pythonで時系列分析

Pythonを使った時系列分析の方法、ひいてはモデル化を通した将来予測の方法について見ていく。

目次

1. 時系列分析とは

1-1. 時系列分析とは

その名の通り、時系列データを解析する手法。

例
「毎日の売り上げデータ」や「日々の気温のデータ」「月ごとの飛行機乗客数」など
毎日（あるいは毎週・毎月・毎年）増えていくデータのこと。

1-2. 特性

時系列データには「昨日の売り上げと今日の売り上げが似ている」といった関係性を
持つことがよくある。
→そのため、時系列データをうまく使えば、昨日の売り上げデータから、未来の売り上げデータを
予測することができる可能性あり。

1-3. 出来ること

→時系列解析を学ぶことで、過去から未来を予測するモデルを作成することができます。

2. 時系列データの読み込み

例：月ごとの飛行機の乗客数データを対象とします。

AirPassengers
https://www.analyticsvidhya.com/wp-content/uploads/2016/02/AirPassengers.csv

2-1. データを読み込み

まずは普通の読み込み方。

```
# url = "https://www.analyticsvidhya.com/wp-content/uploads/2016/02/AirPassengers.csv"
# dataNormal = pd.read_csv(url)
# dataNormal.head()
```

このやり方だと1列目も「データ」として扱われてしまう。
1列目は単なる日付なので「1列目は日付のインデックス」と指定をして読み込む必要がある。

```
# dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m')
# data = pd.read_csv(url, index_col='Month', date_parser=dateparse, dtype='float')
# data.head()
```

乗客数のSeriesだけを取り出して「ts」という名前の変数に格納します。

```
# ts = data['#Passengers']
# ts.head()
```

```
Month
1949-01-01    112.0
1949-02-01    118.0
1949-03-01    132.0
1949-04-01    129.0
1949-05-01    121.0
```

3. 定常性とその判定

3-1. 定常性

tsは、平均、分散などの統計的性質が時間の経過と共に一定である場合、
定常状態(stationary)であると言う。

時系列モデルの大部分は、定常状態を仮定して機能します。
→直感的にはある周期で将来も同じことを続ける可能性が非常に高いということ。

定常性を表す条件

1. 平均値が一定
2. 分散が一定
3. 時間に依存しない自己共分散である

3-2. データのプロット

まずは、データをプロットします。

```
---------------------------------------------------------------------------------------------
plt.plot(ts, label='passengers')
plt.title('passengers time series data')
plt.legend(loc='best')
---------------------------------------------------------------------------------------------
```

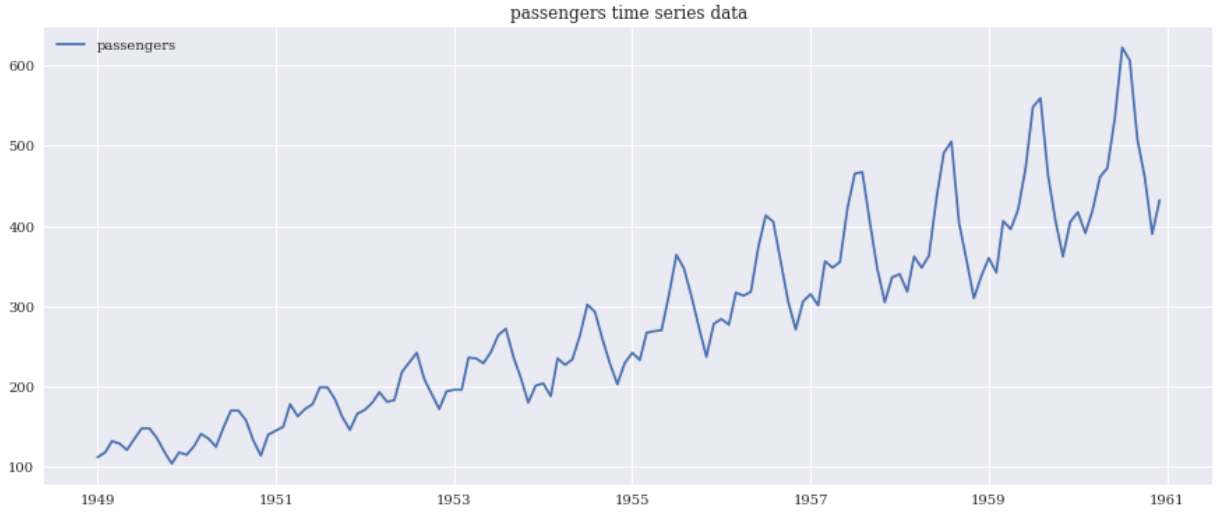


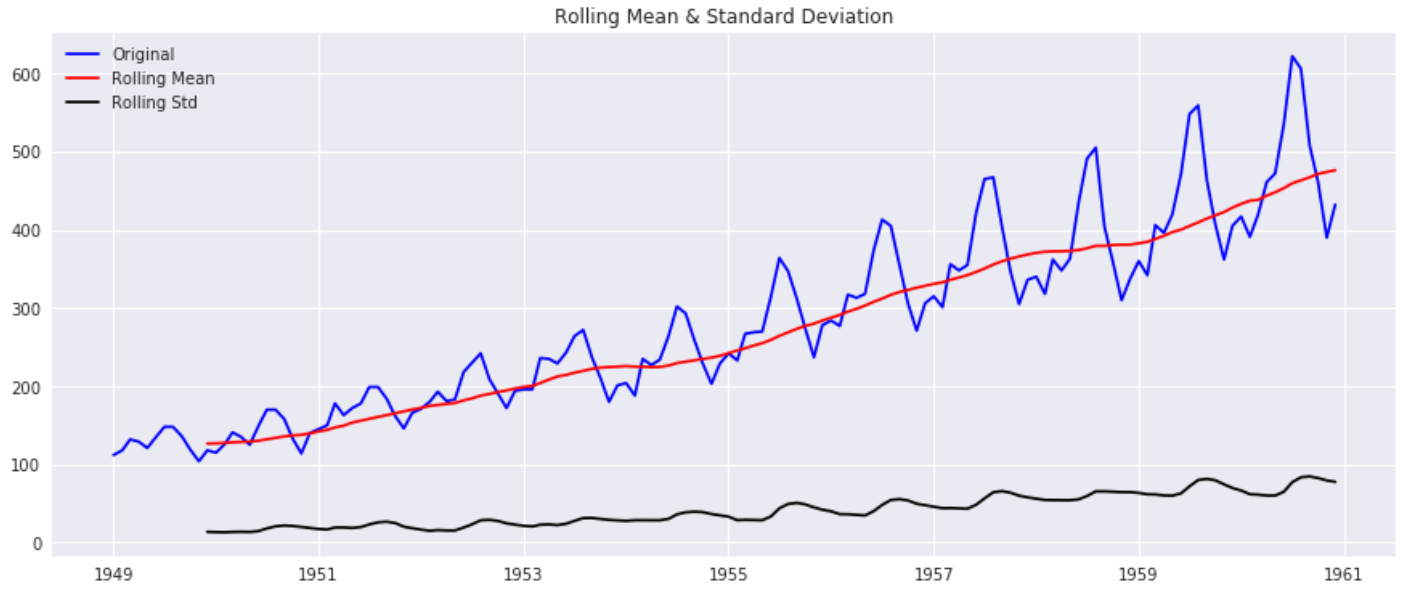
図3-1

季節変動とともにデータに全体的な増加傾向があることは明らか。
統計的に定常性をチェックするには以下を行う。

1. **rolling 統計データをプロットする**
   移動平均または移動分散をプロットして、時間とともに変化するかどうかを確認。

2. **Dickey-Fuller テスト**
   定常性をチェックするための統計的テストの1つ。
   ここで、帰無仮説は、TSが非定常であるということ。
   テスト結果は、「検定統計量(Test Statistic)」と信頼水準の「臨界値(Critical Value)」から構成される。
   「検定統計量」が「臨界値」よりも小さい場合は、帰無仮説を棄却して系列が定常状態と判定できる。

Dickey-Fullerのテスト結果と共にローリング統計をプロットした結果。



Results of Dickey-Fuller Test:
Test Statistic 0.815369
p-value 0.991880
#Lags Used 13.000000
Number of Observations Used 130.000000
Critical Value (1%) -3.481682
Critical Value (5%) -2.884042
Critical Value (10%) -2.578770

・標準偏差のばらつきは小さいが、平均値は明らかに時間と共に増加しており、定常的ではない。
・検定統計量(0.815369) ははるかに臨界値(-3.481682, -2.884042, -2.578770) 以上。
<- 絶対値ではなく、符号付きの値を比較することに注意。

```
from statsmodels.tsa.stattools import adfuller

# Dickey-Fuller test 結果と標準偏差、平均のプロット
def test_stationarity(timeseries):
    # Determing rolling statistics
    rolmean = timeseries.rolling(window=12,center=False).mean()
    rolstd = timeseries.rolling(window=12,center=False).std()

    # Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    # Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value',
                                             '#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

---

4. 定常化の流れ

この系列データを定常性に向かわせる手法について。

定常性仮定は多くの時系列モデルで採用されているが、実用的な時系列のほとんどは静止していない。
→そこで統計家は、系列を静止させる方法を考え出した。
　実際には、シリーズを完全に静止させることはほとんど不可能ですが、可能な限り近づける。

4-1. 定常的でない二つの主な理由

1. **Trend**
時間とともに変化する平均。このケースでは、平均乗客数が時間とともに増加している。

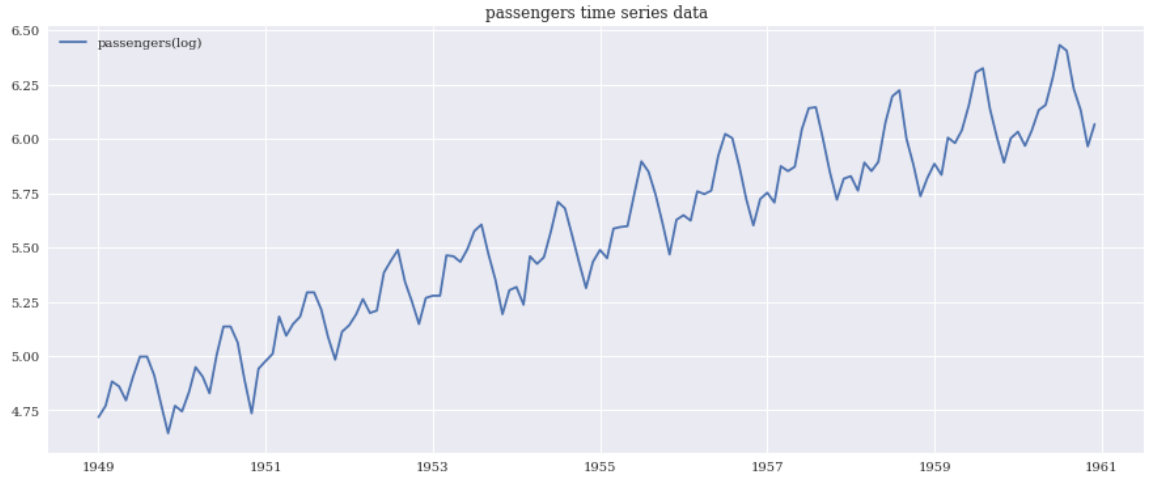
2. **Seasonality**
特定の時間枠での変動。増額や祝祭のために、特定の月に車を購入する傾向があるかも。

定常化の第一歩は、「トレンドと季節性をモデル化または推定」し、それらをシリーズから
除去して固定系列を得ること。
次に、「トレンドと季節性制約を適用」して予測値を元のスケールに変換する。

4-2. トレンドを除去する方法

変換する。このケースでは、明確な傾向が分かる。
したがって、小さい値よりも高い値にペナルティを課す変換を適用。
→対数、平方根、立方根などの処理が可能。ここでは簡単のため対数を採用。

```
ts_log = np.log(ts)
plt.plot(ts_log, label='passengers(log)')
plt.title('passengers time series data')
plt.legend(loc='best')
```



このトレンドを推定またはモデル化し、それをシリーズから除去してみる。

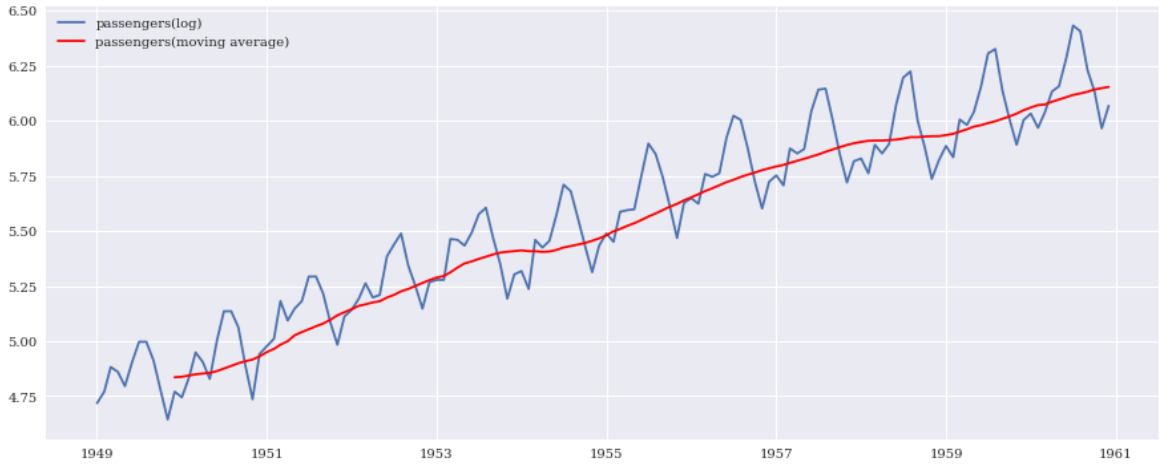
1. Aggregation
　月/週平均などの平均期間で集約する
2. Smoothing
　平滑化。移動平均をとる。
3. Polynomial Fitting
　回帰モデルを適用

ここでは平滑化を適用する。平滑化とは、過去のデータから移動平均を参照する。

5. 移動平均

時系列の直近のデータから特定個数のデータの平均値を取ります。
ここでは、過去1年間、すなわち過去12値の平均値を取る。

```
moving_avg = ts_log.rolling(window=12,center=False).mean()
plt.plot(ts_log, label='passengers(log)')
plt.plot(moving_avg, label='passengers(moving average)', color='red')
plt.legend(loc='best')
```



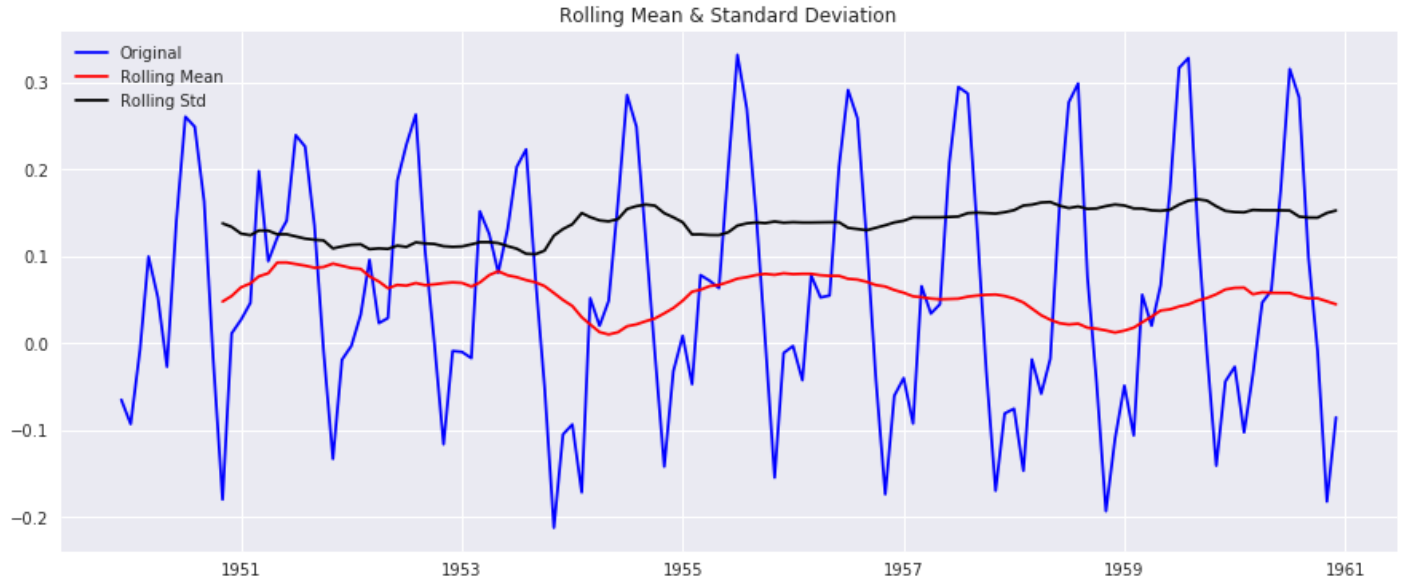
赤い線は移動平均を示します。
この移動平均を元の系列データから差し引いてみます。
注意：直近12個の値の平均値を取るので、最初の11個の値に対して移動平均が
　　定義されていないことに注意。

```
ts_log_moving_avg_diff = ts_log - moving_avg
ts_log_moving_avg_diff.head(12)
```

```
Month
1949-01-01       NaN
1949-02-01       NaN
1949-03-01       NaN
1949-04-01       NaN
1949-05-01       NaN
1949-06-01       NaN
1949-07-01       NaN
1949-08-01       NaN
1949-09-01       NaN
1949-10-01       NaN
1949-11-01       NaN
1949-12-01   -0.065494
Name: #Passengers, dtype: float64
```

最初の11がNanであることに注目。
これらのNaN値を落として、プロットをチェックして定常性をテストする。

```
ts_log_moving_avg_diff.dropna(inplace=True)
test_stationarity(ts_log_moving_avg_diff)
```



Results of Dickey-Fuller Test:
Test Statistic -3.162908
p-value 0.022235
#Lags Used 13.000000
Number of Observations Used 119.000000
Critical Value (1%) -3.486535
Critical Value (5%) -2.886151
Critical Value (10%) -2.579896
.

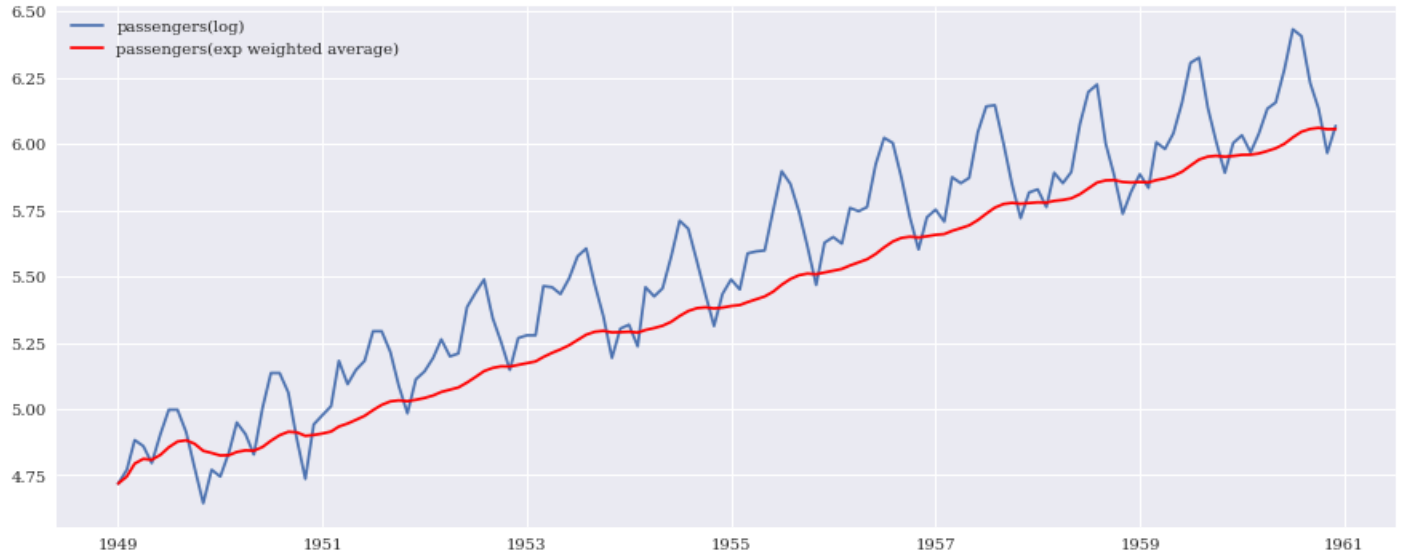
先ほどよりはるかに定常化している系列データに見えます。
移動平均はわずかに変化するようだが、特定の傾向はない。
検定統計量は5%の臨界値よりも小さいので、**95%の信頼度で定常化**された系列だと言える。

6. 加重移動平均

移動平均アプローチの欠点
期間を厳密に定義しなければならないこと。
この例では年平均を取ることができる。株価予測のような複雑な状況では、期間の特定は困難。

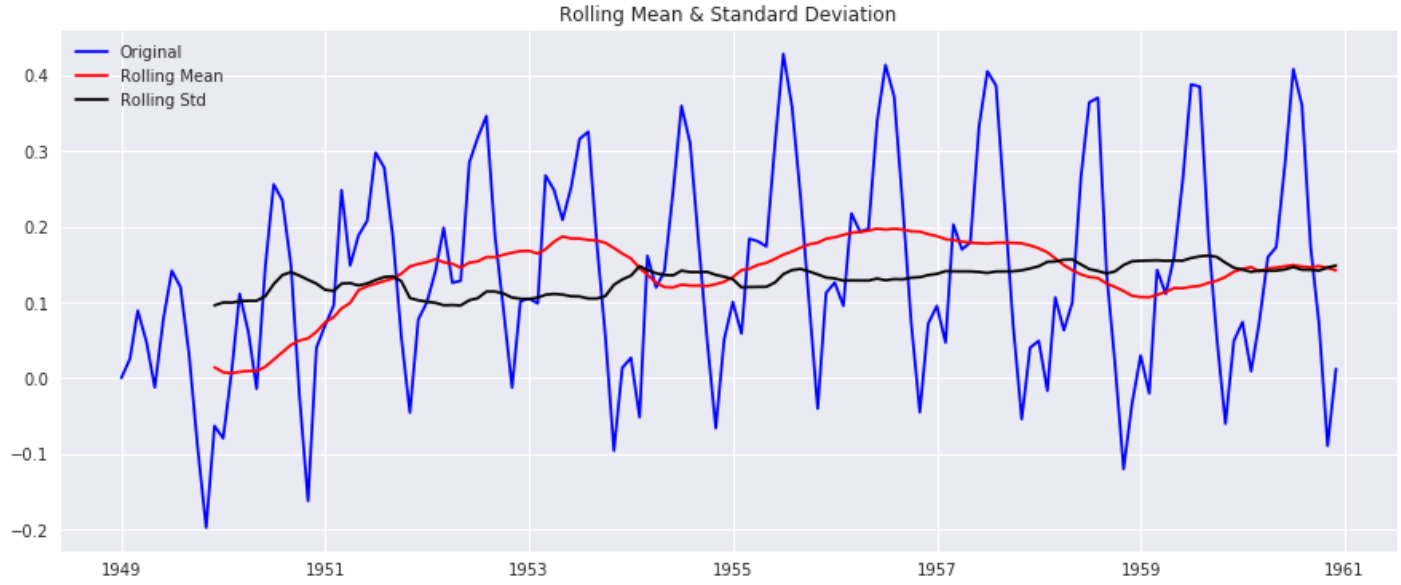
→より最近の値に高い重みが与えられている場合、「加重移動平均」を取る。
　期間中に同じ重みではなく、より最近の値に大きな重みを割り当てる。
例：指数関数的に加重された移動平均。
　　重みは減衰係数を伴うすべての前の値に割り当てられます。

```
expwighted_avg = ts_log.ewm(halflife=12,min_periods=0,adjust=True,ignore_na=False).mean()
plt.plot(ts_log, label='passengers(log)')
plt.plot(expwighted_avg, label='passengers(exp weighted average)', color='red')
plt.legend(loc='best')
```



この加重移動平均を元の系列データから差し引いてみます。

---
ts_log_ewma_diff = ts_log - expwighted_avg
test_stationarity(ts_log_ewma_diff)



Results of Dickey-Fuller Test:
Test Statistic -3.601262
p-value 0.005737
#Lags Used 13.000000
Number of Observations Used 130.000000
Critical Value (1%) -3.481682
Critical Value (5%) -2.884042
Critical Value (10%) -2.578770

この系列データは、平均および標準偏差の変動がさらに小さい。
また、検定統計量は1%の臨界値よりも小さく、移動平均よりも定常性に優れる。

この場合、開始からのすべての値に重みが与えられるので、欠損値は存在しない。

7. トレンドと季節変動の除去

これまでに説明したシンプルなトレンド削減手法(移動平均、加重移動平均を引く手法)では、
すべてのケース、特に季節変動の高い状況では機能しない。

差分	特定のタイムラグで差分をとる。
分解	トレンドと季節性の両方をモデル化し、それらをモデルから削除する。

7-1. 差分

トレンドと季節性の両方を処理する最も一般的な方法の1つは差分。
この手法では、ある瞬間の観測と前の瞬間の観測との違いを取く。
→主に定常性を改善する上で効果的。

シフトして頭だけ取り出し確認

```
# ts.shift().head()
Month
1949-01-01      NaN
1949-02-01    112.0
1949-03-01    118.0
1949-04-01    132.0
1949-05-01    129.0
```

差分は、シフトする前から、シフトした後を引けばいいです。

```
# diff = ts - ts.shift()
```

頭だけ取り出し確認

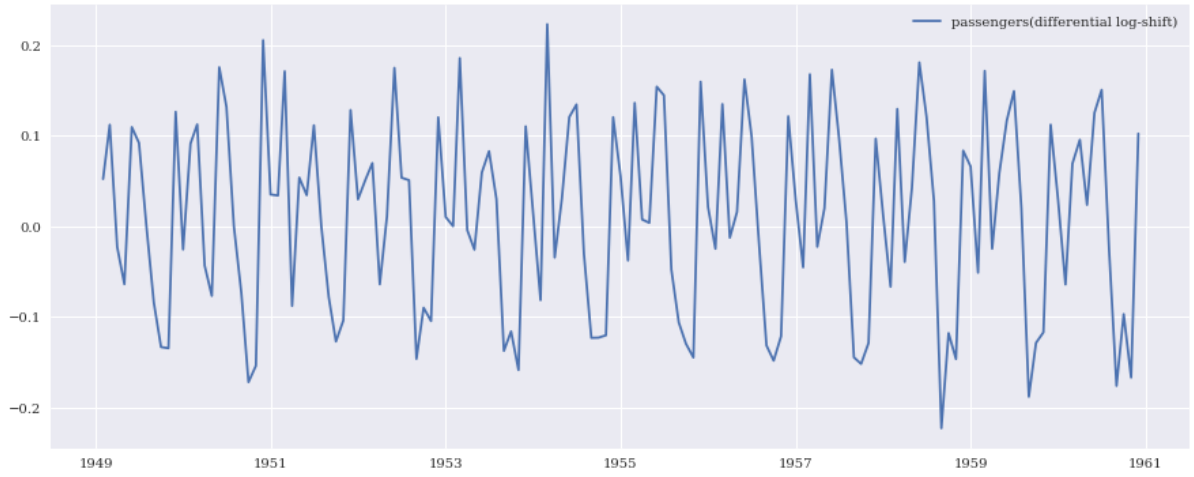
```
# diff.head()
Month
1949-01-01     NaN
1949-02-01     6.0
1949-03-01    14.0
1949-04-01    -3.0
1949-05-01    -8.0
```

対数差分は、単に対数をとってから差分するだけ。

```
# logDiff = np.log(ts) - np.log(ts.shift())
# logDiff.head()
Month
1949-01-01         NaN
1949-02-01    0.052186
1949-03-01    0.112117
1949-04-01   -0.022990
1949-05-01   -0.064022
```

一次差分は、次のようにPandasで実行できます。

```
---------------------------------------------------------------------
ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff, label='passengers(differential log-shift)')
plt.legend(loc='best')
---------------------------------------------------------------------
```

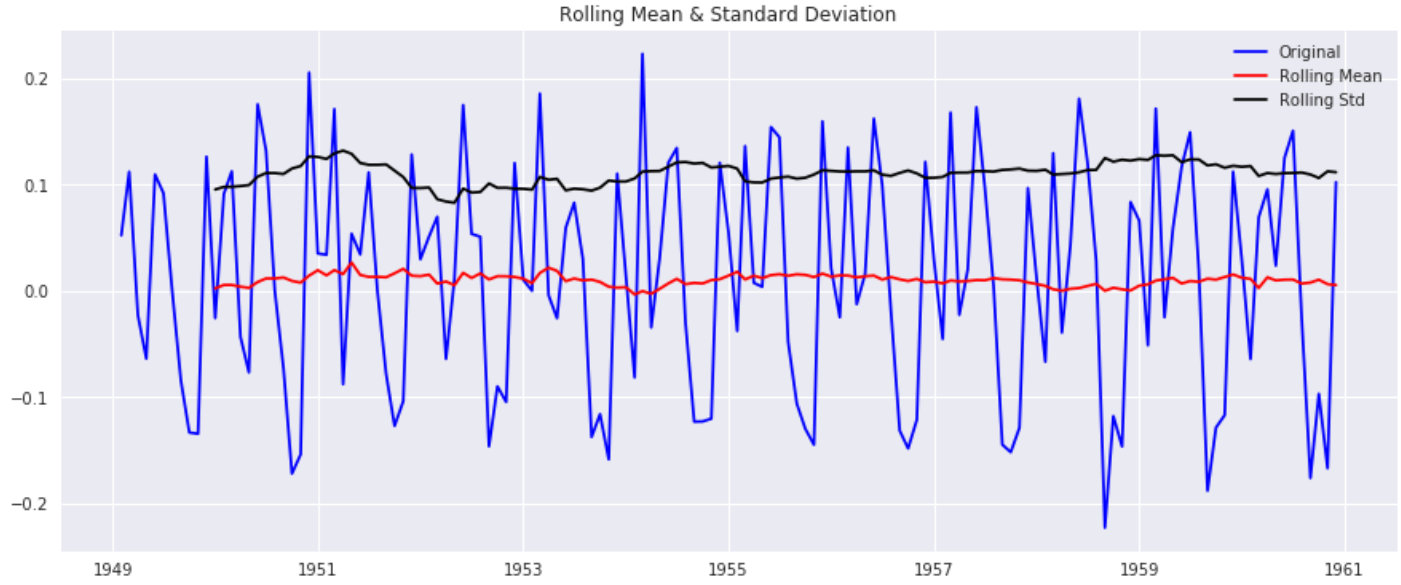


-

傾向がかなり減少したように見えるので、Dickey-Fuller検定で定常性検定をしてみる。

---

ts_log_diff.dropna(inplace=True)
test_stationarity(ts_log_diff)



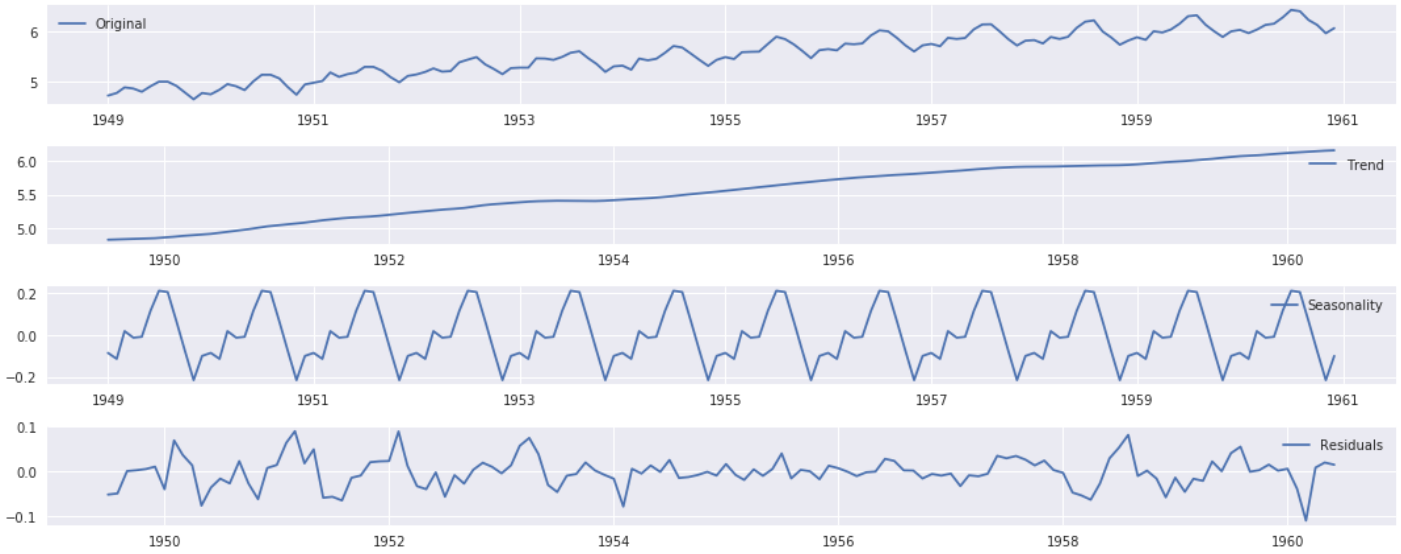
Results of Dickey-Fuller Test:
Test Statistic -2.717131
p-value 0.071121
#Lags Used 14.000000
Number of Observations Used 128.000000
Critical Value (1%) -3.482501
Critical Value (5%) -2.884398
Critical Value (10%) -2.578960

平均と標準偏差にともに小さな変動であることが確認できる。
検定統計量は10%臨界値よりも小さいため、90%信頼できる状態で定常状態。

7-2. 分解

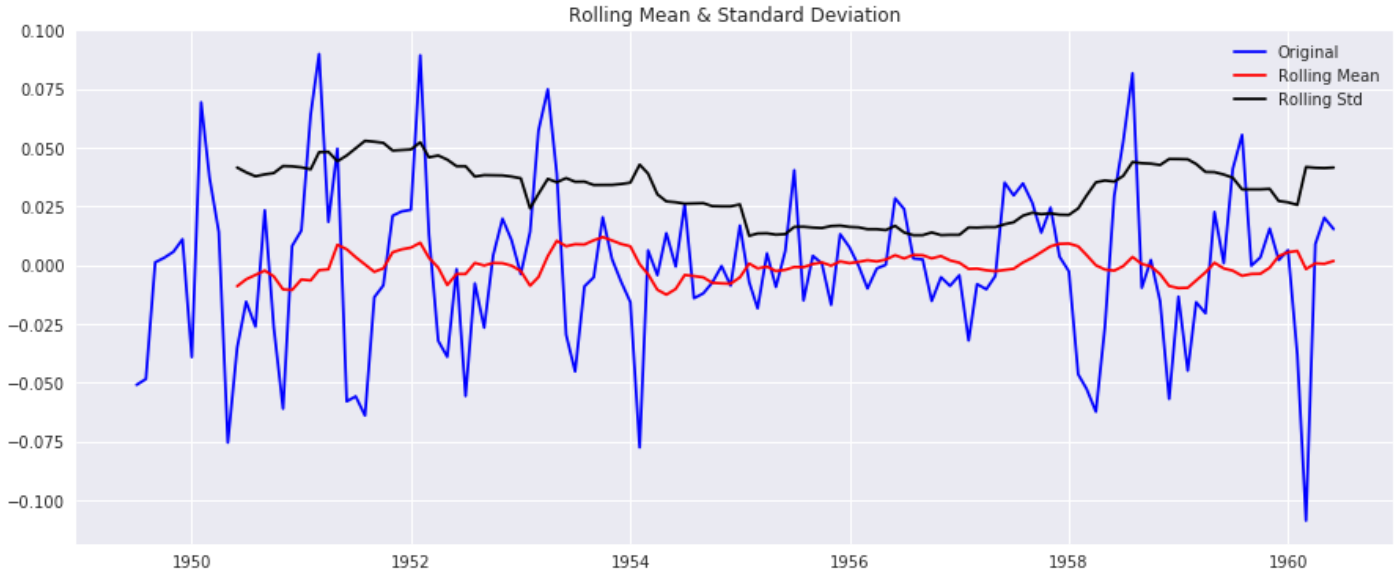
このアプローチでは、傾向(trend)、季節性(seasonal)、残差(residual)に分解してモデル化する。

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts_log)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# オリジナルの時系列データプロット
plt.subplot(411)
plt.plot(ts_log, label='Original')
plt.legend(loc='best')

# trend のプロット
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')

# seasonal のプロット
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')

# residual のプロット
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```



ここで、傾向、季節性がデータから分離され、残差をモデル化できることがわかる。
そこで、残差の定常性をチェックできる。

```
ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)
```



Results of Dickey-Fuller Test:
Test Statistic -6.332387e+00
p-value 2.885059e-08
#Lags Used 9.000000e+00
Number of Observations Used 1.220000e+02
Critical Value (1%) -3.485122e+00
Critical Value (5%) -2.885538e+00
Critical Value (10%) -2.579569e+00

Dickey-Fuller検定統計量は、**1%臨界値よりも有意に低い**。
→この系列の定常性は非常に高い。

8. 時系列の予測

トレンドと季節性の推定手法を実行した結果、次の2つの状況が発生する可能性がある。

1. **厳密に定常状態の系列データの間に依存関係がない**
   残差を白色ノイズとしてモデル化できる簡単なケース。これは非常にまれなケース。

2. **系列データの間に重大な依存関係を持つ**
   ARIMAのような統計モデルを使用してデータを予測する必要がある。

8-1. ARIMA

ARIMA(Auto-Regressive Integrated Moving Averages)は、**自動回帰積分移動平均を**表します。
静止時系列のARIMA予測は線形回帰式のような線形式に過ぎません。
予測子は、ARIMAモデルのパラメータ（p、d、q）に依存します。

1. **AR(自己回帰) 項の数 (p)**
   ARのラグ。pが5の場合、x(t)の予測子はx(t-1)... x(t-5) 。

2. **MA(移動平均) 項数 (q)**
   qが5の場合、x(t) の予測子はe(t-1)... e(t-5) 。
   ここで e(i) は i 番目の移動平均と実際の値の差である。

3. **差分の数 (d)**
   非季節性の違いの数。この例では、一次差をとっています。
   その変数を渡してd = 0とするか、元の変数を渡してd = 1とするかのどちらか。
   両方とも同じ結果を生成します。

ここで重要なのは、 'p'と 'q'の値を決定する方法。

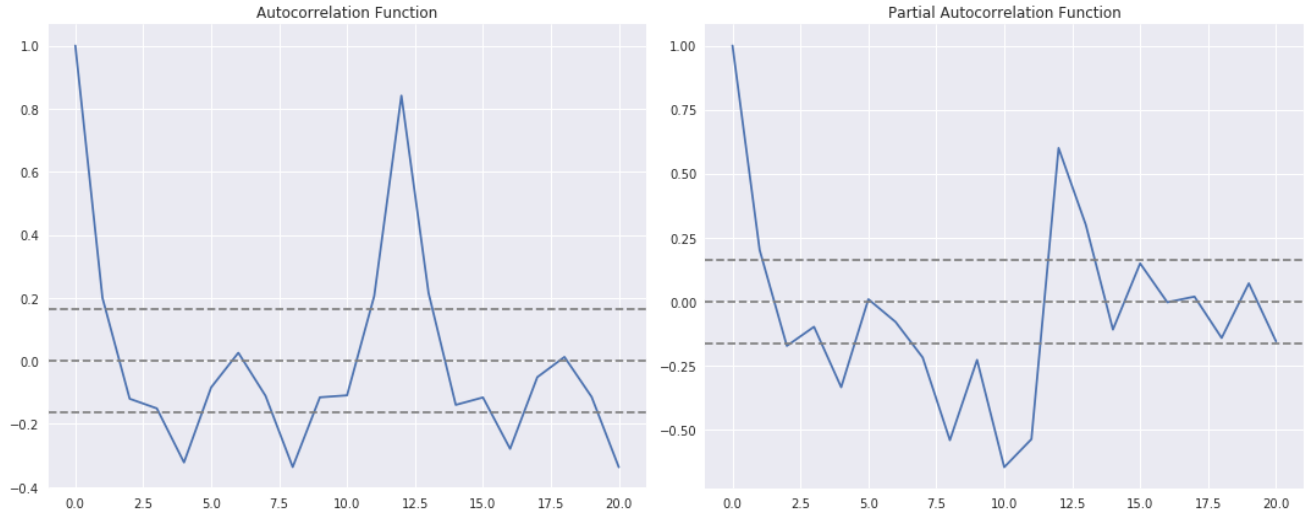
8-2. 自己相関係数

「過去の値とどの程度の相関があるか」を定量的に評価するための手法には
・自己相関（Autocorrelation）
・偏自己相関（Partial Autocorrelation）
がある。

自己相関	当日以前のデータの影響をどの程度受けているか（相関性があるか）を表す。
偏自己相関	当日と該当日の2点の間の相関性を表す。

```
#ACF and PACF plots:
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(ts_log_diff, nlags=20)
lag_pacf = pacf(ts_log_diff, nlags=20, method='ols')

#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')
```

```
#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```



0の両側にある2つの点線は信頼区間。これらを使用して 'p'と 'q'の値を次のように決定できる。

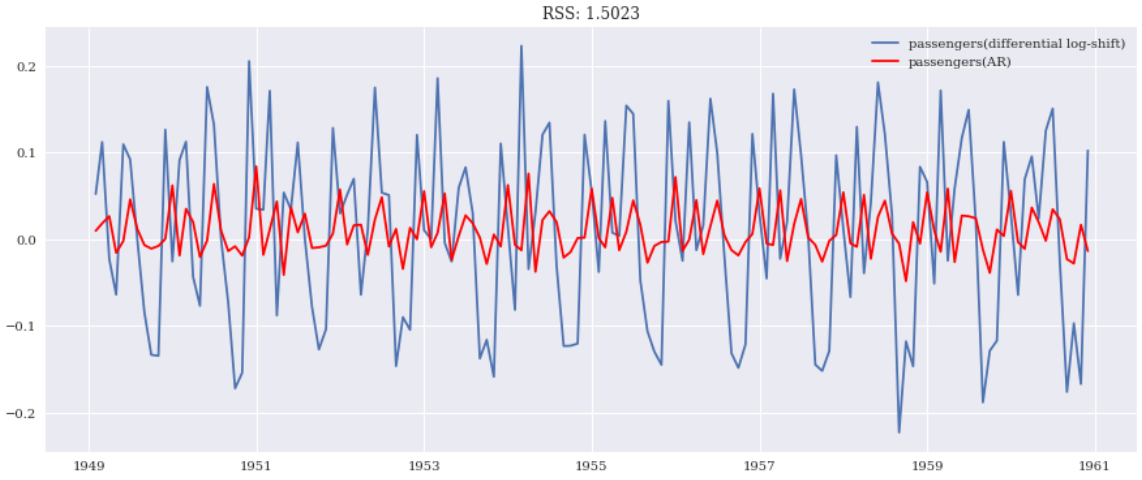
1. **p** - **PACF**チャートが最初に上の信頼区間を横切るラグ値。
   →この場合は p = 2
2. **q** - **ACF**チャートが最初に上の信頼区間を横切るラグ値。
   →この場合 q = 2

個々の効果と組み合わせた効果を考慮して3種類のARIMAモデル(ARモデル,MAモデル,結合モデル)を作成します。
ここでRSSは残差の値であり、実際の系列ではないことに注意。
ARIMAモデルは order引数としてタプル（p、d、q）の値を指定して利用できる。

8-3. ARモデルのプロット

ARモデル　p=2, d=1, q=0を指定。

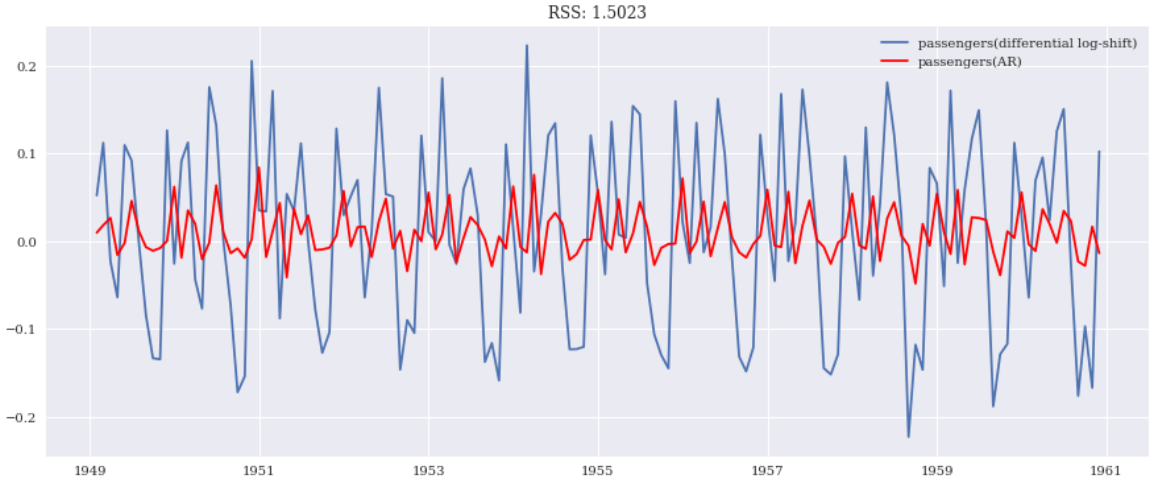
```
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(ts_log, order=(2, 1, 0))
results_AR = model.fit(disp=-1)

plt.plot(ts_log_diff, label='passengers(differential log-shift)')
plt.plot(results_AR.fittedvalues, label='passengers(AR)', color='red')
plt.title('RSS: %.4f'% sum((results_AR.fittedvalues-ts_log_diff)**2))
plt.legend(loc='best')
```



8-4. MAモデルのプロット

MA モデル　p=0, d=1, q=2を指定。

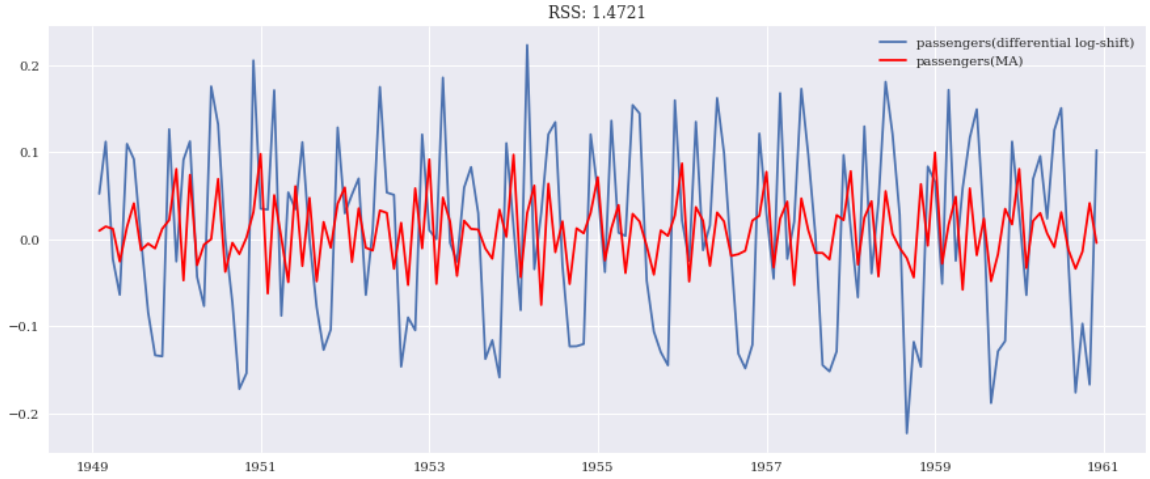
```
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(ts_log, order=(0, 1, 2))
results_AR = model.fit(disp=-1)

plt.plot(ts_log_diff, label='passengers(differential log-shift)')
plt.plot(results_AR.fittedvalues, label='passengers(AR)', color='red')
plt.title('RSS: %.4f'% sum((results_AR.fittedvalues-ts_log_diff)**2))
plt.legend(loc='best')
```



8-5. 結合モデルのプロット

結合 モデル p=2, d=1, q=2を指定。

```
model = ARIMA(ts_log, order=(2, 1, 2))
results_ARIMA = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_ARIMA.fittedvalues-ts_log_diff)**2))
```



ARモデルとMAモデルはほぼ同じRSSを持っていますが、結合モデルは
かなり優れている。
ここでは、最後のステップとして、これらの値を元のスケールに戻す。

8-6. オリジナルスケールに戻す

結合されたモデルを元の値に戻し、予測がどの程度出来るのかを確認できる。
まずは予測結果を別の連続データとして観測する。

```
predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
print(predictions_ARIMA_diff.head())
```

```
Month
1949-02-01    0.009580
1949-03-01    0.017491
1949-04-01    0.027670
1949-05-01   -0.004521
1949-06-01   -0.023890
```

最初の月は '1949-02-01'から始まるのは、1ラグを取り、最初の要素にはその前のデータがないから。
差分を対数スケールに変換するには、これらの差を連続して基数に加算する。
そのために、最初にインデックスの累積合計を決定し、それを基本数に加算する。

```
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
print(predictions_ARIMA_diff_cumsum.head())
```

```
Month
1949-02-01    0.009580
1949-03-01    0.027071
1949-04-01    0.054742
1949-05-01    0.050221
1949-06-01    0.026331
```

次に、それらを基本番号に追加する。このため、すべての値を基数として持つシリーズを作成し、それにその差異を追加する。

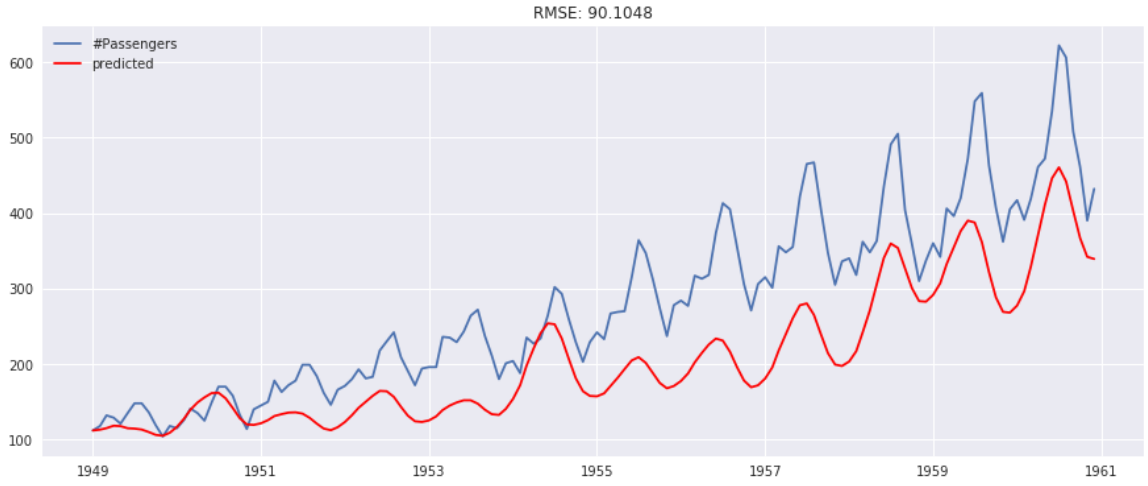
```
predictions_ARIMA_log = pd.Series(ts_log.iloc[0], index=ts_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill_value=0)
predictions_ARIMA_log.head()
```

```
Month
1949-01-01    4.718499
1949-02-01    4.728079
1949-03-01    4.745570
1949-04-01    4.773241
1949-05-01    4.768720
```

最初の要素は基本数そのもの。後はそこから累積的に加算された値。

最後のステップとして指数をとり、元のシリーズと比較する。

```
predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(ts)
plt.plot(predictions_ARIMA, label='predicted', color='red')
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-ts)**2)/len(ts)))
plt.legend(loc='best')
```



予測があまりうまくいっていないように見える。
これはARIMAモデルでは季節変動がうまく表現できていないため。
季節変動がある場合はSARIMAモデルを利用する。

補足：RMSE (Root Mean Squared Error)
数値予測問題における精度評価指標の1つ。
予測値が正解からどの程度乖離しているかを示す。
モデルの予測精度の”悪さ”を表すため0 に近い値であるほど優れている。

下記計算を行いスコアを算出する

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2}$$

N ：全予測対象数
yi : 実績値
y^i: 予測値

8-7. SARIMAモデル

SARIMAモデル（季節自己回帰和分移動平均モデル）
ARIMAモデルに、さらに周期的な変動(季節変動等)を取り入れさせたモデル。

SARIMAモデルは「statsmodels」というライブラリを使って計算するのですが、
このバージョン0.8.0以上が必要。
conda list|grep statsmodels
statsmodels 0.8.0 np112py36_0

```
import statsmodels.api as sm

SARIMA_3_1_2_111 = sm.tsa.SARIMAX(ts, order=(3,1,2), seasonal_order=(1,1,1,12)).fit()
print(SARIMA_3_1_2_111.summary())
```
---

order=(3,1,2)でARIMAモデルの次数
seasonal_order=(1,1,1,12)で季節変動の次数を設定
最後の「12」は「12か月周期」であることを意味しています。

＜サマリ表示結果＞

```
                              Statespace Model Results
==========================================================================================
Dep. Variable:                      #Passengers   No. Observations:                  144
Model:             SARIMAX(3, 1, 2)x(1, 1, 1, 12)   Log Likelihood                -502.993
Date:                            Fri, 18 Aug 2017   AIC                           1021.986
Time:                                    17:52:53   BIC                           1045.744
Sample:                                  01-01-1949   HQIC                        1031.640
                                       - 12-01-1960
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.5290    2.3e+08    2.3e-09      1.000    -4.5e+08     4.5e+08
ar.L2          0.2876   1.63e+08   1.77e-09      1.000   -3.19e+08    3.19e+08
ar.L3         -0.0336   7.21e+07  -4.67e-10      1.000   -1.41e+08    1.41e+08
ma.L1         -0.9174   3.06e+08     -3e-09      1.000      -6e+08       6e+08
ma.L2         -0.0566   2.58e+08  -2.19e-10      1.000   -5.06e+08    5.06e+08
ar.S.L12      -0.8851   2.45e+07  -3.61e-08      1.000    -4.8e+07     4.8e+07
ma.S.L12       0.7926    3.3e+07    2.4e-08      1.000   -6.47e+07    6.47e+07
sigma2       125.1780   2.89e+10   4.34e-09      1.000   -5.66e+10    5.66e+10
==============================================================================
Ljung-Box (Q):                       51.07   Jarque-Bera (JB):                 13.32
Prob(Q):                              0.11   Prob(JB):                         0.00
Heteroskedasticity (H):               2.61   Skew:                             0.14
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.54
==========================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 5.74e+14. Standard errors may be unstable.

/root/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

→ ConvergenceWarning　は最適化の収束しないということなので、うまくモデルで表現できていない。

総当たりで、AICが最小となるSARIMAの次数を探してみる。

```
max_p = 3
max_q = 3
max_d = 1
max_sp = 1
max_sq = 1
max_sd = 1

pattern = max_p*(max_q + 1)*(max_d + 1)*(max_sp + 1)*(max_sq + 1)*(max_sd + 1)
modelSelection = pd.DataFrame(index=range(pattern), columns=["model", "aic"])
pattern
```

192パターンあることがわかる。

```
# 自動SARIMA選択
num = 0

for p in range(1, max_p + 1):
    for d in range(0, max_d + 1):
        for q in range(0, max_q + 1):
            for sp in range(0, max_sp + 1):
                for sd in range(0, max_sd + 1):
                    for sq in range(0, max_sq + 1):
                        sarima = sm.tsa.SARIMAX(
                            ts, order=(p,d,q),
                            seasonal_order=(sp,sd,sq,12),
                            enforce_stationarity = False,
                            enforce_invertibility = False
                        ).fit()
                        modelSelection.iloc[num]["model"] = "order=(" + str(p) + ","+ str(d) + ","+ str(q) + "), season=("+ str(sp) + ","+ str(sd) + "," + str(sq) + ")"
                        modelSelection.iloc[num]["aic"] = sarima.aic
                        num = num + 1
```

AICが最小となるモデルを確認する。

```
# モデルごとの結果確認
print(modelSelection)
# AIC最小モデル
print(modelSelection[modelSelection.aic == min(modelSelection.aic)])
```

```
                        model      aic
0    order=(1,0,0), season=(0,0,0)  1415.91
1    order=(1,0,0), season=(0,0,1)  1205.39
2    order=(1,0,0), season=(0,1,0)  1029.98
3    order=(1,0,0), season=(0,1,1)  944.385
4    order=(1,0,0), season=(1,0,0)  1017.32
...
190  order=(3,1,3), season=(1,1,0)  910.008
191  order=(3,1,3), season=(1,1,1)  903.239

[192 rows x 2 columns]
                        model      aic
187  order=(3,1,3), season=(0,1,1)  898.105
```

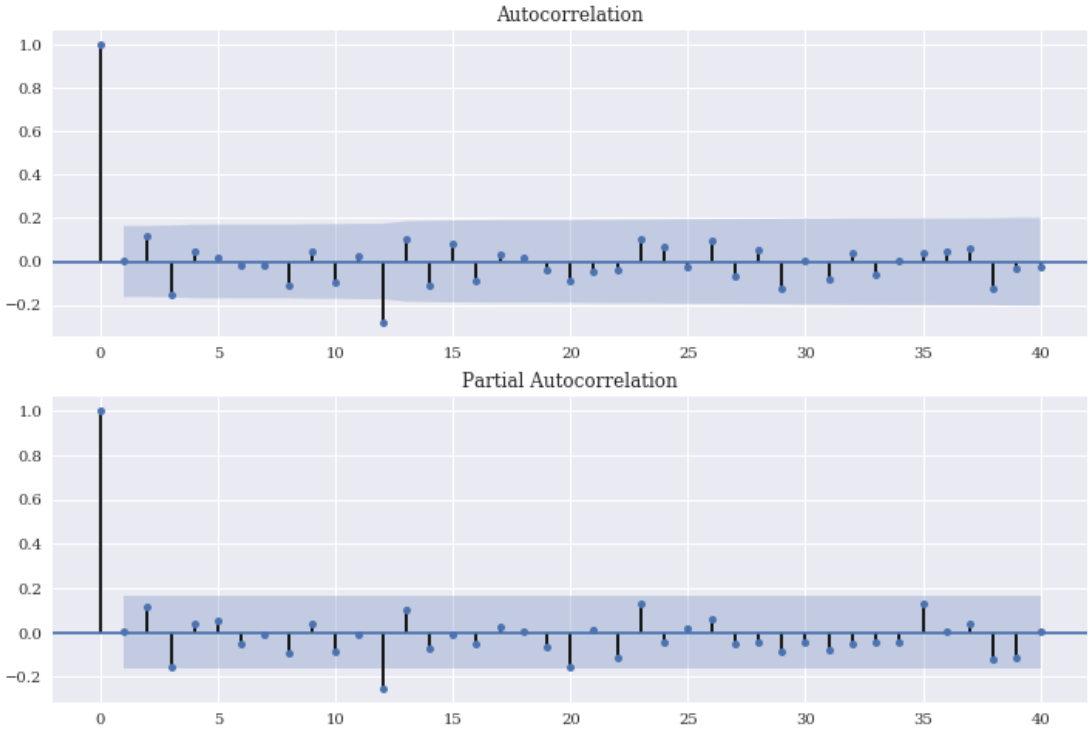
order=(3,1,3), season=(0,1,1)　の組み合わせが最適なモデルだと判明したので
改めてモデル生成して結果を確認する。

```
p=3
d=1
q=3
sp=0
sd=1
sq=1

sarima = sm.tsa.SARIMAX(
   ts, order=(p,d,q),
   seasonal_order=(sp,sd,sq,12),
   enforce_stationarity = False,
   enforce_invertibility = False
).fit()
# 結果確認
print(sarima.summary())
```

```
 Statespace Model Results
==========================================================================================
Dep. Variable:                  #Passengers   No. Observations:                  144
Model:             SARIMAX(3, 1, 3)x(0, 1, 1, 12)   Log Likelihood             -441.052
Date:                            Sat, 19 Aug 2017   AIC                        898.105
Time:                                18:43:32   BIC                        921.863
Sample:                              01-01-1949   HQIC                       907.759
                                   - 12-01-1960
Covariance Type:                        opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.2231   3336.897  -6.69e-05      1.000   -6540.422    6539.975
ar.L2         -0.1642   4448.807  -3.69e-05      1.000   -8719.666    8719.338
ar.L3          0.7244   3656.684      0.000      1.000   -7166.244    7167.693
ma.L1         -0.0837   1.33e+10  -6.29e-12      1.000   -2.61e+10    2.61e+10
ma.L2          0.1221   1.58e+10   7.74e-12      1.000   -3.09e+10    3.09e+10
ma.L3         -0.9797   1.42e+10   -6.9e-11      1.000   -2.78e+10    2.78e+10
ma.S.L12      -0.1583   6.18e+09  -2.56e-11      1.000   -1.21e+10    1.21e+10
sigma2       119.6719   7.63e+08   1.57e-07      1.000   -1.49e+09    1.49e+09
==========================================================================================
Ljung-Box (Q):                       36.68   Jarque-Bera (JB):                 4.39
Prob(Q):                              0.62   Prob(JB):                         0.11
Heteroskedasticity (H):               1.87   Skew:                             0.16
Prob(H) (two-sided):                  0.06   Kurtosis:                         3.90
==========================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 5.18e+18. Standard errors may be unstable.
```

残差の自己相関、偏自己相関について確認する。

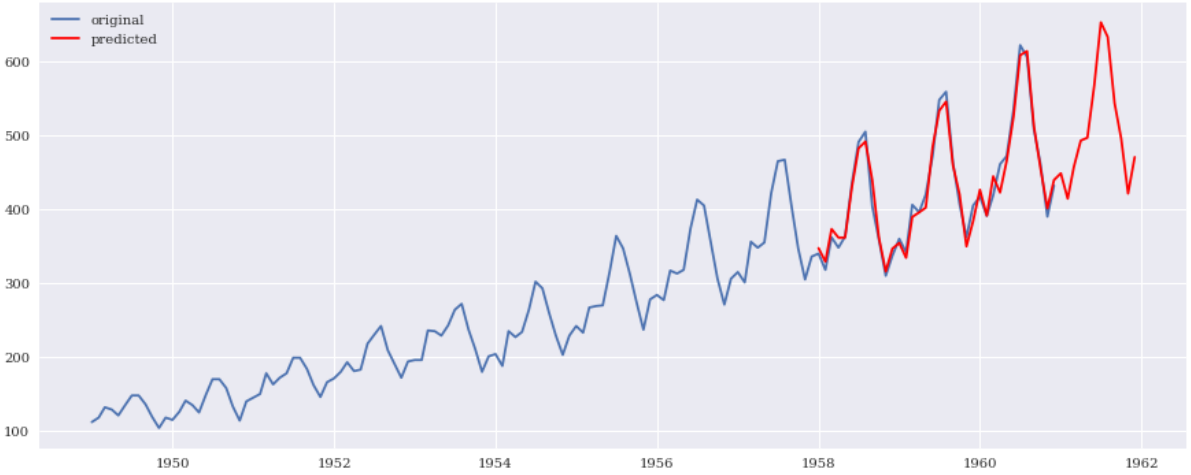
```
# 残差のチェック
residSARIMA = sarima.resid
fig = plt.figure(figsize=(12,8))

# 自己相関
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(residSARIMA, lags=40, ax=ax1)

# 偏自己相関
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(residSARIMA, lags=40, ax=ax2)
```



自己相関、偏自己相関共に過去のデータに依存していないことがわかる。

最後にこのモデルで1958/01/01-1961/12/01 の期間の予測値を取得し、うまく予測できているのかプロットしてみる。

```
# 予測
ts_pred = sarima.predict('1958-01-01', '1961-12-01')

# 実データと予測結果の図示
plt.plot(ts, label='original')
plt.plot(ts_pred, label='predicted', color='red')
plt.legend(loc='best')
```



オリジナルデータは1961/01 までだが、予測データは1961/12まで。
1958/01 - 1961/01 までの予測データもほぼ予測できており、想定通りの予測データが得られたことがわかる。

9. 補足：ARMAモデルの推定

9-1. AR(p) モデル（自己回帰モデル; Auto-regressive model）

自己回帰とは、その名の通り自分のデータと回帰する手法。
一日前の値を横軸に、一日後のデータを縦軸にしてプロットし、線を引っ張ると言うイメージ。
一番簡単な時系列モデルです。

例えば、株価など時系列（日次）で変化する事象を表すデータで、
今日の株価は1日前の株価の値と相関関係があり、式1のように表すことができるモデルを**AR(1)過程**
と呼びます。
1日前と2日前の2世代前までのデータと相関性がある場合には**AR(2)過程**
ということができ、p日前までのデータと相関性がある場合には**AR(p)過程**と呼びます。

AR(*p*) という表記は次数 *p* の自己回帰モデルを表す。AR(*p*)モデルは次の式で表される。
時系列データ X_t について、ARMAモデルはその将来の値を予測するためのツールとして機能する。
モデルは自己回帰(AR)部分と移動平均(MA)部分からなる。一般に ARMA(p,q)モデルと表記され、
p は自己回帰部分の次数、q は移動平均部分の次数を表す。

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t.$$

ここで $\varphi_1, \cdots, \varphi_p$ はモデルの**パラメータ**、c は定数項、ε_t は誤差項。
定数項は単純化するために省かれることが多い。

9-2. MAモデル（移動平均モデル; Moving Average model）

MA(*q*)という表記は、次数 *q* の移動平均モデルを表す。以下の数式で表される。

$$X_t = \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

ここで $\theta_1, \ldots, \theta_q$ はモデルのパラメータ、$\varepsilon_t, \varepsilon_{t-1}, \ldots$ は誤差項。

データ例1
4　2　6　10　14　18

これを3区間移動平均をしてみると、
（4＋2＋6）/3、（2＋6＋10）/3、（6＋10＋14）/3、（10＋14＋18）/3
＝ 4　6　10 14

→これが移動平均。
　3地点の平均をずらしながらやっていったら3点移動平均になる。

ここで、緑の部分に注目。移動平均する際のたんなる計算過程なのですが・・・・・

4番目の移動平均結果	14	＝	10＋14＋18	÷3
3番目の移動平均結果	10	＝	6＋10＋14	÷3
2番目の移動平均結果	6	＝	2＋6＋10	÷3
1番目の移動平均結果	4	＝	4＋2＋6	÷3

青い部分、4番目の結果に使用されているデータが、3番目2番目の移動平均結果にも用いられている。
「移動平均みたいに、一部同じ数字を使って表されたデータは自己相関を持つ」ということになる。

これを利用したのが**移動平均モデル**。MAモデルだと、
・一日前と大きな自己相関がある　とかいう現象や
・周期性
うまくモデリングすることができる。

9-3. ARMAモデル

ARMAモデル（自己回帰移動平均モデル）
= ARモデル、MAモデル を両方突っ込んだモデル

ARMA(p, q)という表記は、p次の自己回帰とq次の移動平均を組合わせたモデルを指す。
以下の数式で表される。

$$X_t = \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}.$$

参考文献

Pythonによる時系列分析の基礎
http://logics-of-blue.com/python-time-series-analysis/

Autoregressive Moving Average (ARMA): Sunspots data
http://www.statsmodels.org/stable/examples/notebooks/generated/tsa_arma_0.html

A comprehensive beginner's guide to create a Time Series Forecast
https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/