Python 時間序列分析

David Chiu

課程資料

- 所有課程補充資料、投影片皆位於
  - https://github.com/ywchiu/ctbcpy

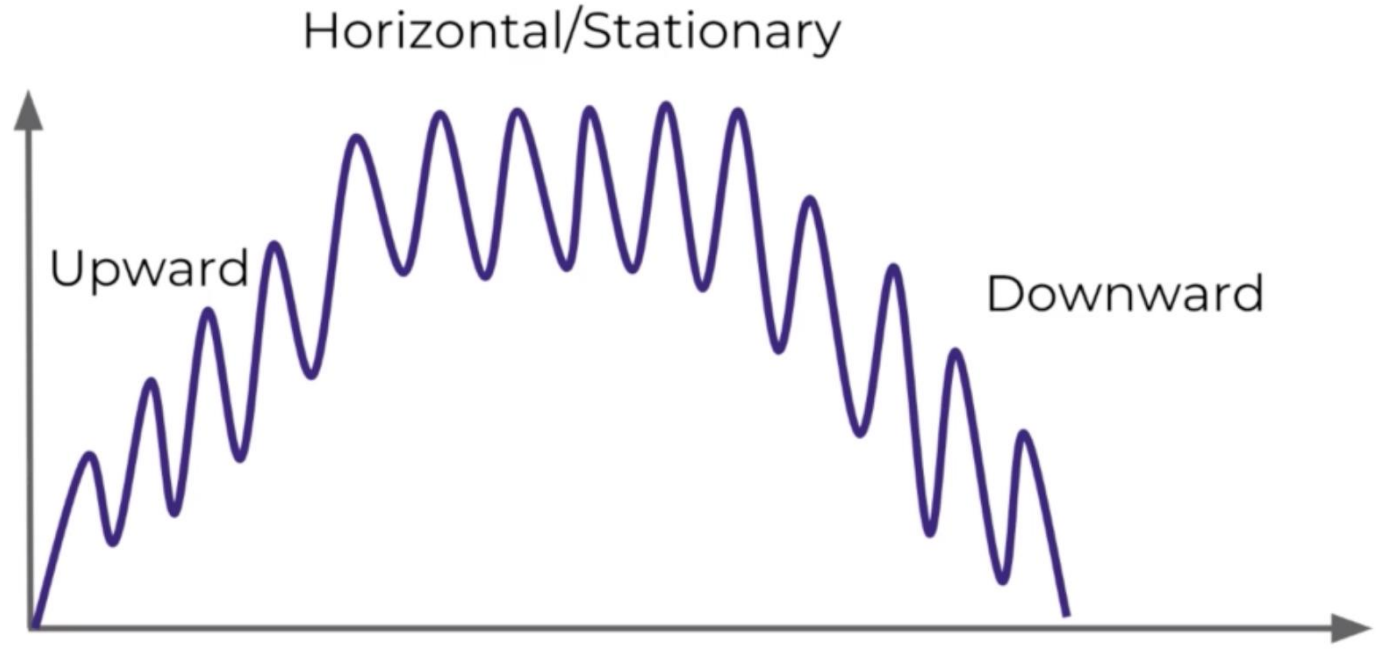
時間序列分析

埃及人的時間序列分析

- 在古埃及，每當天狼星在黎明時從東方地平線升起時，正是一年一度尼羅河水泛濫的時候，尼羅河水的泛濫，灌溉了兩岸大片良田，於是埃及人又開始了他們的耕種
- 古代埃及人發現，天狼星兩次偕日升起的時間間隔不是埃及曆年的365天而是365.25天。古埃及把黎明前天狼星自東方升起的那一天確定為歲首。

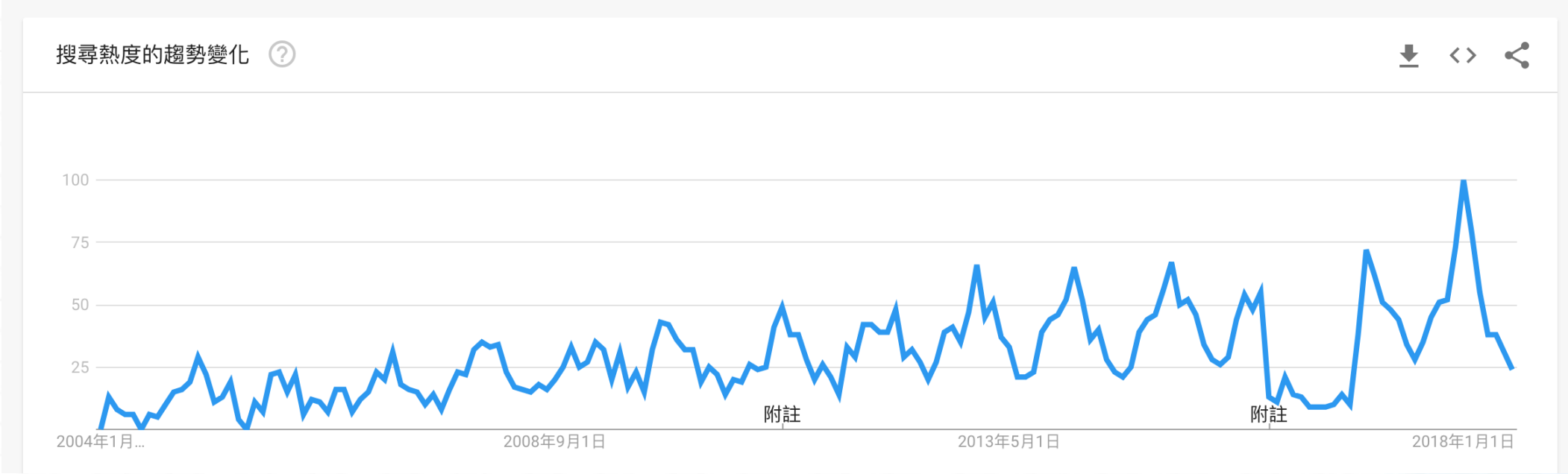
趨勢序列

- 比如線性趨勢，先增加後降低的整體趨勢
- 趨勢序列可以分為: 上升、水平與下降趨勢



季節性序列

- 從感冒的搜尋趨勢 (Google Trends) 發現民眾每年搜尋感冒的時間點是有規律的(季節性)
- 季節性資料可視為重複性的趨勢 (上升-平穩-下降)



週期性序列

- 在以不固定週期不斷震盪，通常週期性至少持續2年

典型時間序列

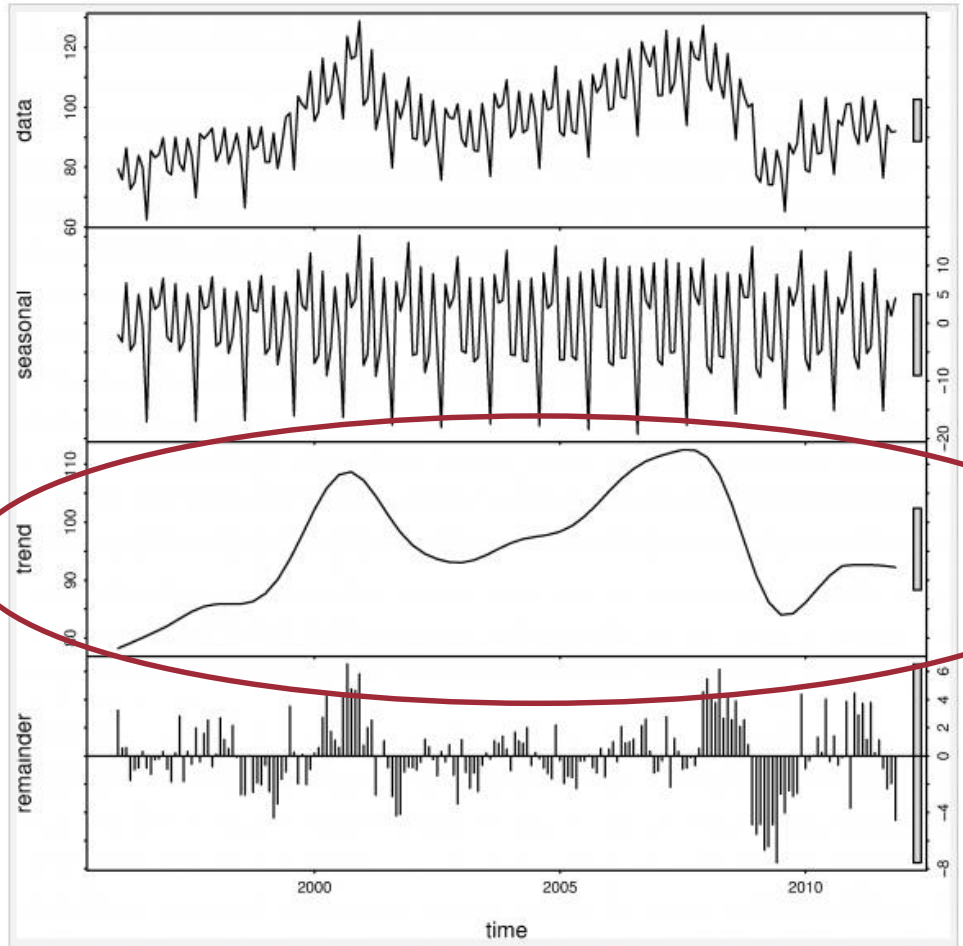


Figure 6.3: The electricial equipment orders (top) and its three additive components.

包含 Trend 與 Cyclic 序列

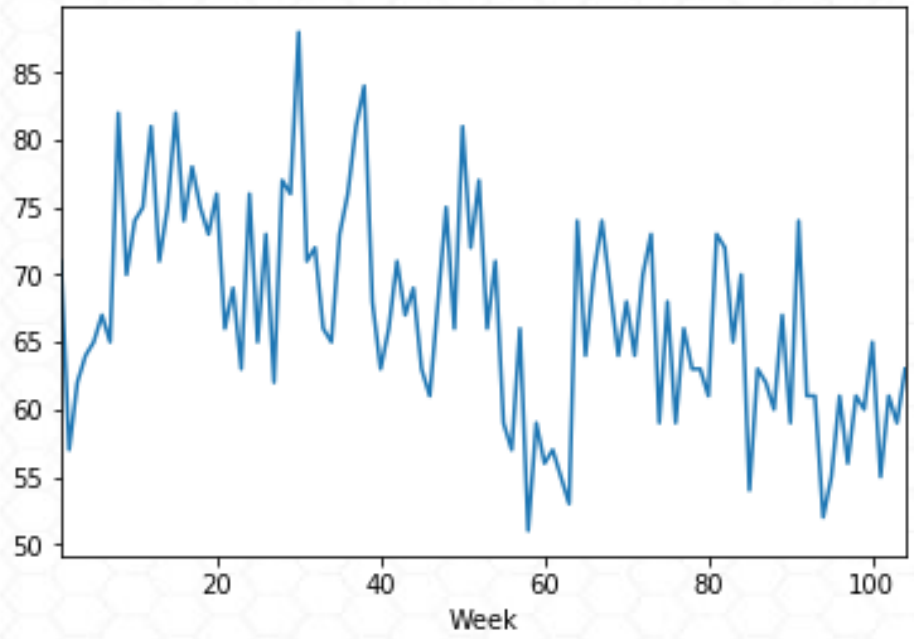
時間序列與迴歸分析

- 時間序列**與時間有關**。線性模型假設觀察值是獨立的情形不適用於時間序列

- 大多數時間序列具有**季節性趨勢**
  - 例如，冬天在Google 的感冒搜尋量會比較高

讀取貸款申請數量的時間序列資料

```
import pandas
df = pandas.read_csv('LoanApplications.csv')
df.set_index('Week', inplace=True)
df['Applications'].plot(kind = 'line')
```



時間序列的平穩性

- 大部分時間序列模型是在假設它是穩定的前提下建立的
  - 如果一個時間序列隨著時間產生特定的行為，就有很高的可能性認為它在未來的行為是一樣的
- 如果一個時間序列的統計特徵如平均數(Mean)，變異數(Variance)隨著時間保持不變，我們就可以認為它是穩定的
- 平穩的定義
  - 恆定的平均數
  - 恆定的變異數
  - 共變異數Covariance不隨時間變化

如何使一個時間序列趨於平穩

- 雖然許多時間序列模型都採用了平穩性假設，但幾乎沒有一個實際的時間序列是平穩的。實際上，幾乎不可能使一個序列完全平穩，但我們要盡可能地接近它

- 導致時間序列不穩定的兩個主要原因：
  - 趨勢-均值隨著時間變化而變化
  - 季節性-特定時間的變化

消除趨勢的方式

1. 聚合:取一段時間的平均值（月/周平均值）

2. 平滑:取移動平均數

3. 多項式擬合:擬合一個回歸模型

簡單均線法

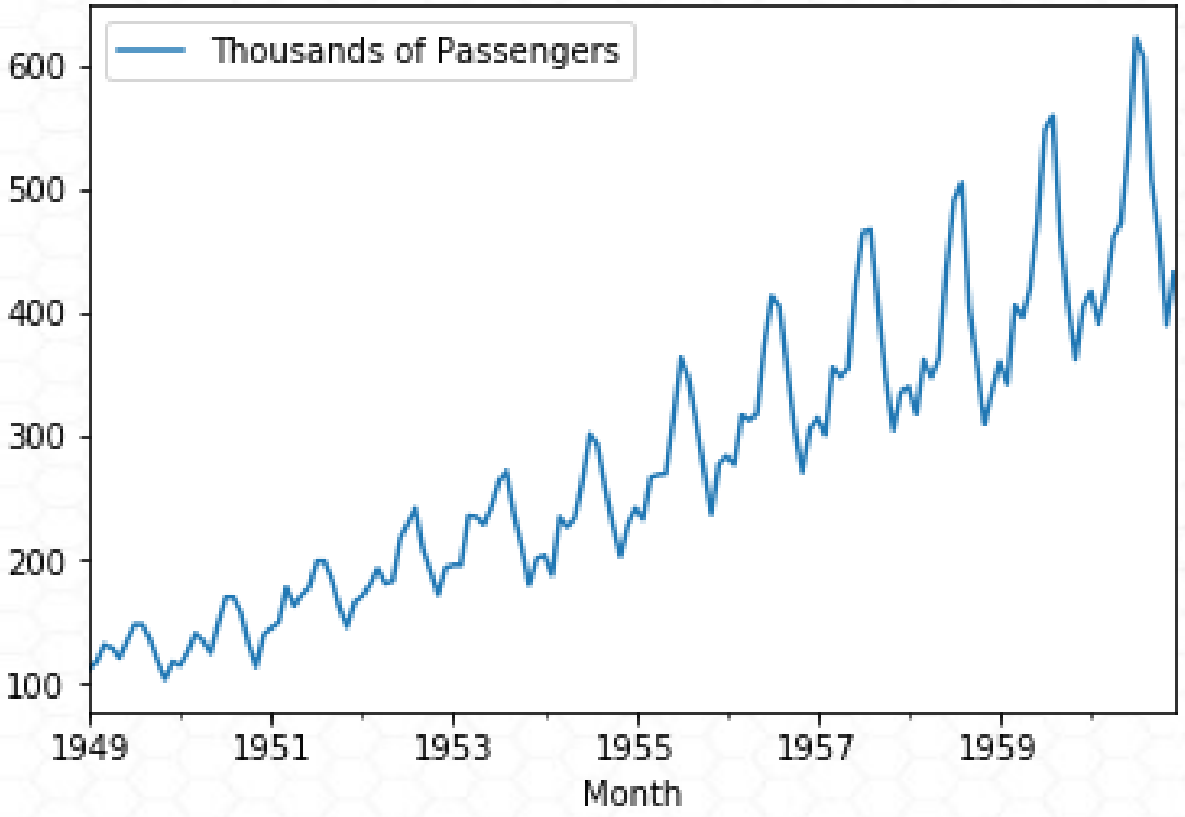
讀取機場旅客資料

```
import pandas as pd
import numpy as np
airline = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/airline_passengers.csv',index_col='Month',parse_dates=True)
airline.dropna(inplace=True)
airline.head()
```

Month	Thousands of Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

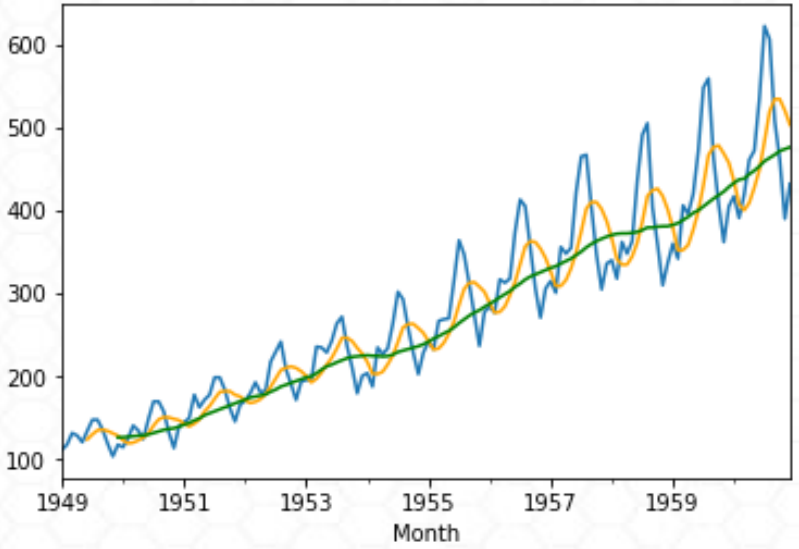
繪製機場旅客數折線圖

airline.plot()



找出非季節因素的趨勢 (使用均線)

```
moving_avg_6  = airline['Thousands of Passengers'].rolling(6).mean()
moving_avg_12 = airline['Thousands of Passengers'].rolling(12).mean()
airline['Thousands of Passengers'].plot()
moving_avg_6.plot(color='orange')
moving_avg_12.plot(color='green')
```



ETS 模型

ETS 模型 (Error Trend Seasonality)

1. 指數平滑 (Exponential Smoothing)
2. 趨勢方法 (Trend Method Models)
3. ETS 分解 (ETS Decomposition)

時間序列的分解

- 長期趨勢 T: 時間序列隨時間逐漸增加或減少的變化趨勢
- 季節變動 S: 時間在固定時間內出現固定規則(週期性)的變動
- 循環變動 C: 隨著趨勢性出現鐘擺的變動
- 不規則變動 e: 時間序列受到隨機因素而產生的變動

- 加法模型
  - Y = T+S+C+e

- 乘法模型
  - Y= T * S * C * e

加法或乘法模型

- 其各自的適用範圍：
  - 季節性和trend-cycle變動的量級不隨時間變化，用加法模式
    - 貸款的申請數量每年增加10,000件
  - 兩者的波動隨時間成比例，適用於乘法模式
    - 貸款的申請數量每年倍數增長
  - 乘法模式可以通過log變為加法模式

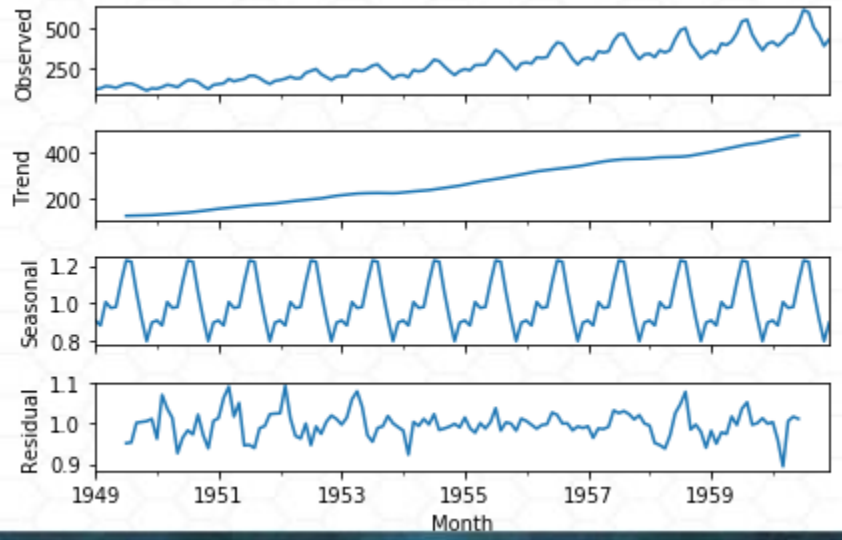
$$y_t = S_t \times T_t \times E_t \quad \Leftrightarrow \quad \log y_t = \log S_t + \log T_t + \log E_t$$

Statsmodels

- statsmodels是一個Python模組，提供許多統計模型的函數與套件，可用於統計測試和資料探索等任務

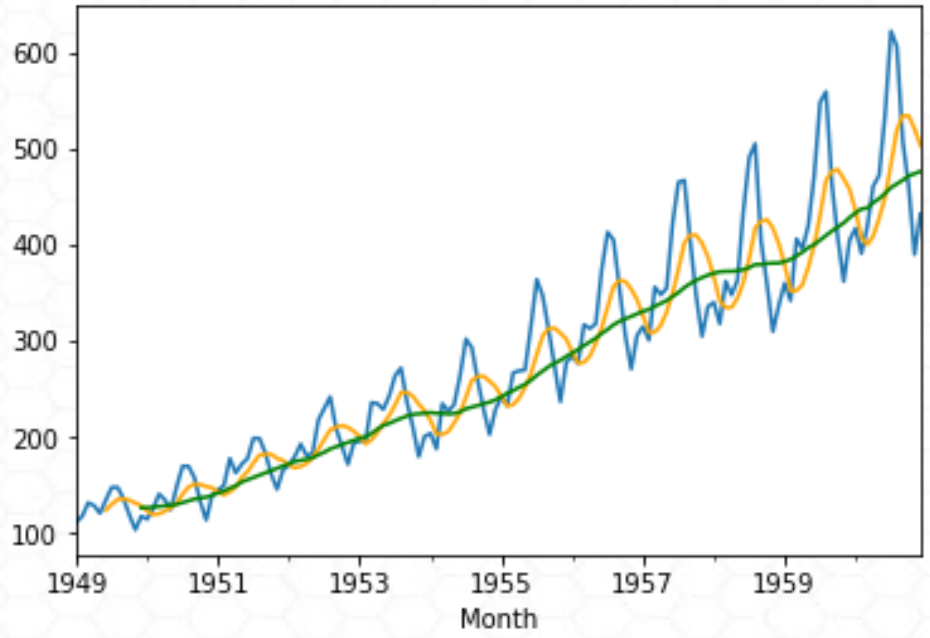
時間序列分解

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(airline['Thousands of Passengers'],
model='multiplicative')
result.plot()
```



簡單移動平均

- 簡單移動平均需要給一個固定大小的區間 (e.g. 6 個月或12個月)
- 是否可以只考慮近期的資料，並且給予近期的資料比較大的權重?



指數平滑法

- 計算移動平均時，每出現一新的觀察值，就要從移動平均中減去最遠的觀測值，加入一個最新觀測值以計算移動平均，新的平均移動可做為下一期的預測值。

- 求出指數平滑：
  - 簡單指數平滑法(1 個參數)
  - Holt 雙參數線性指數平滑法(2 個參數)
  - Winters 線性與季節性指數平滑法(3 個參數)

指數平滑克服以下缺點

- 比較小的時間區間會產生許多噪音
- 必須設定固定的時間區間
- 無法反映出資料的最大最小值
- 無法反映資料未來的行為
- 資料出現極值會導致簡單移動平均線歪斜

簡單指數平滑法(1 個參數)

- 使用於可用加法模型描述的時間序列，用做恆定水平(數據有平穩性)且沒有季節性的數據做預測

- 定義公式

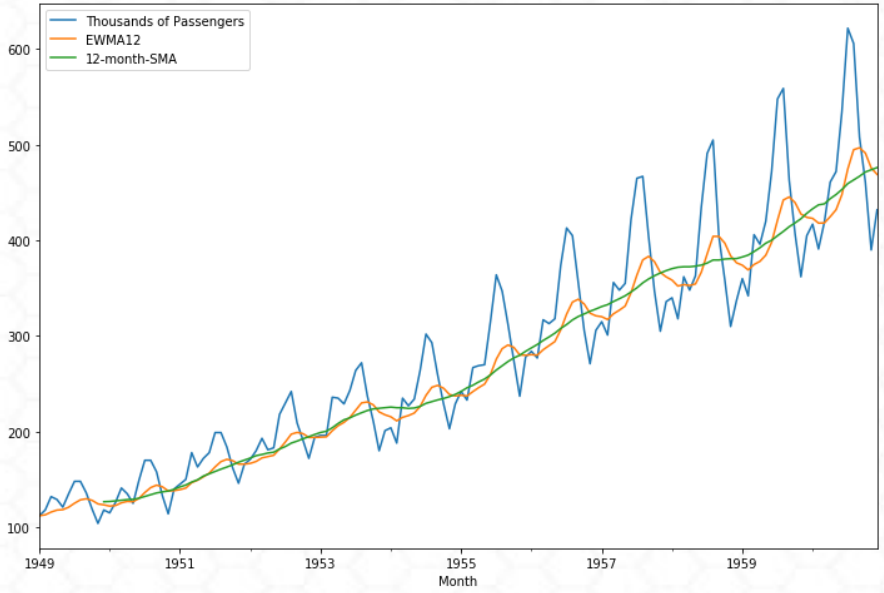
$$y_0 = x_0$$

$$y_t = (1-\alpha)y_{t-1} + \alpha x_t,$$

  - a 代表加權，參數介於0~1之間

指數移動平均

```
airline['EWMA12'] = airline['Thousands of Passengers'].ewm(span=12,adjust=False).mean()
airline['12-month-SMA'] = airline['Thousands of Passengers'].rolling(window=12).mean()
airline[['Thousands of Passengers','EWMA12','12-month-SMA']].plot(figsize=(12,8)).autoscale(axis='x',tight=True)
```



EMW 的Adjust 參數

- adjust = True
  - 假設歷史資料是有限長度

$$y_t = \frac{x_t+(1-\alpha)x_{t-1}+(1-\alpha)^2x_{t-2}+...+(1-\alpha)^t x_0}{1+(1-\alpha)+(1-\alpha)^2+...+(1-\alpha)^t}$$

- adjust = False
  - 假設歷史資料為無限長

$$y_0 = x_0$$
$$y_t = (1-\alpha)y_{t-1} + \alpha x_t,$$

HoltWinters

- 簡單指數平滑法的缺陷
  - 只使用到一個參數Alpha
  - 沒有考慮到趨勢(Trend)與季節性(Seasonality)等因素

- HoltWinters
  - Holt(1957), Winters(1960) 將趨勢性與季節性的因素考量進來
  - Holt’s Method - Double Exponential Smoothing
  - Holt-Winters Method - Triple Exponential Smoothing

Holt 雙參數線性指數平滑法(2 個參數)

- 使用於時間序列有增長或降低趨勢，且可用加法模型描述的時間序列，可以估計當前時間點的水平與斜率

- 定義公式

$$l_t = (1-\alpha)l_{t-1} + \alpha x_t, \quad \text{level}$$

$$b_t = (1-\beta)b_{t-1} + \beta(l_t - l_{t-1}) \quad \text{trend}$$

$$y_t = l_t + b_t \quad \text{fitted model}$$

$$\hat{y}_{t+h} = l_t + hb_t \quad \text{forecasting model } (h = \#\text{ periods into the future})$$

Winters 線性與季節性指數平滑法(3 個參數)

- 使用於時間序列有增長或降低趨勢，存在季節性，且可用加法模型描述的時間序列

- 定義公式
  - L (循環的週期，若要表示一年中每個月當一個週期，L=12)

$$l_t = (1-\alpha)l_{t-1} + \alpha x_t, \quad \text{level}$$

$$b_t = (1-\beta)b_{t-1} + \beta(l_t - l_{t-1}) \quad \text{trend}$$

$$c_t = (1-\gamma)c_{t-L} + \gamma(x_t - l_{t-1} - b_{t-1}) \quad \text{seasonal}$$

$$y_t = (l_t + b_t)c_t \quad \text{fitted model}$$

$$\hat{y}_{t+m} = (l_t + mb_t)c_{t-L+1+(m-1) \bmod L} \quad \text{forecasting model } (m = \#\text{ periods into the future})$$

設定時間序列頻率

```
import pandas as pd
import numpy as np
df = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/airline_passengers.csv',index_col='Month',parse_dates=True)
df.dropna(inplace=True)
df.index
df.index.freq = 'MS'
df.index
```

簡單指數平滑法(1 個參數)

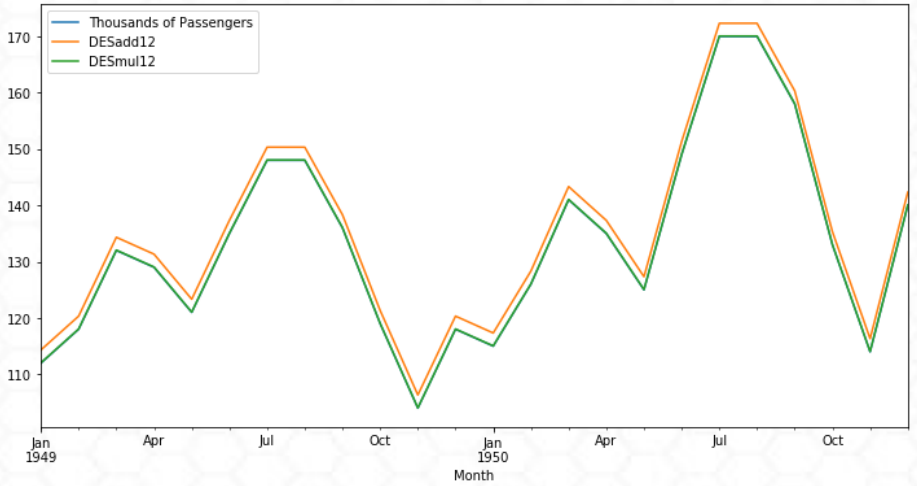
```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing

span = 12
alpha = 2/(span+1)

df['EWMA12'] = df['Thousands of Passengers'].ewm(alpha=alpha,adjust=False).mean()
df['SES12']=SimpleExpSmoothing(df['Thousands of Passengers']).fit(smoothing_level=alpha,optimized=False).fittedvalues.shift(-1)
df.head()
```

Holt 雙參數線性指數平滑法(2 個參數) -加法模型

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

df['DESadd12'] = ExponentialSmoothing(df['Thousands of Passengers'],
trend='add').fit().fittedvalues.shift(-1)
df[['Thousands of
Passengers','EWMA12','DESadd12']].iloc[:24].plot(figsize=(12,6)).autoscale(axis='x',tight=True)
```

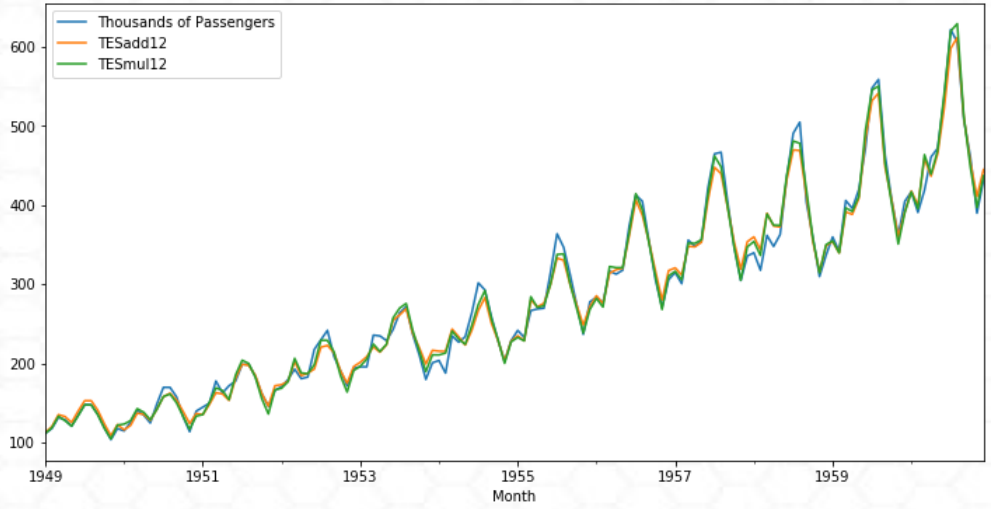
Holt 雙參數線性指數平滑法(2 個參數) – 乘法模型

```
df['DESmul12'] = ExponentialSmoothing(df['Thousands of Passengers'], trend='mul').fit().fittedvalues.shift(-1)
df[['Thousands of Passengers','DESadd12','DESmul12']].iloc[:24].plot(figsize=(12,6)).autoscale(axis='x',tight=True)
```



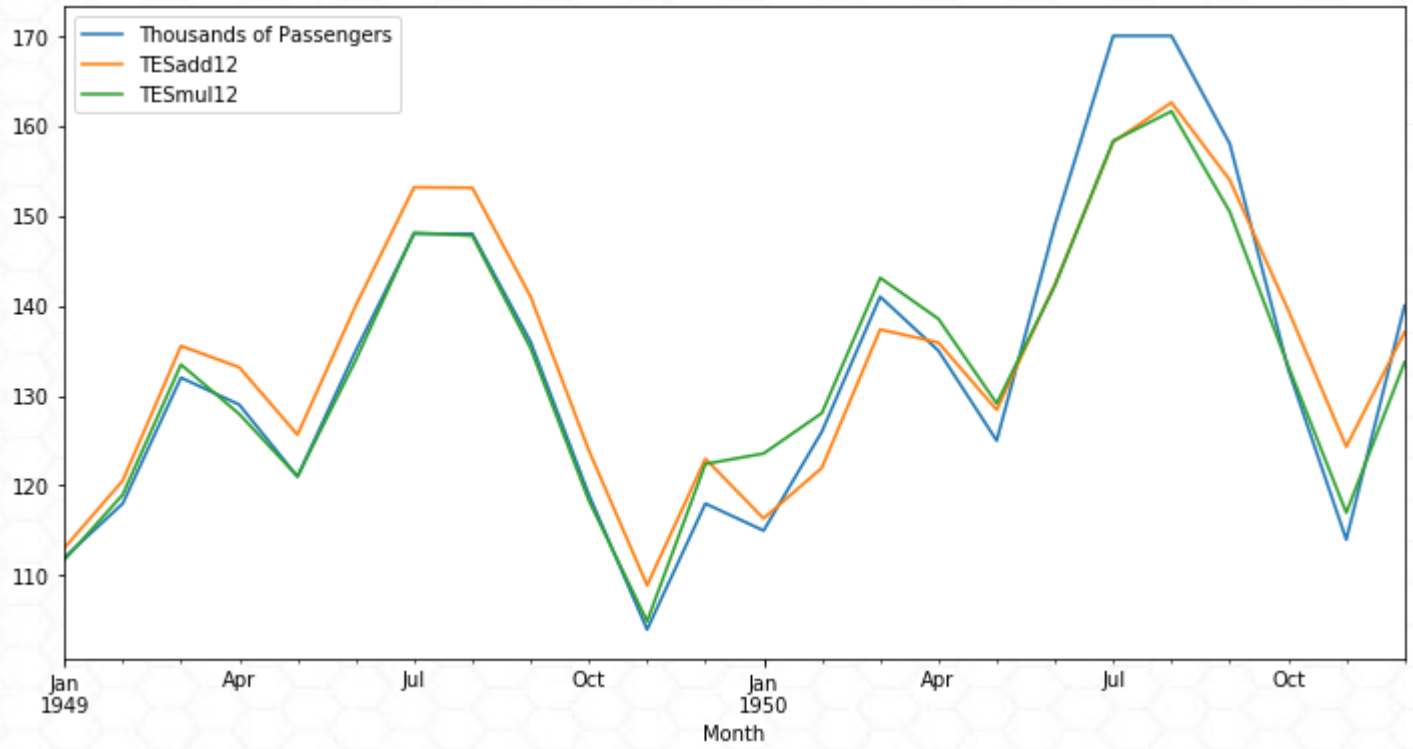
Winters 線性與季節性指數平滑法(3 個參數)

```
df['TESadd12'] = ExponentialSmoothing(df['Thousands of Passengers'],trend='add',seasonal='add',seasonal_periods=12).fit().fittedvalues
df['TESmul12'] = ExponentialSmoothing(df['Thousands of Passengers'],trend='mul',seasonal='mul',seasonal_periods=12).fit().fittedvalues
df[['Thousands of Passengers','TESadd12','TESmul12']].plot(figsize=(12,6)).autoscale(axis='x',tight=True)
```



Winters 線性與季節性指數平滑法(3 個參數)

```
df[['Thousands of Passengers','TESadd12','TESmul12']].iloc[:24].plot(figsize=(12,6)).autoscale(axis='x',tight=True)
```



建立時間序列模型

時間序列預測步驟

1. 選擇模型
2. 將資料分為訓練與測試資料集
3. 將訓練資料輸入至模型中
4. 在測試資料及上測試模型
5. 預測未來資料

讀取資料

```
import pandas as pd
import numpy as np

df = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/airline_passengers.csv',index_col='Month',parse_dates=True)
df.index.freq = 'MS'
df.head()
```

將資料分為訓練與測試資料集

```
train_data = df.iloc[:108]
test_data = df.iloc[108:]
```

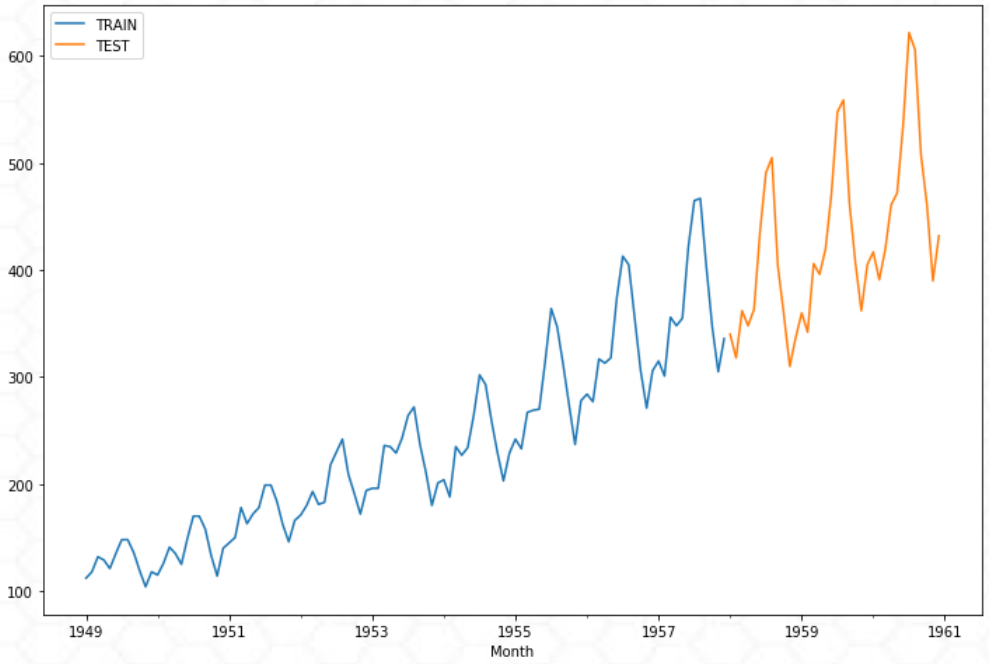
通常會放20%的資料當訓練資料集

建立HoltWinters 模型

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

fitted_model = ExponentialSmoothing(train_data['Thousands of Passengers'],trend='mul',seasonal='mul',seasonal_periods=12).fit()
```

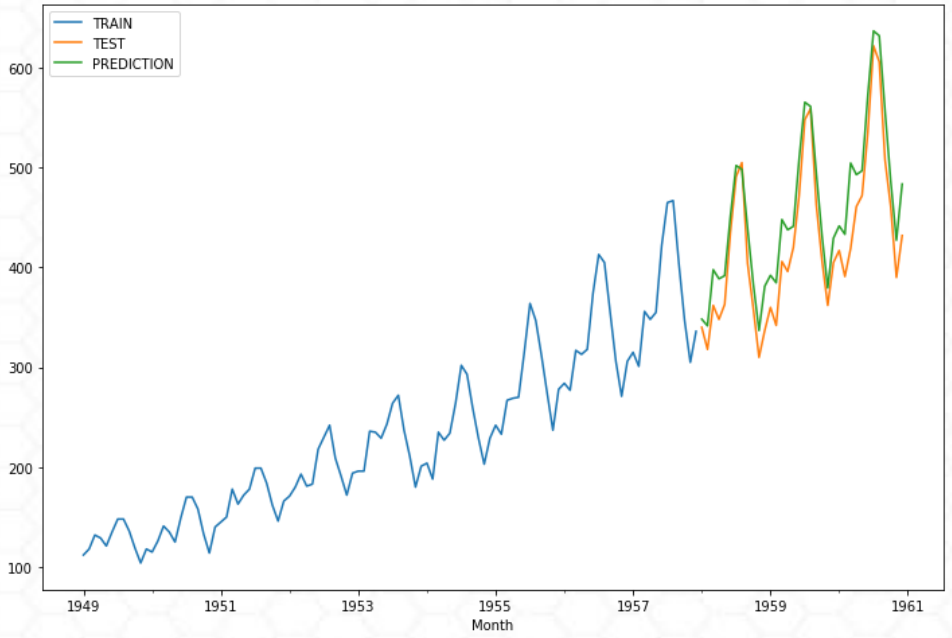
產生預測結果

```
test_predictions = fitted_model.forecast(36).rename('HW Forecast')
train_data['Thousands of Passengers'].plot(legend=True,label='TRAIN')
test_data['Thousands of Passengers'].plot(legend=True,label='TEST',figsize=(12,8));
```



繪製預測結果

```
train_data['Thousands of Passengers'].plot(legend=True,label='TRAIN')
test_data['Thousands of Passengers'].plot(legend=True,label='TEST',figsize=(12,8))
test_predictions.plot(legend=True,label='PREDICTION');
```



評估預測結果

- MAE (Mean Absolute Error)

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- MSE (Mean Squared Error) – 對少數較大的偏差值較靈敏

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- RMSE (Root Mean Squared Error) – 單位相同

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

評估預測結果

```
from sklearn.metrics import
mean_squared_error,mean_absolute_error
mean_absolute_error(test_data,test_predictions)
mean_squared_error(test_data,test_predictions)
np.sqrt(mean_squared_error(test_data,test_predictions))
```

ARIMA

ARIMA

- ARIMA代表差分自回歸移動平均。平穩時間序列的ARIMA預測的只是個線性方程(如線性回歸)。ARIMA使用三個模型參數(p,d,q)產生預測：
  - 自回歸項的數目(p)：AR項僅僅是因變數的時間區間。例如，如果p等於5，那麼預測x(t)將是x(t-1)…x(t-5)
  - 移動平均的數目(q)：MA項是預測誤差的時間區間。例如，如果q等於5，那麼預測x(t)將是e(t-1)…e(t-5)，e(i)是第i個時刻的移動平均和真實值的差值。
  - 差分次數(d)：這裡指的是非季節性的差分次數，即這種情況下我們採用一階差分。因為無論我們傳遞差分後的變數且d=0，還是傳遞原始變數且d=1，得到的結果是一樣的

ARIMA 建模過程

- 對時間序列做平穩化處理
- 根據ACF 與 PACF 選擇參數
- 進行參數估計
- 利用模型做預測分析

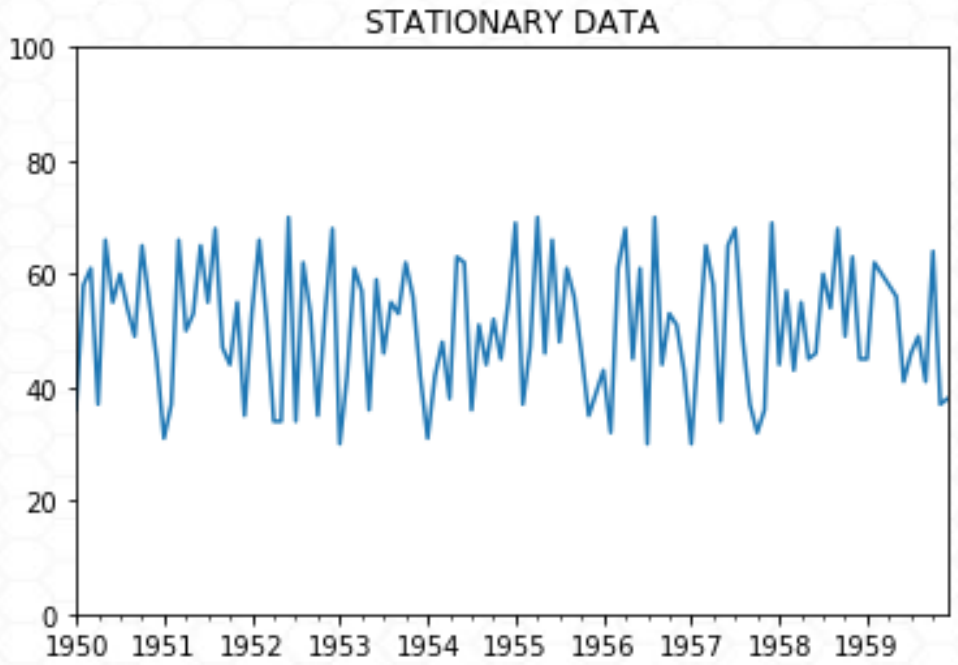
差分

消除趨勢和季節性

- 消除趨勢與季節性的方式
  - **差分(Difference)**—取一個特定時間間隔的差值
  - **分解(Decomposition)**—建立有關趨勢和季節性的模型然後從模型中刪除它們

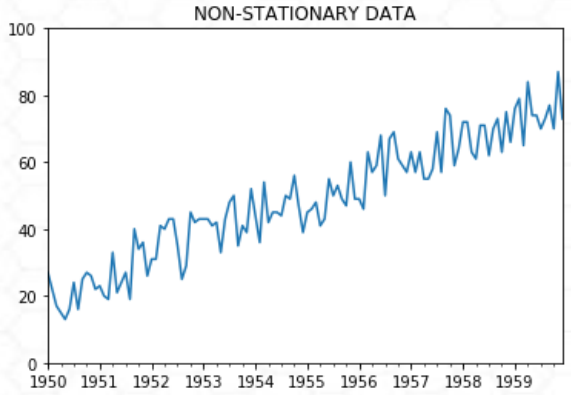
平穩序列

```
df2 = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/samples.csv',index_col=0,parse_dates=True)
df2['a'].plot(ylim=[0,100],title="STATIONARY DATA").autoscale(axis='x',tight=True);
```

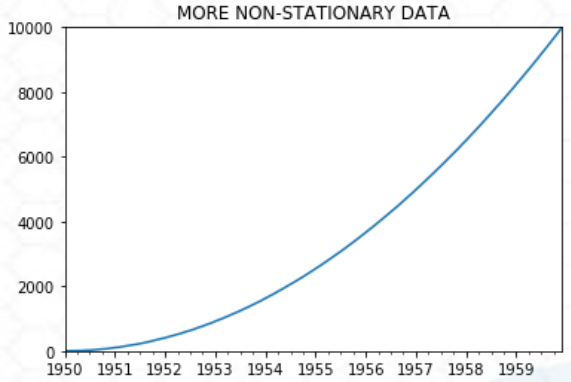


非平穩序列

```
df2['b'].plot(ylim=[0,100],title="NON-STATIONARY
DATA").autoscale(axis='x',tight=True);
```

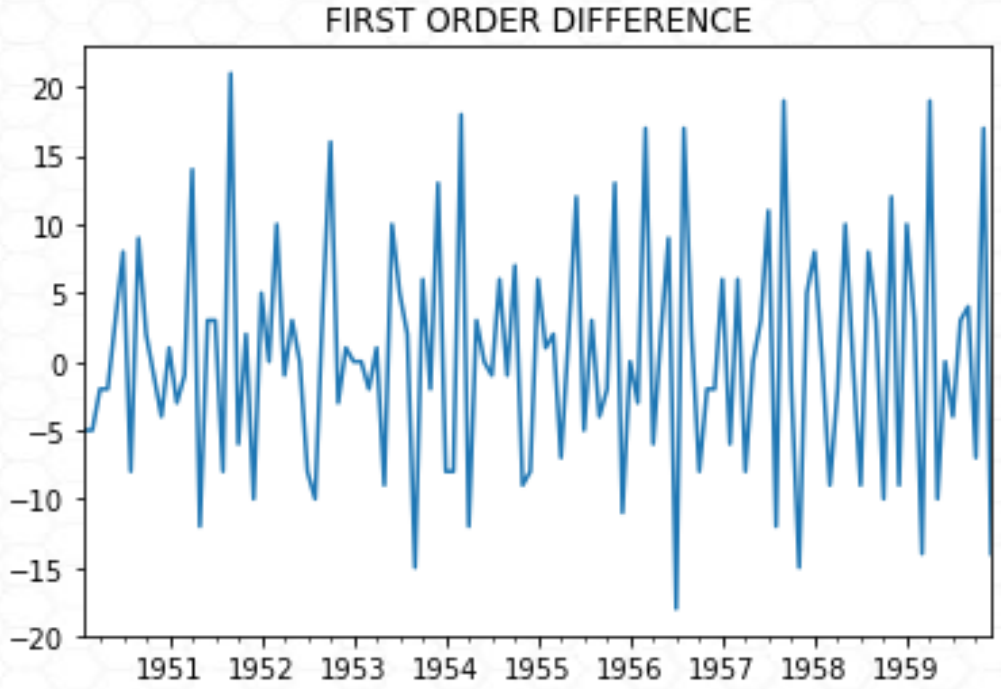



```
df2['c'].plot(ylim=[0,10000],title="NON-STATIONARY
DATA").autoscale(axis='x',tight=True);
```



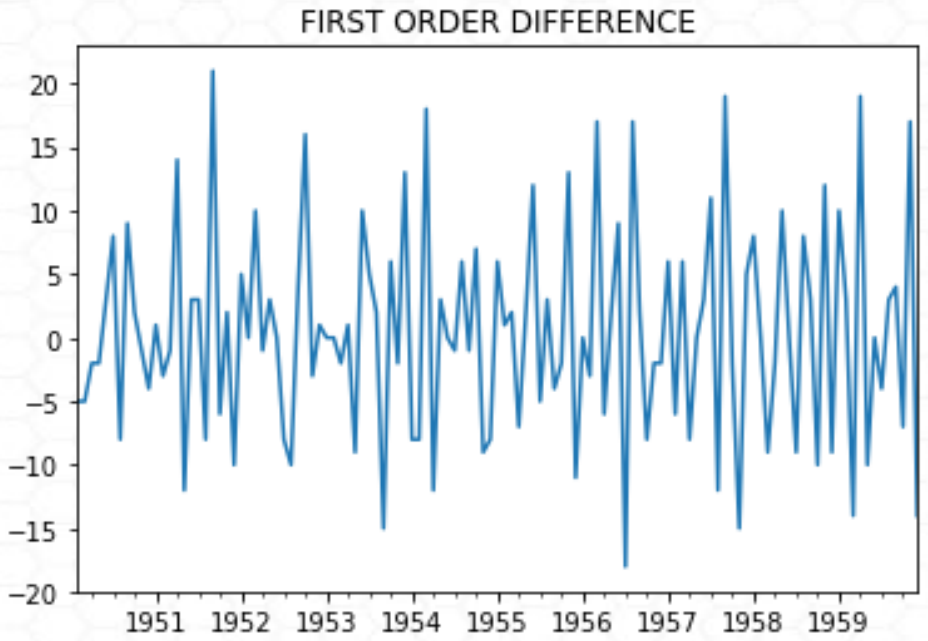
一階差分(First Order Difference)

```
df2['d1b'] = df2['b'] - df2['b'].shift(1)
df2['d1b'].plot(title="FIRST ORDER DIFFERENCE").autoscale(axis='x',tight=True);
```



一階差分(First Order Difference)

```
df2['d1b'] = df2['b'].diff()

df2['d1b'].plot(title="FIRST ORDER DIFFERENCE").autoscale(axis='x',tight=True);
```



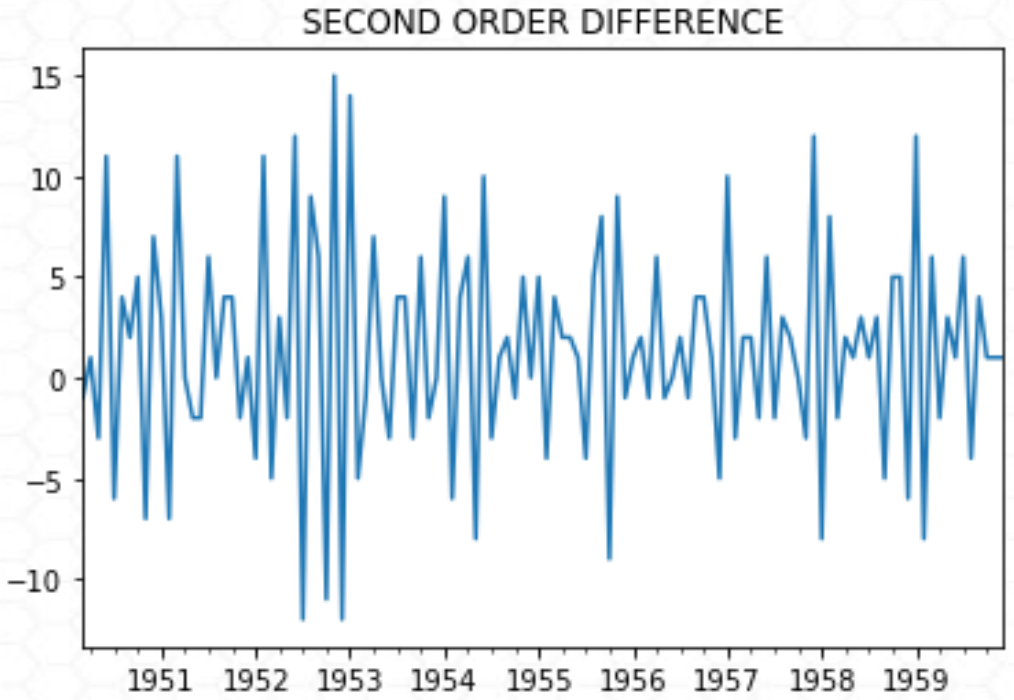
二階差分(Second Order Difference)

- 當趨勢為非線性時，若一階差分無效，便需要使用二階差分

$$
\begin{aligned}
y_t'' &= y_t' - y_{t-1}' \\
&= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\
&= y_t - 2y_{t-1} + y_{t-2}
\end{aligned}
$$

二階差分(Second Order Difference)

```
df2['d2c'] = df2['c'].diff().diff()
df2['d2c'].plot(title="SECOND ORDER DIFFERENCE").autoscale(axis='x',tight=True)
```



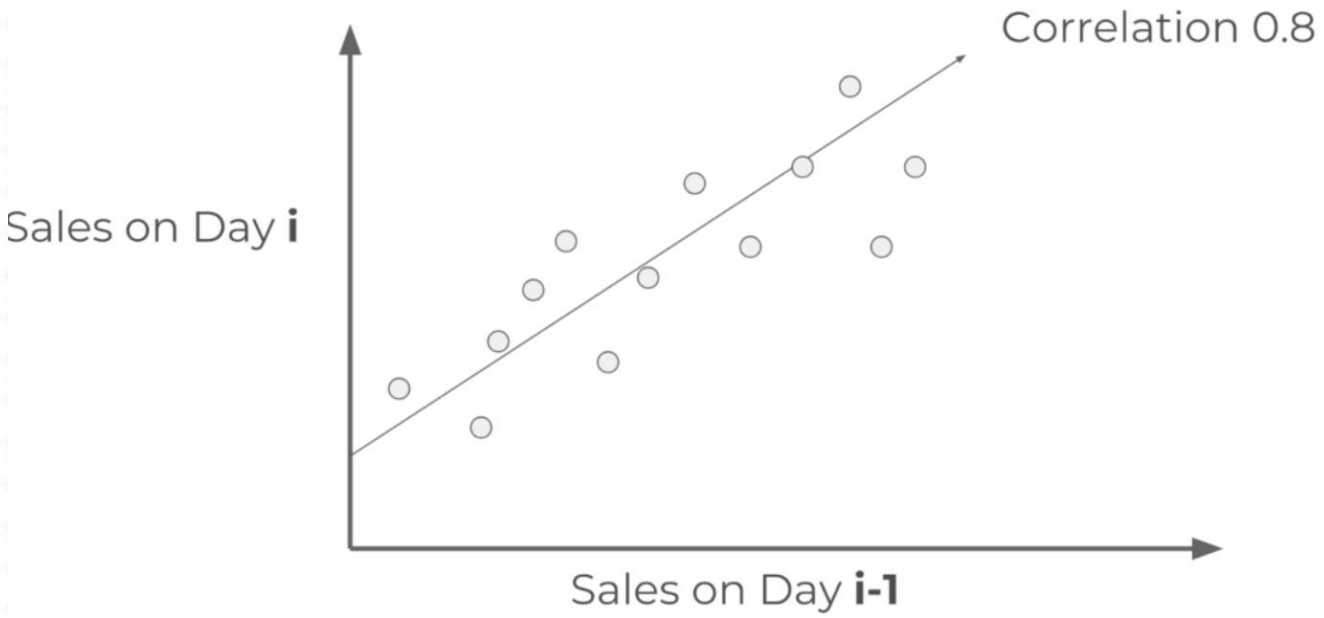
ACF & PACF

ACF v.s. PACF

- ACF
  - Autocorrelation Function Plot
  - 一個序列於其自身在不同時間點的相關係數
- PACF
  - Partial Autocorrelation Function Plot
  - 去除該變數所有落後其小於延遲項的影響後，考慮當期與落後q期間的相關係數

自相關 (Autocorrelation)

- 自相關 (Autocorrelation , aka Correlogram) 指序列跟自身延遲 x 時間點是否存在相關性



ACF 圖

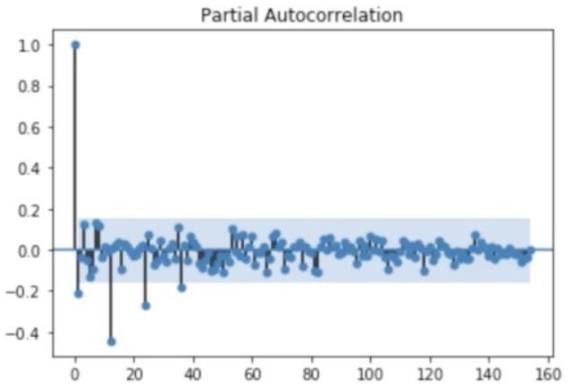

Correlation 0.8
Sales on Day i
Sales on Day i-1
Correlation 0.7
Sales on Day i
Sales on Day i-2
1
Autocorrelation
0
1
2
Shift

PACF


Residual
Sales on Day i
Sales on Day i-1


Residuals
Sales on Day i-1


Residuals fitted on Day i-1
Sales on Day i-2

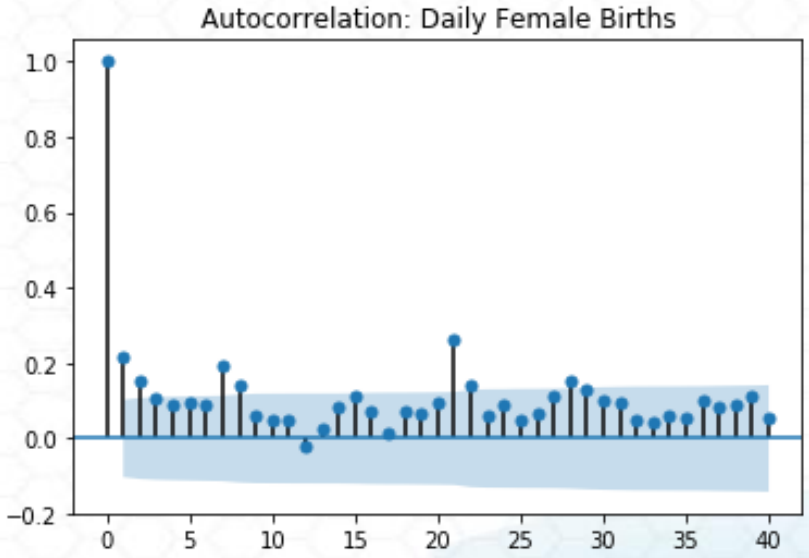

Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
20
40
60
80
100
120
140
160

讀取資料

```
import pandas as pd
import numpy as np
import statsmodels.api as sm

# Load a non-stationary dataset
df1 =
pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/airline_passengers.csv',index_col='Month',p
arse_dates=True)
df1.index.freq = 'MS'

# Load a stationary dataset
df2 =
pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/DailyTotalFemaleBirths.csv',index_col='Date'
,parse_dates=True)
df2.index.freq = 'D'
```

ACF 圖

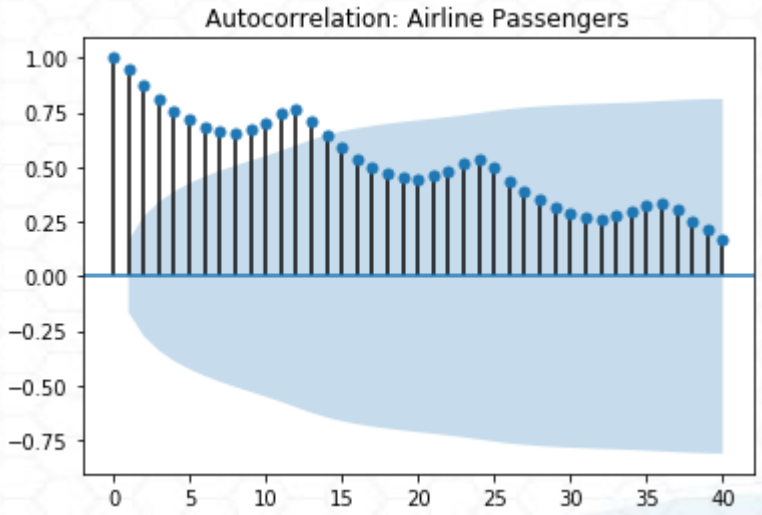
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.stattools import acf,pacf
title = 'Autocorrelation: Daily Female Births'
lags = 40
plot_acf(df2,title=title,lags=lags)
```



平穩資料集自我相關性快速跌落

ACF 圖

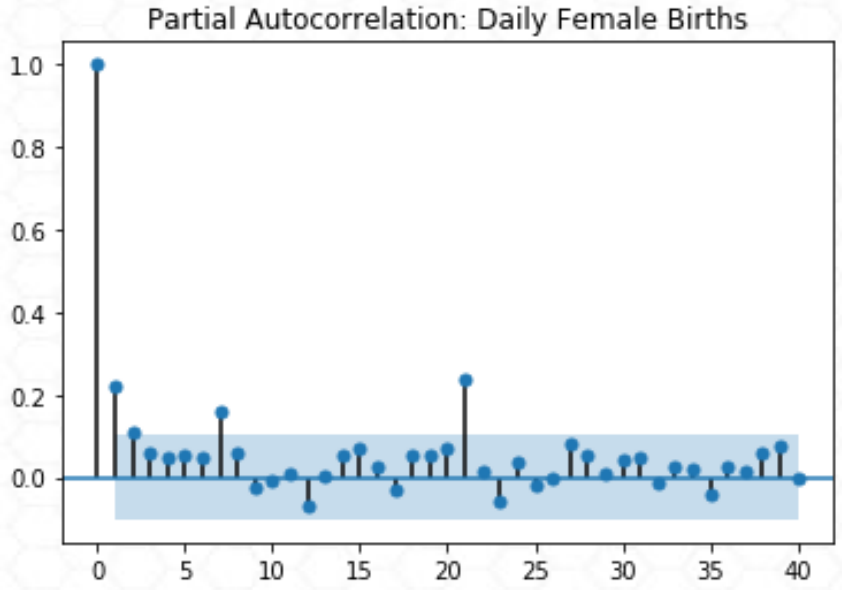
```
acf(df1['Thousands of Passengers'])
title = 'Autocorrelation: Airline Passengers'
lags = 40
plot_acf(df1,title=title,lags=lags);
```



非平穩資料集自我相關性緩慢下降

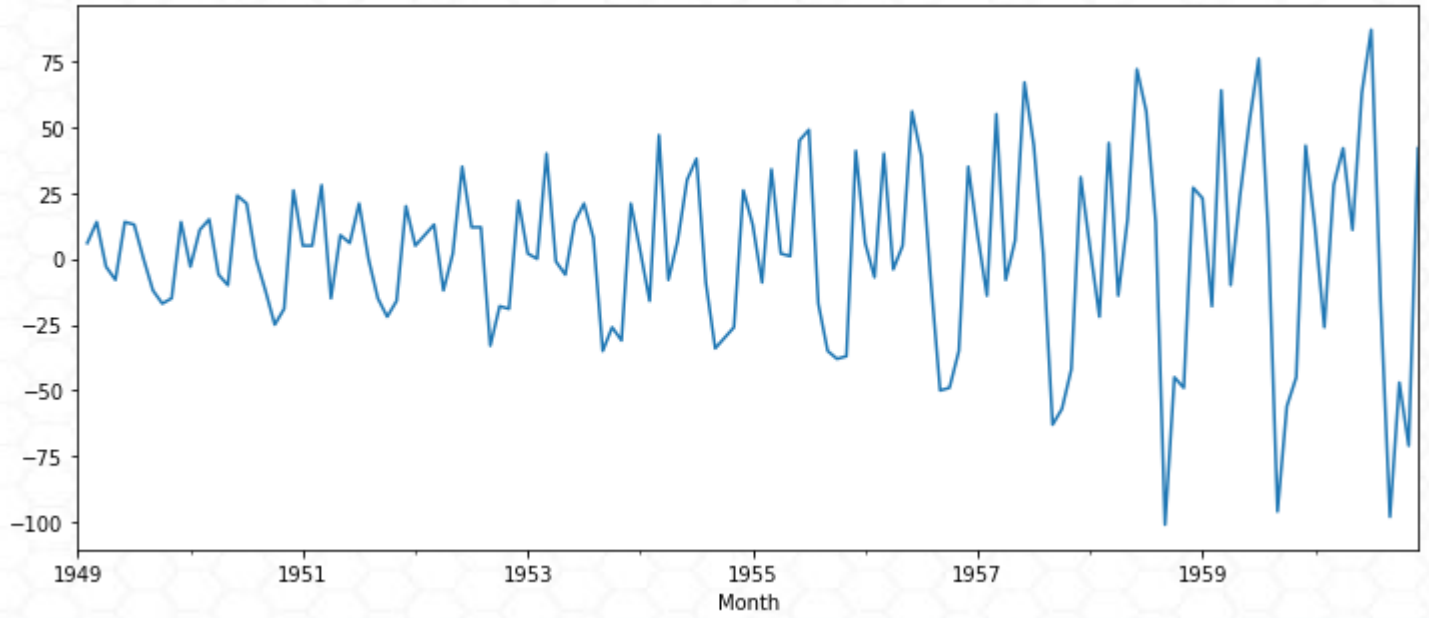
PACF圖

```
title='Partial Autocorrelation: Daily Female Births'
lags=40
plot_pacf(df2,title=title,lags=lags)
```



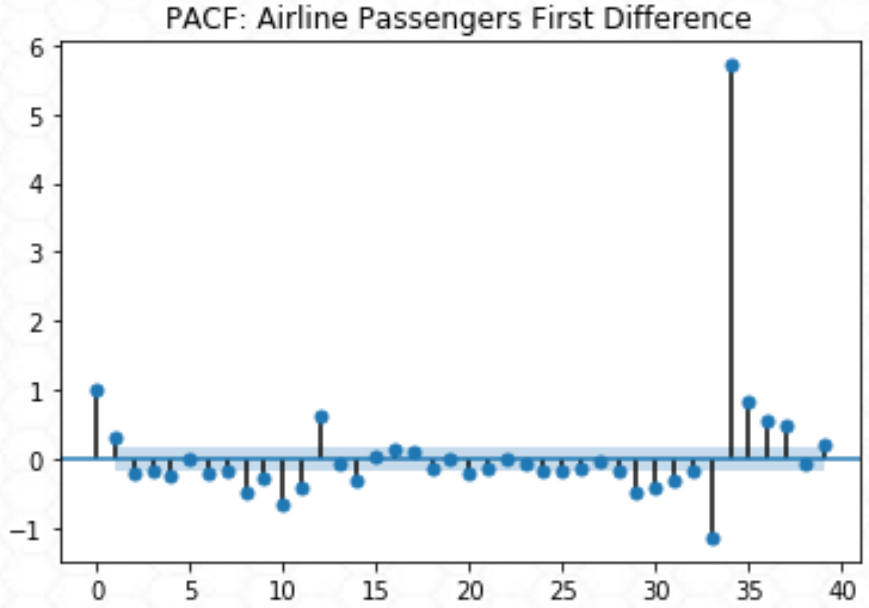
將時間序列變為平穩

```
from statsmodels.tsa.statespace.tools import diff

df1['d1'] = diff(df1['Thousands of Passengers'],k_diff=1)
df1['d1'].plot(figsize=(12,5))
```



PACF圖

```
title='PACF: Airline Passengers First Difference'
lags=40
plot_pacf(df1['d1'].dropna(),title=title,lags=np.arange(lags));
```



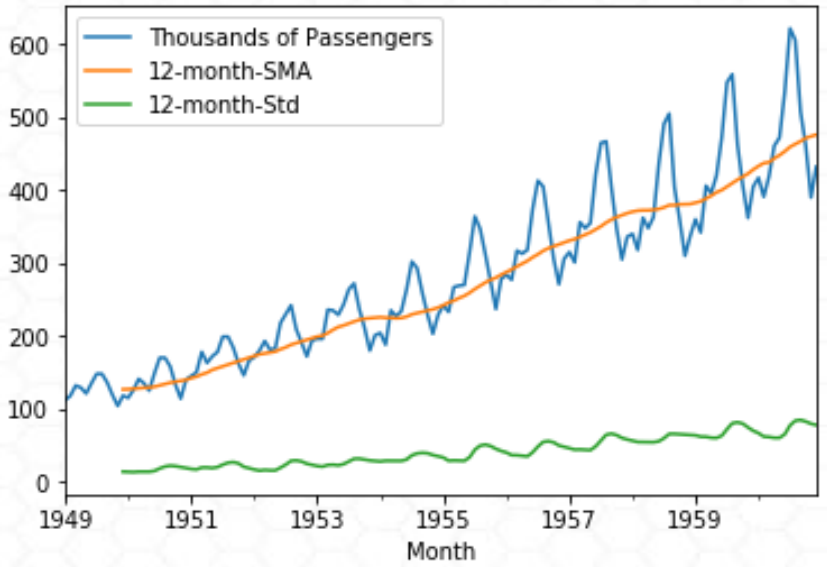
檢測序列是否平穩

Augmented Dickey-Fuller Test

- 為了確定一個序列是否平穩，我們可以使用Augmented Dickey-FullerTest。在這個測試虛無假設ϕ是否為 1
- 如果($p<0.05$)表明該檢驗拒絕原假設

繪製資料

```
df1['12-month-SMA'] = df1['Thousands of Passengers'].rolling(window=12).mean()
df1['12-month-Std'] = df1['Thousands of Passengers'].rolling(window=12).std()

df1[['Thousands of Passengers','12-month-SMA','12-month-Std']].plot();
```

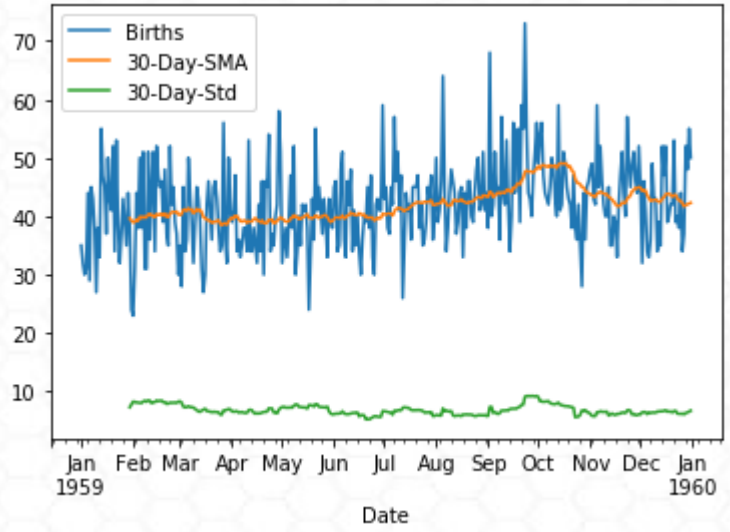


檢驗 P 值

```
from statsmodels.tsa.stattools import adfuller
print('Augmented Dickey-Fuller Test on Airline Data')
dftest = adfuller(df1['Thousands of Passengers'],autolag='AIC')
dfout = pd.Series(dftest[0:4],index=['ADF test statistic','p-value','#
lags used','# observations'])

print(dfout)
```

繪製資料

```
df2['30-Day-SMA'] = df2['Births'].rolling(window=30).mean()
df2['30-Day-Std'] = df2['Births'].rolling(window=30).std()

df2[['Births','30-Day-SMA','30-Day-Std']].plot();
```



檢驗 P 值

```
from statsmodels.tsa.stattools import adfuller
print('Augmented Dickey-Fuller Test on Airline Data')
dftest = adfuller(df2['Births’], autolag='AIC')
dfout = pd.Series(dftest[0:4],index=['ADF test statistic','p-value','# lags used','# observations'])

print(dfout)
```

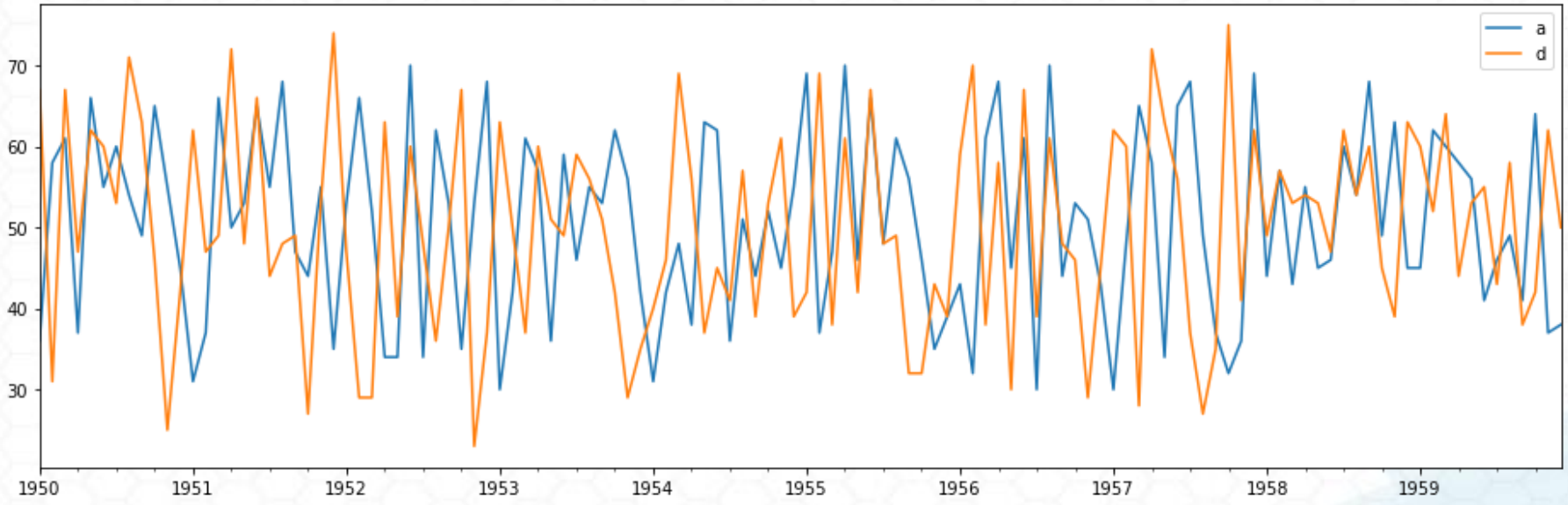
檢測序列是否存在因果關係

Granger Causality Tests

- **Granger Causality Tests** 是一種假設檢驗，用來確定一個時間序列是否有助於預測另一個時間序列
- 測量序列之間的相關性相當容易
  - e.g. 當一個序列上升時，另一個序列上升
- 但觀察一個序列的變化與另一個序列在一段時間後的變化之間的因果關係則不然
  - 即第一個序列的變化影響第二個序列的行為
  - 也有可能這兩個序列都受到第三個序列的影響，只是速度不同

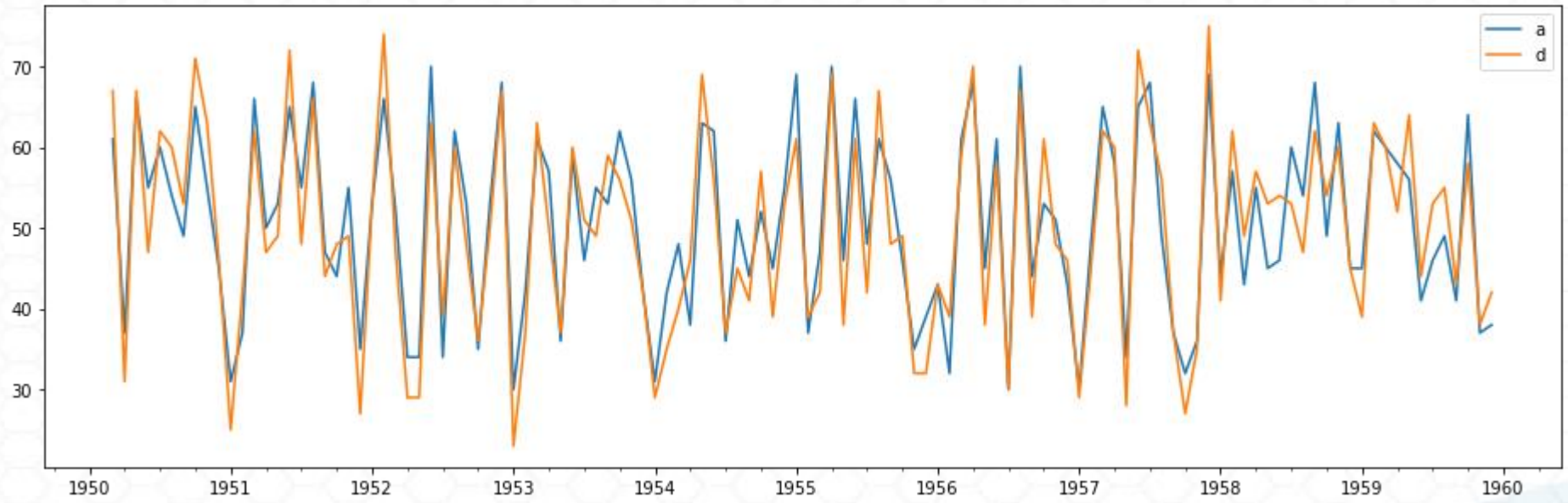
繪製兩序列

```
df3 = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/samples.csv',index_col=0,parse_dates=True)

df3.index.freq = 'MS'
df3[['a','d']].plot(figsize=(16,5));
```



比較遞延後的序列

```
df3['a'].iloc[2:].plot(figsize=(16,5),legend=True);
df3['d'].shift(2).plot(legend=True);
```



Granger Causality Tests

```
from statsmodels.tsa.stattools import grangercausalitytests
grangercausalitytests(df3[['a','d']],maxlag=3)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=1.7051  , p=0.1942  , df_denom=116, df_num=1
ssr based chi2 test:   chi2=1.7492  , p=0.1860  , df=1
likelihood ratio test: chi2=1.7365  , p=0.1876  , df=1
parameter F test:         F=1.7051  , p=0.1942  , df_denom=116, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=286.0339, p=0.0000  , df_denom=113, df_num=2
ssr based chi2 test:   chi2=597.3806, p=0.0000  , df=2
likelihood ratio test: chi2=212.6514, p=0.0000  , df=2
parameter F test:         F=286.0339, p=0.0000  , df_denom=113, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=188.7446, p=0.0000  , df_denom=110, df_num=3
ssr based chi2 test:   chi2=602.2669, p=0.0000  , df=3
likelihood ratio test: chi2=212.4789, p=0.0000  , df=3
parameter F test:         F=188.7446, p=0.0000  , df_denom=110, df_num=3
```

AR Model

ARIMA (Autoregressive Integrated Moving Average)

- 由以下三個模型組成:
  - **AR(p)** Autoregression - 利用當前觀測值與前一時期觀測值之間的依賴關係的回歸模型
  - **I(d)** Integration - 使用不同的觀測值(從前一個時間點的觀測值減去當下觀測值)來使時間序列保持平穩
  - **MA(q)** Moving Average - 利用觀測值與延遲觀測值的移動平均模型的殘差之間的相關性

AR Model

- **AR(p)** Autoregression - 利用當前觀測值與前一時期觀測值之間的依賴關係的回歸模型

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

where c is a constant, ϕ_1 and ϕ_2 are lag coefficients up to order p, and ε_t is white noise.

- AR(1)

$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t$$

- AR(2)

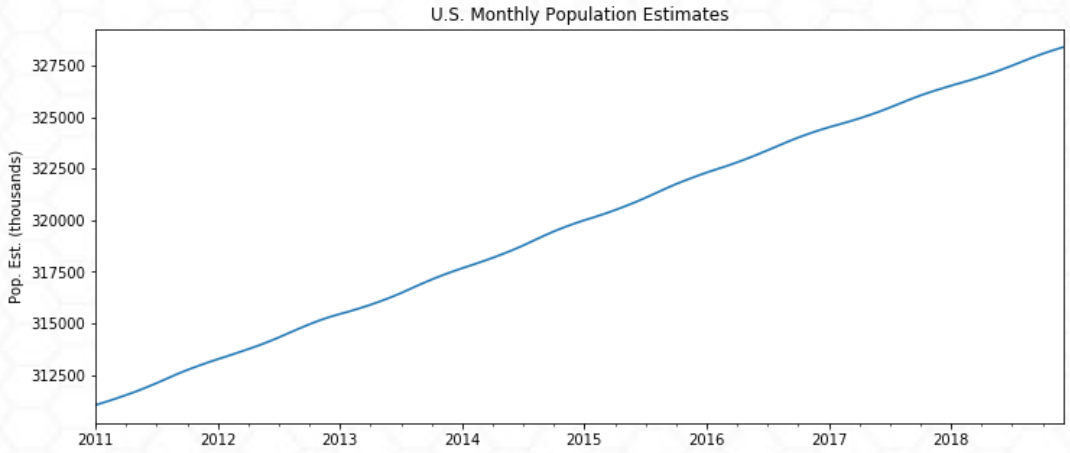
$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \varepsilon_t$$

讀取資料

```
import pandas as pd
import numpy as np
from statsmodels.tsa.ar_model import AR,ARResults
df =
pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/uspopulation.csv',index_col='DATE',parse_dates=True)
df.index.freq = 'MS'
df.head()
```

繪製資料圖

```
title='U.S. Monthly Population Estimates'
ylabel='Pop. Est. (thousands)'
xlabel=''

ax = df['PopEst'].plot(figsize=(12,5),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
```

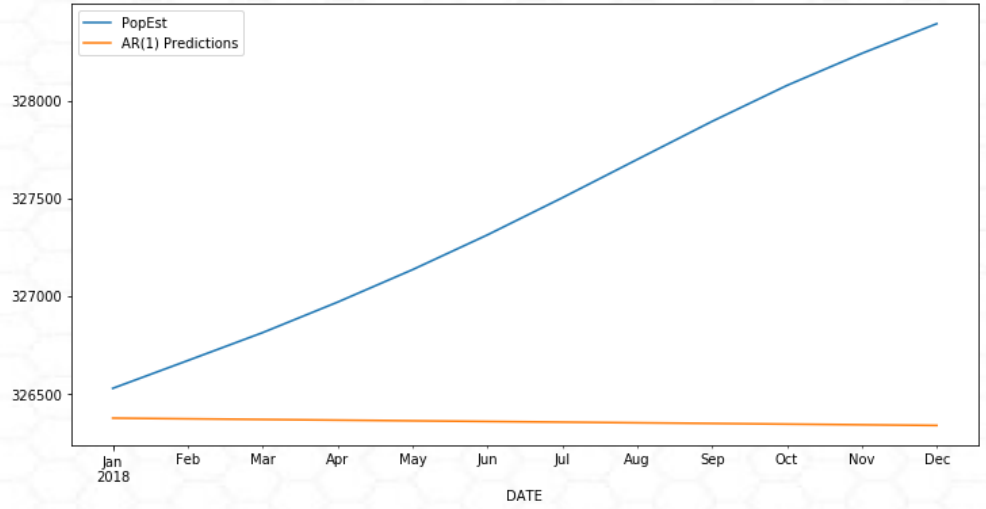


將資料分為訓練與測試資料集

```
train = df.iloc[:84]
test = df.iloc[84:]
```

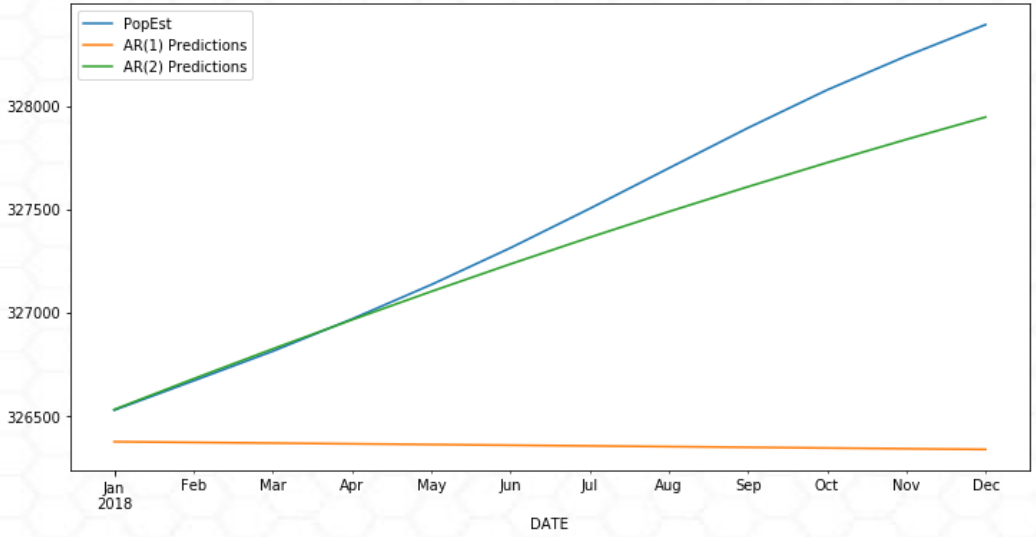
AR(1)

```
model = AR(train['PopEst'])
AR1fit = model.fit(maxlag=1,method='mle')
print(f'Lag: {AR1fit.k_ar}')
print(f'Coefficients:\n{AR1fit.params}')

start=len(train)
end=len(train)+len(test)-1
predictions1 = AR1fit.predict(start=start, end=end, dynamic=False).rename('AR(1) Predictions')

for i in range(len(predictions1)):
    print(f"predicted={predictions1[i]:<11.10}, expected={test['PopEst'][i]}")

test['PopEst'].plot(legend=True)
predictions1.plot(legend=True,figsize=(12,6));
```



AR(2)

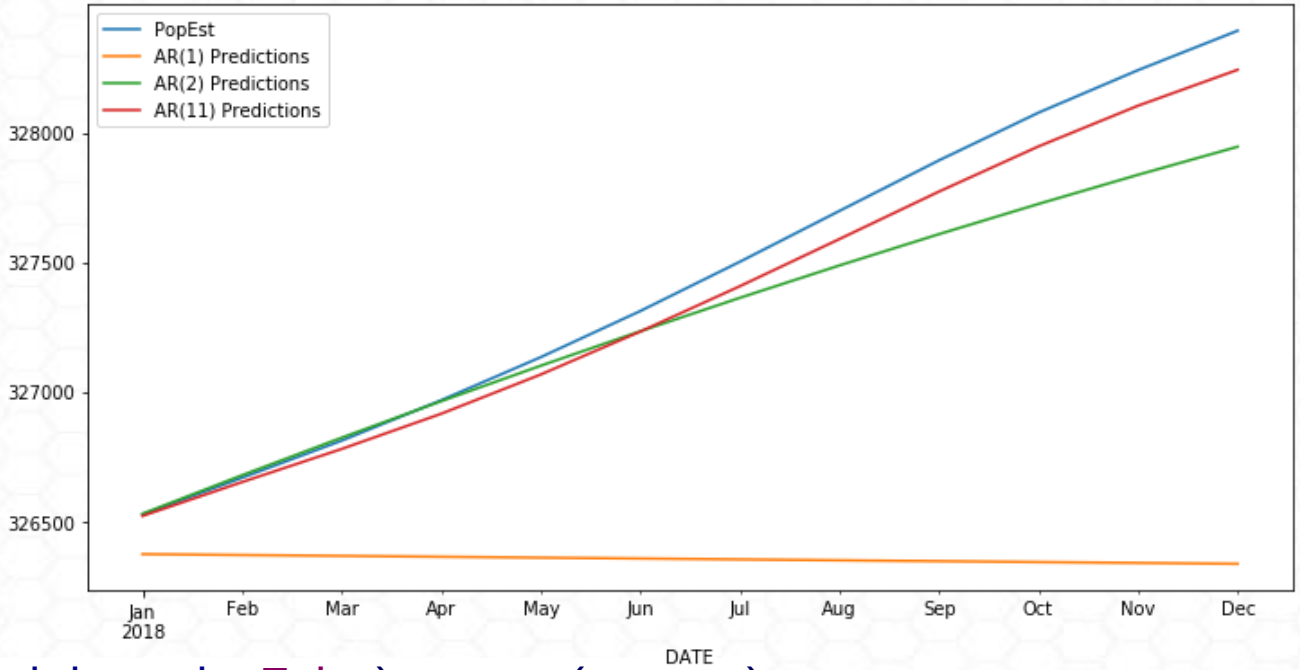
```
AR2fit = model.fit(maxlag=2,method='mle')
print(f'Lag: {AR2fit.k_ar}')
print(f'Coefficients:\n{AR2fit.params}')
```




```
start=len(train)
end=len(train)+len(test)-1
predictions2 = AR2fit.predict(start=start, end=end, dynamic=False).rename('AR(2) Predictions')

test['PopEst'].plot(legend=True)
predictions1.plot(legend=True)
predictions2.plot(legend=True,figsize=(12,6));
```

Auto - AR

```
ARfit = model.fit(method='mle')
print(f'Lag: {ARfit.k_ar}')
print(f'Coefficients:\n{ARfit.params}')

start = len(train)
end = len(train)+len(test)-1
rename = f'AR(11) Predictions'
```




```
predictions11 = ARfit.predict(start=start,end=end,dynamic=False).rename(rename)

test['PopEst'].plot(legend=True)
predictions1.plot(legend=True)
predictions2.plot(legend=True)
predictions11.plot(legend=True,figsize=(12,6));
```

評估模型

```
from sklearn.metrics import mean_squared_error

labels = ['AR(1)','AR(2)','AR(11)']
preds = [predictions1, predictions2, predictions11]

for i in range(3):
    error = mean_squared_error(test['PopEst'], preds[i])
    print(f'{labels[i]} Error: {error:11.10}')

modls = [AR1fit,AR2fit,ARfit]

for i in range(3):
    print(f'{labels[i]} AIC: {modls[i].aic:6.5}')
```

產生預測

```
model = AR(df['PopEst'])
ARfit = model.fit(maxlag=11,method='mle')
fcast = ARfit.predict(start=len(df), end=len(df)+12,
dynamic=False).rename('Forecast')
df['PopEst'].plot(legend=True)
fcast.plot(legend=True,figsize=(12,6));
```

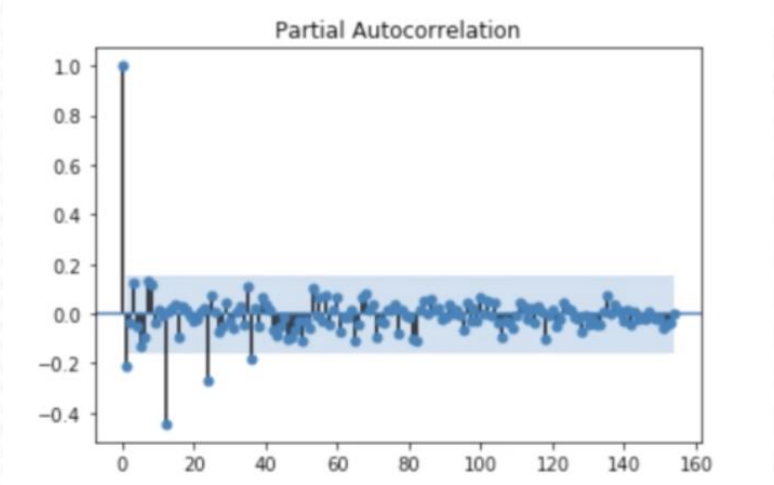
選擇Arima參數

選擇ARIMA 的參數

- P: 模型中的延遲觀測值數量
- d: 採取差分的次數
- q: 移動平均的區間大小

AR v.s. MA 模型

- 在k值快速下降的話，使用AR-K模型



人工判讀有太多不確定性

- 慢慢下降的話，使用ＭＡ模型

pmdarima Auto-ARIMA

- 安裝 pmdarima
  - pip install pmdarima
- pmadarima 使用Grid Search (網格搜尋法) 搜尋所有可能p,d,q參數
- 使用AIC挑選最佳模型

Auto Arima

```
auto_arima(df2['Births'],error_action='ignore').summary()
```

Dep. Variable:	y	**No. Observations:**	365
Model:	SARIMAX(1, 1, 1)	**Log Likelihood**	-1226.077
Date:	Fri, 22 Mar 2019	**AIC**	2460.154
Time:	10:46:05	**BIC**	2475.743
Sample:	0	**HQIC**	2466.350
	- 365		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0132	0.014	0.975	0.330	-0.013	0.040
ar.L1	0.1299	0.059	2.217	0.027	0.015	0.245
ma.L1	-0.9694	0.016	-62.235	0.000	-1.000	-0.939
sigma2	48.9989	3.432	14.279	0.000	42.273	55.725

Ljung-Box (Q):	36.69	**Jarque-Bera (JB):**	26.17
Prob(Q):	0.62	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.97	**Skew:**	0.58
Prob(H) (two-sided):	0.85	**Kurtosis:**	3.62

(p,d,q) ARIMA Order of (1,1,1)
no seasonal_order component

Stepwise Auto Arima

```
stepwise_fit = auto_arima(df2['Births'], start_p=0, start_q=0,
                          max_p=6, max_q=3, m=12,
                          seasonal=False,
                          d=None, trace=True,
                          error_action='ignore
                          suppress_warnings=True,
                          stepwise=True)

stepwise_fit.summary()
```

Dep. Variable:	D.y	**No. Observations:**	364
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-1226.077
Method:	css-mle	**S.D. of innovations**	7.000
Date:	Mon, 06 May 2019	**AIC**	2460.154
Time:	00:13:19	**BIC**	2475.742
Sample:	1	**HQIC**	2466.350

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0152	0.014	1.068	0.286	-0.013	0.043
ar.L1.D.y	0.1299	0.056	2.334	0.020	0.021	0.239
ma.L1.D.y	-0.9694	0.019	-51.415	0.000	-1.006	-0.932

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	7.6996	+0.0000j	7.6996	0.0000
MA.1	1.0316	+0.0000j	1.0316	0.0000

Stepwise Auto Arima

```
stepwise_fit = auto_arima(df1['Thousands of Passengers'],
                          start_p=1, start_q=1,
                          max_p=3, max_q=3, m=12,
                          start_P=0, seasonal=True,
                          d=None, D=1, trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)

stepwise_fit.summary()
```

Dep. Variable:	y	**No. Observations:**	144
Model:	SARIMAX(1, 1, 0)x(2, 1, 1, 12)	**Log Likelihood**	-502.480
Date:	Fri, 22 Mar 2019	**AIC**	1016.960
Time:	10:46:39	**BIC**	1034.211
Sample:	0	**HQIC**	1023.970
	- 144		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0045	0.178	0.025	0.980	-0.345	0.354
ar.L1	-0.3766	0.077	-4.890	0.000	-0.527	-0.226
ar.S.L12	0.6891	0.140	4.918	0.000	0.414	0.964
ar.S.L24	0.3091	0.107	2.883	0.004	0.099	0.519
ma.S.L12	-0.9742	0.511	-1.906	0.057	-1.976	0.028
sigma2	113.2075	48.855	2.317	0.020	17.453	208.961

Ljung-Box (Q):	58.67	**Jarque-Bera (JB):**	12.12
Prob(Q):	0.03	**Prob(JB):**	0.00
Heteroskedasticity (H):	2.70	**Skew:**	0.10
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.48

ARMA v.s. ARIMA

ARMA v.s. ARIMA

- 對平穩序列
  - ARMA
- 對不平穩序列
  - ARIMA
  - 使用I 讓序列便平穩

ARMA

- AR(1)

$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t$$

- MA(1)

$$y_t = \mu + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

- ARMA(1,1)

$$y_t = c + \phi_1 y_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

c is a constant

μ the expectation of y_t (often assumed to be zero)

φ_1 AR lag coefficient,

θ_1 MA lag coefficient

ε white noise

讀取資料

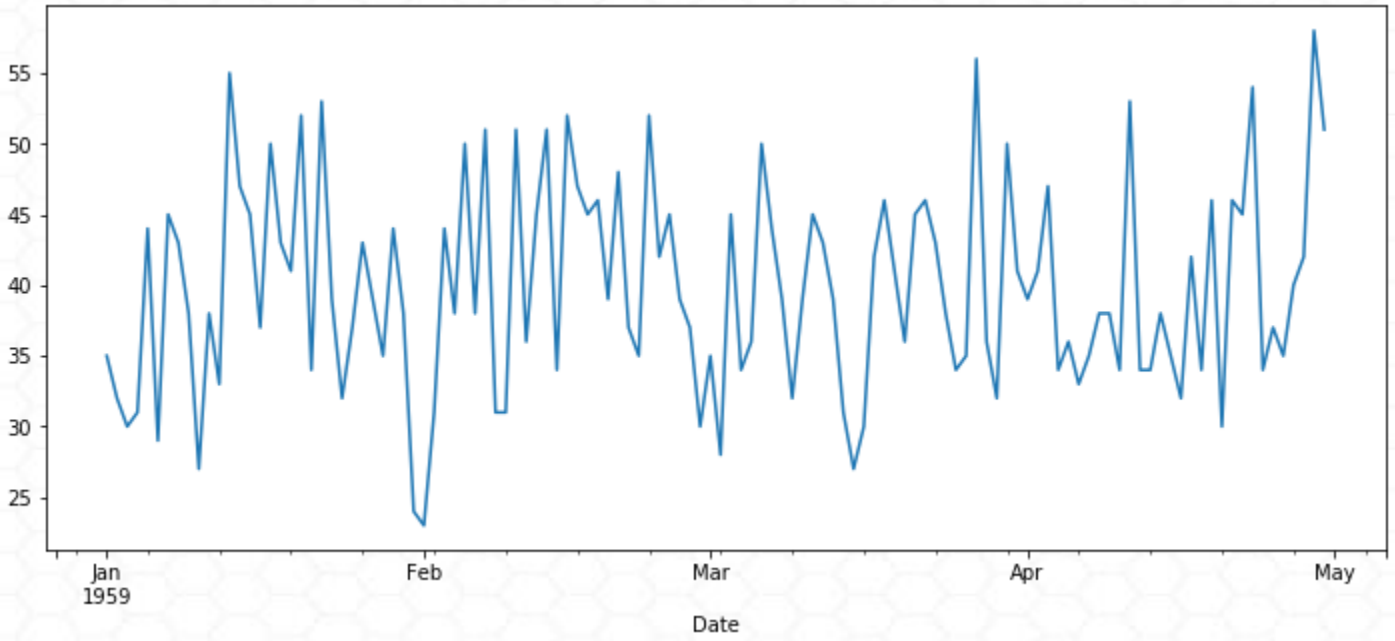
```
df1 = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/DailyTotalFemaleBirths.csv',index_col='Date',parse_dates=True)
df1.index.freq = 'D'
df1 = df1[:120]

df2 = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/TradeInventories.csv',index_col='Date',parse_dates=True)
df2.index.freq='MS'
```

Dickey-Fuller Test

```
from statsmodels.tsa.stattools import adfuller

def adf_test(series,title=''):
    """
    Pass in a time series and an optional title, returns an ADF report
    """
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles differenced data

    labels = ['ADF test statistic','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())          # .to_string() removes the line "dtype: float64"

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is non-stationary")
```

繪製資料

```
df1['Births'].plot(figsize=(12,5));
```



Dickey-Fuller Test

adf_test(df1['Births'])

```
Augmented Dickey-Fuller Test: None
ADF test statistic     -9.855384e+00
p-value                 4.373545e-17
# lags used             0.000000e+00
# observations          1.190000e+02
critical value (1%)    -3.486535e+00
critical value (5%)    -2.886151e+00
critical value (10%)   -2.579896e+00
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary
```

Auto ARIMA

```
auto_arima(df1['Births'],seasonal=False).summary()
```

Dep. Variable:	y	**No. Observations:**	120
Model:	ARMA(2, 2)	**Log Likelihood**	-405.370
Method:	css-mle	**S.D. of innovations**	6.991
Date:	Sat, 23 Mar 2019	**AIC**	822.741
Time:	12:02:45	**BIC**	839.466
Sample:	0	**HQIC**	829.533

	coef	std err	z	P>|z|	[0.025	0.975]
const	39.8162	0.108	368.841	0.000	39.605	40.028
ar.L1.y	1.8568	0.081	22.933	0.000	1.698	2.016
ar.L2.y	-0.8814	0.073	-12.030	0.000	-1.025	-0.738
ma.L1.y	-1.8634	0.109	-17.126	0.000	-2.077	-1.650
ma.L2.y	0.8634	0.108	8.020	0.000	0.652	1.074

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0533	-0.1582j	1.0652	-0.0237
AR.2	1.0533	+0.1582j	1.0652	0.0237
MA.1	1.0000	+0.0000j	1.0000	0.0000
MA.2	1.1583	+0.0000j	1.1583	0.0000

建立模型

```
train = df1.iloc[:90]
test = df1.iloc[90:]

from statsmodels.tsa.arima_model import ARMA
model = ARMA(train['Births'],order=(2,2))
results = model.fit()
results.summary()
```

產生預測結果

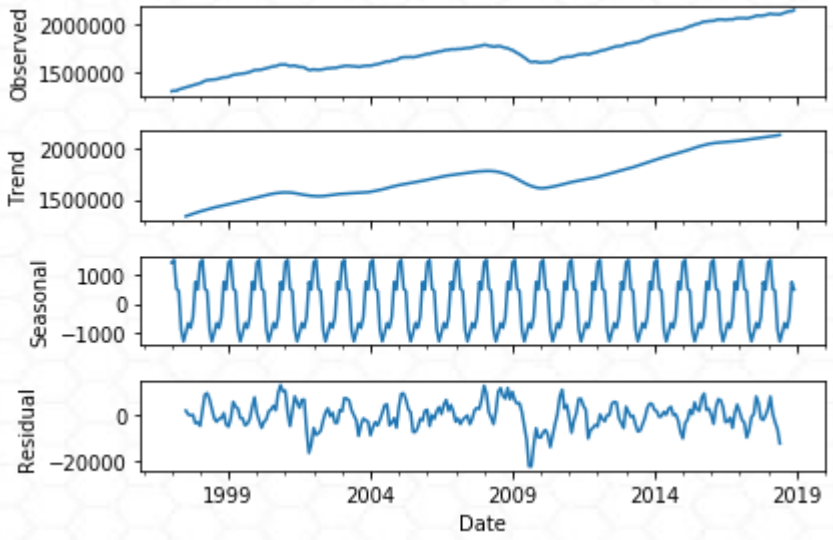
```
start=len(train)
end=len(train)+len(test)-1
predictions = results.predict(start=start, end=end).rename('ARMA(2,2) Predictions')

title = 'Daily Total Female Births'
ylabel='Births'
xlabel=''

ax = test['Births'].plot(legend=True,figsize=(12,6),title=title)
predictions.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel);
```

繪製資料

```
import matplotlib.ticker as ticker
formatter = ticker.StrMethodFormatter('{x:,.0f}')

title = 'Real Manufacturing and Trade Inventories'
ylabel='Chained 2012 Dollars'
xlabel=''

ax = df2['Inventories'].plot(figsize=(12,5),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```

ETS 分解

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df2['Inventories'], model='additive')
result.plot();
```



Auto ARIMA

```
auto_arima(df2['Inventories'],seasonal=False).summary()
```

Dep. Variable:	D.y	**No. Observations:**	263
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-2610.252
Method:	css-mle	**S.D. of innovations**	4938.258
Date:	Sat, 23 Mar 2019	**AIC**	5228.505
Time:	12:18:53	**BIC**	5242.794
Sample:	1	**HQIC**	5234.247

	coef	std err	z	P>|z|	[0.025	0.975]
const	3472.9857	1313.669	2.644	0.009	898.241	6047.731
ar.L1.D.y	0.9037	0.039	23.414	0.000	0.828	0.979
ma.L1.D.y	-0.5732	0.076	-7.545	0.000	-0.722	-0.424

Roots

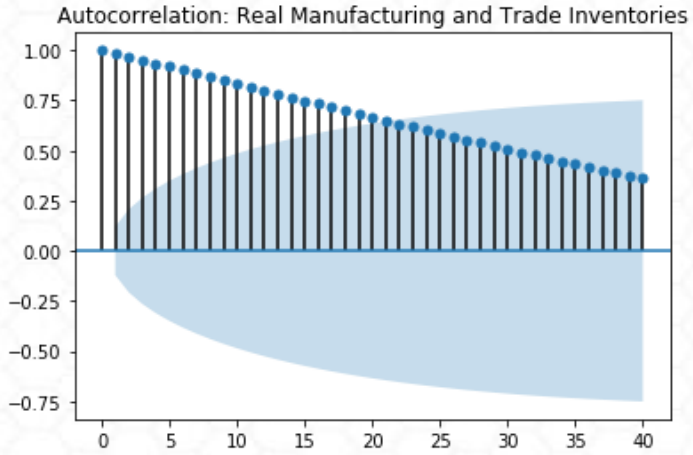
	Real	Imaginary	Modulus	Frequency
AR.1	1.1065	+0.0000j	1.1065	0.0000
MA.1	1.7446	+0.0000j	1.7446	0.0000

一階差分

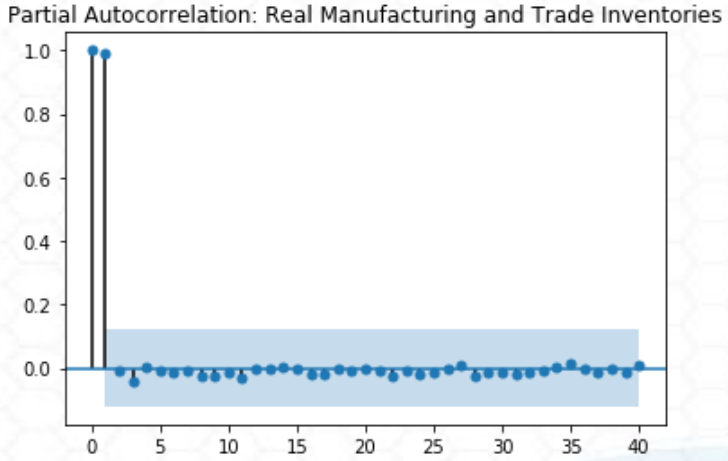
```
from statsmodels.tsa.statespace.tools import diff
df2['d1'] = diff(df2['Inventories'],k_diff=1)


adf_test(df2['d1'],'Real Manufacturing and Trade Inventories')
```

PACF v.s. ACF

```
title = 'Autocorrelation'
lags = 40
plot_acf(df2['Inventories'],title=title,lags=lags);
```




```
title = 'Partial Autocorrelation'
lags = 40
plot_pacf(df2['Inventories'],title=title,lags=lags);
```



Stepwise Auto ARIMA

```
stepwise_fit = auto_arima(df2['Inventories'], start_p=0, start_q=0,
                          max_p=2, max_q=2, m=12,
                          seasonal=False,
                          d=None, trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)

stepwise_fit.summary()
```

適配模型

```
train = df2.iloc[:252]
test = df2.iloc[252:]

model = ARIMA(train['Inventories'],order=(1,1,1))
results = model.fit()
results.summary()
```

產生預測

```
start=len(train)
end=len(train)+len(test)-1
predictions = results.predict(start=start, end=end, dynamic=False, typ='levels').rename('ARIMA(1,1,1) Predictions')

for i in range(len(predictions)):
    print(f"predicted={predictions[i]:<11.10}, expected={test['Inventories'][i]}")

title = 'Real Manufacturing and Trade Inventories'
ylabel='Chained 2012 Dollars'
xlabel=''

ax = test['Inventories'].plot(legend=True,figsize=(12,6),title=title)
predictions.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```

評估模型

```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(test['Inventories'], predictions)
print(f'ARIMA(1,1,1) MSE Error: {error:11.10}')

from statsmodels.tools.eval_measures import rmse

error = rmse(test['Inventories'], predictions)
print(f'ARIMA(1,1,1) RMSE Error: {error:11.10}')
```

適配所有資料

```
model = ARIMA(df2['Inventories'],order=(1,1,1))
results = model.fit()
fcast = results.predict(len(df2),len(df2)+11,typ='levels').rename('ARIMA(1,1,1) Forecast')

title = 'Real Manufacturing and Trade Inventories'
ylabel='Chained 2012 Dollars'
xlabel=''

ax = df2['Inventories'].plot(legend=True,figsize=(12,6),title=title)
fcast.plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter);
```

THANK YOU