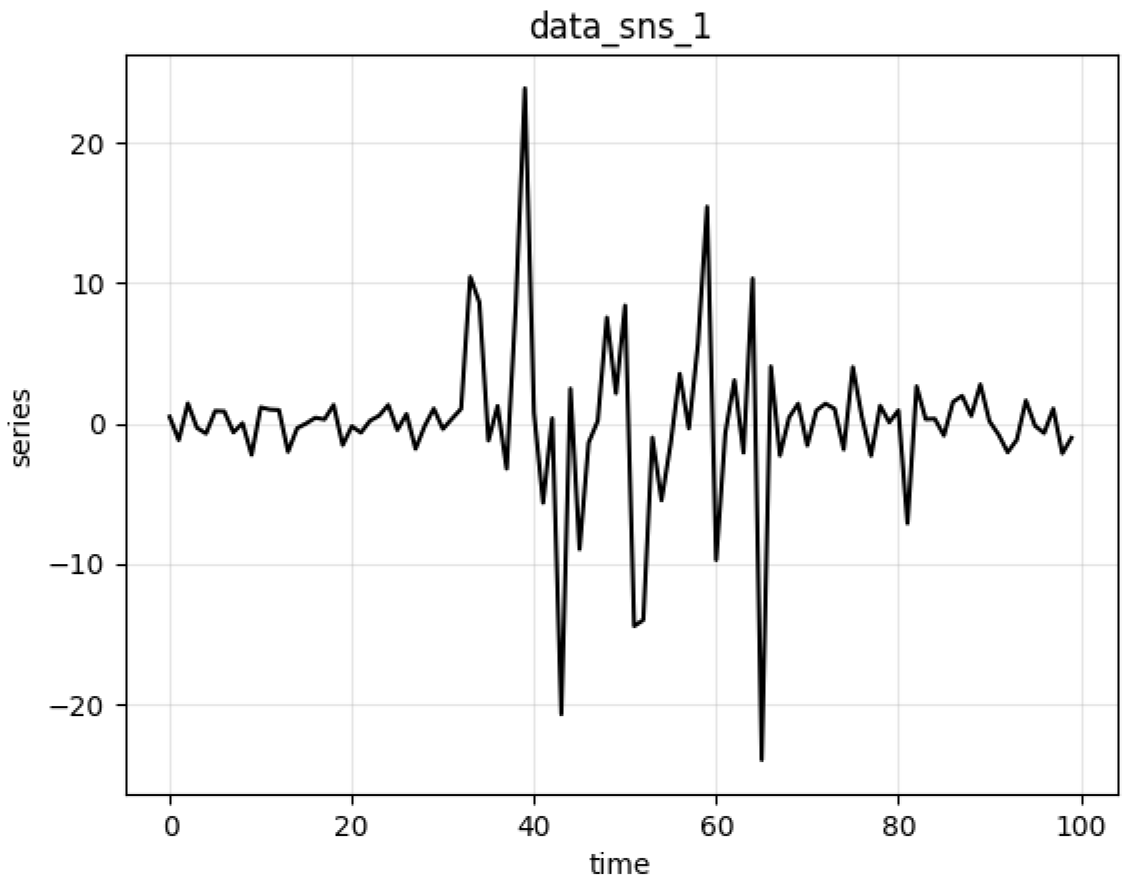
Think back to the two datasets from Exercise #2.1.

You should have the tools to answer whether each is stationary or not.

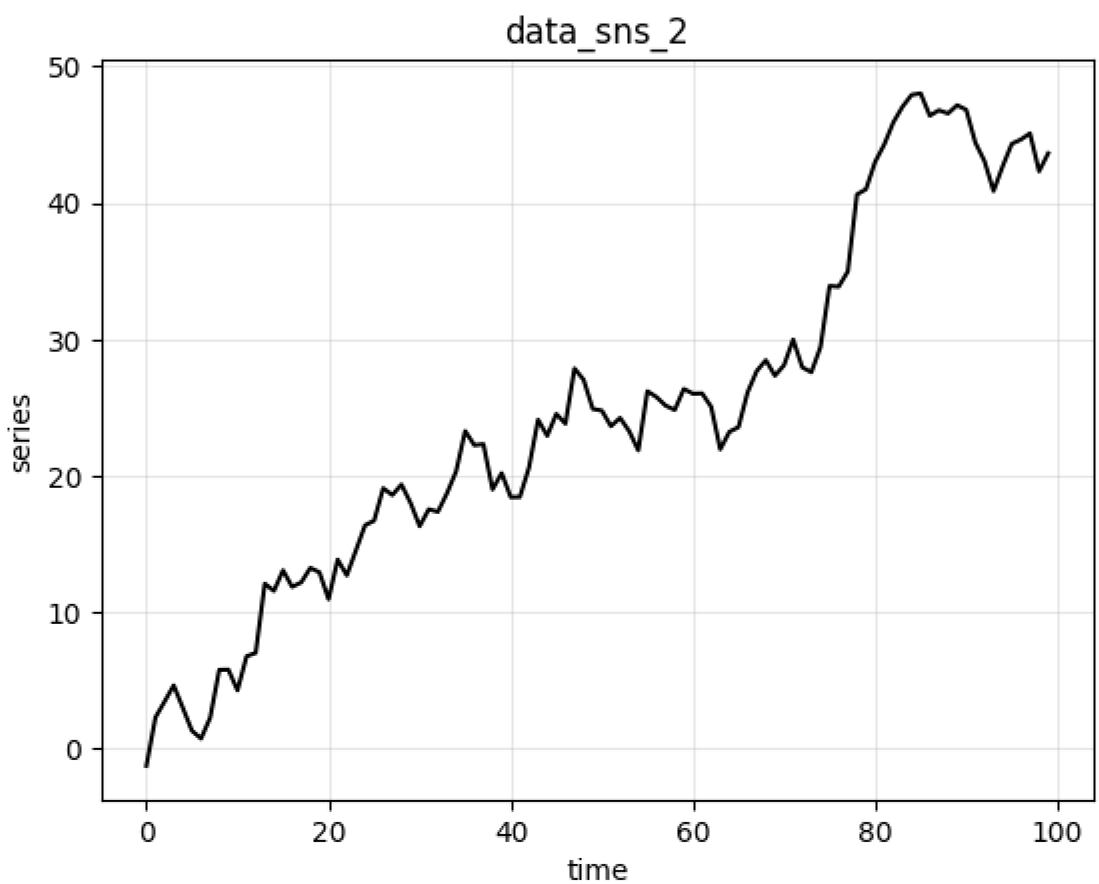
Provide your answers and explanations below.

In [12]:
```
def run_sequence_plot(x, y, title, xlabel="time", ylabel="series"):
    plt.plot(x, y, 'k-')
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.grid(alpha=0.3);
```

In [13]:
```
# run-sequence plots

run_sequence_plot(time, dataset_SNS_1,
                  title="data_sns_1")
```



In [14]:
```
run_sequence_plot(time, dataset_SNS_2,
                  title="data_sns_2")
```



Explanation:

the observation is similar to the preliminary observation recorded in the first exercise, that being dataset 2 appearing as non stationary and dataset 1 appearing to be non-stationary due to it not having constant variance although appearig to have consrtant mean.

In [15]:
```
# chunked stats

# split data into 10 chunks
chunks_1 = np.split(dataset_SNS_1, indices_or_sections=10)

chunks_2 = np.split(dataset_SNS_2, indices_or_sections=10)

np.mean(chunks_1, axis=1)
```

Out[15]:
```
array([-0.14368349,  0.121089  ,  0.04714784,  4.92083495, -2.32626967,
       -0.3706503 , -1.95084875,  0.34634898,  0.30483126, -0.51907842])
```

In [16]:
```
np.var(chunks_1, axis=1)
```

Out[16]:
```
array([ 1.1064868 ,  1.16521661,  0.76635153, 59.91993023, 55.94682032,
       79.168351  , 77.81561226,  3.18839451,  7.29302955,  1.34694601])
```

In [17]:
```
np.mean(chunks_2, axis=1)
```

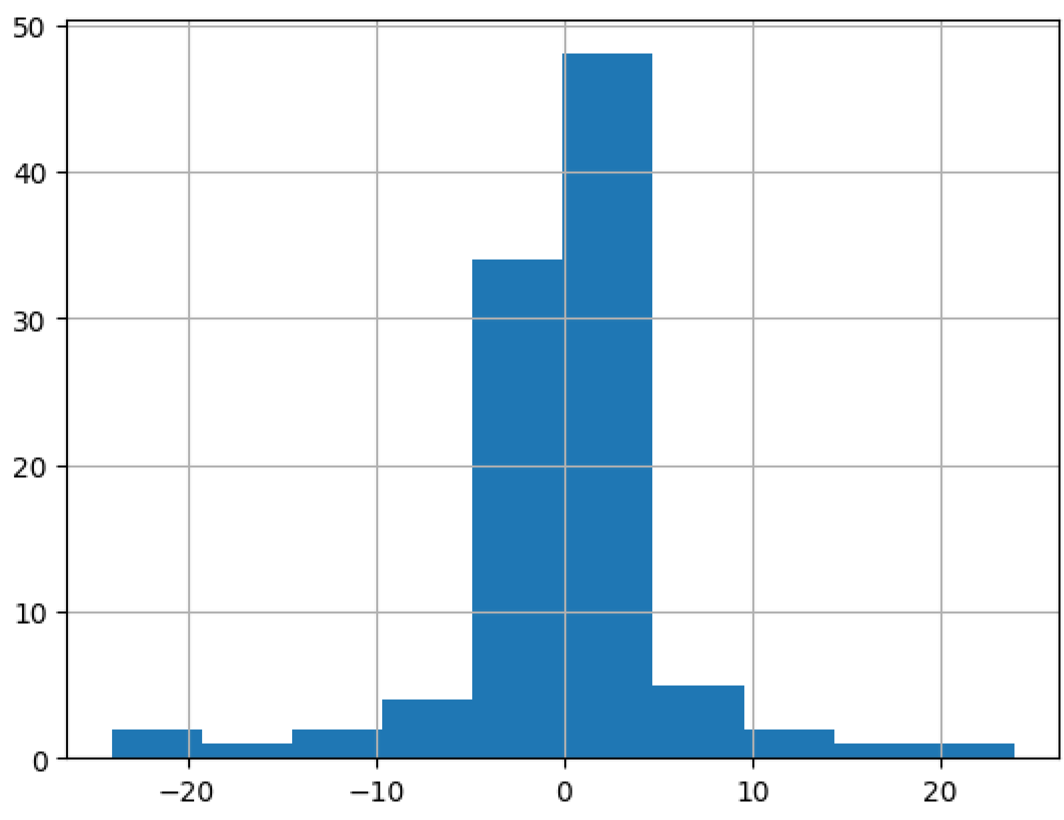
Out[17]:
```
array([ 2.8186155 , 10.51083038, 16.02252814, 19.73251898, 23.26775272,
       24.60790371, 25.54323527, 32.73577811, 46.28942898, 43.77348672])
```

In [18]:
```
np.var(chunks_2, axis=1)
```

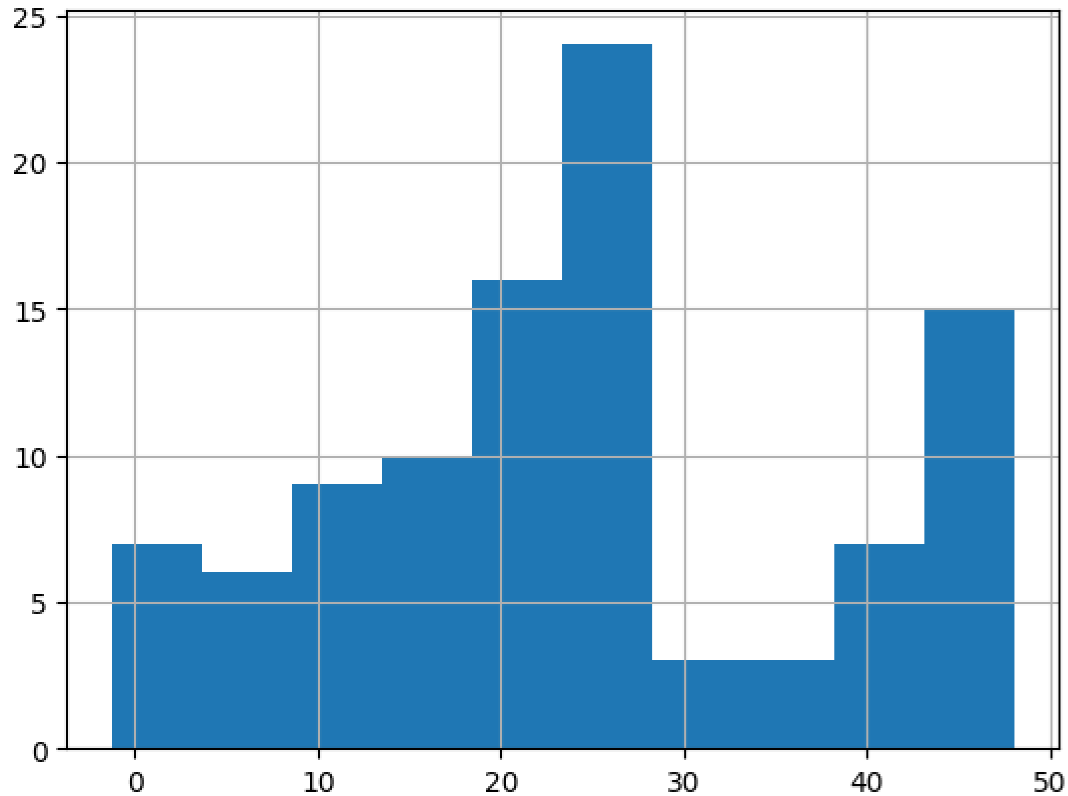
Out[18]:
```
array([ 4.50258195,  9.30343813,  7.51911006,  4.97894604,  9.42707844,
        1.77171835,  3.96080036, 22.71299455,  2.22467478,  2.47819651])
```

varying means and variances from both the datasets makes them out to be non stationary

In [19]:
```
# histograms
pd.Series(dataset_SNS_1).hist();
```



In [20]:
```
pd.Series(dataset_SNS_2).hist();
```



Explanation:

the second dataset is clearly not a normal distribution proving its non stationary property, while the first data set appears to be closely normal distributed suggesting it may be stationary.

In [21]:
```
# ADF tests

from statsmodels.tsa.stattools import adfuller

adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(dataset_SNS_1)

print("ADF: ", adf)
print("p-value for 1st data set:", pvalue)
```

```
ADF:  -3.032415903501602
p-value for 1st data set: 0.03197606455861599
```

In [22]:
```
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(dataset_SNS_2)

print("ADF: ", adf)
print("p-value for 2nd data set:", pvalue)
```

```
ADF:  -1.3222642986946496
p-value for 2nd data set: 0.6189258221979334
```

Explanation:

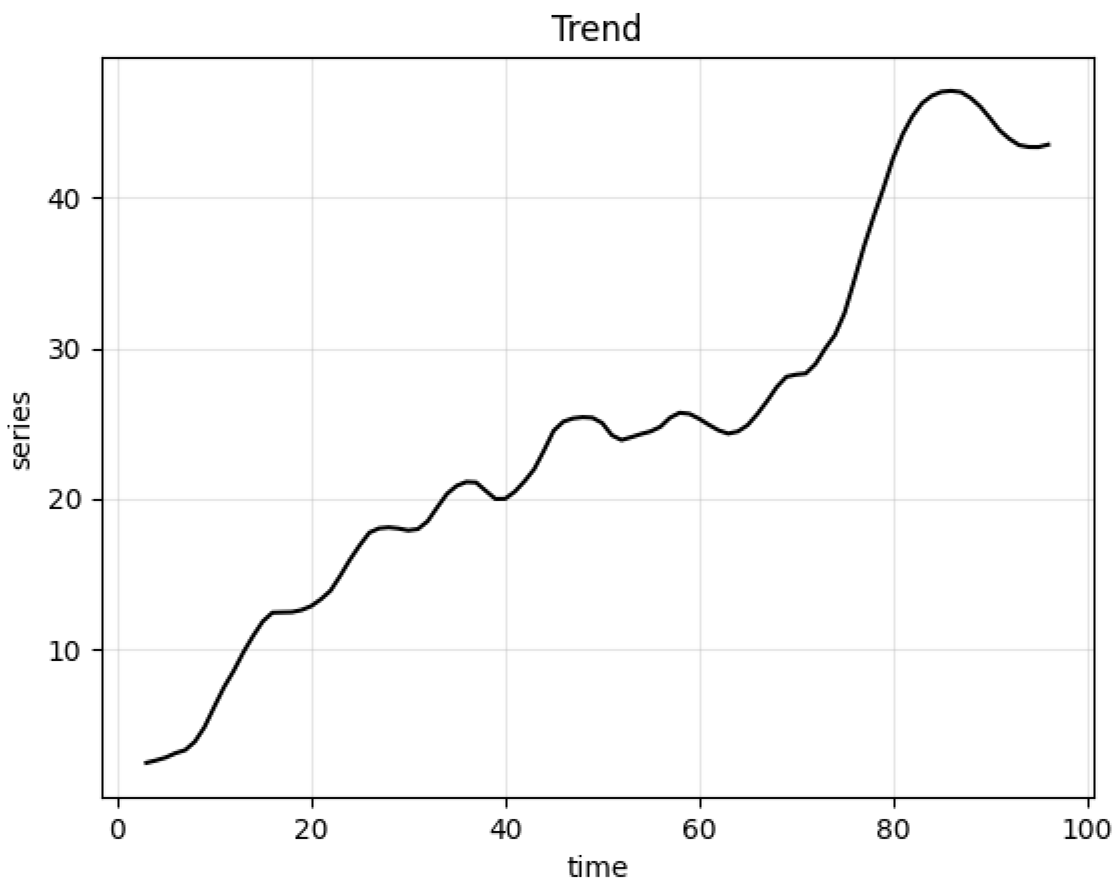
In the second dataset , p-value being well over 0.05 implies that we cannot reject the null hypthothesis . Hence dataset 2 is non stationary. on the other we we can reject the null hypthesis in favor of alternate in case of the forst data set as the p-value is below 0.05. Thus the first dataset can be considered stationary.

Exercise #2.3

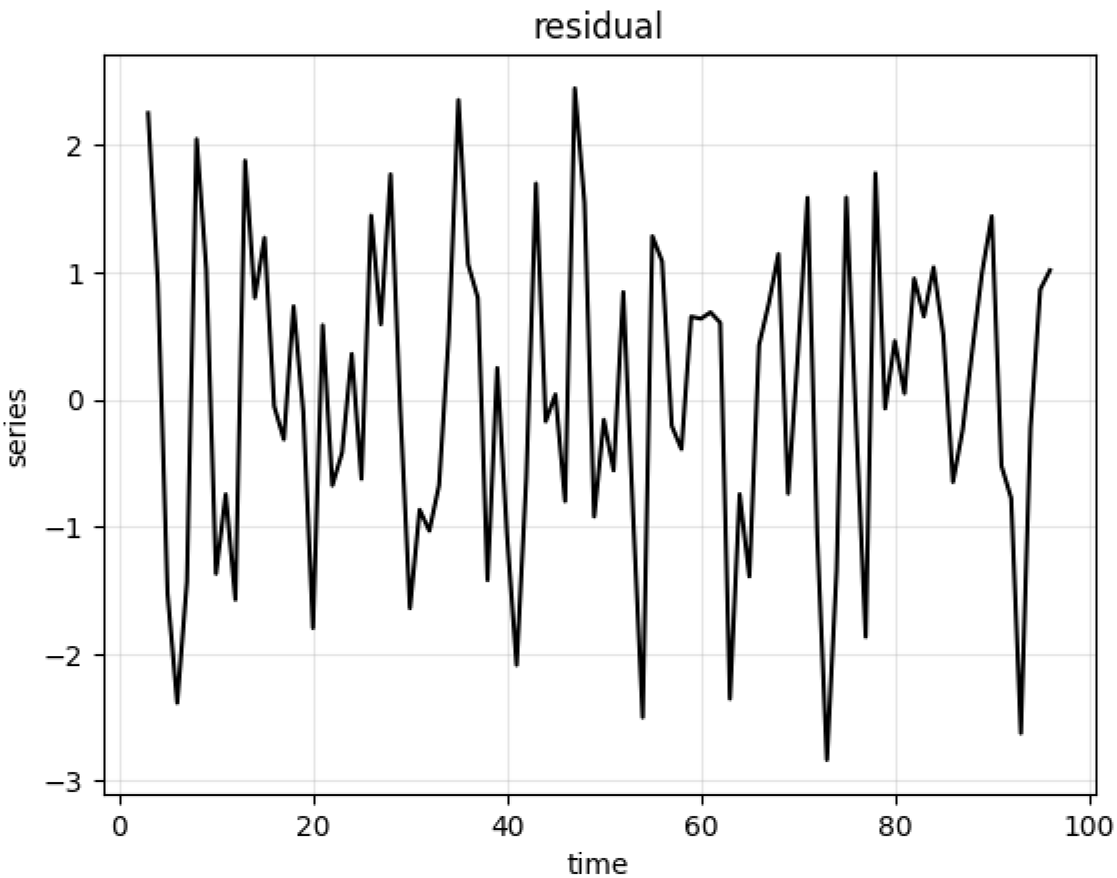
If either or both datasets from exercises one and two are nonstationary, apply the transformations you learned in this section to make them so. Then apply the methods you learned to ensure stationarity.

In [23]:
```
#removing heteroscedasticity in dataset 1 by taking log

#removing trend in dataset 2 by statsmodel

from statsmodels.tsa.seasonal import seasonal_decompose

ss_decomposition = seasonal_decompose(x=dataset_SNS_2, model='additive', period=6)
est_trend = ss_decomposition.trend
est_seasonal = ss_decomposition.seasonal
est_residual = ss_decomposition.resid
```

In [24]:
```
run_sequence_plot(time, est_trend, title="Trend", ylabel="series")
```



In [25]:
```
run_sequence_plot(time, est_residual, title="residual", ylabel="series")
```



In [26]:
```
print(est_residual)
```

```
[         nan          nan          nan  2.25164332  0.91311651 -1.5222694
 -2.3861046  -1.43438458  2.04765693  1.02213979 -1.36989103 -0.74622427
 -1.57415546  1.87792254  0.8004592   1.27164413 -0.05283683 -0.31420964
  0.7321111  -0.09920165 -1.79714322  0.58406459 -0.67627894 -0.41738726
  0.3586735  -0.62448903  1.44611184  0.59194224  1.77082168 -0.07267433
 -1.64137035 -0.86710074 -1.03296816 -0.67411847  0.50378415  2.35538619
  1.06488261  0.79967166 -1.4207001   0.2479742  -1.05510569 -2.08826227
 -0.61832126  1.69836598 -0.17246994  0.03988933 -0.79967962  2.44734969
  1.54748355 -0.92071178 -0.15927568 -0.55751889  0.84458616 -0.88716597
 -2.49946731  1.28485174  1.08726871 -0.20703547 -0.39028846  0.65156252
  0.63522219  0.68562812  0.60399827 -2.3539276  -0.74505928 -1.39172098
  0.42817036  0.75334466  1.14428859 -0.73939069  0.34680138  1.58671918
 -1.07113773 -2.83438054 -1.34636137  1.58845154 -0.16128733 -1.86664715
  1.77905834 -0.07086648  0.45967517  0.04808054  0.95101104  0.65333646
  1.03926848  0.50875097 -0.64876726 -0.2378658   0.39098803  0.99824046
  1.44094873 -0.52168038 -0.77605248 -2.62453756 -0.21924656  0.86540196
  1.01617305         nan         nan         nan]
```

In [27]:
```
adf_after, pvalue_after, usedlag_, nobs_, critical_values_, icbest_ = adfuller(est_resid
print("ADF: ", adf_after)
print("p-value: ", pvalue_after)
```

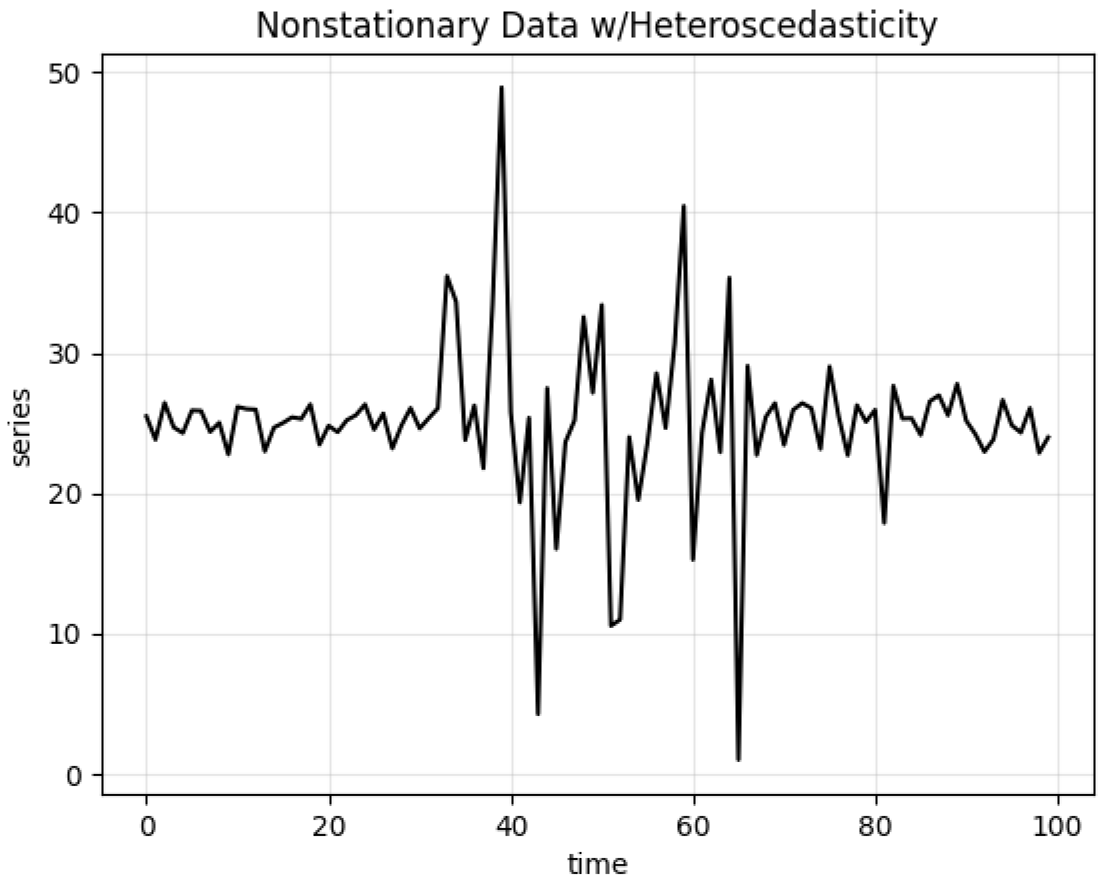
```
ADF:  -6.238390718569909
p-value:  4.763184090747469e-08
```

the p-value is sufficiently close to zero to reject the null hypothesis and consider the resiodual data as stationary.

removing heteroscedasticity in dataset 1 by taking log

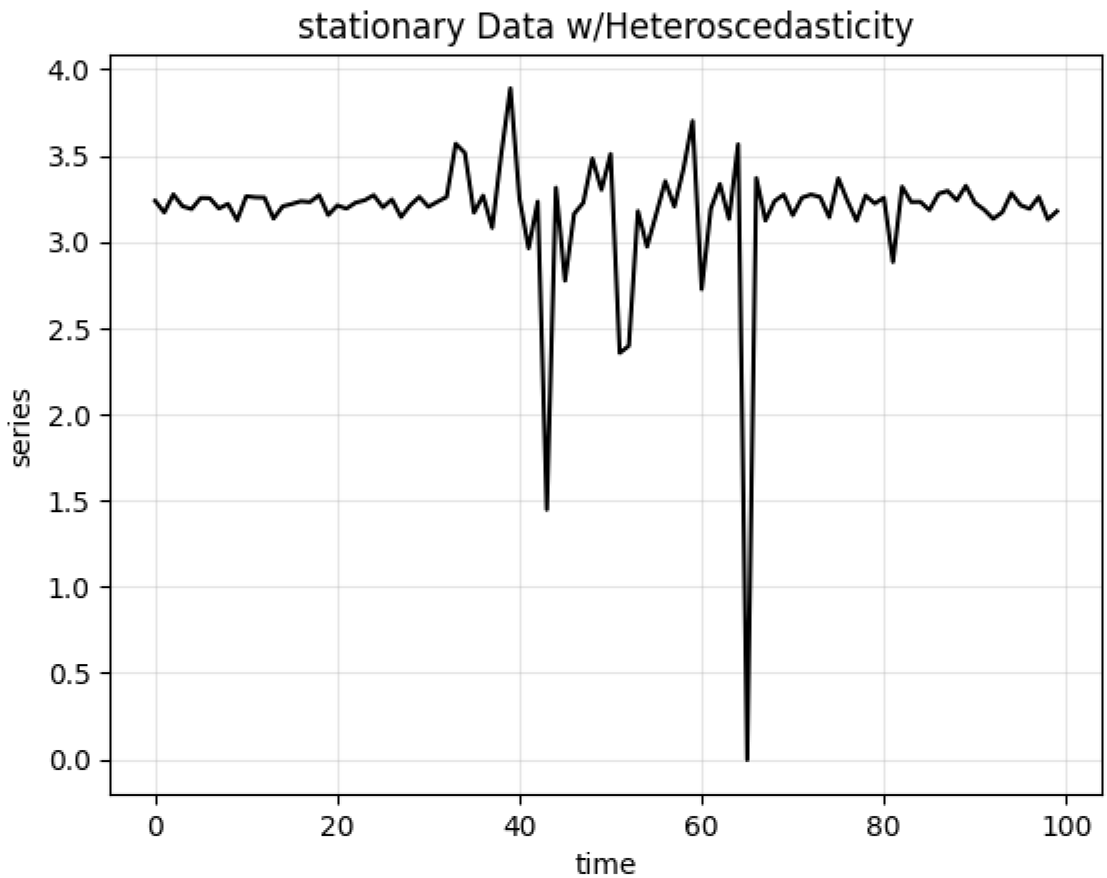
In [28]:
```
positive_data = dataset_SNS_1 + 25
```

```
run_sequence_plot(time, positive_data,
                  title="Nonstationary Data w/Heteroscedasticity")
```



In [29]:
```
log_new_pos_data = np.log(positive_data)

run_sequence_plot(time, log_new_pos_data,
                  title="stationary Data w/Heteroscedasticity")

adf_after, pvalue_after, usedlag_, nobs_, critical_values_, icbest_ = adfuller(log_new_p
print("ADF: ", adf_after)
print("p-value: ", pvalue_after)
```

```
ADF:  -10.750411172845046
p-value:  2.674288989362179e-19
```



the first data set has relatively less heteroscedascity.

Exercise #3.1

You have been provided two datasets:

1. **smooth_1.npy**
2. **smooth_2.npy**

Your task is to leverage what you've learned in this and previous courses.

More specifically, you will do the following:

1. Read in **smooth_1.npy** and **smooth_2.npy**.
2. Create a time variable called **mytime** that starts at 0 and is as long as each dataset.
3. Split each dataset into train and test sets (test set is last 5 observations).
4. Identify trend and seasonality, if present.
5. Identify if trend and/or seasonality are additive or multiplicative, if present.
6. Create smoothed model on the train set and use to forecast on the test set.
7. Calculate MSE on test data.
8. Plot training data, test data, and your model's forecast for each dataset.

1. Get Data

In [30]:
```
# get data
path_to_file = "./"
```

```
smooth_1 = np.load(path_to_file + "smooth_1.npy")
smooth_2 = np.load(path_to_file + "smooth_2.npy")

# Find the length of smooth_1 array
length_smooth_1 = len(smooth_1)

# Find the length of smooth_2 array
length_smooth_2 = len(smooth_2)

print("Length of smooth_1:", length_smooth_1)
print("Length of smooth_2:", length_smooth_2)
```

```
Length of smooth_1: 144
Length of smooth_2: 1000
```

2. Create mytime

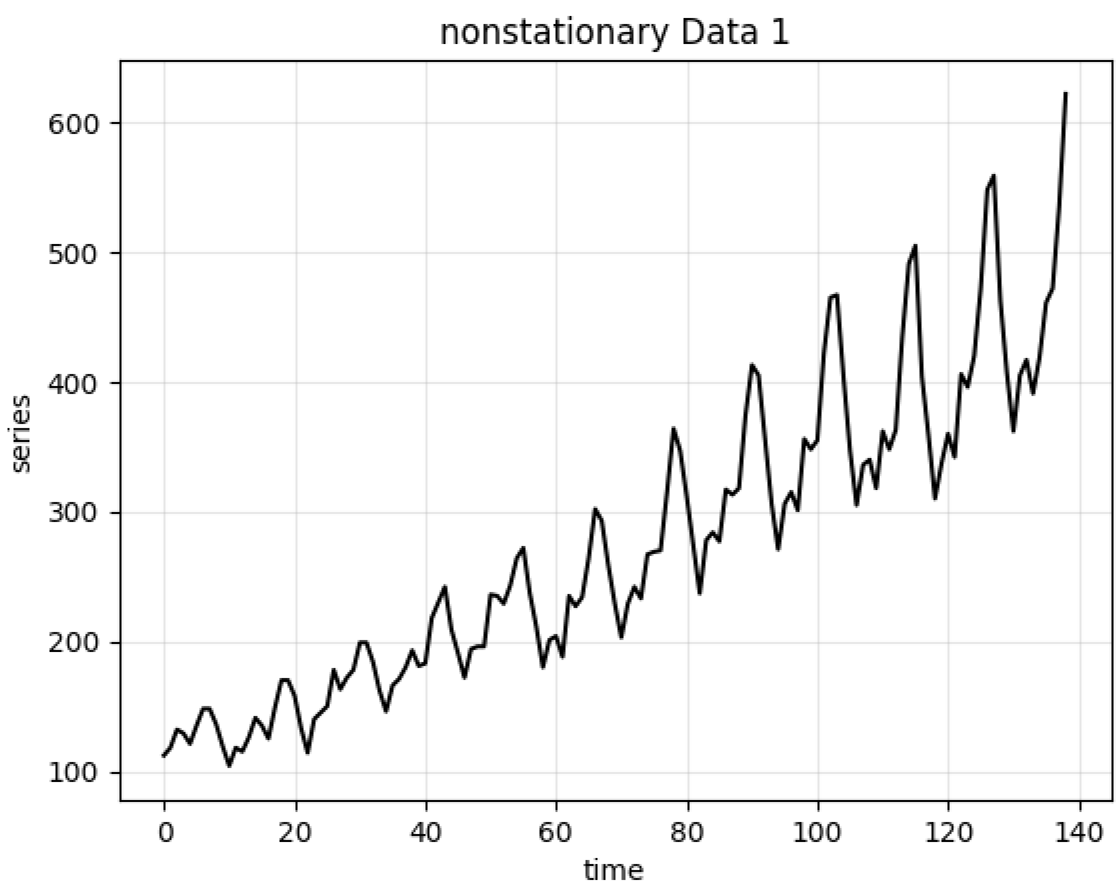
In [31]:
```
mytime = np.arange(0, length_smooth_1)
mytime_2 = np.arange(0, length_smooth_2)
```

3. Train/Test Split

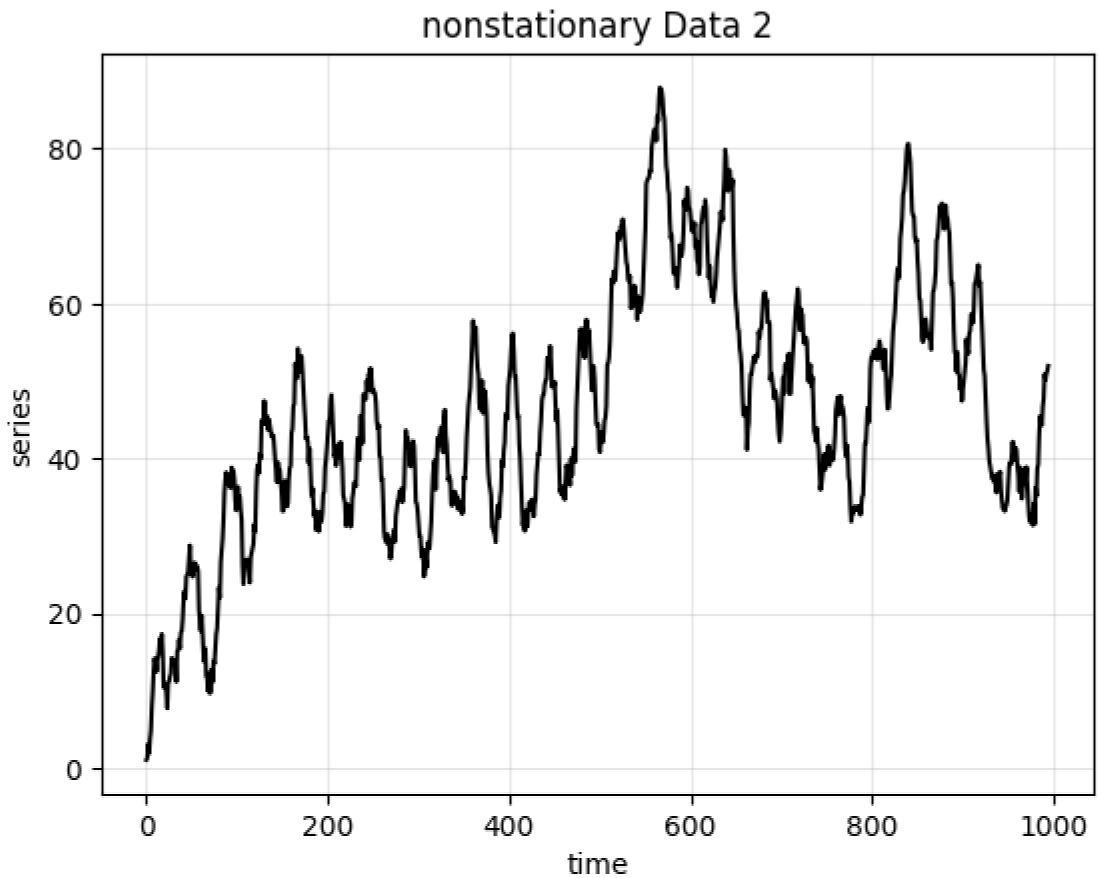
In [32]:
```
train_1 = smooth_1[:-5]
test_1 = smooth_1[-5:]

train_2 = smooth_2[:-5]
test_2 = smooth_2[-5:]
```

4. ID Trend / Seasonality

In [33]:
```
run_sequence_plot(mytime[:-5], train_1,
                  title="nonstationary Data 1")
```



In [34]:
```
run_sequence_plot(mytime_2[:-5], train_2,
                  title="nonstationary Data 2")
```



5. ID Additive vs Multicplicative

from the plots, by running a visual plot, the first time series appears to be multiplicative model.

from the plots, by running a visual plot , the second time series appears to be additive model.

6. Create Smoothed Models

In [35]:
```
from statsmodels.tsa.api import ExponentialSmoothing

triple_1 = ExponentialSmoothing(train_1,
                                trend="additive",
                                seasonal="multiplicative",
                                seasonal_periods=12).fit(optimized=True)
```

In [36]:
```
triple_2 = ExponentialSmoothing(train_2,
                                trend="additive",
                                seasonal="additive",
                                seasonal_periods=35).fit(optimized=True)
```

7. Calculate MSE

In [37]:
```
def mse(observations, estimates):
    '''
    INPUT:
        observations - numpy array of values indicating observed values
        estimates - numpy array of values indicating an estimate of values
    OUTPUT:
        Mean Square Error value
```

```python
    '''
    # check arg types
    assert type(observations) == type(np.array([])), "'observations' must be a numpy arr
    assert type(estimates) == type(np.array([])), "'estimates' must be a numpy array"
    # check length of arrays equal
    assert len(observations) == len(estimates), "Arrays must be of equal length"

    # calculations
    difference = observations - estimates
    sq_diff = difference ** 2
    mse = sum(sq_diff)

    return mse
```

In [38]:

```python
triple_preds_1 = triple_1.forecast(len(test_1))
triple_mse_1 = mse(test_1, triple_preds_1)

print("MSE for first dataset : ", triple_mse_1)
```

```
MSE for first dataset :  633.505320669449
```

In [39]:

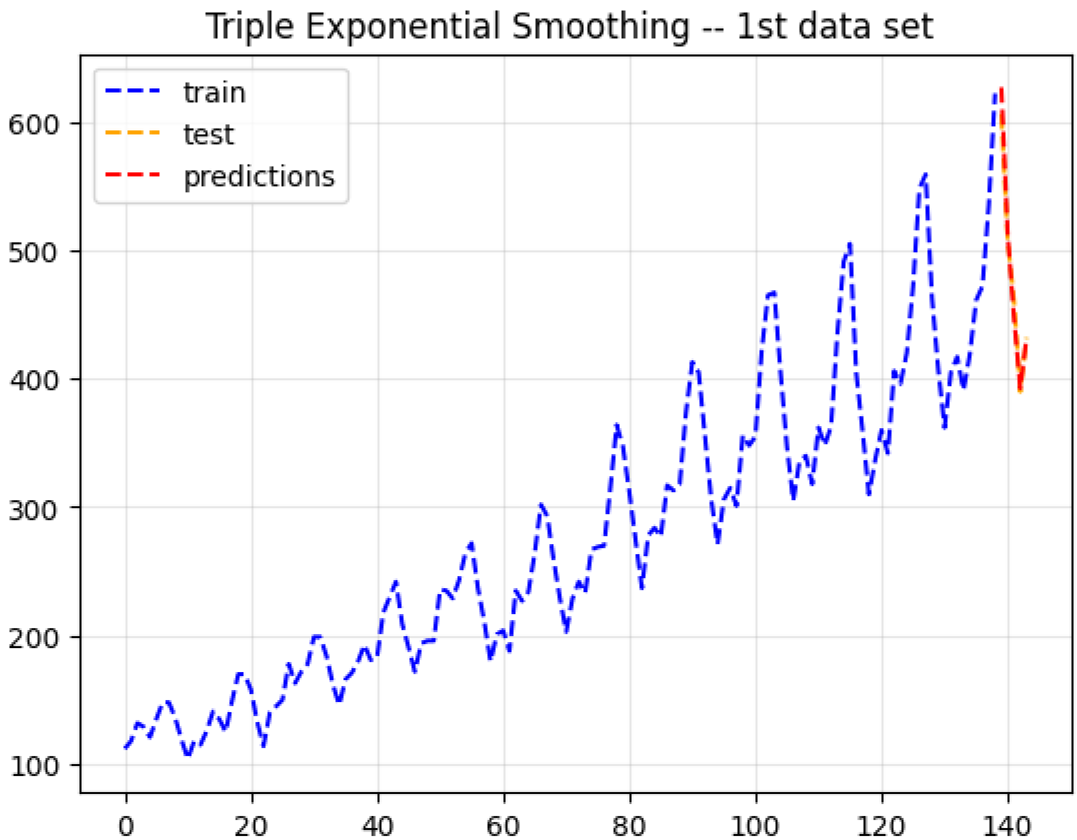
```python
triple_preds_2 = triple_2.forecast(len(test_2))
triple_mse_2 = mse(test_2, triple_preds_2)

print("MSE for second dataset : ", triple_mse_2)
```

```
MSE for second dataset :  68.93466739372548
```

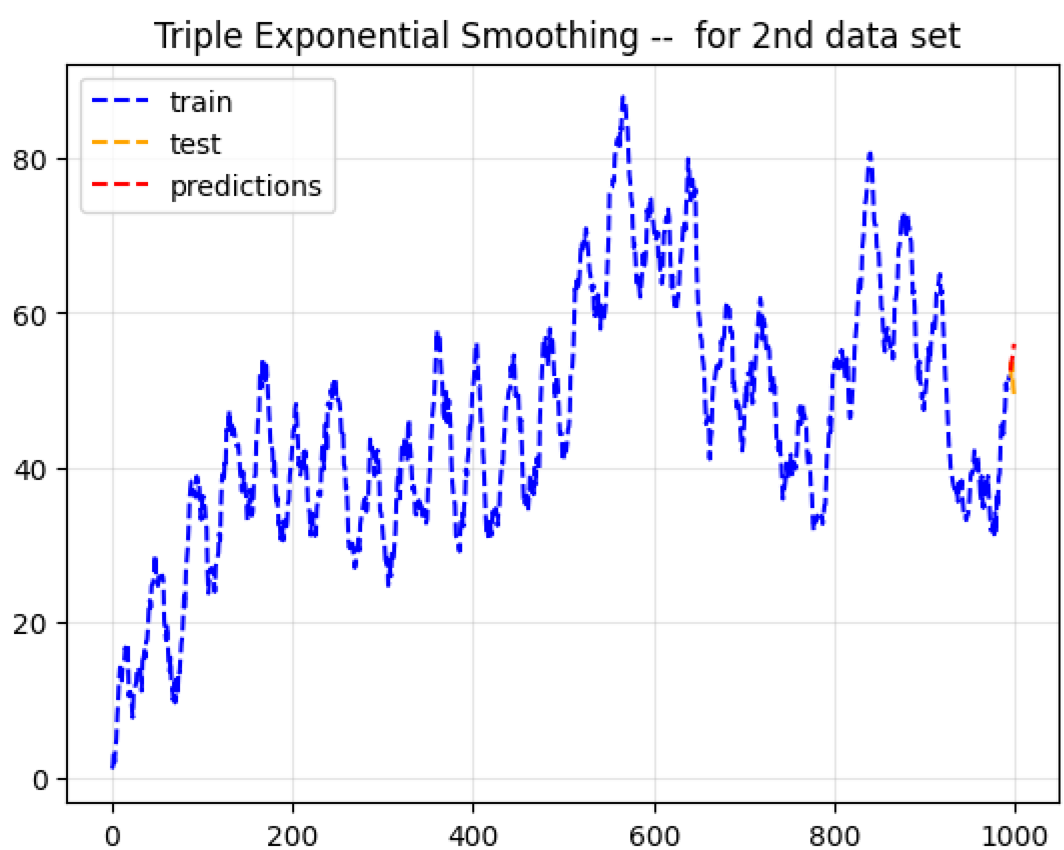
8. Plot Train, Test, Forecast

In [40]:

```python
plt.plot(mytime[:-5], train_1, 'b--', label="train")
plt.plot(mytime[-5:], test_1, color='orange', linestyle="--", label="test")
plt.plot(mytime[-5:], triple_preds_1, 'r--', label="predictions")
plt.legend(loc='upper left')
plt.title("Triple Exponential Smoothing -- 1st data set ")
plt.grid(alpha=0.3);
```



In [41]:
```
plt.plot(mytime_2[:-5], train_2, 'b--', label="train")
plt.plot(mytime_2[-5:], test_2, color='orange', linestyle="--", label="test")
plt.plot(mytime_2[-5:], triple_preds_2, 'r--', label="predictions")
plt.legend(loc='upper left')
plt.title("Triple Exponential Smoothing --  for 2nd data set ")
plt.grid(alpha=0.3);
```



Exercise #4.1

You have been provided two datasets:

1. **auto_1.npy**
2. **auto_2.npy**

Your task is to leverage what you've learned in this and previous lectures.

More specifically, you will do the following:

1. Read in **auto_1.npy** and **auto_2.npy**.
2. Create a time variable called **mytime** that starts at 0 and is as long as both datasets.
3. Generate run-sequence plots of auto_1 and auto_2.
4. Determine the order of p and q.

1. Get Data

In [42]:

```
# get data
path_to_file = "./"
auto_1 = np.load(path_to_file + "auto_1.npy")
auto_2 = np.load(path_to_file + "auto_2.npy")



# Find the length of smooth_1 array
length_auto_1 = len(auto_1)

# Find the length of smooth_2 array
length_auto_2 = len(auto_2)

print("Length of auto_1:", length_auto_1)
print("Length of auto_2:", length_auto_2)
```

```
Length of auto_1: 100
Length of auto_2: 100
```

Create mytime

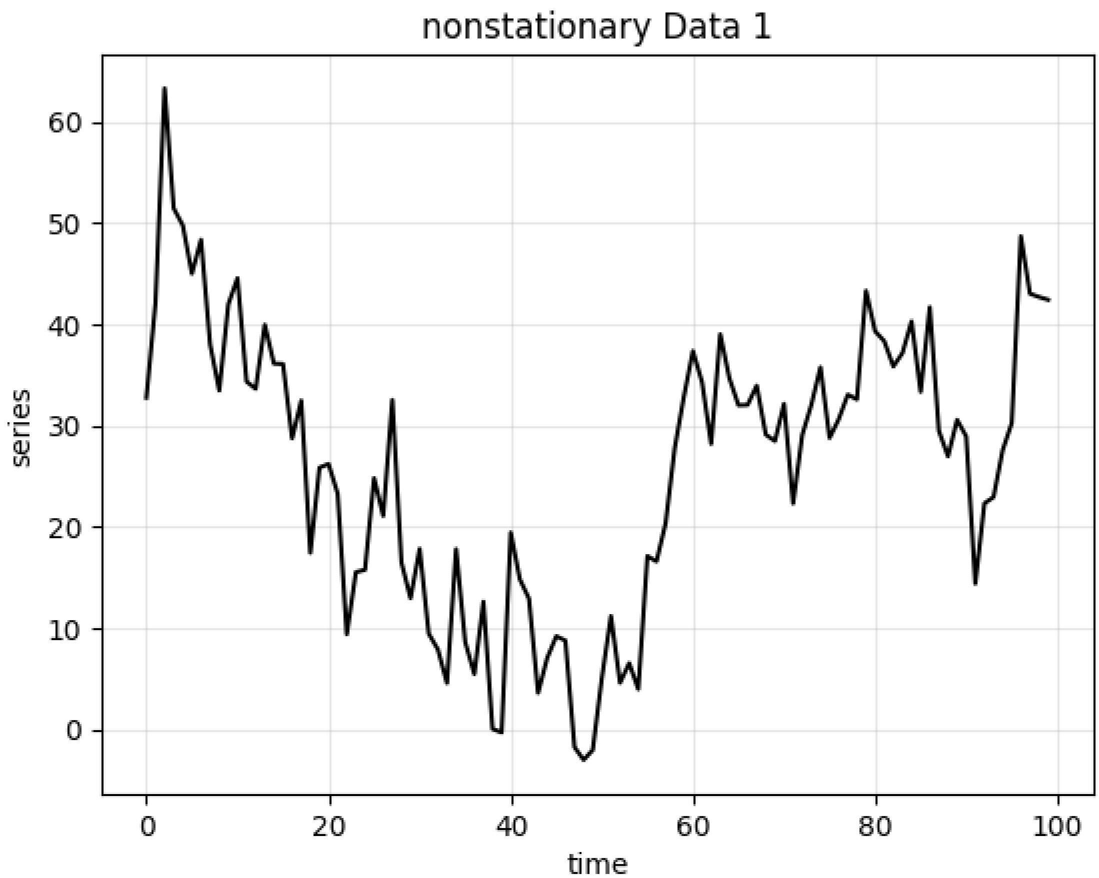
In [43]:

```
# time component


mytime = np.arange(0, length_auto_1)
mytime_2 = np.arange(0, length_auto_2)
```

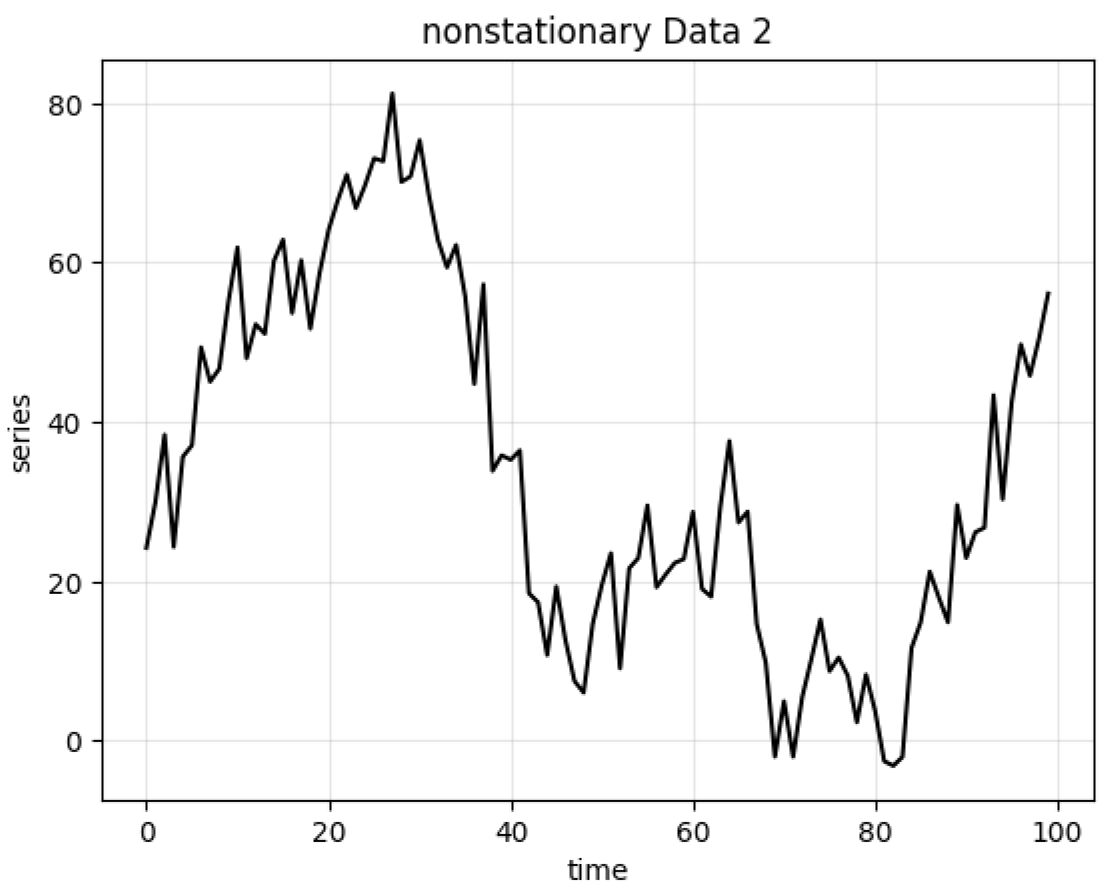
Run-Sequence Plots

In [44]:

```
run_sequence_plot(mytime, auto_1,
                  title="nonstationary Data 1")
```



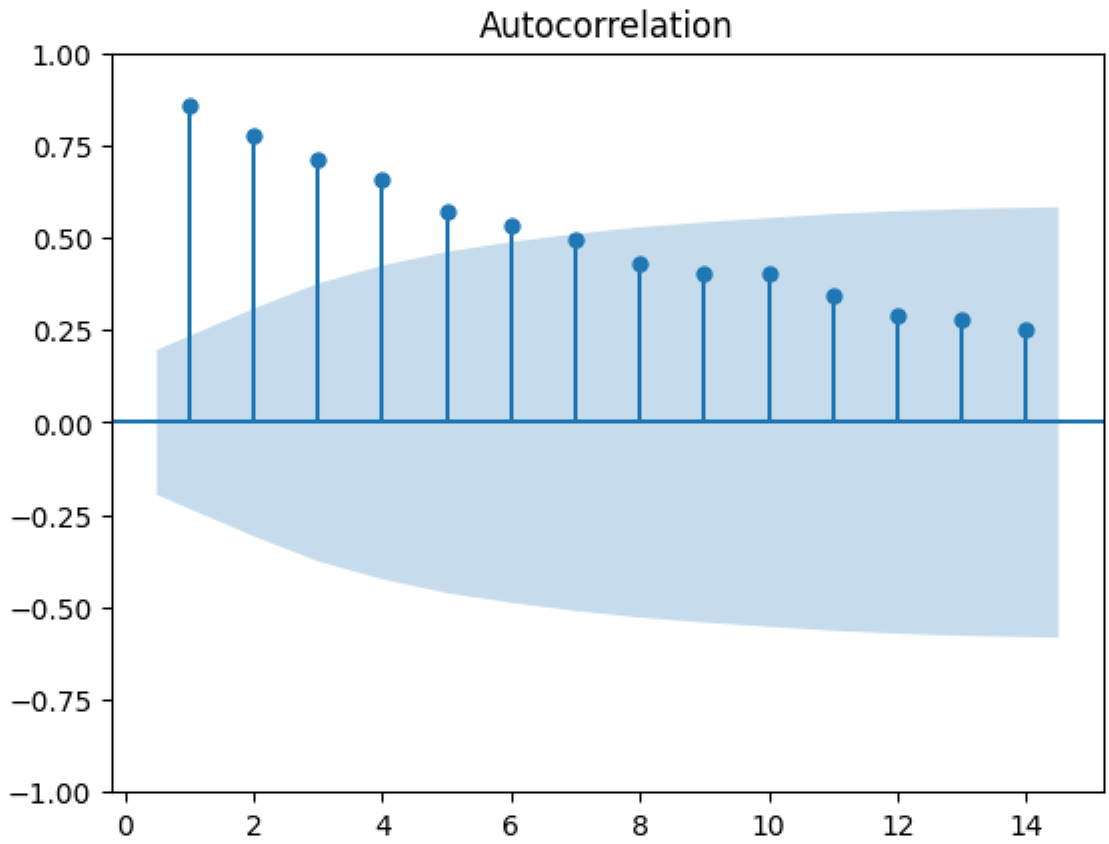
In [45]:
```
run_sequence_plot(mytime_2, auto_2,
                  title="nonstationary Data 2")
```



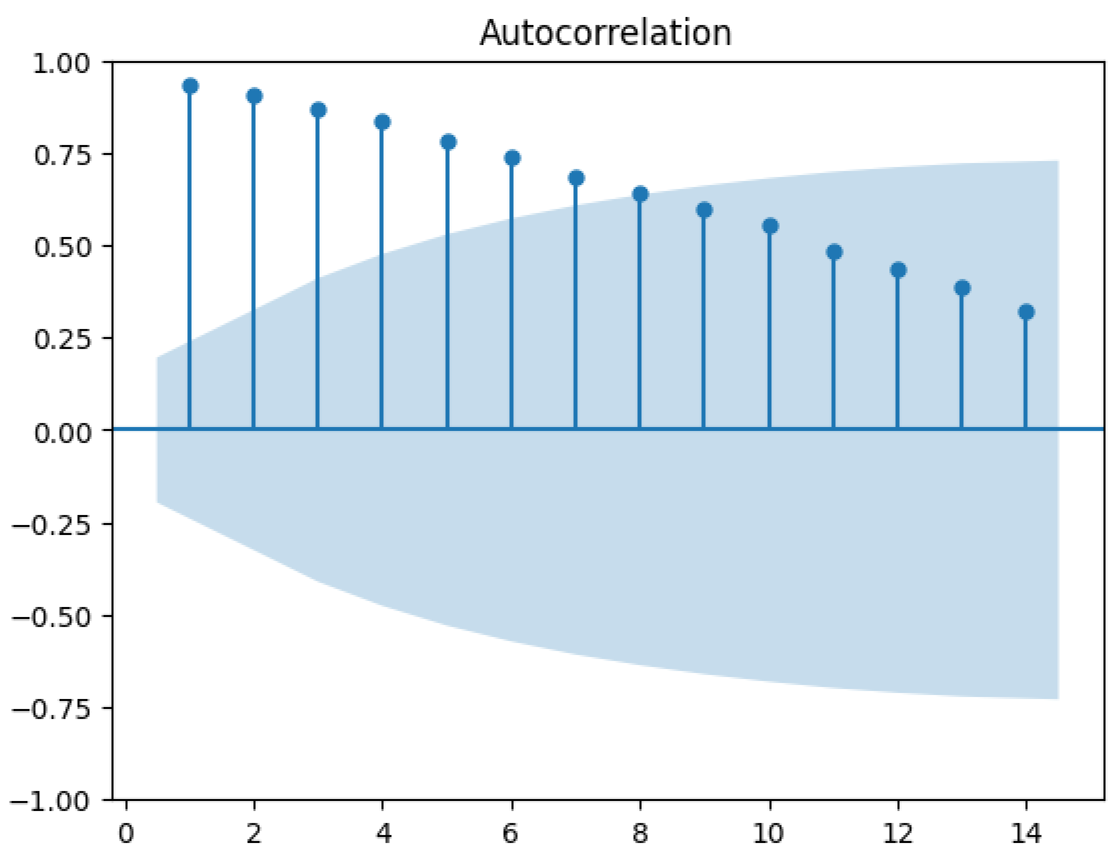
Determine Order (p & q)

In [46]:
```
from statsmodels.graphics.tsaplots import plot_acf
```

```
fig = plot_acf(auto_1, lags=range(1,15), alpha=0.05)
```



In [47]:
```
fig = plot_acf(auto_2, lags=range(1,15), alpha=0.05)
```



In [48]:
```
from statsmodels.tsa.stattools import adfuller

difference_1 = auto_1[:-1] - auto_1[1:]
```

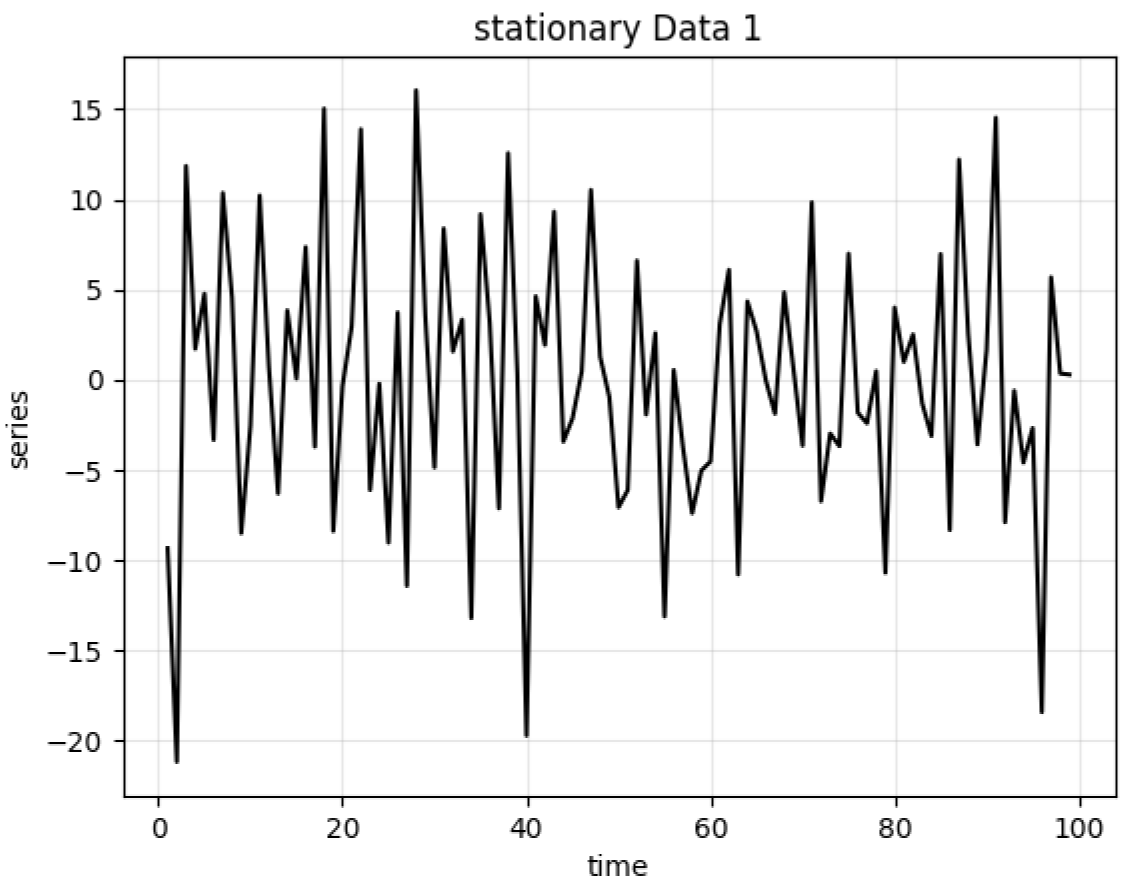
```python
_, pvalue, _, _, _, _ = adfuller(difference_1)
print(f"p-value: {pvalue:.3f}")
print(pvalue)
```

```
p-value: 0.000
3.816622935918145e-24
```

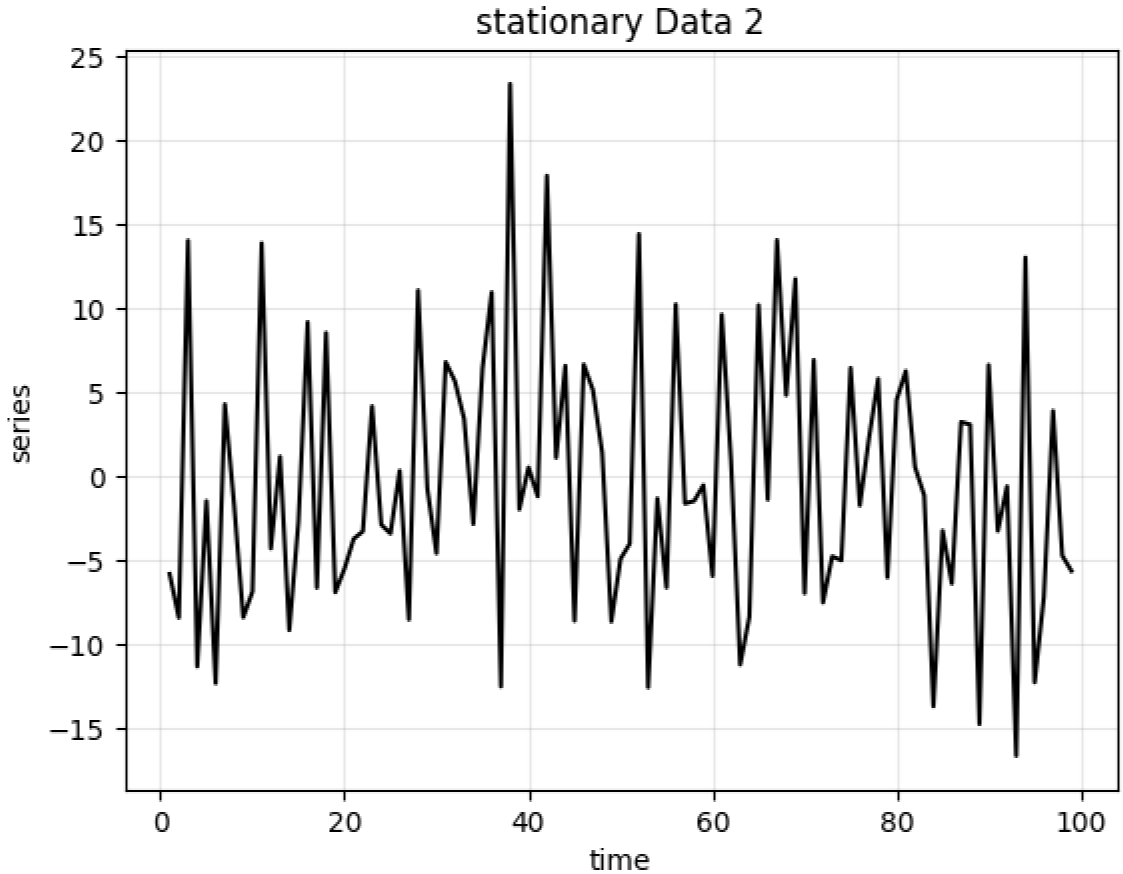
In [49]:
```python
difference_2 = auto_2[:-1] - auto_2[1:]

_, pvalue, _, _, _, _ = adfuller(difference_2)
print(f"p-value: {pvalue:.3f}")
print(pvalue)
```

```
p-value: 0.000
4.667882463338721e-25
```

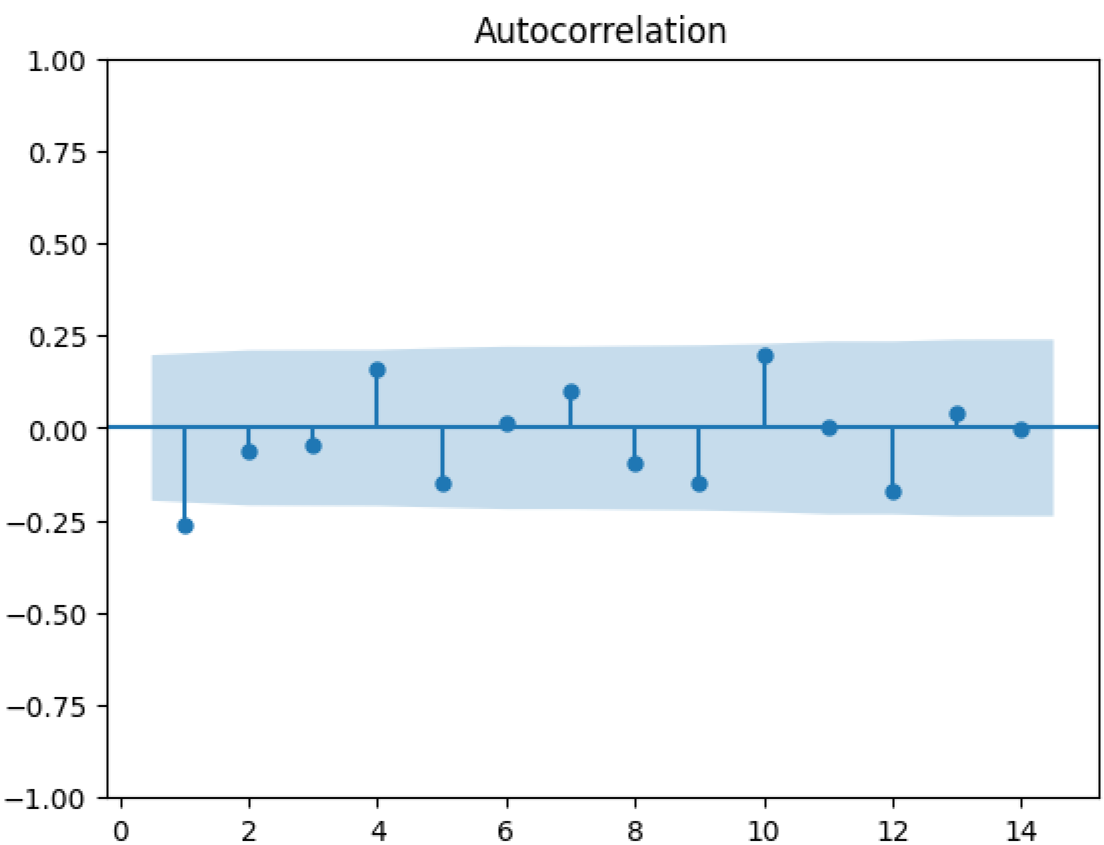
In [50]:
```python
run_sequence_plot(mytime[1:], difference_1,
                  title="stationary Data 1")
```



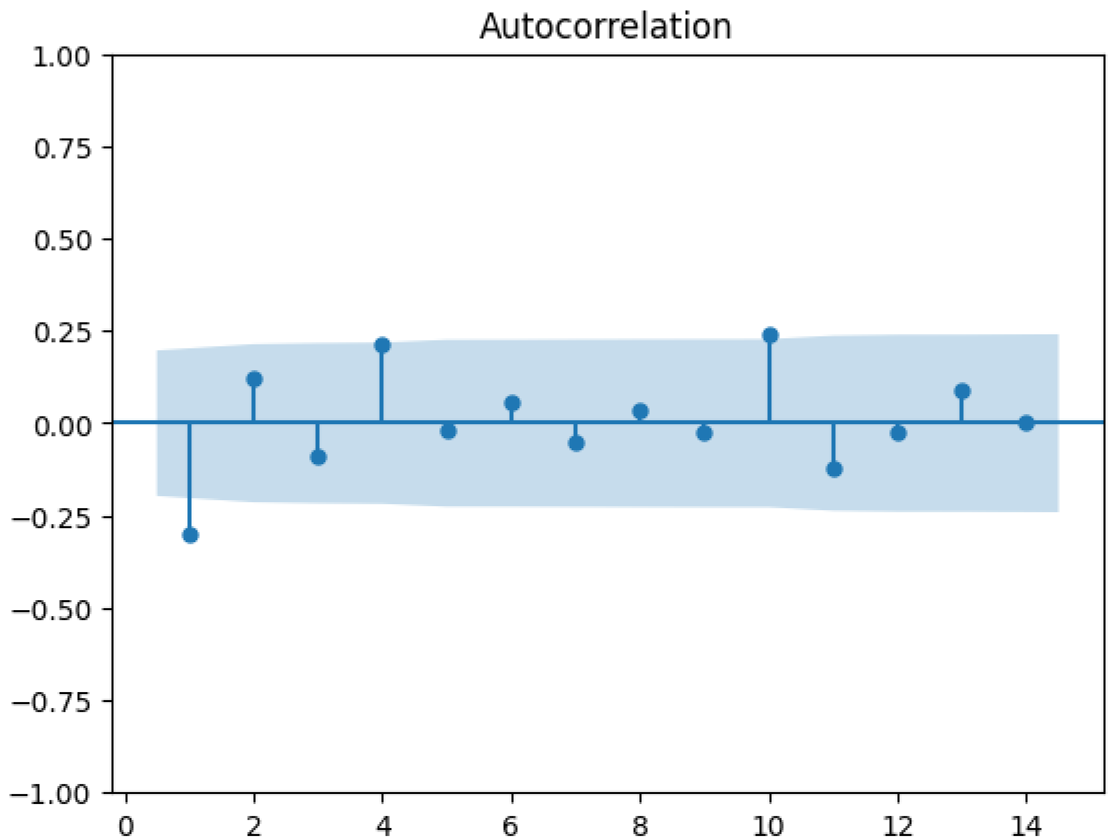
In [51]:
```python
run_sequence_plot(mytime_2[1:], difference_2,
                  title="stationary Data 2")
```



In [52]:
```
fig = plot_acf(difference_1, lags=range(1,15), alpha=0.05)
```

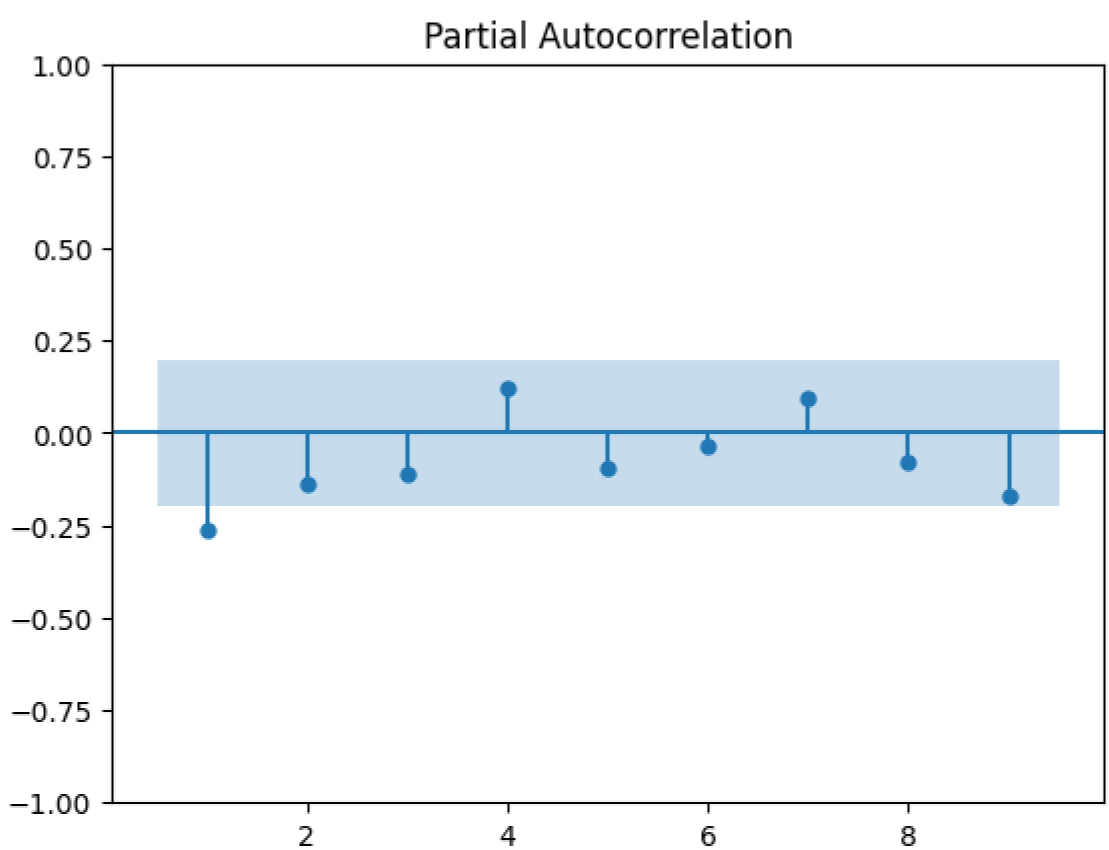


In [53]:
```
fig = plot_acf(difference_2, lags=range(1,15), alpha=0.05)
```

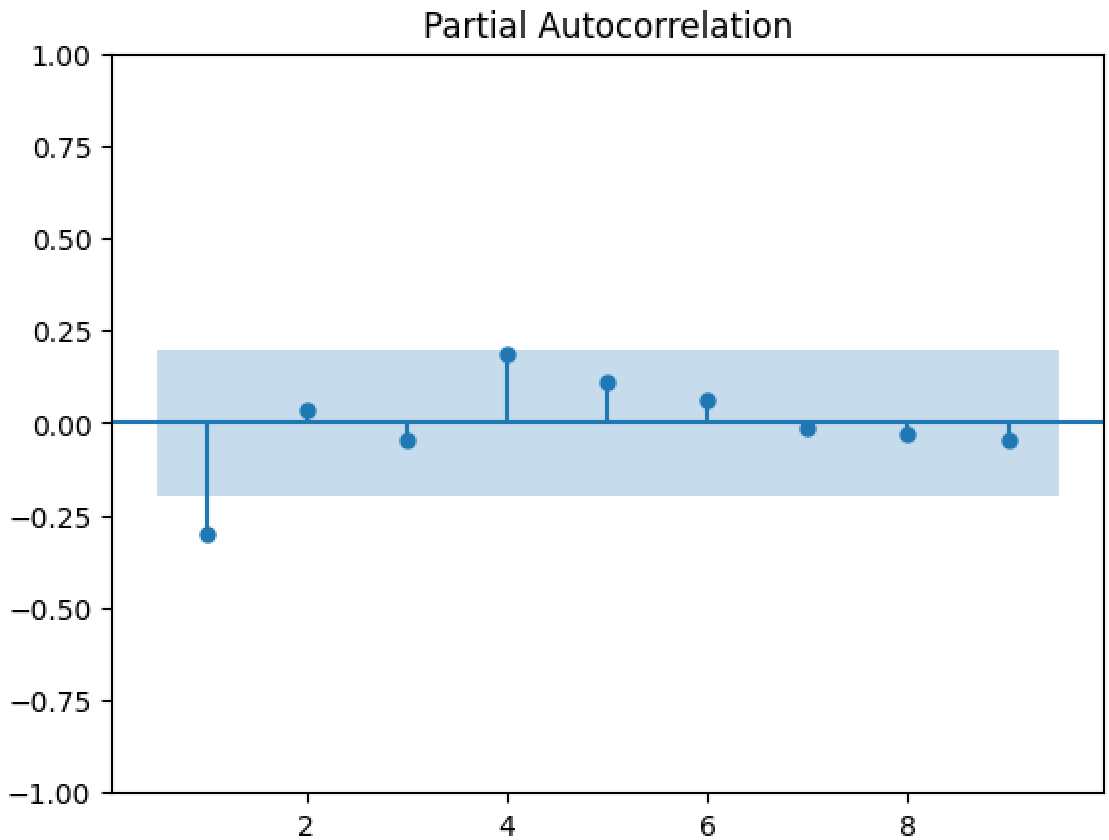


thus q value is chosen to be 1 in both the data sets as all the other branches in the above autocorrelation plot os almost zero

In [54]:
```
from statsmodels.graphics.tsaplots import plot_pacf
fig = plot_pacf(difference_1, lags=range(1,10), alpha=0.05)
```



In [55]:
```
fig = plot_pacf(difference_2, lags=range(1,10), alpha=0.05)
```



thus deciding the p value to be 1 as all the branches in the partial auto correlation plot drops to almost 0 after that

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: