时间序列预测全攻略（附带Python代码） (http://www.36dsj.com/archives/44065)

Optimus Prime 2016-03-18 12:18:12 Python (http://www.36dsj.com/archives/category/large-data-tools/python) 评论(8)

介绍

时间序列（简称TS）被认为是分析领域比较少人知道的技能。（我也是几天前才知道它）。但是你一定知道最近的小型编程马拉松就是基于时间序列发展起来的，我参加了这项活动去学习了解决时间序列问题的基本步骤，在这儿我要分享给大家。这绝对能帮助你在编程马拉松中获得一个合适的模型。

文章之前，我极力推荐大家阅读《基于R语言的时间序列建模完整教程》A Complete Tutorial on Time Series Modeling in R，它就像这篇文章的前篇。它关注基本概念和基于R语言，我将重点使用这些概念来解决Python编程里面端到端的问题。R语言存在许多关于时间序列的资源，但是很少关于Python的，所以本文将使用Python。

36大数据专稿，原文作者：AARSHAY JAIN ***本文由36大数据翻译****，任何不标明译者和出处以及本文链接http://www.36dsj.com/archives/43811的均为侵权。*

我们的过程包括下面几步：

1、时间序列有什么特别之处？

2、在Pandas上传和加载时间序列（pandas 是基于 Numpy 构建的含有更高级数据结构和工具的数据分析包，类似于 Numpy 的核心是 ndarray，pandas 也是围绕着 Series 和 DataFrame 两个核心数据结构展开的。）

3、如何检验时间序列的稳定性？

4、如何令时间序列稳定？

5、时间序列预测。

1、时间序列有什么特别之处？

顾名思义，时间序列是时间间隔不变的情况下收集的时间点集合。这些集合被分析用来了解长期发展趋势，为了预测未来或者表现分析的其他形式。但是是什么令时间序列与常见的回归问题的不同？

有两个原因：

1、时间序列是跟时间有关的。所以基于线性回归模型的假设：观察结果是独立的在这种情况下是不成立的。

2、随着上升或者下降的趋势，更多的时间序列出现季节性趋势的形式，如：特定时间框架的具体变化。即：如果你看到羊毛夹克的销售上升，你就一定会在冬季做更多销售。

因为时间序列的固有特性，有各种不同的步骤可以对它进行分析。下文将详细分析。通过在Python上传时间序列对象开始。我们将使用飞机乘客数据集。

请记住本文的目的是希望使你熟悉关于时间序列的不同使用方法。本文的例子只是用来方便解释时间序列对象，我重点关注题目的广泛性，不会做非常精确的预测。

2、在pandas上传和加载时间序列

Pandas有专门处理时间序列对象的库，特别是可以存储时间信息和允许人们执行快速合作的datatime64(ns)类。从激发所需的库开始。

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6
```

现在，我们可以上传数据集和查看一些最初的行以及列的数据类型。

```
data = pd.read_csv('AirPassengers.csv')
print data.head()
print '\n Data Types:'
print data.dtypes
```

```
     Month  #Passengers
0  1949-01          112
1  1949-02          118
2  1949-03          132
3  1949-04          129
4  1949-05          121

 Data Types:
Month          object
#Passengers     int64
dtype: object
```

数据包含了指定的月份和该月的游客数量。但是时间序列对象的读取和数据类型的“对象”和“整数类型”的读取是不一样的。为了将读取的数据作为时间序列，我们必须通过特殊的参数读取csv指令。

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m')
data = pd.read_csv('AirPassengers.csv', parse_dates='Month', index_col='Month',date_parser=dateparse)
print data.head()
```

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

我们逐个解释这些参数：

1、parse_dates:这是指定含有时间数据信息的列。正如上面所说的，列的名称为“月份”。

1、index_col:使用pandas 的时间序列数据背后的关键思想是：目录成为描述时间数据信息的变量。所以该参数告诉pandas使用“月份”的列作为索引。

2、date_parser：指定将输入的字符串转换为可变的时间数据。Pandas默认的数据读取格式是‘YYYY-MM-DD HH:MM:SS’。如需要读取的数据没有默认的格式，就要人工定义。这和dataparse的功能部分相似，这里的定义可以为这一目的服务。

现在我们看到数据有作为索引的时间对象和作为列的乘客（#Passengers）。我们可以通过以下指令再次检查索引的数据类型。

```
data.index
```

```
DatetimeIndex(['1949-01-01', '1949-02-01', '1949-03-01', '1949-04-01',
               '1949-05-01', '1949-06-01', '1949-07-01', '1949-08-01',
               '1949-09-01', '1949-10-01',
               ...
               '1960-03-01', '1960-04-01', '1960-05-01', '1960-06-01',
               '1960-07-01', '1960-08-01', '1960-09-01', '1960-10-01',
               '1960-11-01', '1960-12-01'],
              dtype='datetime64[ns]', name=u'Month', length=144, freq=None)
```

注意： **dtype=’datetime[ns]’** 确认它是一个时间数据对象。个人而言，我会将列转换为序列对象，这样当我每次使用时间序列的时候，就不需要每次都要提及列名称。当然，这因人而异，如果能令你更好工作，可以使用它作为数据框架。

ts = data[‘#Passengers’] ts.head(10)

```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
1949-06-01    135
1949-07-01    148
1949-08-01    148
1949-09-01    136
1949-10-01    119
Name: #Passengers, dtype: int64
```

在进一步深入前，我会探讨一些关于时间序列数据的索引技术。先在序列对象选择一个特殊值。可以通过以下两种方式实现：

```
#1. Specific the index as a string constant:
ts['1949-01-01']

#2. Import the datetime library and use 'datetime' function:
from datetime import datetime
ts[datetime(1949,1,1)]
```

两种方法都会返回值“112”，这可以通过先前的结果确认。希望我们可以获得1949年5月（包括1949年5月）之前的数据。这又可以用以下两种方法实现：

```
#1. Specify the entire range:
ts['1949-01-01':'1949-05-01']

#2. Use ':' if one of the indices is at ends:
ts[:'1949-05-01']
```

两种方法都会输出以下结果：

```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
Name: #Passengers, dtype: int64
```

我们要注意两点：

- 跟数值索引不一样，结束索引在这儿是被包含的。比如，如果令一个列表成为索引作为一个数组[:5]，它将在索引返回值-[0,1,2,3,4]. 在这里索引‘1949-05-01’是包含在结果输出里面的。
- 目录必须为了工作区间而进行分类。如果你随意打乱这些索引，将不能工作。

考虑到可以使用另外一个例子：你需要1949年所有的值。可以这样做：

```
ts['1949']
```

```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
1949-06-01    135
1949-07-01    148
1949-08-01    148
1949-09-01    136
1949-10-01    119
1949-11-01    104
1949-12-01    118
Name: #Passengers, dtype: int64
```

可见，月份部分已经省略。如果你要获得某月所有的日期，日期部分也可以省略。

现在，让我们开始分析时间序列。

3、如何检验时间序列的稳定性？

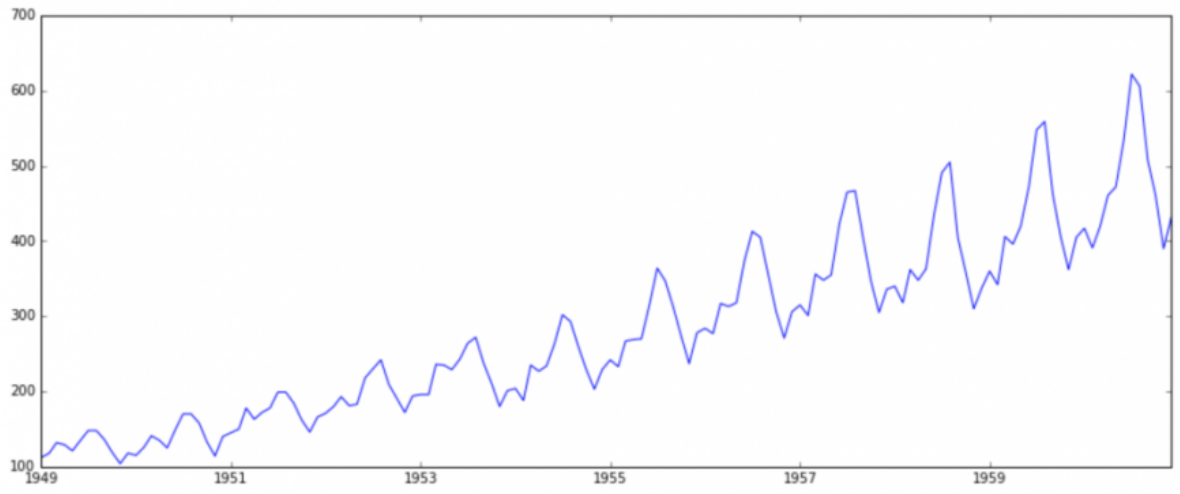
如果一个时间序列的统计特征如平均数，方差随着时间保持不变，我们就可以认为它是稳定的。为什么时间序列的稳定性这么重要？大部分时间序列模型是在假设它是稳定的前提下建立的。直观地说，我们可以这样认为，如果一个时间序列随着时间产生特定的行为，就有很高的可能性认为它在未来的行为是一样的。同时，根据稳定序列得出的理论是更加成熟的， 也是更容易实现与非稳定序列的比较。

稳定性的确定标准是非常严格的。但是，如果时间序列随着时间产生恒定的统计特征，根据实际目的我们可以假设该序列是稳定的。如下：

- **恒定的平均数**
- **恒定的方差**
- **不随时间变化的自协方差**

我会跳过一些细节，因为在文章已经说的非常清楚。接下来，是关于测试稳定性的方法。首先是简化坐标和数据，进行可视分析。数据可以通过以下指令定位。

```
ts['1949']
```



非常清晰的看到，随着季节性的变动，飞机乘客的数量总体上是在不断增长的。但是，不是经常都可以获得这样清晰的视觉推论（下文给出相应例子）。所以，更正式的讲，我们可以通过下面的方法测试稳定性。

1、绘制滚动统计：我们可以绘制移动平均数和移动方差，观察它是否随着时间变化。随着移动平均数和方差的变化，我认为在任何“t”瞬间，我们都可以获得去年的移动平均数和方差。如：上一个12个月份。但是，这更多的是一种视觉技术。

2、DF检验：这是一种检查数据稳定性的统计测试。无效假设：时间序列是不稳定的。测试结果由测试统计量和一些置信区间的临界值组成。如果“测试统计量”少于“临界值”，我们可以拒绝无效假设，并认为序列是稳定的。

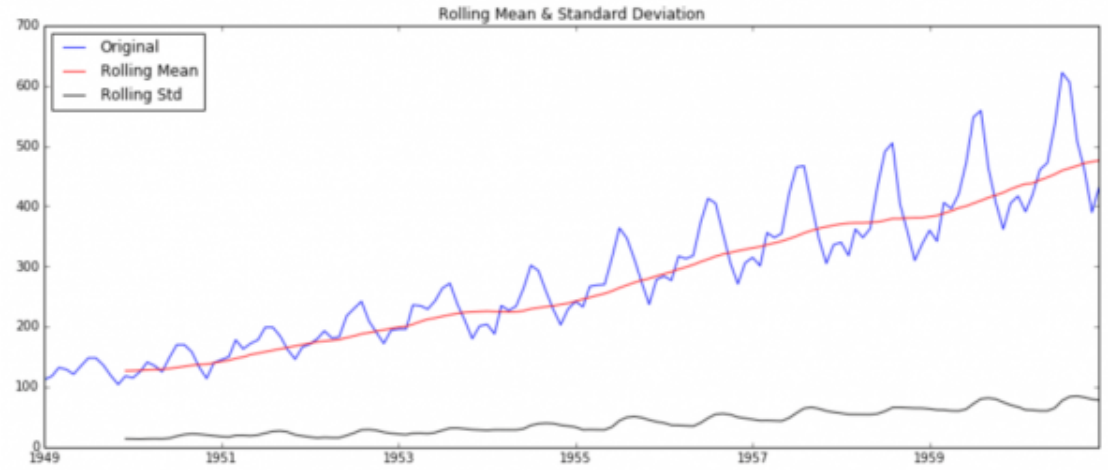
这些概念不是凭直觉得出来的。我推荐大家浏览前篇文章。如果你对一些理论数据感兴趣，你可以参考Brockwell 和 Davis关于时间序列和预测的介绍的书。这本书的数据很多，但是如果你能读懂言外之意，你会明白这些概念和正面接触这些数据。

回到检查稳定性这件事上，我们将使用滚动数据坐标连同许多DF测试结果，我已经定义了一个需要时间序列作为输入的函数，为我们生成结果。请注意,我已经绘制标准差来代替方差，为了保持单元和平均数相似。

```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = pd.rolling_mean(timeseries, window=12)
    rolstd = pd.rolling_std(timeseries, window=12)

    #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
```

代码是非常直观的，如果你在阅读过程中遇到挑战可以在评论处提出。

输入序列开始运行：




```
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 (5%)             -2.884042
Critical Value (1%)             -3.481682
Critical Value (10%)            -2.578770
dtype: float64
```

基本上是不可能使序列完全稳定，我们只能努力让它尽可能的稳定。

先让我们弄明白是什么导致时间序列不稳定。这儿有两个主要原因。

- **趋势-随着时间产生不同的平均值。**举例：在飞机乘客这个案例中，我们看到总体上，飞机乘客的数量是在不断增长的。
- **季节性-特定时间框架内的变化。**举例：在特定的月份购买汽车的人数会有增加的趋势，因为车价上涨或者节假日到来。

模型的根本原理或者预测序列的趋势和季节性，从序列中删除这些因素，将得到一个稳定的序列。然后统计预测技术可以在这个序列上完成。最后一步是通过运用趋势和季节性限制倒回到将预测值转换成原来的区间。

注意：我将探讨一系列方法。可能有些对文中情况有用，有些不能。但是我的目的是得到一系列可用方法，而不是仅仅关注目前的问题。

让我们通过分析趋势的一部分开始工作吧。

预测&消除趋势

消除趋势的第一个方法是转换。例如,在本例中,我们可以清楚地看到,有一个显著的趋势。所以我们可以通过变换，惩罚较高值而不是较小值。这可以采用日志,平方根,立方跟等等。让我们简单在这儿转换一个对数。

```
ts_log = np.log(ts)
plt.plot(ts_log)
```

在这个简单的例子中,很容易看到一个向前的数据趋势。但是它表现的不是很直观。所以我们可以使用一些技术来估计或对这个趋势建模,然后将它从序列中删除。这里有很多方法,最常用的有:

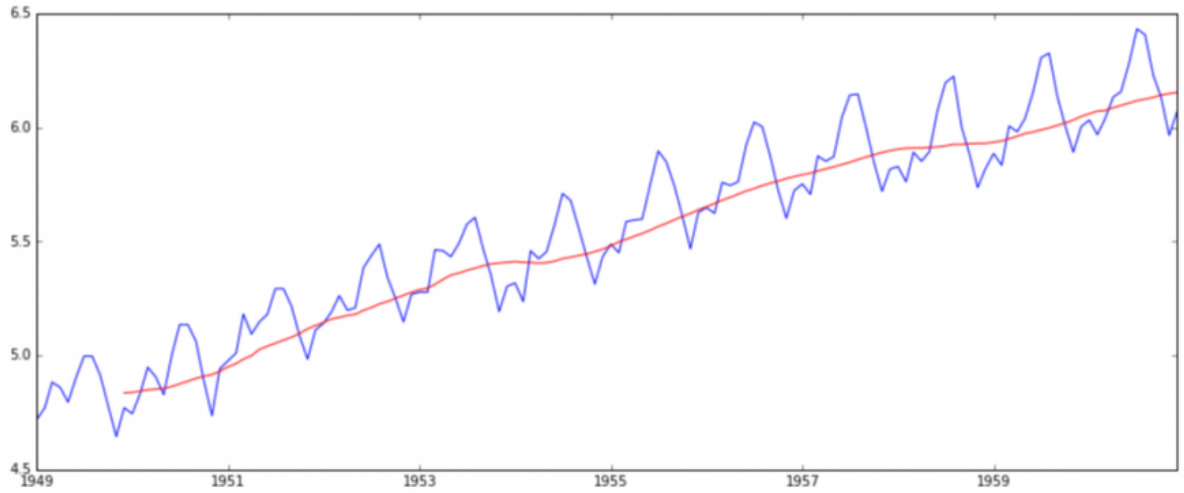
- **聚合-取一段时间的平均值（月/周平均值）**
- **平滑-取滚动平均数**
- **多项式回归分析-适合的回归模型**

我在这儿讨论将平滑,你也应该尝试其他可以解决的问题的技术。平滑是指采取滚动估计,即考虑过去的几个实例。有各种方法可以解决这些问题，但我将主要讨论以下两个。

移动平均数

在这个方法中，根据时间序列的频率采用“K”连续值的平均数。我们可以采用过去一年的平均数，即过去12个月的平均数。关于确定滚动数据，pandas有特定的功能定义。

```
moving_avg = pd.rolling_mean(ts_log,12)
plt.plot(ts_log)
plt.plot(moving_avg, color='red')
```



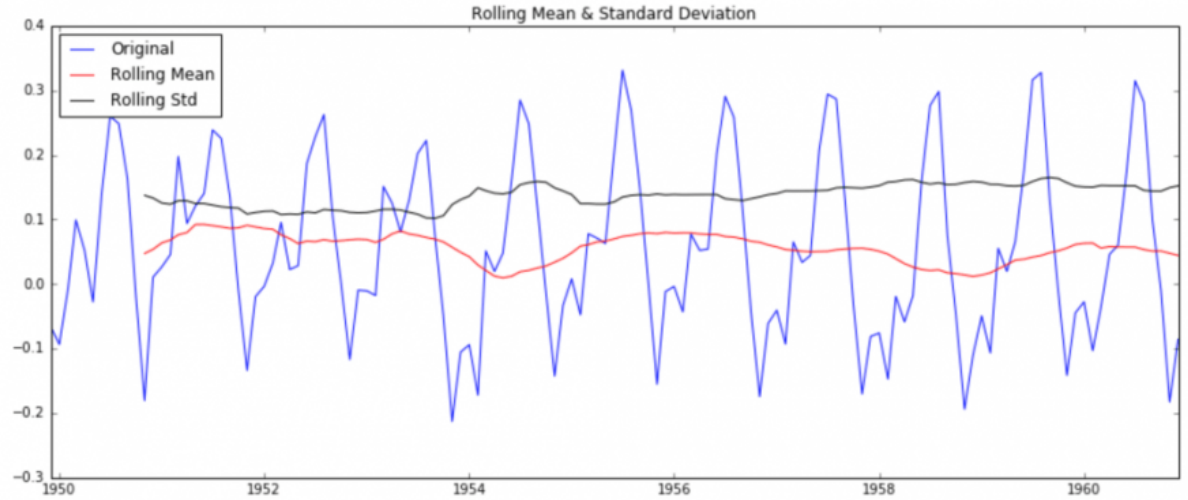
红色表示了滚动平均数。让我们从原始序列中减去这个平均数。注意，从我们采用过去12个月的值开始，滚动平均法还没有对前11个月的值定义。我们可以看到：

```
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
```

注意前11个月是非数字的，现在让我们对这11个月降值和检查这些模块去测试稳定性。

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



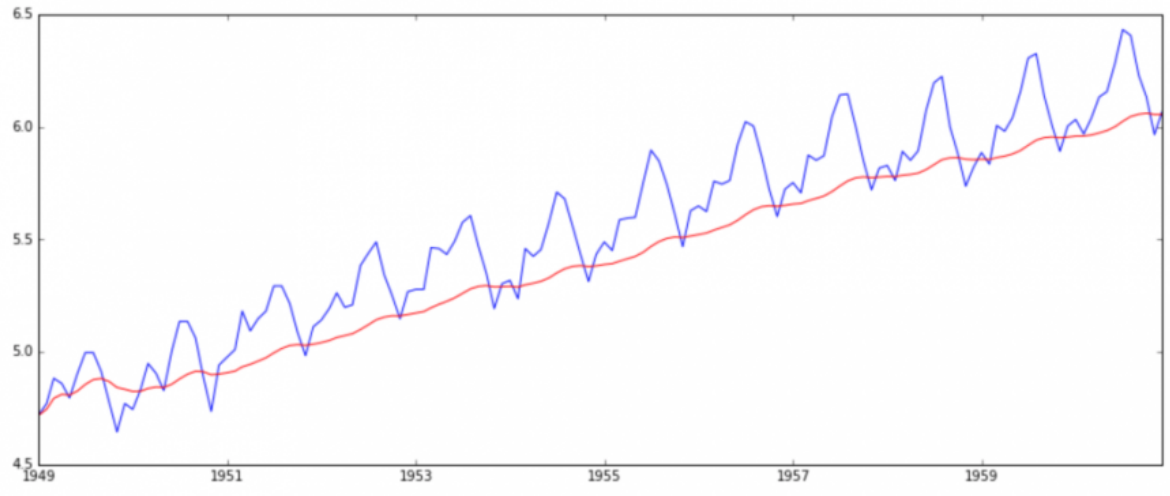

```
Results of Dickey-Fuller Test:
Test Statistic                  -3.162908
p-value                          0.022235
#Lags Used                      13.000000
Number of Observations Used    119.000000
Critical Value (5%)             -2.886151
Critical Value (1%)             -3.486535
Critical Value (10%)            -2.579896
dtype: float64
```

这看起来像个更好的序列。滚动平均值出现轻微的变化，但是没有明显的趋势。同时，检验统计量比5%的临界值小，所以我们在95%的置信区间认为它是稳定序列。

但是，这个方法有一个缺陷：要严格定义时段。在这种情况下，我们可以采用年平均数，但是对于复杂的情况的像预测股票价格，是很难得到一个数字的。所以，我们采取“加权移动平均法”可以对最近的值赋予更高的权重。关于指定加重这儿有很多技巧。

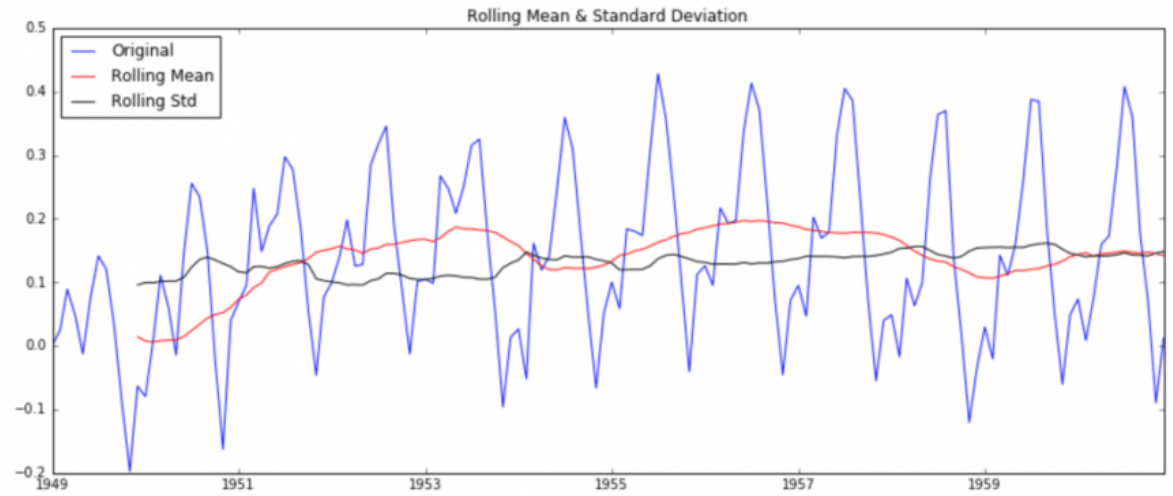
指数加权移动平均法是很受欢迎的方法，所有的权重被指定给先前的值连同衰减系数。这可以通过pandas实现：

```
expwighted_avg = pd.ewma(ts_log, halflife=12)
plt.plot(ts_log)
plt.plot(expwighted_avg, color='red')
```



注意,这里使用了参数“半衰期”来定义指数衰减量。这只是一个假设,将很大程度上取决于业务领域。其他参数,如跨度和质心也可以用来定义衰减，正如上面链接分享的探讨。现在让我们从这个序列转移，继续检查稳定性。

```
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 (5%)             -2.884042
Critical Value (1%)             -3.481682
Critical Value (10%)            -2.578770
dtype: float64
```

这个时间序列有更少的平均值变化和标准差大小变化。同时，检验统计量小于1%的临界值,这比以前的情况好。请注意在这种情况下就不会有遗漏值因为所有的值在一开始就被赋予了权重。所以在运行的时候，它没有先前的值参与。

消除趋势和季节性

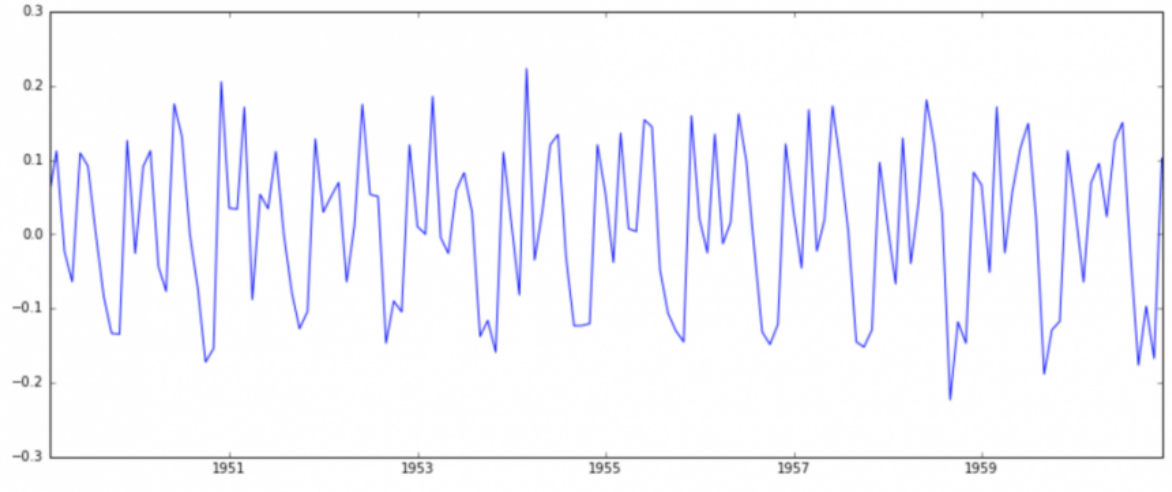
之前讨论来了简单的趋势减少技术不能在所有情况下使用，特别是在高季节性情况下。让我们谈论一下两种消除趋势和季节性的方法。

- **差分–采用一个特定时间差的差值**
- **分解——建立有关趋势和季节性的模型和从模型中删除它们。**

差分

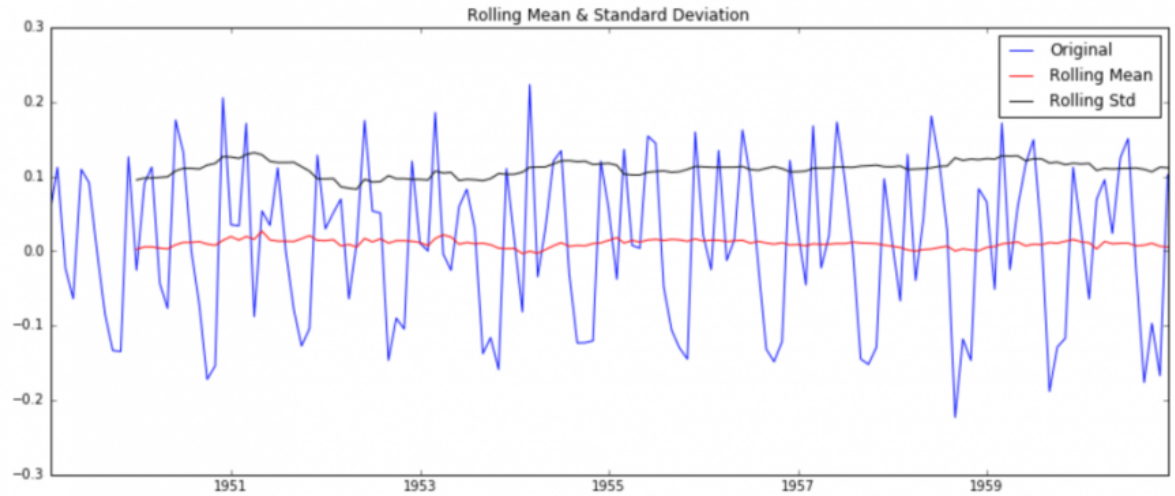
处理趋势和季节性的最常见的方法之一就是差分法。在这种方法中,我们采用特定瞬间和它前一个瞬间的不同的观察结果。这主要是在提高平稳性。pandas可以实现一阶差分：

```
ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff)
```



表中可以看出很大程度上减少了趋势。让我们通过模块验证一下：

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



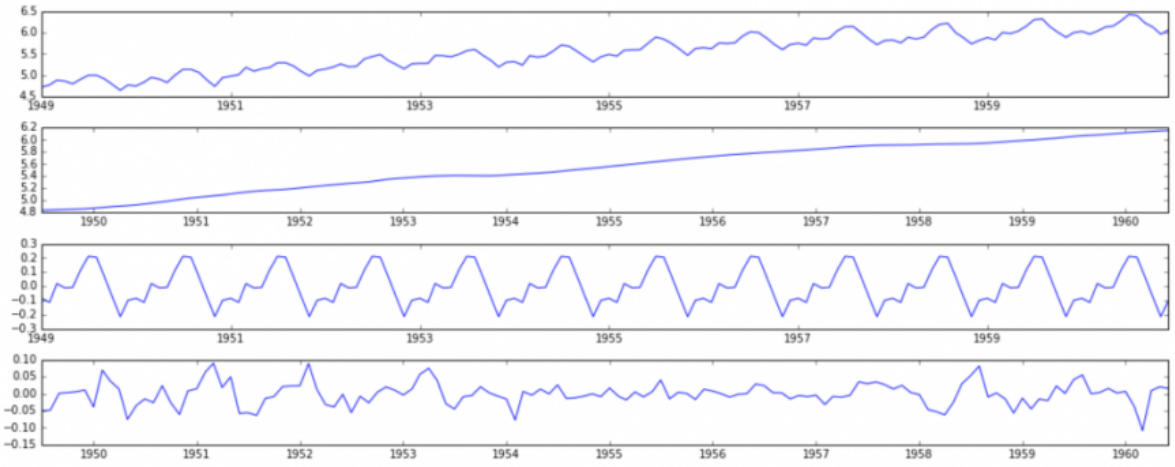

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.717131
p-value                          0.071121
#Lags Used                      14.000000
Number of Observations Used    128.000000
Critical Value (5%)             -2.884398
Critical Value (1%)             -3.482501
Critical Value (10%)            -2.578960
dtype: float64
```

我们可以看到平均数和标准差随着时间有小的变化。同时，DF检验统计量小于10% 的临界值，因此该时间序列在90%的置信区间上是稳定的。我们同样可以采取二阶或三阶差分在具体应用中获得更好的结果。这些方法你可以自己尝试。

分解

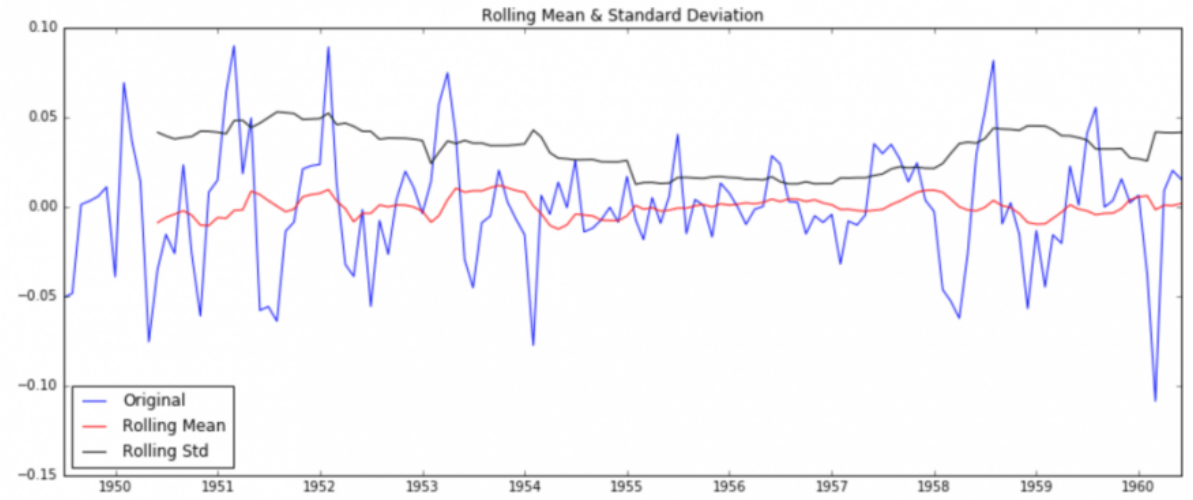
在这种方法中,趋势和季节性是分别建模的并倒回到序列的保留部分。我将跳过统计数据，直接给出结果:

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

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

plt.subplot(411)
plt.plot(ts_log, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```



在这里我们可以看到趋势,季节性从数据分离，我们可以建立残差的模型，让我们检查残差的稳定性：

```
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 (5%)           -2.885538e+00
Critical Value (1%)           -3.485122e+00
Critical Value (10%)          -2.579569e+00
dtype: float64
```

DF测试统计量明显低于1%的临界值，这样时间序列是非常接近稳定。你也可以尝试高级的分解技术产生更好的结果。同时,你应该注意到, 在这种情况下将残差转换为原始值对未来数据不是很直观。

预测时间序列

我们看到不同的技术和它们有效的工作使得时间序列得以稳定。让我们建立差分后的时间序列模型，因为它是很受欢迎的技术，也相对更容易添加噪音和季节性倒回到预测残差。在执行趋势和季节性评估技术上,有两种情况:

- **不含依赖值的严格稳定系列。**简单的情况下,我们可以建立残差模型作为白噪音（指功率谱密度在整个频域内均匀分布的噪声）。但这是非常罕见的。

- **序列含有明显的依赖值。**在这种情况下,我们需要使用一些统计模型像ARIMA（差分自回归移动平均模型）来预测数据。

让我给你简要介绍一下ARIMA，我不会介绍技术细节,但如果你希望更有效地应用它们，你应该理解这些概念的细节。ARIMA代表自回归整合移动平均数。平稳时间序列的ARIMA预测的只不过是一个线性方程(如线性回归)。预测依赖于ARIMA模型参数(p d q)。

- **自回归函数（AR）的条件（p）**：AR条件仅仅是因变量的滞后。如：如果P等于5，那么预测x（t）将是x（t-1）。。。（t-5）。
- **移动平均数(MA)的条件(q):**MA条件是预测方程的滞后预测错误。如：如果q等于5，预测x（t）将是e（t-1）。。。e(t-5),e(i)是移动平均叔在第ith个瞬间和实际值的差值。
- **差分（d）：**有非季节性的差值，即这种情况下我们采用一阶差分。所以传递变量，令d=0或者传递原始变量，令d=1。两种方法得到的结果一样。

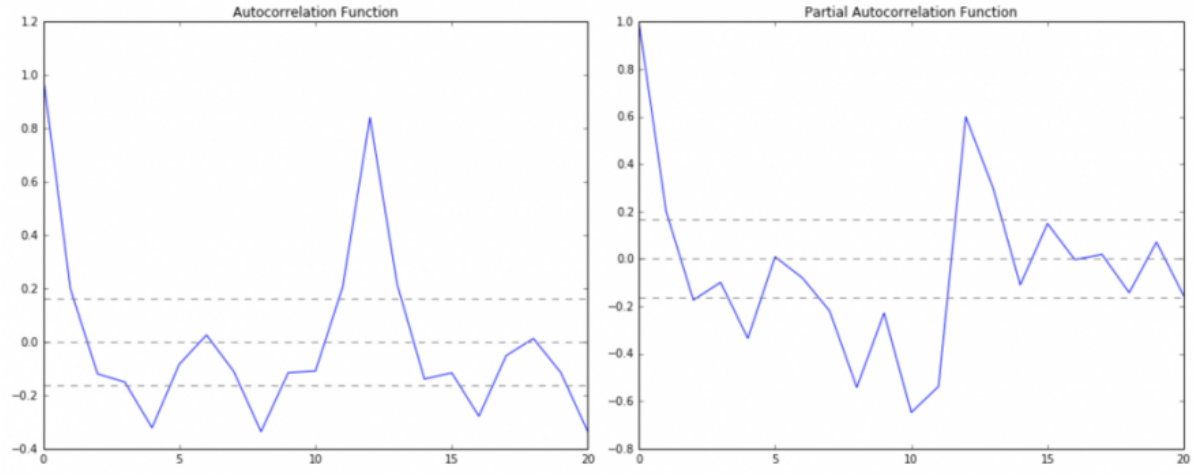
在这里一个重要的问题是如何确定“p”和“q”的值。我们使用两个坐标来确定这些数字。我们来讨论它们。

1. 、**自相关函数（ACF）**：这是时间序列和它自身滞后版本之间的相关性的测试。比如在自相关函数可以比较时间的瞬间‘t1’…’t2’以及序列的瞬间‘t1-5’…’t2-5’ (t1-5和t2 是结束点)。
2. **部分自相关函数(PACF)**:这是时间序列和它自身滞后版本之间的相关性测试，但是是在预测（已经通过比较干预得到解释）的变量后。如：滞后值为5，它将检查相关性，但是会删除从滞后值1到4得到的结果。

时间序列的自回归函数和部分自回归函数可以在差分后绘制为:

```
#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()
```



在这个点上,0的每一条边上的两条虚线之间是置信区间。这些可以用来确定“p”和“q”的值:

1、p-部分自相关函数表第一次截断的上层置信区间是滞后值。如果你仔细看，该值是p=2。

2、q- 自相关函数表第一次截断的上层置信区间是滞后值。如果你仔细看，该值是q=2。

现在，考虑个体以及组合效应建立3个不同的ARIMA模型。我也会发布各自的RSS（是一种描述和同步网站内容的格式，是使用最广泛的XML应用）。请注意,这里的RSS是指残差值,而不是实际序列。

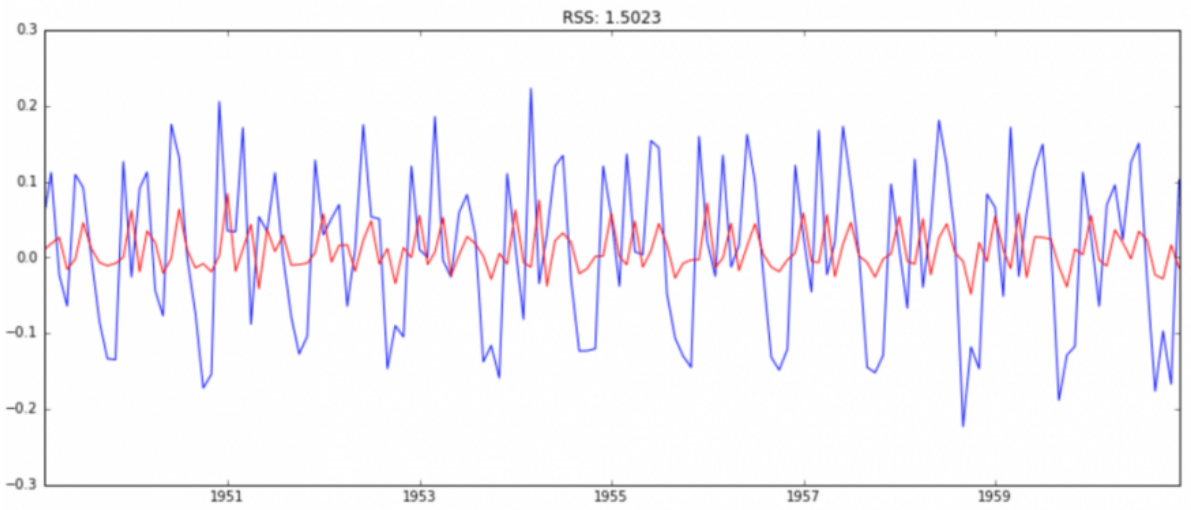
首先，我们需要上传ARIMA模型。

```
from statsmodels.tsa.arima_model import ARIMA
```

p,d,q值可以指定使用ARIMA的命令参数即采用一个元(p,d,q)。建立三种情况下的模型：

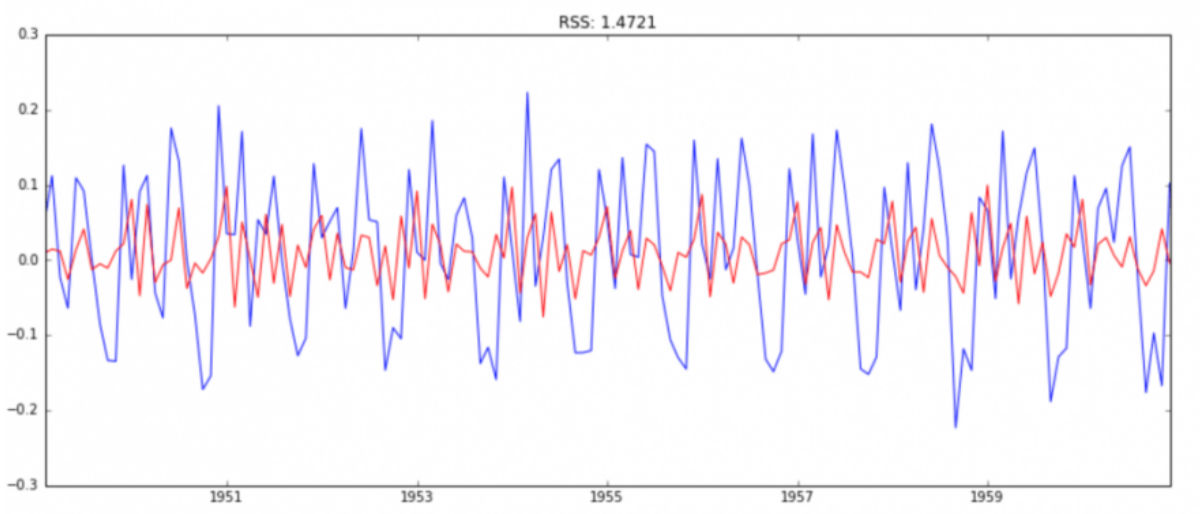
自回归（AR）模型：

```
model = ARIMA(ts_log, order=(2, 1, 0))
results_AR = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_AR.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_AR.fittedvalues-ts_log_diff)**2))
```



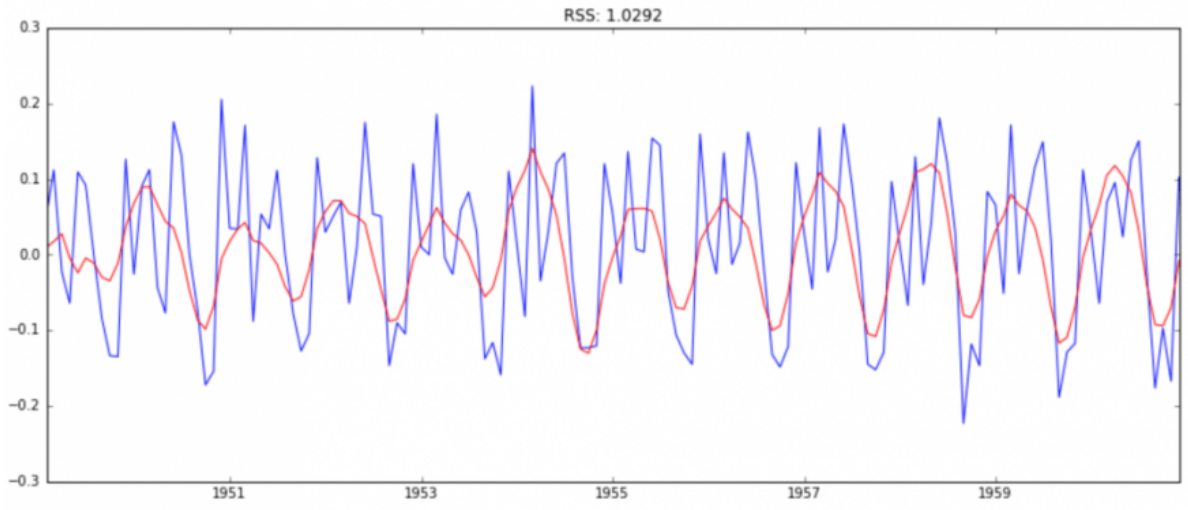
组合模型

```
model = ARIMA(ts_log, order=(0, 1, 2))
results_MA = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_MA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_MA.fittedvalues-ts_log_diff)**2))
```



移动平均数（MA）模型

```
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))
```



在这里我们可以看到,自回归函数模型和移动平均数模型几乎有相同的RSS，但相结合效果显著更好。现在,我们只剩下最后一步,即把这些值倒回到原始区间。

倒回到原始区间

既然组合模型获得更好的结果，让我们将它倒回原始值，看看它如何执行。第一步是作为一个独立的序列，存储预测结果,观察它。

```
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   -0.023889
dtype: float64
```

注意,这些是从‘1949-02-01’开始,而不是第一个月。为什么?这是因为我们将第一个月份取为滞后值，一月前面没有可以减去的元素。将差分转换为对数尺度的方法是这些差值连续地添加到基本值。一个简单的方法就是首先确定索引的累计总和,然后将其添加到基本值。累计总和可以在下面找到:

```
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
dtype: float64
```

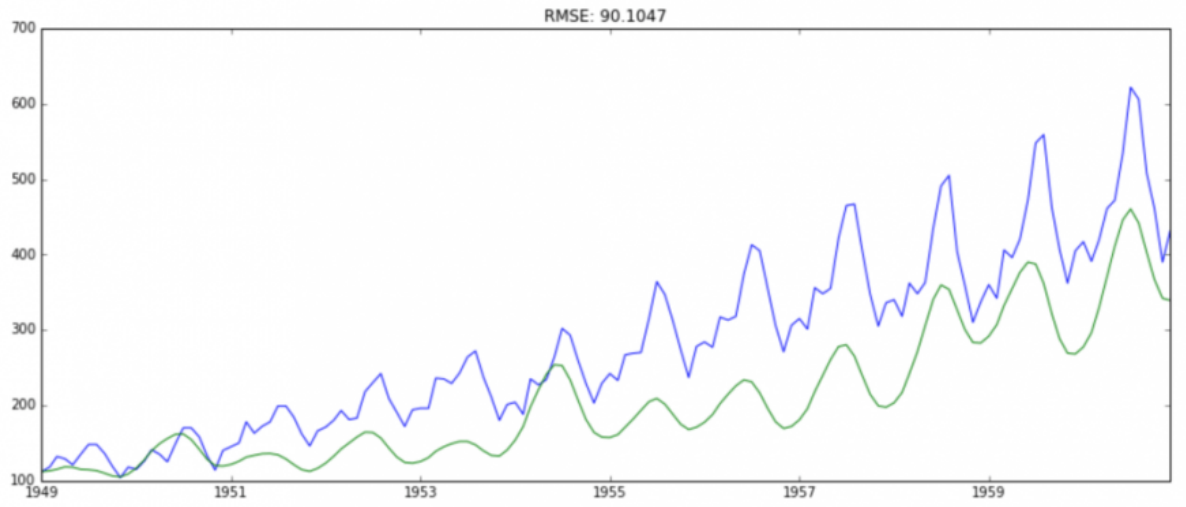
你可以在头脑使用之前的输出结果进行回算，检查这些是否正确的。接下来我们将它们添加到基本值。为此我们将使用所有的值创建一个序列作为基本值，并添加差值。我们这样做：

```
predictions_ARIMA_log = pd.Series(ts_log.ix[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
dtype: float64
```

第一个元素是基本值本身，从基本值开始值累计添加。最后一步是将指数与原序列比较。

```
predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(ts)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-ts)**2)/len(ts)))
```



最后我们获得一个原始区间的预测结果。虽然不是一个很好的预测。但是你获得了思路对吗?现在,我把它留个你去进一步改进，做一个更好的方案。

最后注意

在本文中,我试图提供你们一个标准方法去解决时间序列问题。这个不可能达到一个更好的时间,因为今天是我们的小型编程马拉松，挑战你们是否可以解决类似的问题。我们广泛的讨论了稳定性的概念和最终的预测残差。这是一个漫长的过程,我跳过了一些统计细节,我鼓励大家使用这些作为参考材料。如果你不想复制粘贴,你可以从我的GitHub库下载iPython笔记本的代码。 (https://github.com/aarshayj/Analytics_Vidhya/tree/master/Articles)

原文：Complete guide to create a Time Series Forecast (with Codes in Python) (http://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/)

优就业Python教程-Python集合使用详解 (http://www.36dsj.com/archives/43644)

【Python】爬虫+ K-means 聚类分析电影海报主色 (http://www.36dsj.com/archives/42208)

Python 爬虫的工具列表 附Github代码下载链接 (http://www.36dsj.com/archives/36417)

End.

转载请注明来自36大数据（36dsj.com）：36大数据 (http://www.36dsj.com) » 时间序列预测全攻略（附带Python代码） (http://www.36dsj.com/archives/44065)

标签： Python (http://www.36dsj.com/archives/tag/python) R语言 (http://www.36dsj.com/archives/tag/r%e8%af%ad%e8%a8%80)
数据分析 (http://www.36dsj.com/archives/tag/data-analysis) 数据建模 (http://www.36dsj.com/archives/tag/%e6%95%b0%e6%8d%ae%e5%bb%ba%e6%a8%a1)
数据结构 (http://www.36dsj.com/archives/tag/%e6%95%b0%e6%8d%ae%e7%bb%93%e6%9e%84)
时间序列预测 (http://www.36dsj.com/archives/tag/%e6%97%b6%e9%97%b4%e5%ba%8f%e5%88%97%e9%a2%84%e6%b5%8b)

猜你喜欢

- ECharts, PHP, MySQL, Ajax, JQuery 实现前后端数据可视化 (http://www.36dsj.com/archives/98824) 2017-10-11
- Java数据结构与算法解析(八)——伸展树 (http://www.36dsj.com/archives/98348) 2017-10-10
- 如何科学地蹭热点：用python爬虫获取热门微博评论并进行情感分析 (http://www.36dsj.com/archives/98485) 2017-10-09

- Java数据结构与算法解析(九)——B树 (http://www.36dsj.com/archives/98327) 2017-10-09
- Python NLP入门教程 (http://www.36dsj.com/archives/98211) 2017-09-30
- 数据结构与算法–关键路径 (http://www.36dsj.com/archives/98184) 2017-09-30
- 选Python还是Java ？ (http://www.36dsj.com/archives/98154) 2017-09-29
- 用于构建优秀命令行的 4 个 Python 库 (http://www.36dsj.com/archives/98019) 2017-09-28
- Tensorflow + PyCharm (http://www.36dsj.com/archives/97770) 2017-09-27
- 如何为使用 Python 语言而辩论 (http://www.36dsj.com/archives/97749) 2017-09-27

评论 8

骚年哟，既然都来了，为何不吐槽几句呢？((o(^_ ^)o))

提交评论

昵称 昵称 (必填)

邮箱 邮箱 (必填)

网址 网址

匿名 2年前 (2016-03-21)
好多概念不懂...

匿名 2年前 (2016-04-17)
里面用arima预测的结果进行还原时有错，并不是累计求和，原始只是一阶差分，所以还原时应该是预测的结果与一阶滞后的求和

匿名 1年前 (2016-04-22)
翻译的错误太多。还好提供了原文

匿名 1年前 (2016-05-04)
原文在哪啊？

匿名 1年前 (2016-04-28)
刚开始接触时间序列的的小白，老师叫我们找一篇完整的源代码了解用机器学习方法进行时间序列的步骤，这篇代码算吗，哪一部分用到机器学习了呢？

匿名 1年前 (2016-06-30)
翻译错误太多！原文的错误没消除，还翻译出新的错误！

匿名 12个月前 (10-19)
为什么我根据你的代码做 acf和pacf的图都是空的呢

XU QIAN 7个月前 (03-21)
关于Arima还原预测值，如果我是进行样本外预测（预测未来），请问如何还原一阶差分的值？