Statsmodels를 이용한 시계열분석

- 시계열 분석 (Time Series Analysis)
  - 시계열: 보통 동일한 시간 간격마다 측정한 데이터(x축: 시간, y축: 측정 데이터)
  - 독립 변수로 오직 시간만 존재
  - 과거에 관측한 값들을 기반으로 미래의 값 예측
  - 활용 예: 과거 행위 분석, 현재 성취도 분석, 미래 예측 및 계획 수립에 활용

- Components of Time Series
  - Trend: 장기간에 걸쳐서 지속적으로 증가/감소/유지하는 성향
  - Seasonality: 고정된 시간 주기마다 반복되는 패턴 (예: 아이스크림 판매)
  - Irregularity: 규칙성 X, 예측 불가능, 우연적으로 발생, 짧게 발생 (예: 홍수, 파업..)

◉ 항공 탑승객 데이터

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
dataset = pd.read_csv('AirPassengers.csv')
dataset.tail()
```

Out [2]:

	Month	#Passengers
139	1960-08	606
140	1960-09	508
141	1960-10	461
142	1960-11	390
143	1960-12	432

◉ 시간 데이터 수정

In [3]:
```
dataset['Month'] = pd.to_datetime(dataset['Month'], infer_datetime_format = True)
dataset.tail()
```

Out[3]:

	Month	#Passengers
139	1960-08-01	606
140	1960-09-01	508
141	1960-10-01	461
142	1960-11-01	390
143	1960-12-01	432

◉ index 변경 및 timeseries 선택

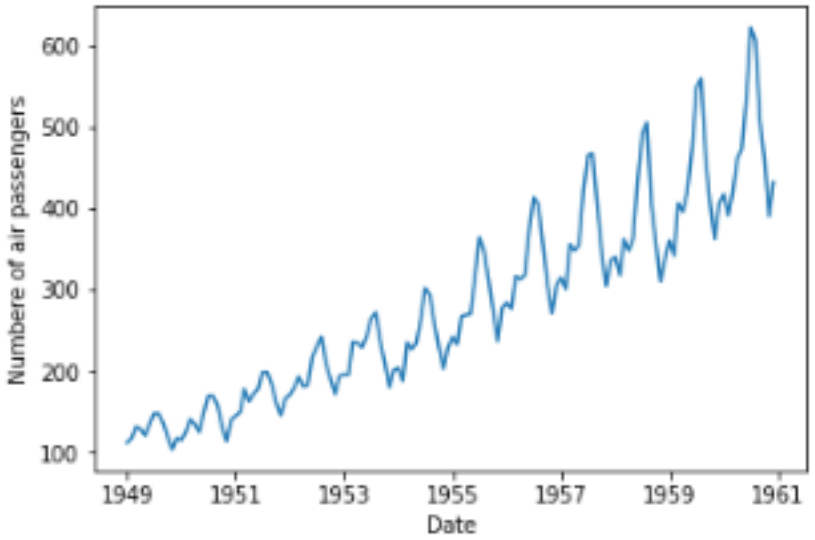
In [4]:
```
dataset.set_index('Month', inplace = True)
timeseries = dataset['#Passengers']
timeseries.tail()
```

Out[4]:
```
Month
1960-08-01    606
1960-09-01    508
1960-10-01    461
1960-11-01    390
1960-12-01    432
Name: #Passengers, dtype: int64
```

◉ 그래프 그리기

```
In [5]: plt.xlabel('Date')
        plt.ylabel('Numbere of air passengers')
        plt.plot(timeseries)

Out[5]: [<matplotlib.lines.Line2D at 0x1a5a0478b38>]
```



- Stationarity (정상성)
  - 비유동적임을 의미
  - 모델링이 쉽고 과거 데이터를 기반으로 미래 예측이 쉬움

- Stationarity 조건
  - Constant mean: 시간에 따라 평균이 고정적이어야 함
  - Constant variance: 시간 간격마다 변화량이 일정해야 함
  - Autocovariance: 시간에 의존적이지 않음
  - Autocovariance: 특정 시간 간격만큼 이동시킨 자기자신과의 공분산

Stationarity 확인

- Rolling statistics: 시간이 흐름에 따라 평균과 분산의 변화를 그려 변화하는지 평가

- Dickey-Fuller Test: non-stationary하다는 귀무 가설 테스트.
  - reject 조건: test statistic > critical value && p-value < 0.05

Stationarize

- 차분(differencing): 1차 차분, trend 제거에 용이
- 로그변환(logarithm): 표준편차가 비례/지수함수적으로 증가하는 추세일 때 용이

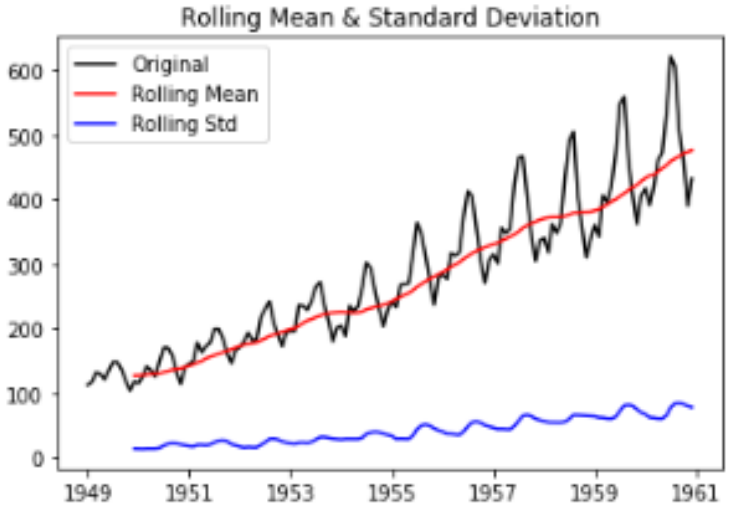
- Rolling statistics
  - 1년치마다의 평균과 표준편차

In [6]:
```
rolmean = timeseries.rolling(window=12).mean()
rolstd  = timeseries.rolling(window=12).std() #standard deviation
print(rolmean, rolstd)
```

Rolling statistics

- 시각화

In [7]:
```
#Plot rolling statistics
plt.plot(timeseries, color = 'black', label = 'Original')
plt.plot(rolmean, color = 'red', label = 'Rolling Mean')
plt.plot(rolstd, color = 'blue', label = 'Rolling Std')
plt.title("Rolling Mean & Standard Deviation")
plt.legend()
plt.show()
```



Dickey-Fuller Test

- Test 결과

In [8]:
```
# Dickey-Fuller test
from statsmodels.tsa.stattools import adfuller
df_test = adfuller(timeseries, autolag = 'AIC')
df_test
```

Out [8]:
```
(0.8153688792060433,
 0.9918802434376409,
 13,
 130,
 {'1%': -3.4816817173418295,
  '5%': -2.8840418343195267,
  '10%': -2.578770059171598},
 996.692930839019)
```

Dickey-Fuller Test

- Test 결과 – 정리하기

In [9]:
```
df_result = pd.Series(df_test[:4],
   index = ['Test Statistic', 'p-valaue', '#Lags used', '#Observations used'])
for key, value in df_test[4].items():
   df_result['Critical values (%s)' % key] = value

df_result
```

Out[9]:
```
Test Statistic                 0,815369
p-valaue                       0,991880
#Lags used                    13,000000
#Observations used           130,000000
Critical values (1%)          -3,481682
Critical values (5%)          -2,884042
Critical values (10%)         -2,578770
dtype: float64
```

- p-value: 0.5보다 작아야함
- Critical values: Test statistic 값보다 커야함

⊙ Stationarity test 함수화

In [10]:
```
def test_stationarity(timeseries):
    from statsmodels.tsa.stattools import adfuller

    #Determing rolling statistics
    rolmean = timeseries.rolling(window=12).mean()
    rolstd  = timeseries.rolling(window=12).std() #standard deviation

    #Plot rolling statistics:
    plt.plot(timeseries, color = 'black', label = 'Original')
    plt.plot(rolmean, color = 'red', label = 'Rolling Mean')
    plt.plot(rolstd, color = 'blue', label = 'Rolling Std')
    plt.title("Rolling Mean & Standard Deviation")
    plt.legend()
    plt.show()

    #Perform Dickey-Fuller test:
    print ('<Results of Dickey-Fuller Test>')
    df_test = adfuller(timeseries, autolag='AIC')
    df_result = pd.Series(df_test[:4],
        index = ['Test Statistic', 'p-valaue', '#Lags used', '#Observations used'])
    for key, value in df_test[4].items():
        df_result['Critical values (%s)' % key] = value
    print (df_result)
```

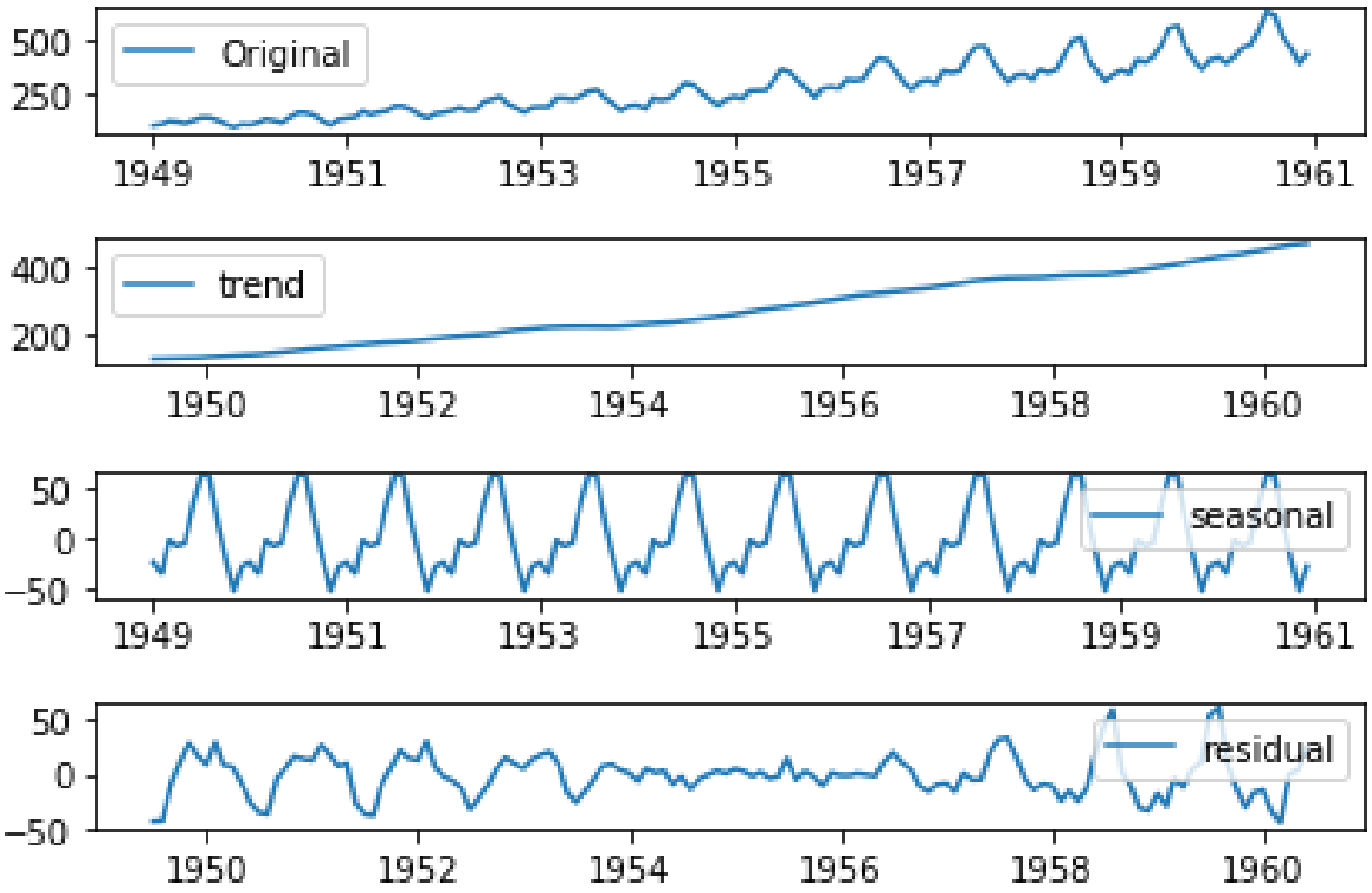
◉ Component 추출

In [11]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
decom = seasonal_decompose(timeseries)

trend = decom.trend
seasonal = decom.seasonal
residual = decom.resid #잔차

plt.subplot(411); plt.plot(timeseries, label = 'Original'); plt.legend()
plt.subplot(412); plt.plot(trend, label = 'trend'); plt.legend()
plt.subplot(413); plt.plot(seasonal, label = 'seasonal'); plt.legend()
plt.subplot(414); plt.plot(residual, label = 'residual'); plt.legend()
plt.tight_layout()
```

Component 추출 결과



ARIMA 모델의 의미

- 시계열 분석에 사용되는 대표적인 수리적 모형
- 자기회귀모형(AR, Auto Regression): 이전 값이 이후의 값에 영향을 미치는 경향
- 이동평균 모형(MA, Moving Average): 변수의 평균값이 지속적으로 증가하거나 감소화는 경향

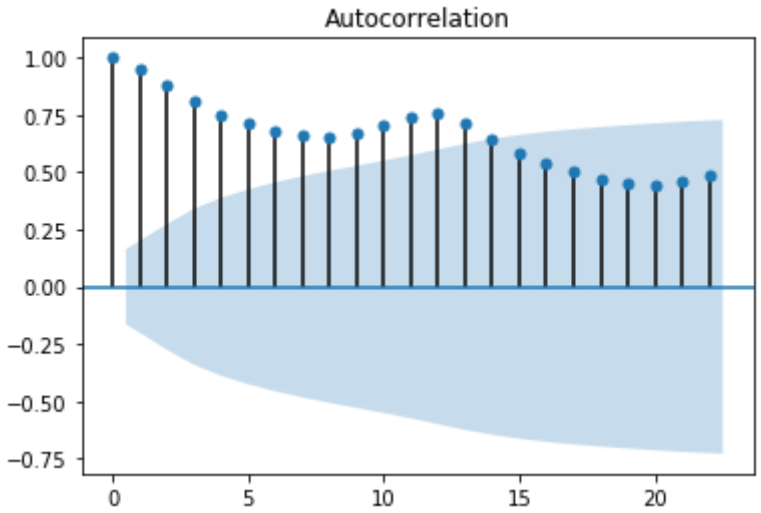
필요한 입력변수

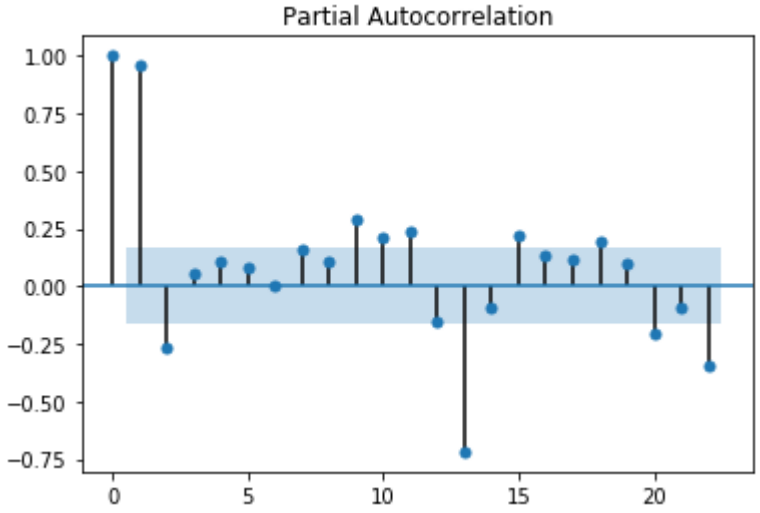
- p: AR에 필요한 autoregressive lags → PACF(Partial autocorrelation graph)로 결정
- d: I에 필요한 order of differentiation
- q: MA에 필요한 moving average → ACF(Autocorrelation graph)로 결정

ACF, PACF

- no difference

```
In [12]: from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
         plot_acf(timeseries)
         plot_pacf(timeseries)
         plt.show()
```



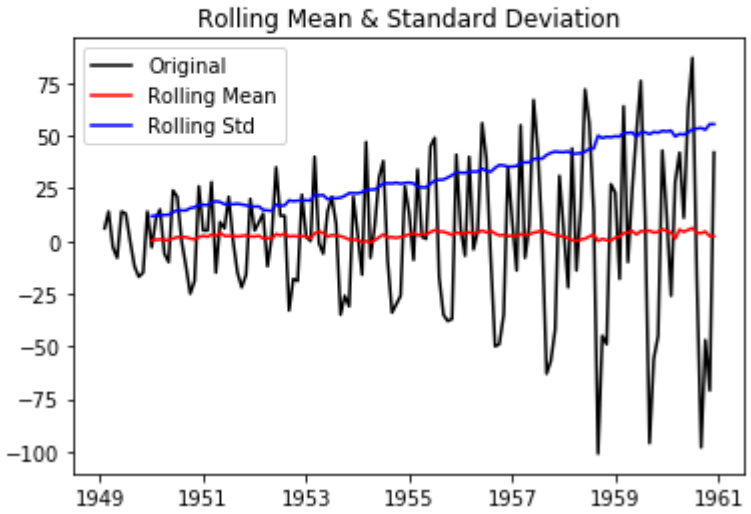


◉ ACF, PACF

- difference = 1

```
In [13]: timeseries_diff1 = timeseries.diff(periods = 1).dropna()

In [14]: test_stationarity(timeseries_diff1)
```

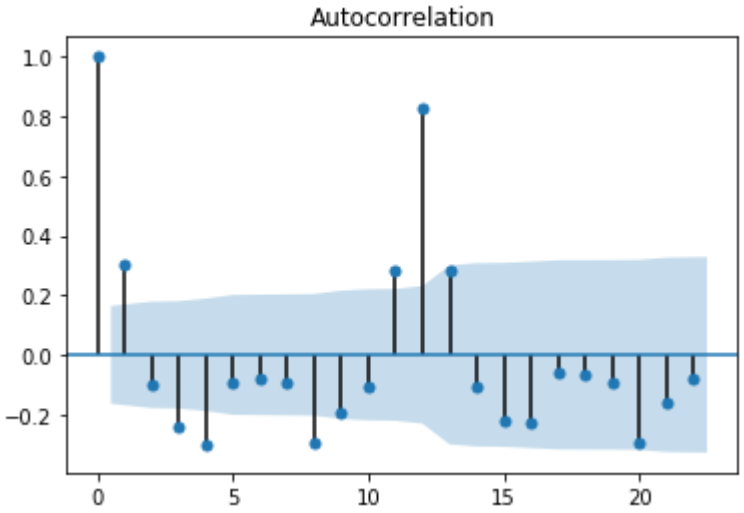


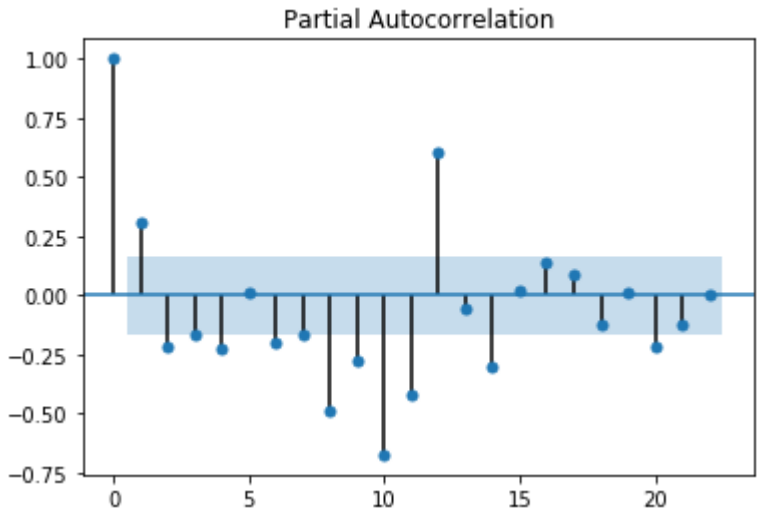

```
<Results of Dickey-Fuller Test>
Test Statistic                  -2.829267
p-valaue                         0.054213
#Lags used                      12.000000
#Observations used             130.000000
Critical values (1%)            -3.481682
Critical values (5%)            -2.884042
Critical values (10%)           -2.578770
dtype: float64
```

ACF, PACF

- difference = 1

```
In [15]: plot_acf(timeseries_diff1)
         plot_pacf(timeseries_diff1)
         plt.show()
```





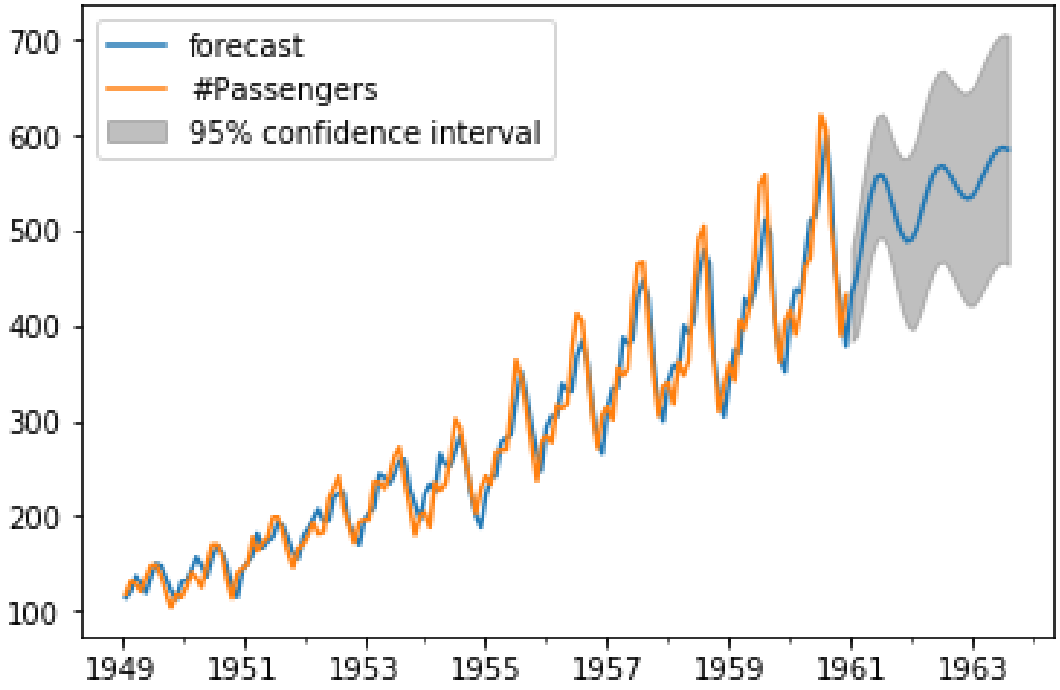
◉ 모델 적용

In [16]:
```
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima_model import ARIMAResults

model = ARIMA(timeseries, order = (2, 1, 2))
fit = model.fit(trend = 'c', full_output=True, disp = 1)
fit.summary()
```

◉ 예측치와 비교하기

```
In [18]: fit.plot_predict(end = '1963-08-01')
         plt.show()
```



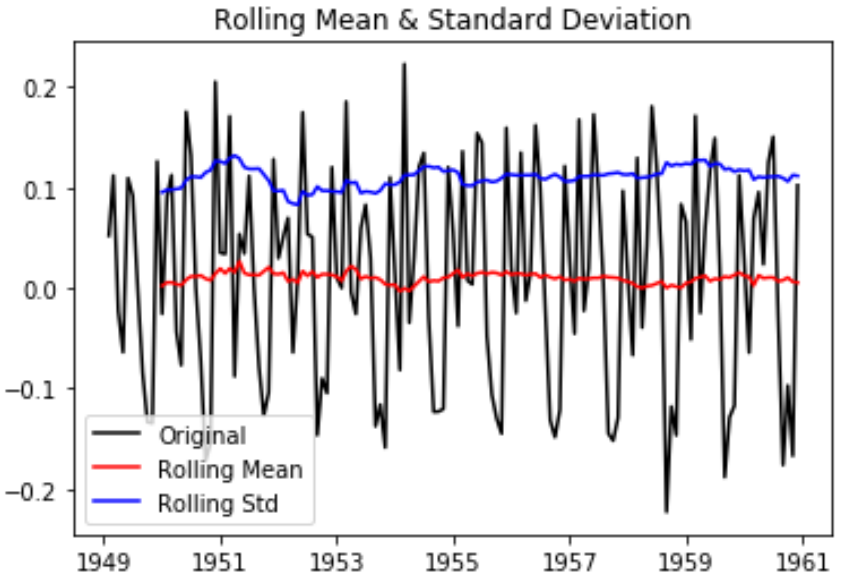
◉ 예측치 추출

```
In [19]: fit.forecast(steps = 10)
```

```
Out[19]: (array([433.10902642, 450.86859474, 479.78513435, 511.94330609,
                 539.29668454, 555.78519429, 558.74091437, 549.27886267,
                 531.66246762, 511.90266692]),
          array([24.71202286, 30.70942638, 32.23158701, 32.2801079 , 32.49895866,
                 32.97657601, 33.1691927 , 33.19978351, 34.10709378, 36.79361374]),
          array([[384.67435163, 481.5437012 ],
                 [390.67922504, 511.05796444],
                 [416.61238464, 542.95788405],
                 [448.67545718, 575.21115499],
                 [475.59989603, 602.99347305],
                 [491.15229298, 620.41809559],
                 [493.73049129, 623.75133745],
                 [484.2084827 , 614.34924264],
                 [464.8137922 , 598.51114304],
                 [439.78850912, 584.01682472]]))
```

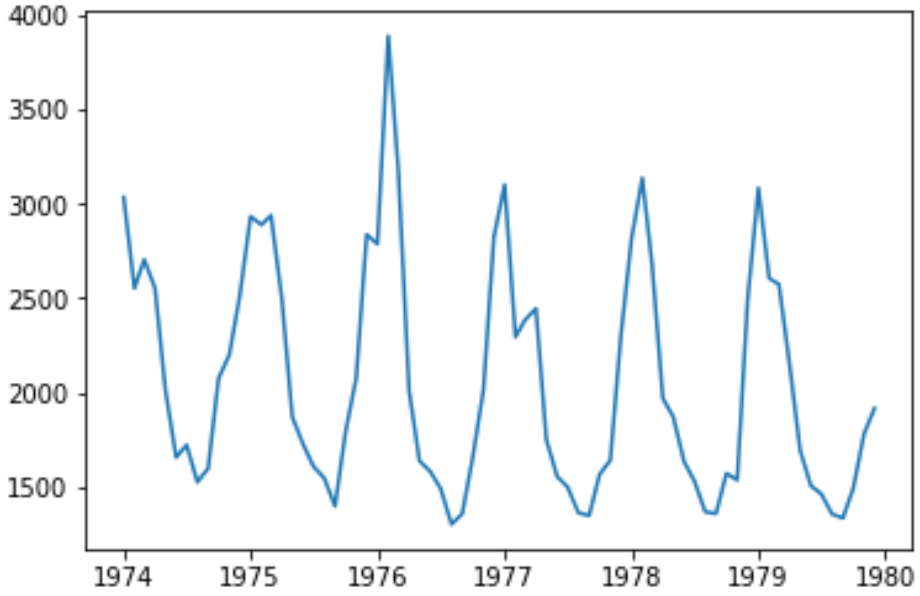
- log를 취한후에 다시 해보기
  - log 취한 후 차분
  - acf, pacf 확인
  - ARIMA 적용
  - predict 해보기
  - forecast 해보기

```
In [20]: timeseries_Log = np.log(timeseries)
         timeseries_LogDiff = timeseries_Log.diff().dropna()
         test_stationarity(timeseries_LogDiff)
```



데이터 개요

- 1974년부터 1979년까지의 영국의 호흡기 질환 사망자 수를 나타내는 시계열 데이터
- 시간 표기: 1년은 1.0, 1개월은 1/12



◉ ARIMA 모델 분석 결과

In [8]:
```
fit.plot_predict()
plt.show()
```