EXPLORING TIME SERIES MODELING

How To Analyse A Single Time Series Variable

Time Series Modeling With Python Code

Jiahui Wang Apr 5 · 8 min read ★

Photo by tangi bertin on Unsplash

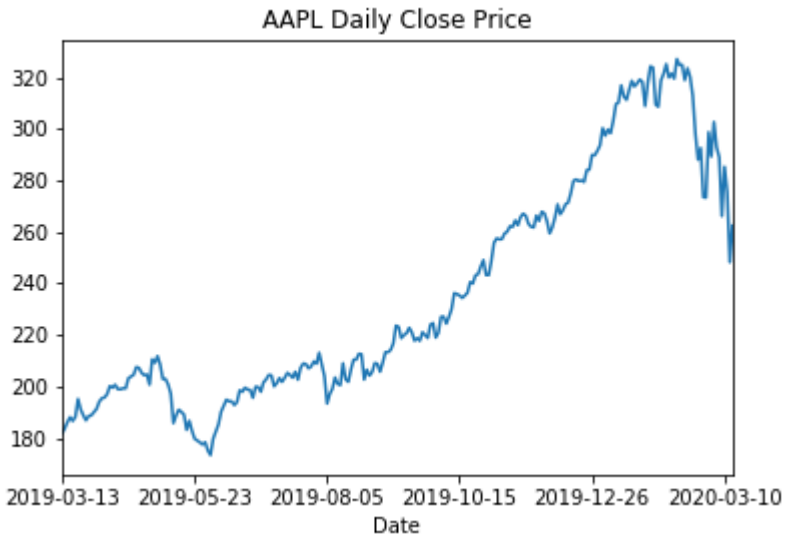
Welcome back! This is the 2nd post in the column to explore analysing and modeling time series data with Python code. If you are not familiar with the basic statistical concepts, such as estimator, hypothesis testing, and p value, please check out my previous post: ***Time Series Modeling With Python Code: Fundamental Statistics***.

In this post, we will start to explore analysing a single time series variable. Given a single time series variable, where should we start the analysis? How can we get some insights into the data? To be honest, the first time I was

asked to generate insights of a time series data which just came in a .csv file without any further information about the data source and application, I got no idea of where to start. After this post, hopefully, the next time you will have some ideas on how to kick start analysing a single time series variable.

Here I will analyse the historical Apple stock price data. The .csv file can be downloaded from Yahoo finance. I downloaded the last year AAPL data, ranging from 3/13/2019 to 3/13/2020. The raw data includes daily open, high, low, close, adj close price and volume. In this post, I will analyse the daily close price of AAPL.

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

AAPL_price = pd.read_csv('AAPL.csv',usecols=['Date', 'Close'])
AAPL_price.set_index('Date',inplace=True,drop=True)

AAPL_price.plot(legend=False)
plt.title('AAPL Daily Close Price')
```



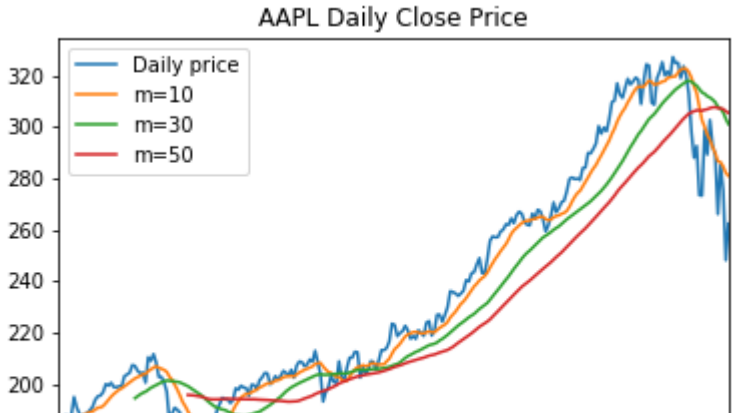
1. Rolling Mean and Variance

Sample parameters are not constant, and they are also changing with time. A single point sample mean or sample variance won't reveal much information to us. Without knowing the variance of the sample parameter, we cannot know how well the sample parameter estimates the population parameter. Thus, we cannot rely on the point sample mean or sample variance to estimate the population parameters.

Instead of calculating a point sample mean and sample variance from the one-year AAPL stock price data, we will apply rolling window method to get multiple sample mean and sample variance values over the one year period.

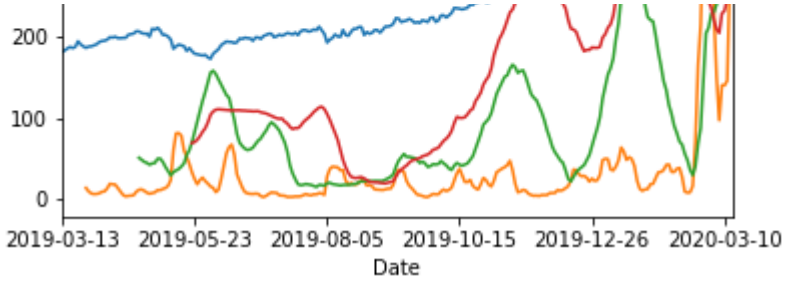
The following code shows how to visualize sample mean with different window size of 10, 30, and 50.

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

AAPL_price = pd.read_csv('AAPL.csv',usecols=['Date', 'Close'])
AAPL_price.set_index('Date',inplace=True,drop=True)

ax = AAPL_price.plot(legend=False)
ax.set_title('AAPL Daily Close Price')

AAPL_price.rolling(window=10).mean().plot(ax=ax)
AAPL_price.rolling(window=30).mean().plot(ax=ax)
AAPL_price.rolling(window=50).mean().plot(ax=ax)

ax.legend(['Daily price', 'm=10', 'm=20', 'm=30'])
```



Similarly, we can visualize sample variance with different window size.

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


AAPL_price = pd.read_csv('AAPL.csv',usecols=['Date', 'Close'])
AAPL_price.set_index('Date',inplace=True,drop=True)


ax = AAPL_price.plot(legend=False)
ax.set_title('AAPL Daily Close Price')


AAPL_price.rolling(window=10).var().plot(ax=ax)
AAPL_price.rolling(window=30).var().plot(ax=ax)
AAPL_price.rolling(window=50).var().plot(ax=ax)


ax.legend(['Daily price', 'm=10', 'm=30', 'm=50'])
```



When choosing the window size, there is always a tradeoff. Take rolling sample mean as an example, we will find that larger window size generates smoother rolling sample mean plot. I like to think about the choice of window size in terms of overfitting and underfitting. A small window size tends to catch more detailed information of each time point, while a large window size includes more overall information of a longer time period. In this way, a small window size may cause overfitting, as it pays too much attention to the movement of each time point. While a large window size may cause underfitting, as it captures too much overall trend but neglects the local movement of each point. Thus, a proper window size should be carefully chosen to avoid overfitting and underfitting.

2. Stationarity and Weak Dependence

In order to use ordinary least squares (OLS) to estimate the generation process, time series data need to stationary and weakly dependent. OLS is a commonly used method in linear regression and will be discussed in detail

in the 4th post of this series: ***Time Series Modeling With Python Code: How To Model Time Series Data With Linear Regression.***

Stationarity has three requirements. The mean and variance of the time series data both are constant. In addition, the covariance of two time points with a lag (h) is a function of the lag, while it should not be dependent on time point (t).

Weak dependence requires that the correlation of two time points becomes zero when the lag h becomes infinity.

Stationarity	Weak Dependence
$E[x_t] = \mu$	$Corr(x_t, x_{t+h}) \to 0$
$Var[x_t] = \sigma^2$	when $h \to \infty$
$Cov(x_t, x_{t+h}) = f(h)$	

When x_t is stationary and weak dependent, OLS can be used to estimate the process.

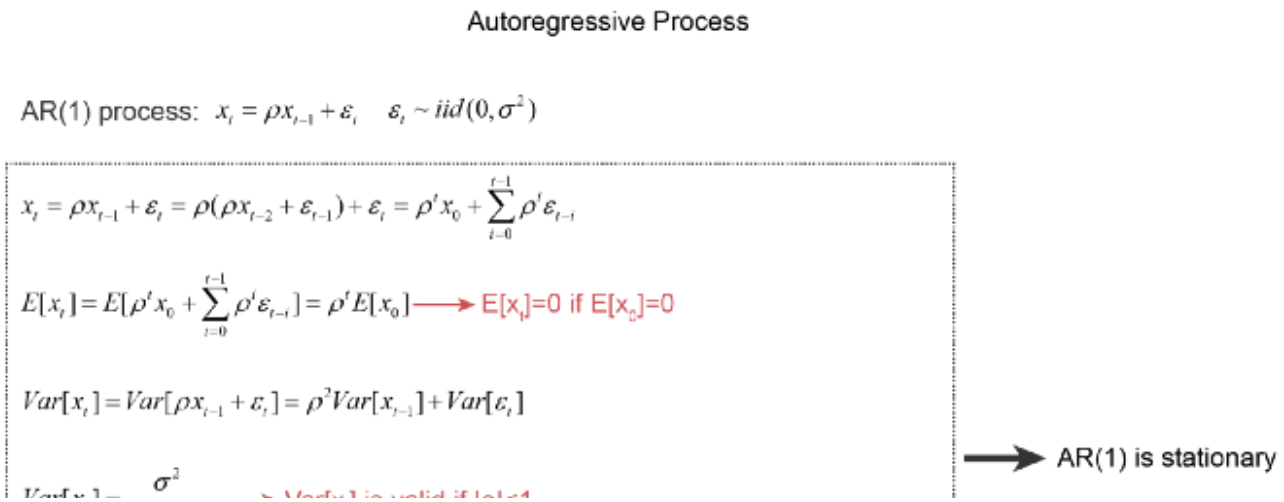
3. Autoregressive Process and Moving Average Process

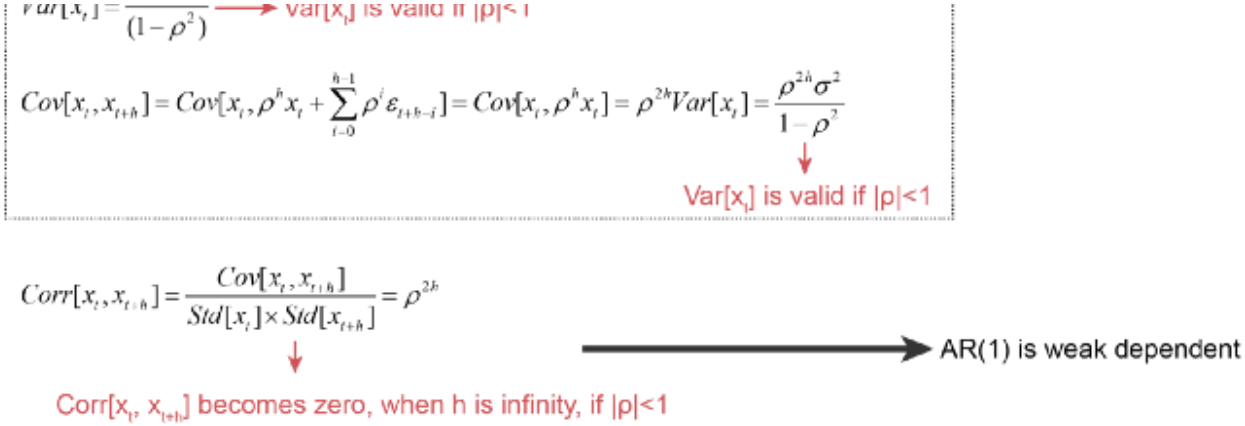
There are two common time series processes: autoregressive process and moving average process. We will discuss these two processes in detail.

3.1 Autoregressive Process Property

For autoregressive process, the time series data depends on itself with a time lag. When the time series data only depends on itself with a time lag of 1, the process is called AR(1). If the time series data depends on itself with a time lag of N, then the process is called AR(N).

Here, take AR(1) as an example. AR(1) process is stationary and weak dependent if two requirements are met: the expected value of the first time point is zero, and the time series is dependent on the previous time point with a multiplying parameter that lies in between -1 and 1.



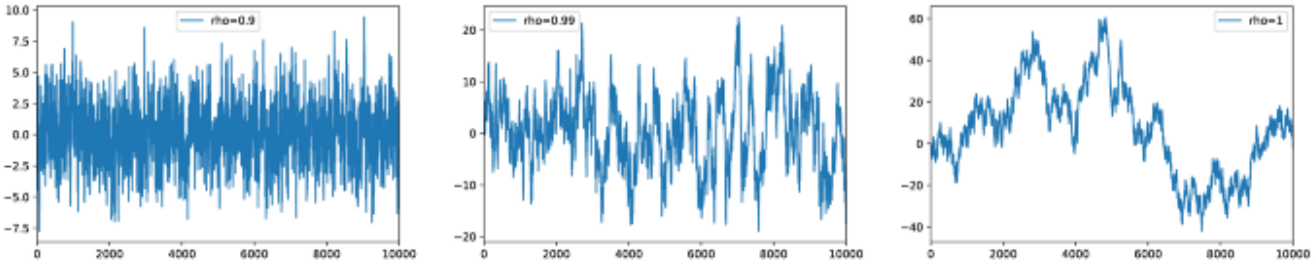


AR(1) process properties

In AR(1) process, the value of ρ determines whether the AR(1) process is stationary. The following is a simple visualization of how ρ affects the AR(1) process. From the results, we can tell when ρ is closer to 1, the AR(1) process crosses zero line less frequently.

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


N = 10000
rho = 1


sigma = np.random.normal(loc=0, scale=1, size=N)
x = [0]
for i in range(1,N):
    x.append(rho*x[-1]+sigma[i])
```

```
plt.plot(x, label='rho=1')
plt.legend()
plt.xlim(0,N)
```



AR(1) process with rho of 0.9, 0.99, 1

3.2 Moving Average Process Property

MA(1) process is stationary and weak dependent.

Moving Average Process

MA(1) process: $x_t = \varepsilon_t + \theta\varepsilon_{t-1} \quad \varepsilon_t \sim iid(0,\sigma^2)$

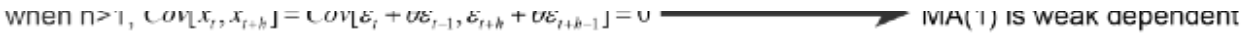
$E[x_t] = E[\varepsilon_t + \theta\varepsilon_{t-1}] = E[\varepsilon_t] + \theta E[\varepsilon_{t-1}] = 0$

$Var[x_t] = Var[\varepsilon_t + \theta\varepsilon_{t-1}] = Var[\varepsilon_t] + \theta^2 Var[\varepsilon_t] = (1+\theta^2)\sigma^2$

$Cov[x_t, x_{t+h}] = Cov[\varepsilon_t + \theta\varepsilon_{t-1}, \varepsilon_{t+h} + \theta\varepsilon_{t+h-1}]$

when h=1, $Cov[x_t, x_{t+1}] = Cov[\varepsilon_t + \theta\varepsilon_{t-1}, \varepsilon_{t+1} + \theta\varepsilon_t] = Cov[\varepsilon_t, \theta\varepsilon_t] = \theta^2\sigma^2$

→ MA(1) is stationary



MA(1) process properties

3.3 Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF)

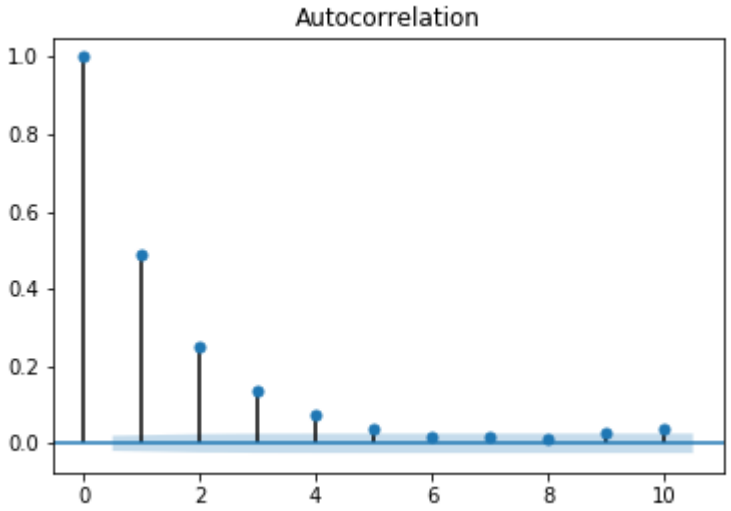
According to the above MA(1) and AR(1) properties, we can write the correlation as follows:

MA(1) process $$Corr[x_t, x_{t+h}] = \begin{cases} \frac{\theta}{1+\theta^2} & h = 1 \\ 0 & h > 1 \end{cases}$$

AR(1) process $$Corr[x_t, x_{t+h}] = \rho^{2h}$$

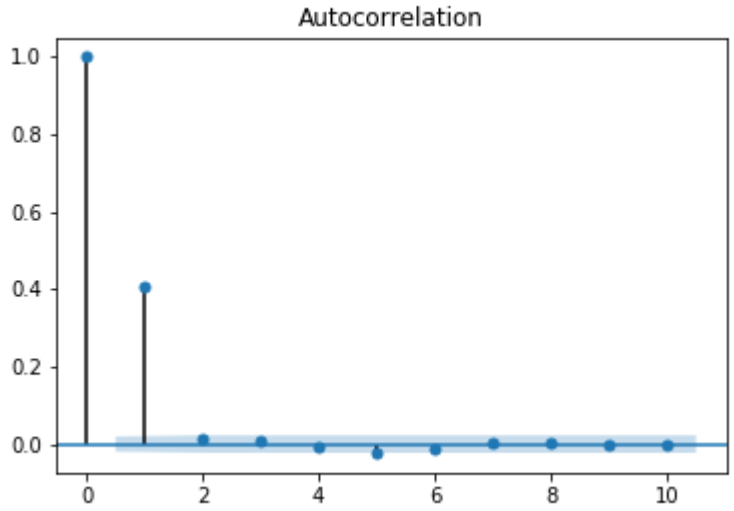
By plotting the correlation of different time lags, we can clearly see the difference between AR(1) process and MA(1) process. This correlation graph is called autocorrelation function (ACF). The ACF plot of AR process shows exponential decrease, and the correlation decreases to almost zero after serval time points. However, the ACF plot of MA(1) process shows the correlation quickly drops towards zero after the first 2 time points.

```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf


N = 10000
rho = 0.5


sigma = np.random.normal(loc=0, scale=1, size=N)
x = [0]
for i in range(1,N):
    x.append(rho*x[-1]+sigma[i])


plot_acf(np.array(x),lags=10)
```



ACF of AR(1) process

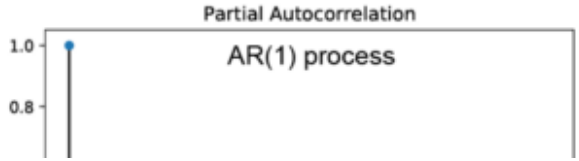
```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf


N = 10000
theta = 0.5


sigma = np.random.normal(loc=0, scale=1, size=N+1)
x = []
for i in range(1,N+1):
    x.append(sigma[i]+theta*sigma[i-1])


plot_acf(np.array(x),lags=10)
```

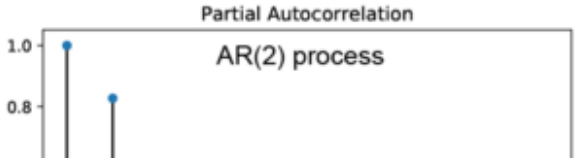


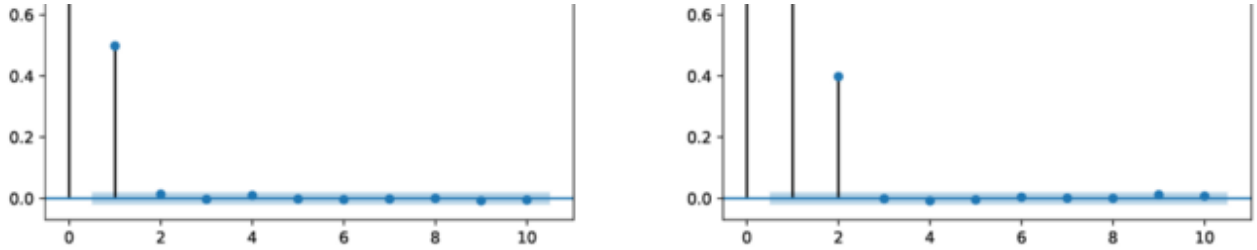
ACF of MA(1) process

Although ACF plot can be used to differentiate MA and AR process, it cannot well differentiate AR(1) process from AR(2) process. Partial autocorrelation function (PACF) can be used to differentiate AR(1) and AR(2) process. As shown in the following example, PACF of AR(1) process shows the correlation quickly drops toward zero after the first 2 time points, while the PACF of AR(2) process shows the correlation drops toward zero after the first 3 time points.

```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_pacf


N = 10000
rho = 0.5
rho2 = 0.4


sigma = np.random.normal(loc=0, scale=1, size=N)
x = [0,0]
for i in range(2,N):
    x.append(rho*x[-1]+rho2*x[-2]+sigma[i])


plot_pacf(np.array(x),lags=10)
```







PACF of AR(1) and AR(2) process

4. Order of Integration Process

4.1 Unit Root and Dickey-Fuller Test

As shown above, when ρ is 1, AR(1) process is non-stationary. The scenario where the ρ=1 in AR process is called unit root. The following is a simple proof of why unit root is non-stationary.

Unit Root

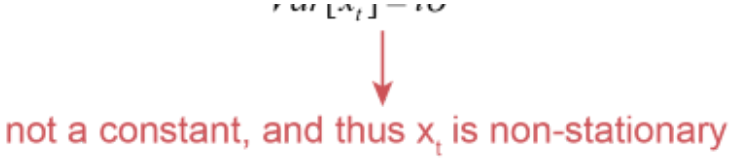
$$\text{AR(1) process: } x_t = \rho x_{t-1} + \varepsilon_t \quad \varepsilon_t \sim iid(0,\sigma^2)$$

$$\downarrow \rho = 1$$

$$\text{Unit Root: } x_t = x_{t-1} + \varepsilon_t \quad \varepsilon_t \sim iid(0,\sigma^2)$$

$$x_t = x_{t-1} + \varepsilon_t = (x_{t-2} + \varepsilon_{t-1}) + \varepsilon_t = x_0 + \sum_{i=1}^{t} \varepsilon_i$$

$$Var[x_t] = t\sigma^2$$



A simple proof that unit root is non-stationary

Unit root can be tested with Augmented Dickey-Fuller (ADF) test.

```
import pandas as pd
from statsmodels.tsa.stattools import adfuller

AAPL_price = pd.read_csv('AAPL.csv',usecols=['Close'])
result = adfuller(AAPL_price.iloc[:,0].values)
print(f'p value is {result[1]}')
```

Output:

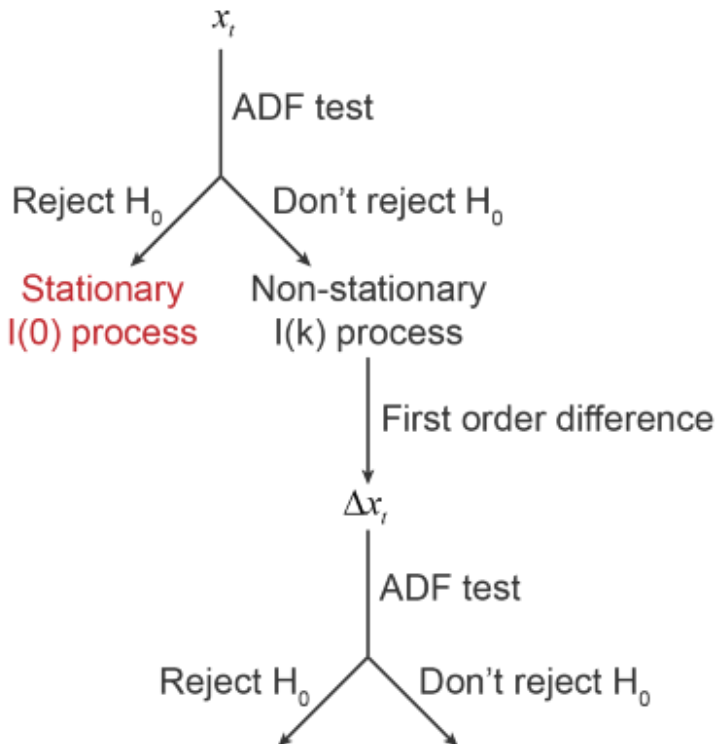
p value is 0.5961654850033034

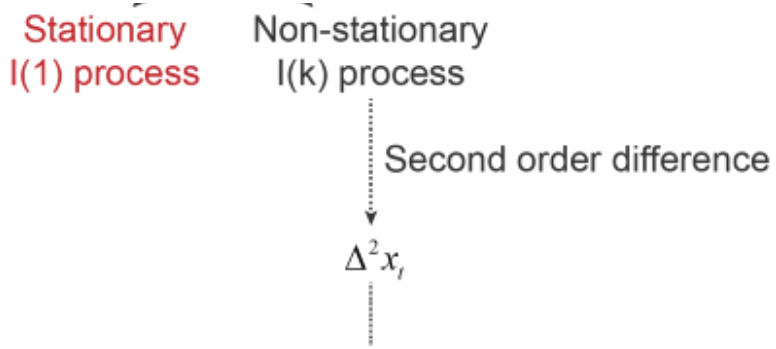
Since the p value is larger than the significance level of 0.05, we cannot reject the null hypothesis that the time series data is non-stationary. Thus, the time series data is non-stationary.

AR process with unit root is serially correlated. However, serially correlated time series data might not necessarily be AR process with unit root.

4.2 Order of Integration Process

For AR process with unit root, if the first order difference of the time series data is stationary, then the time series data follows I(1) process. Similarly, if the second order difference is required to get stationary data, then the process follows I(2) process. To find out the order of the integration process, a series of ADF need to be tested.





How to test the order of integration process

Summary

In this post, we discussed on how to analyse a single time series variable. Typically, we can start the analysis by plotting the rolling mean and variance of the time series data. Then we can use ACF test to see if the time series data follows the autoregressive process or moving average process. If the data follows the autoregressive process, we can then use the PACF test to find the order of the autoregressive process. In addition, we can use a Dickey-Fuller test to find if the time series data follows the integration process.

In the next post, we will continue to discuss how to analyse multiple time series variables. Please stay tuned!