×

-

(http://play.google.com/store/apps/details?id=com.analyticsvidhya.android)

(https://www.analyticsvidhya.com/blog/)

(https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+DS101+2018T2/about?utm_source=DS101AVBlogBanner&utm_medium=Topbanner&utm_campaign=DS101)

BUSINESS ANALYTICS (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/BUSINESS-ANALYTICS/)

MACHINE LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/MACHINE-LEARNING/)

PYTHON (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTHON-2/)

TIME SERIES (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/TIME-SERIES/)

A comprehensive beginner's guide to create a Time Series Forecast (with Codes in Python)

AARSHAY JAIN (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/AARSHAY/), FEBRUARY 6, 2016

Introduction

Time Series (referred as TS from now) is considered to be one of the less known skills in the analytics space (Even I had little clue about it a couple of days back). But as you know our inaugural **Mini Hackathon (http://datahack.analyticsvidhya.com/contest/mini-datahack)** is based on it, I set myself on a journey to learn the basic steps for solving a Time Series problem and here I am sharing the same with you. These will definitely help you get a decent model in our hackathon today.

Analytics Vidhya
Learn Everything About Analytics
COMPLETE GUIDE TO CREATE A TIME SERIES FORECAST WITH

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/guide.jpg)

Before going through this article, I highly recommend reading A Complete Tutorial on Time Series Modeling in R (https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/), which is like a prequel to this article. It focuses on fundamental concepts and is based on R and I will focus on using these concepts in solving a problem end-to-end along with codes in Python. Many resources exist for TS in R but very few are there for Python so I'll be using Python in this article.

Our journey would go through the following steps:

1. What makes Time Series Special?
2. Loading and Handling Time Series in Pandas
3. How to Check Stationarity of a Time Series?
4. How to make a Time Series Stationary?
5. Forecasting a Time Series

1. What makes Time Series Special?

As the name suggests, TS is a collection of data points collected at **constant time intervals**. These are analyzed to determine the long term trend so as to forecast the future or perform some other form of analysis. But what makes a TS different from say a regular regression problem? There are 2 things:

1. It is **time dependent**. So the basic assumption of a linear regression model that the observations are independent doesn't hold in this case.
2. Along with an increasing or decreasing trend, most TS have some form of **seasonality trends**, i.e. variations specific to a particular time frame. For example, if you see the sales of a woolen jacket over time, you will invariably find higher sales in winter seasons.

Because of the inherent properties of a TS, there are various steps involved in analyzing it. These are discussed in detail below. Lets start by loading a TS object in Python. We'll be using the popular AirPassengers data set which can be downloaded here (https://www.analyticsvidhya.com/wp-content/uploads/2016/02/AirPassengers.csv).

Please note that the aim of this article is to familiarize you with the various techniques used for TS in general. The example considered here is just for illustration and I will focus on coverage a breadth of topics and not making a very accurate forecast.

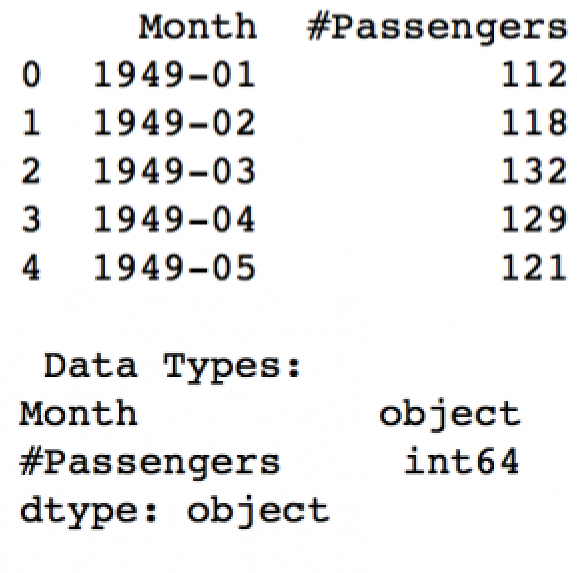
2. Loading and Handling Time Series in Pandas

Pandas has dedicated libraries for handling TS objects, particularly the **datatime64[ns]** class which stores time information and allows us to perform some operations really fast. Lets start by firing up the required libraries:

```
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
```

Now, we can load the data set and look at some initial rows and data types of the columns:

```
data = pd.read_csv('AirPassengers.csv')

print data.head()

print '\n Data Types:'

print data.dtypes
```

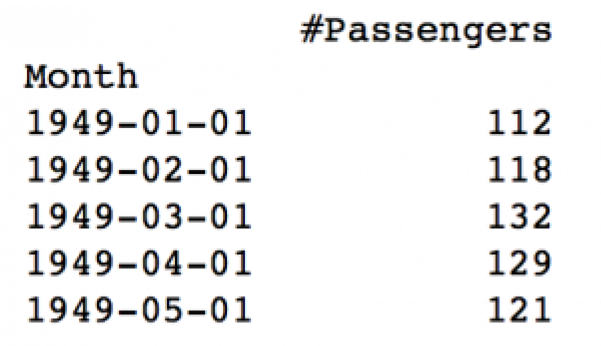


```
     Month  #Passengers
0  1949-01          112
1  1949-02          118
2  1949-03          132
3  1949-04          129
4  1949-05          121

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

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/1.-dataload-1.png)

The data contains a particular month and number of passengers travelling in that month. But this is still not read as a TS object as the data types are 'object' and 'int'. In order to read the data as a time series, we have to pass special arguments to the read_csv command:

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



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

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/2.-dataload-2.png)

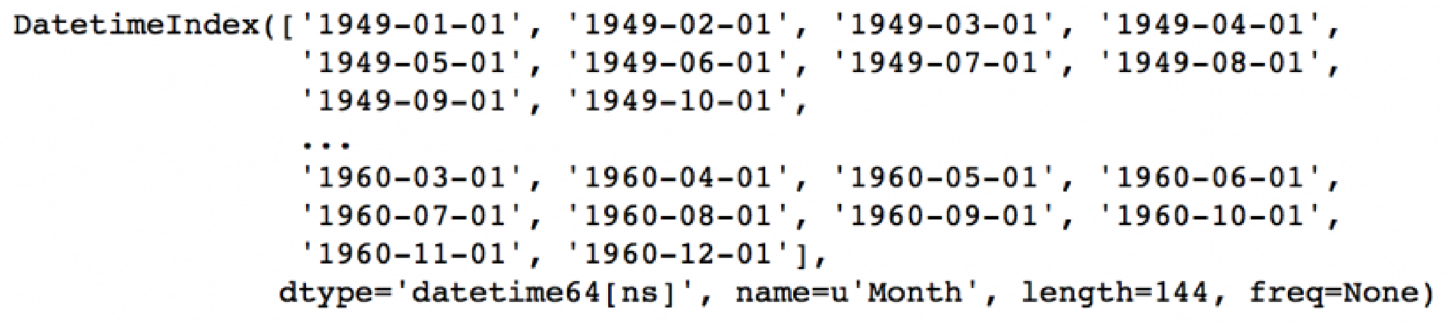
Let's understand the arguments one by one:

1. **parse_dates**: This specifies the column which contains the date-time information. As we say above, the column name is 'Month'.

2. **index_col:** A key idea behind using Pandas for TS data is that the index has to be the variable depicting date-time information. So this argument tells pandas to use the 'Month' column as index.
3. **date_parser:** This specifies a function which converts an input string into datetime variable. Be default Pandas reads data in format 'YYYY-MM-DD HH:MM:SS'. If the data is not in this format, the format has to be manually defined. Something similar to the dataparse function defined here can be used for this purpose.

Now we can see that the data has time object as index and #Passengers as the column. We can cross-check the datatype of the index with the following command:

```
data.index
```

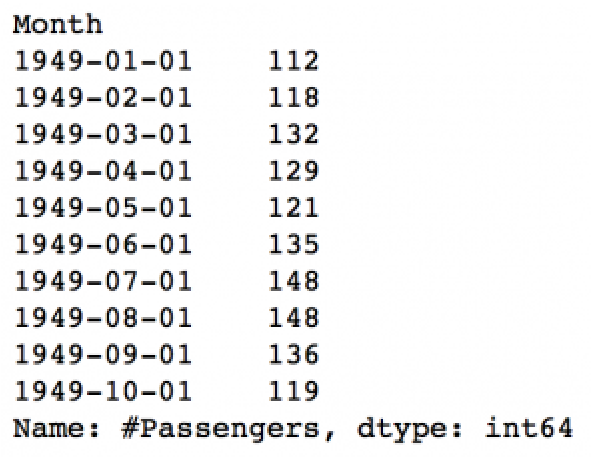


```
DatetimeIndex(['1949-01-01', '1949-02-01', '1949-03-01', '1949-04-01',
               '1949-05-01', '1949-06-01', '1949-07-01', '1949-08-01',
               '1949-09-01', '1949-10-01',
               ...
               '1960-03-01', '1960-04-01', '1960-05-01', '1960-06-01',
               '1960-07-01', '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)
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/3.-index-type.png)

Notice the **dtype='datetime[ns]'** which confirms that it is a datetime object. As a personal preference, I would convert the column into a Series object to prevent referring to columns names every time I use the TS. Please feel free to use as a dataframe is that works better for you.

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
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/4.-series.png)

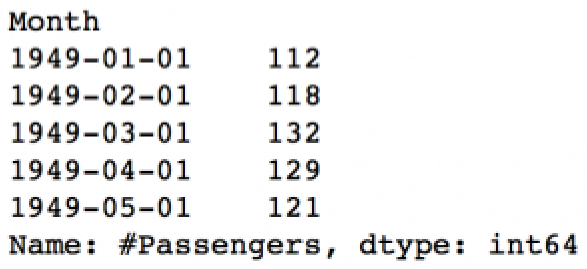
Before going further, I'll discuss some indexing techniques for TS data. Lets start by selecting a particular value in the Series object. This can be done in following 2 ways:

```
#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)]
```

Both would return the value '112' which can also be confirmed from previous output. Suppose we want all the data upto May 1949. This can be done in 2 ways:

```
#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']
```

Both would yield following output:


```
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
```

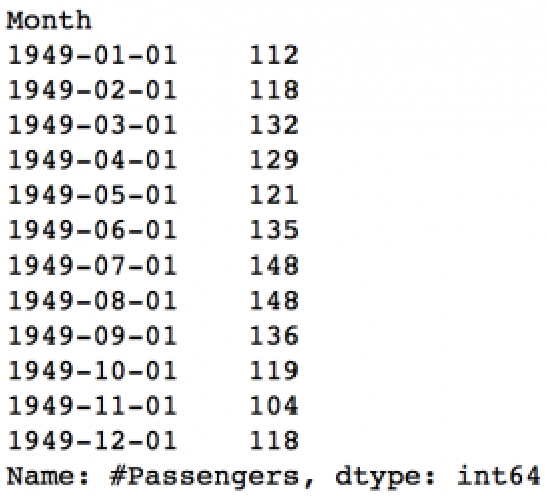
(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/5.-index-range.png)

There are 2 things to note here:

1. Unlike numeric indexing, the **end index is included here**. For instance, if we index a list as a[:5] then it would return the values at indices – [0,1,2,3,4]. But here the index '1949-05-01' was included in the output.
2. The **indices have to be sorted** for ranges to work. If you randomly shuffle the index, this won't work.

Consider another instance where you need all the values of the year 1949. This can be done as:

```
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
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/6.-index-year.png)

The month part was omitted. Similarly if you all days of a particular month, the day part can be omitted.

Now, lets move onto the analyzing the TS.

3. How to Check Stationarity of a Time Series?

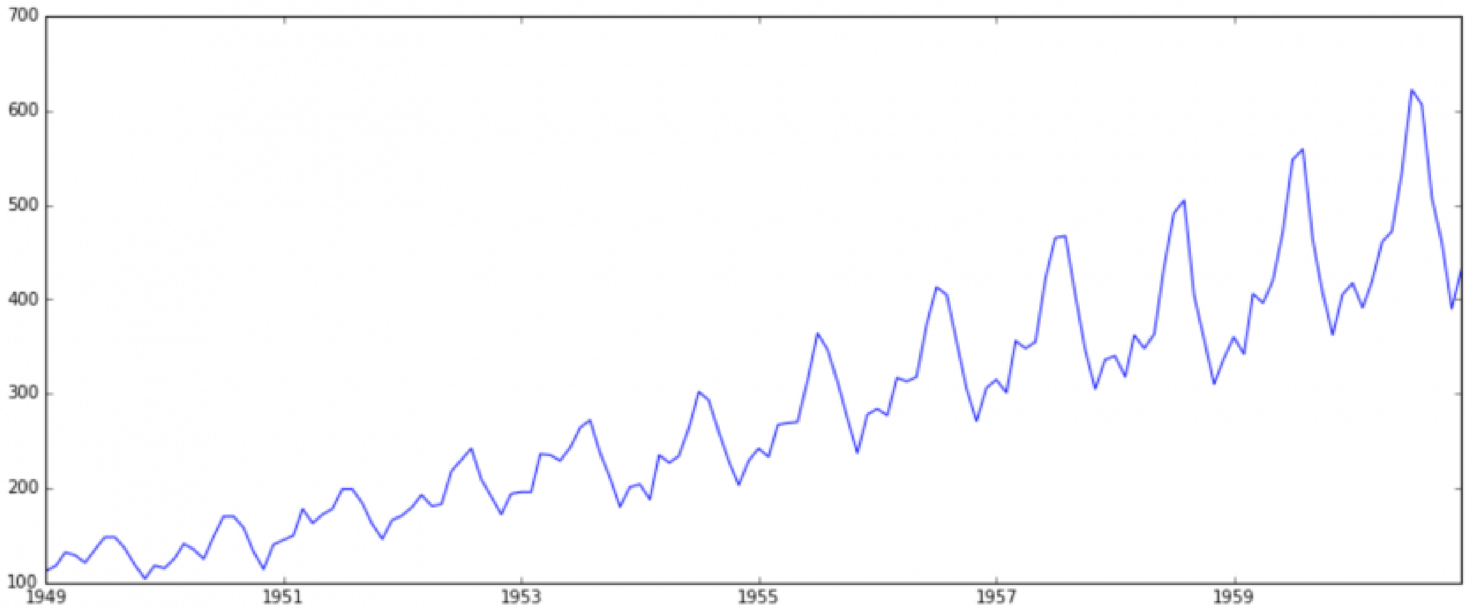
A TS is said to be stationary if its **statistical properties** such as mean, variance remain **constant over time**. But why is it important? Most of the TS models work on the assumption that the TS is stationary. Intuitively, we can sat that if a TS has a particular behaviour over time, there is a very high probability that it will follow the same in the future. Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series.

Stationarity is defined using very strict criterion. However, for practical purposes we can assume the series to be stationary if it has constant statistical properties over time, ie. the following:

1. constant mean
2. constant variance
3. an autocovariance that does not depend on time.

I'll skip the details as it is very clearly defined in this article (https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/). Lets move onto the ways of testing stationarity. First and foremost is to simple plot the data and analyze visually. The data can be plotted using following command:

```
plt.plot(ts)
```



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/7.-ts.png)

It is clearly evident that there is an **overall increasing trend** in the data along with some seasonal variations. However, it might not always be possible to make such visual inferences (we'll see such cases later). So, more formally, we can check stationarity using the following:

1. **Plotting Rolling Statistics:** We can plot the moving average or moving variance and see if it varies with time. By moving average/variance I mean that at any instant 't', we'll take the average/variance of the last year, i.e. last 12 months. But again this is more of a visual technique.
2. **Dickey-Fuller Test:** This is one of the statistical tests for checking stationarity. Here the null hypothesis is that the TS is non-stationary. The test results comprise of a **Test Statistic** and some **Critical Values** for difference confidence levels. If the 'Test Statistic' is less than the 'Critical Value', we can reject the null hypothesis and say that the series is stationary. Refer this article (https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/) for details.

These concepts might not sound very intuitive at this point. I recommend going through the prequel article. If you're interested in some theoretical statistics, you can refer **Introduction to Time Series and Forecasting** by **Brockwell and Davis**. The book is a bit stats-heavy, but if you have the skill to read-between-lines, you can understand the concepts and tangentially touch the statistics.

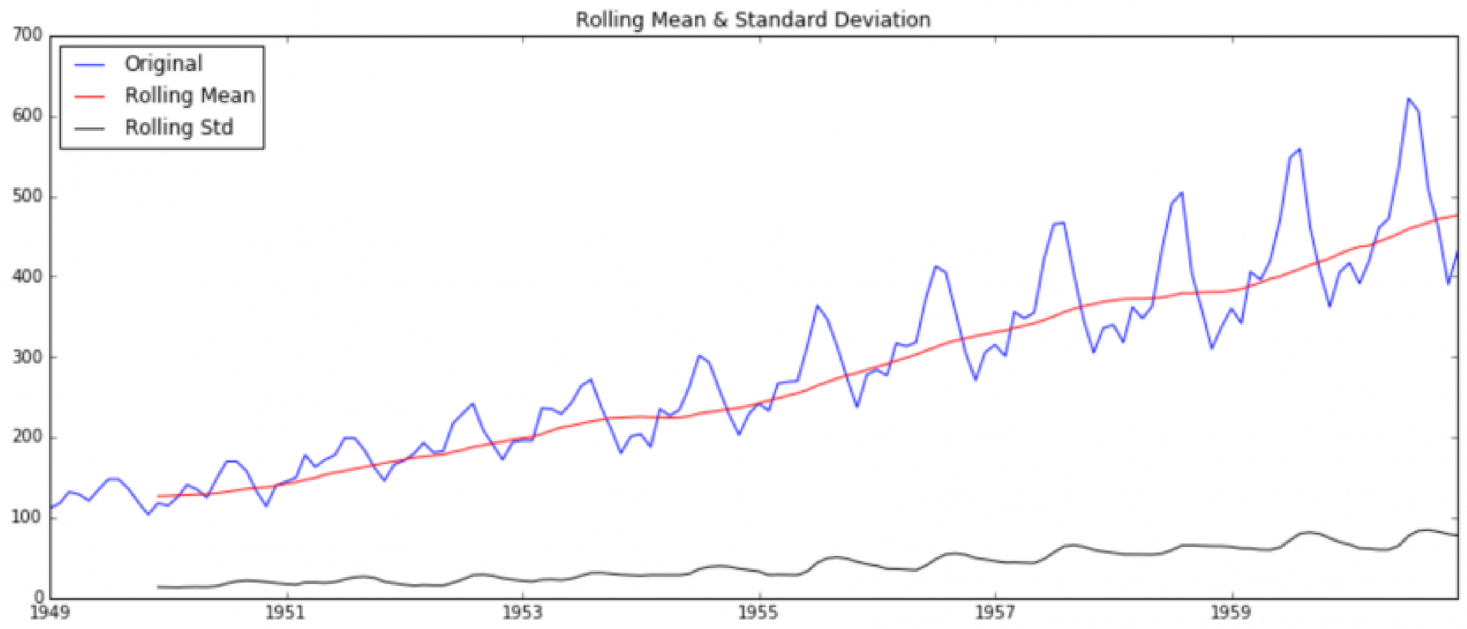
Back to checking stationarity, we'll be using the rolling statistics plots along with Dickey-Fuller test results a lot so I have defined a function which takes a TS as input and generated them for us. Please note that I've plotted standard deviation instead of variance to keep the unit similar to mean.

```
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 Obser
vations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print dfoutput
```

The code is pretty straight forward. Please feel free to discuss the code in comments if you face challenges in grasping it.

Let's run it for our input series:

```
test_stationarity(ts)
```



```
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
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/1.-dfuller-ts.png)

Though the variation in standard deviation is small, mean is clearly increasing with time and this is not a stationary series. Also, the test statistic is way more than the critical values. Note that the **signed values should be compared** and not the absolute values.

Next, we'll discuss the techniques that can be used to take this TS towards stationarity.

4. How to make a Time Series Stationary?

Though stationarity assumption is taken in many TS models, almost none of practical time series are stationary. So statisticians have figured out ways to make series stationary, which we'll discuss now. Actually, its almost impossible to make a series perfectly stationary, but we try to take it as close as possible.

Lets understand what is making a TS non-stationary. There are 2 major reasons behind non-stationaruty of a TS:
1. **Trend** – varying mean over time. For eg, in this case we saw that on average, the number of passengers was growing over time.

2. **Seasonality** – variations at specific time-frames. eg people might have a tendency to buy cars in a particular month because of pay increment or festivals.

The underlying principle is to model or estimate the trend and seasonality in the series and remove those from the series to get a stationary series. Then statistical forecasting techniques can be implemented on this series. The final step would be to convert the forecasted values into the original scale by applying trend and seasonality constraints back.

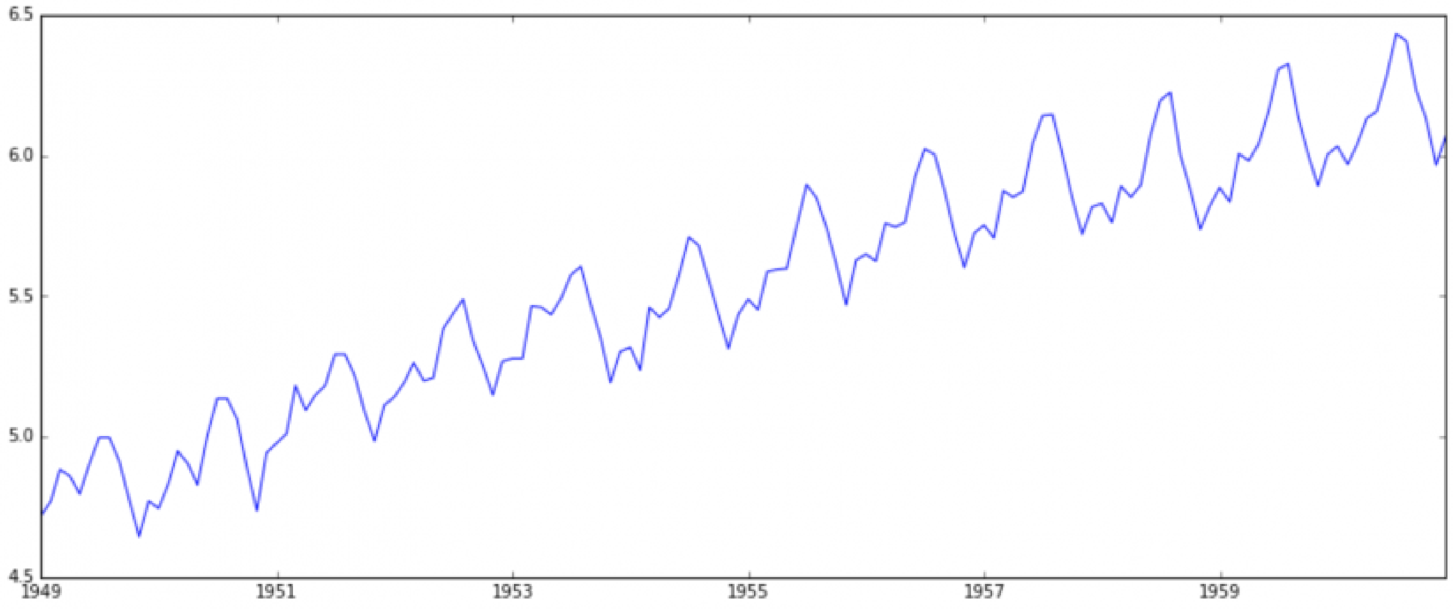
Note: I'll be discussing a number of methods. Some might work well in this case and others might not. But the idea is to get a hang of all the methods and not focus on just the problem at hand.

Let's start by working on the trend part.

Estimating & Eliminating Trend

One of the first tricks to reduce trend can be **transformation**. For example, in this case we can clearly see that the there is a significant positive trend. So we can apply transformation which penalize higher values more than smaller values. These can be taking a log, square root, cube root, etc. Lets take a **log transform** here for simplicity:

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



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/9.-ts-log.png)

In this simpler case, it is easy to see a forward trend in the data. But its not very intuitive in presence of noise. So we can use some techniques to estimate or model this trend and then remove it from the series. There can be many ways of doing it and some of most commonly used are:

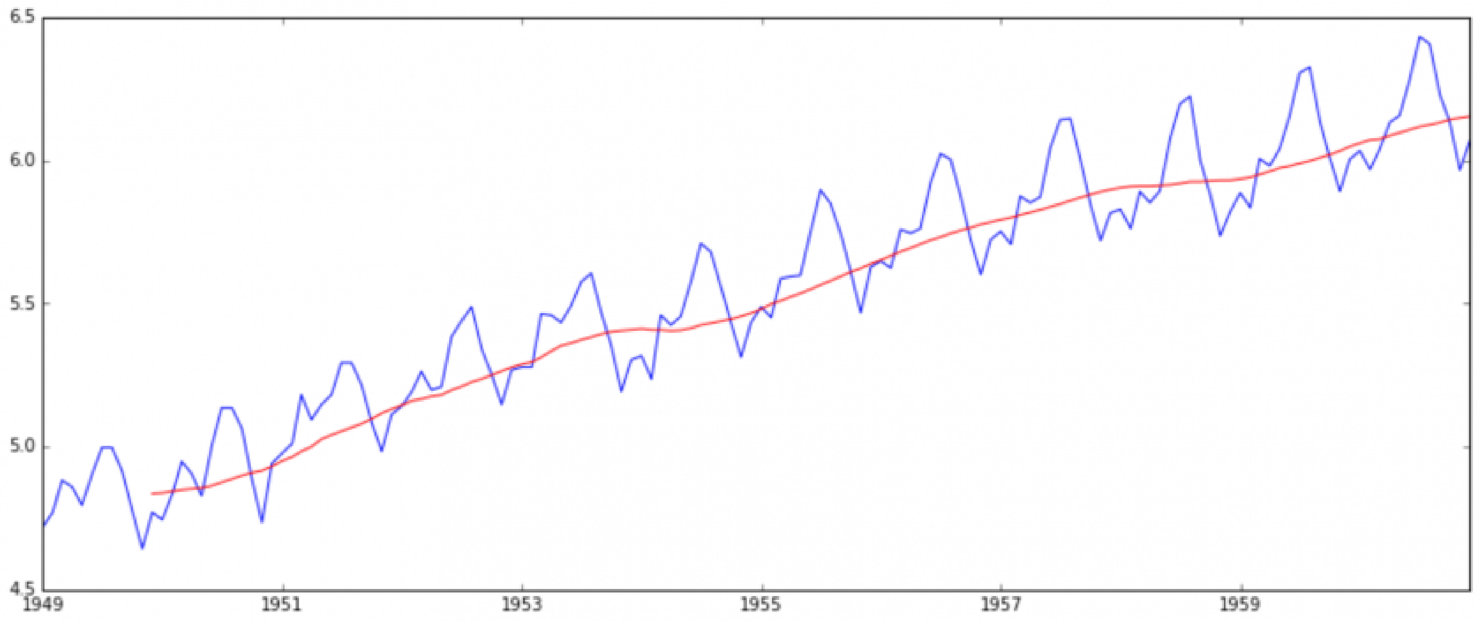
1. **Aggregation** – taking average for a time period like monthly/weekly averages
2. **Smoothing** – taking rolling averages
3. **Polynomial** **Fitting** – fit a regression model

I will discuss smoothing here and you should try other techniques as well which might work out for other problems. Smoothing refers to taking rolling estimates, i.e. considering the past few instances. There are can be various ways but I will discuss two of those here.

Moving average

In this approach, we take average of 'k' consecutive values depending on the frequency of time series. Here we can take the average over the past 1 year, i.e. last 12 values. Pandas has specific functions defined for determining rolling statistics.

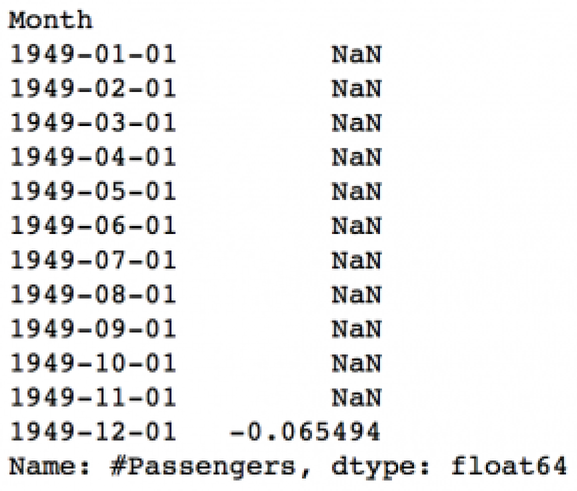
```
moving_avg = pd.rolling_mean(ts_log,12)

plt.plot(ts_log)

plt.plot(moving_avg, color='red')
```



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/10.-smooth-1.png)

The red line shows the rolling mean. Lets subtract this from the original series. Note that since we are taking average of last 12 values, rolling mean is not defined for first 11 values. This can be observed as:

```
ts_log_moving_avg_diff = ts_log - moving_avg

ts_log_moving_avg_diff.head(12)
```

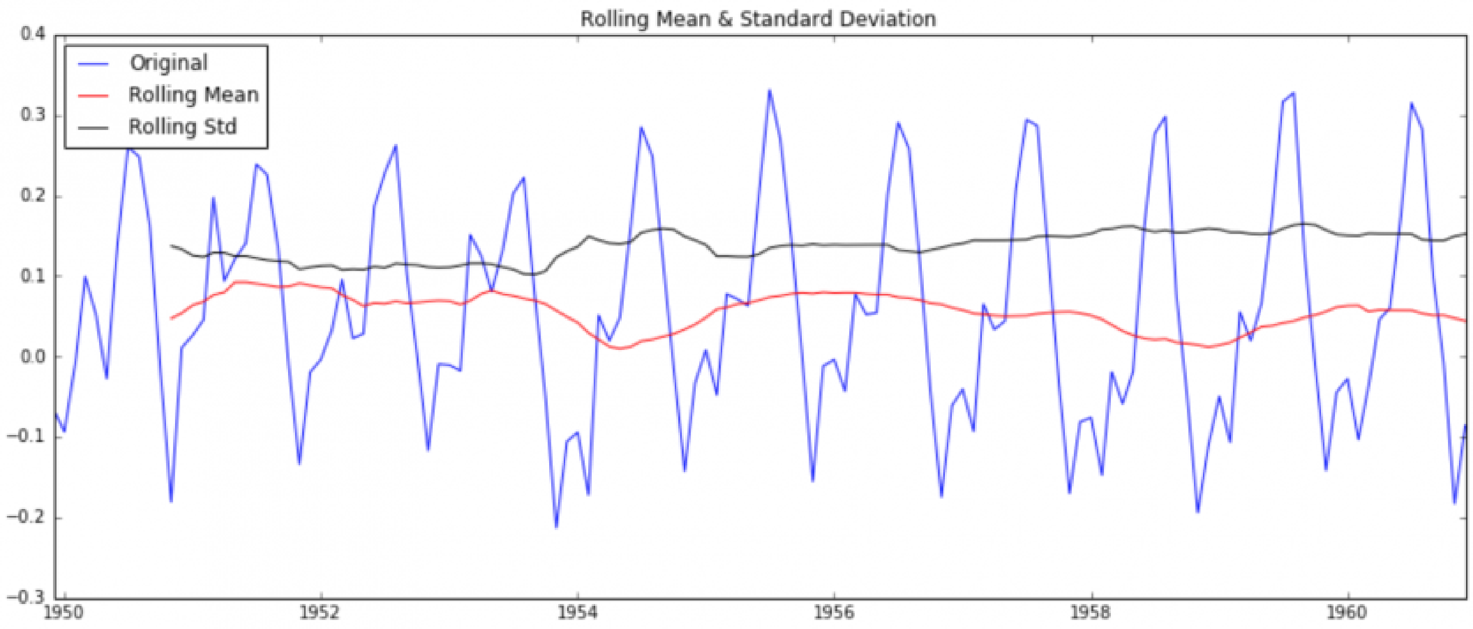


```
Month
1949-01-01         NaN
1949-02-01         NaN
1949-03-01         NaN
1949-04-01         NaN
1949-05-01         NaN
1949-06-01         NaN
1949-07-01         NaN
1949-08-01         NaN
1949-09-01         NaN
1949-10-01         NaN
1949-11-01         NaN
1949-12-01   -0.065494
Name: #Passengers,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

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/10.5-missing-rolling.png)

Notice the first 11 being Nan. Lets drop these NaN values and check the plots to test stationarity.

```
ts_log_moving_avg_diff.dropna(inplace=True)

test_stationarity(ts_log_moving_avg_diff)
```



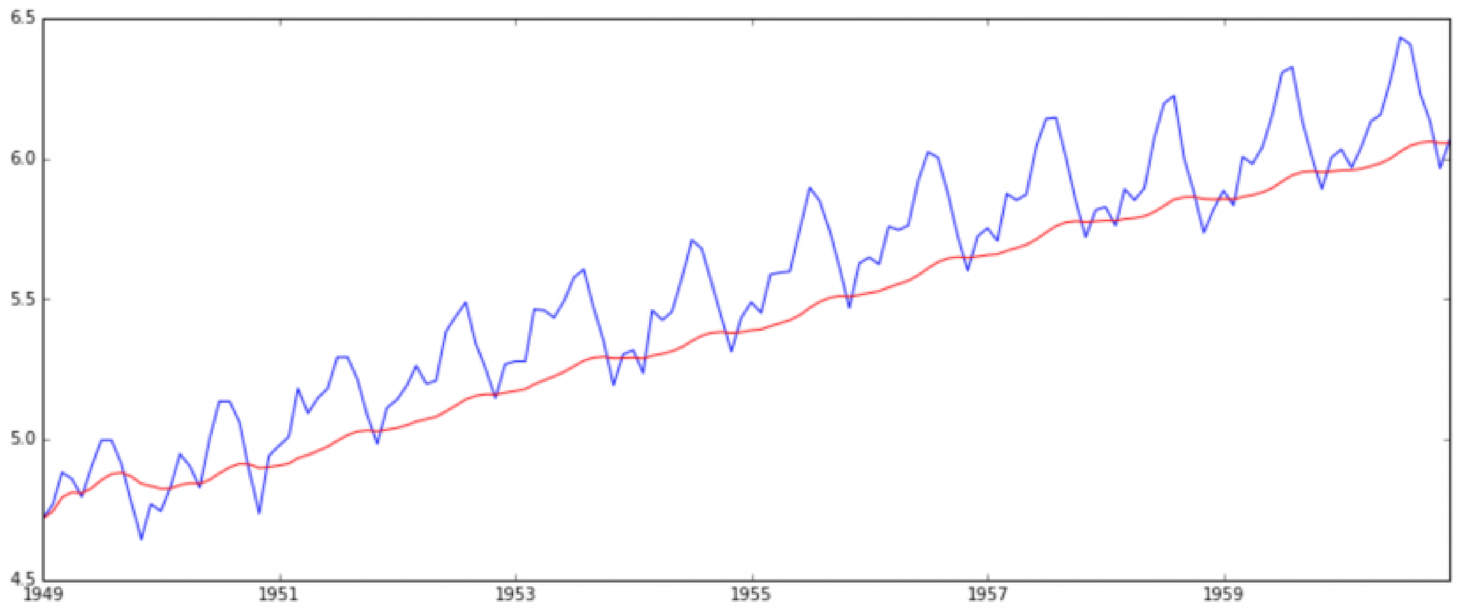
```
Results of Dickey-Fuller Test:
Test Statistic                  -3.162908
p-value                          0.022235
#Lags Used                      13.000000
Number of Observations Used    119.000000
Critical Value (5%)             -2.886151
Critical Value (1%)             -3.486535
Critical Value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

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/2.-dfuller-smooth-1.png)

This looks like a much better series. The rolling values appear to be varying slightly but there is no specific trend. Also, the test statistic is **smaller than the 5% critical values** so we can say with 95% confidence that this is a stationary series.

However, a drawback in this particular approach is that the time-period has to be strictly defined. In this case we can take yearly averages but in complex situations like forecasting a stock price, its difficult to come up with a number. So we take a 'weighted moving average' where more recent values are given a higher weight. There can be many technique for assigning weights. A popular one is **exponentially weighted moving average** where weights are assigned to all the previous values with a decay factor. Find details here (http://pandas.pydata.org/pandas-docs/stable/computation.html#exponentially-weighted-moment-functions). This can be implemented in Pandas as:

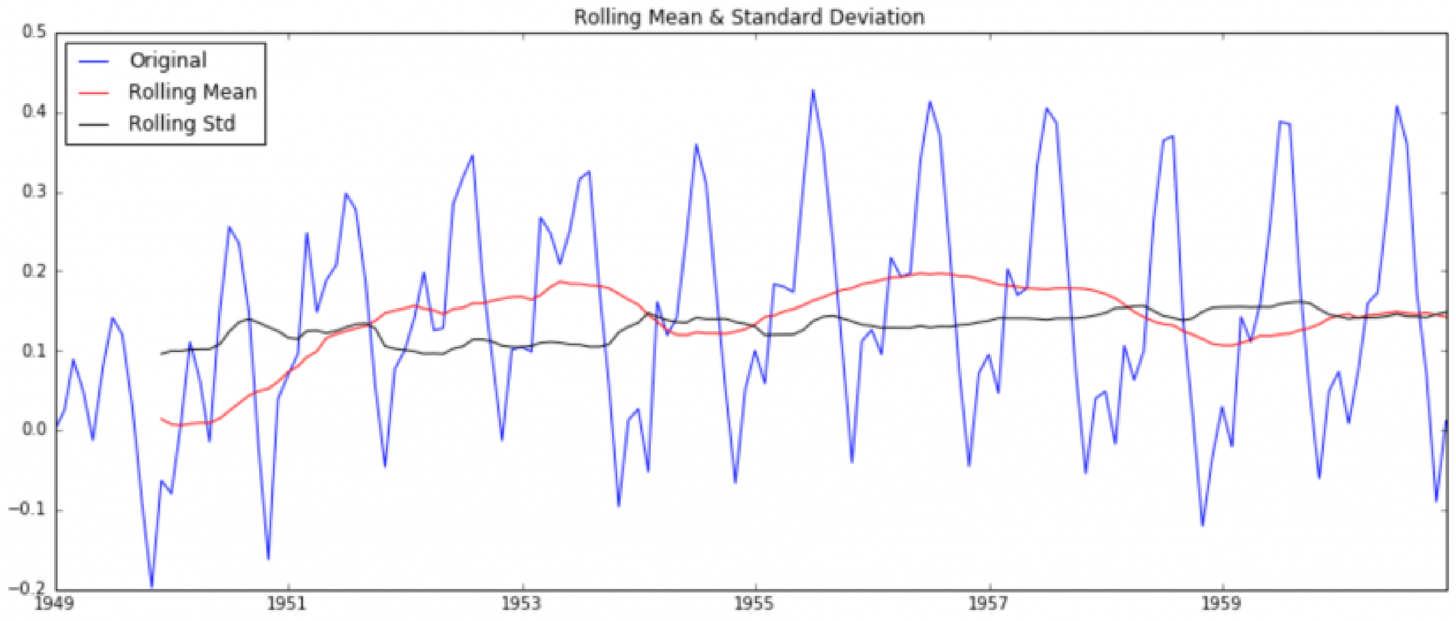
```
expwighted_avg = pd.ewma(ts_log, halflife=12)
plt.plot(ts_log)
plt.plot(expwighted_avg, color='red')
```



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/12.-smooth-2.png)

Note that here the parameter 'halflife' is used to define the amount of exponential decay. This is just an assumption here and would depend largely on the business domain. Other parameters like span and center of mass can also be used to define decay which are discussed in the link shared above. Now, let's remove this from series and check stationarity:

```
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
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/3.-dfuller-smooth-2.png)

This TS has even lesser variations in mean and standard deviation in magnitude. Also, the test statistic is **smaller than the 1% critical value**, which is better than the previous case. Note that in this case there will be no missing values as all values from starting are given weights. So it'll work even with no previous values.

Eliminating Trend and Seasonality

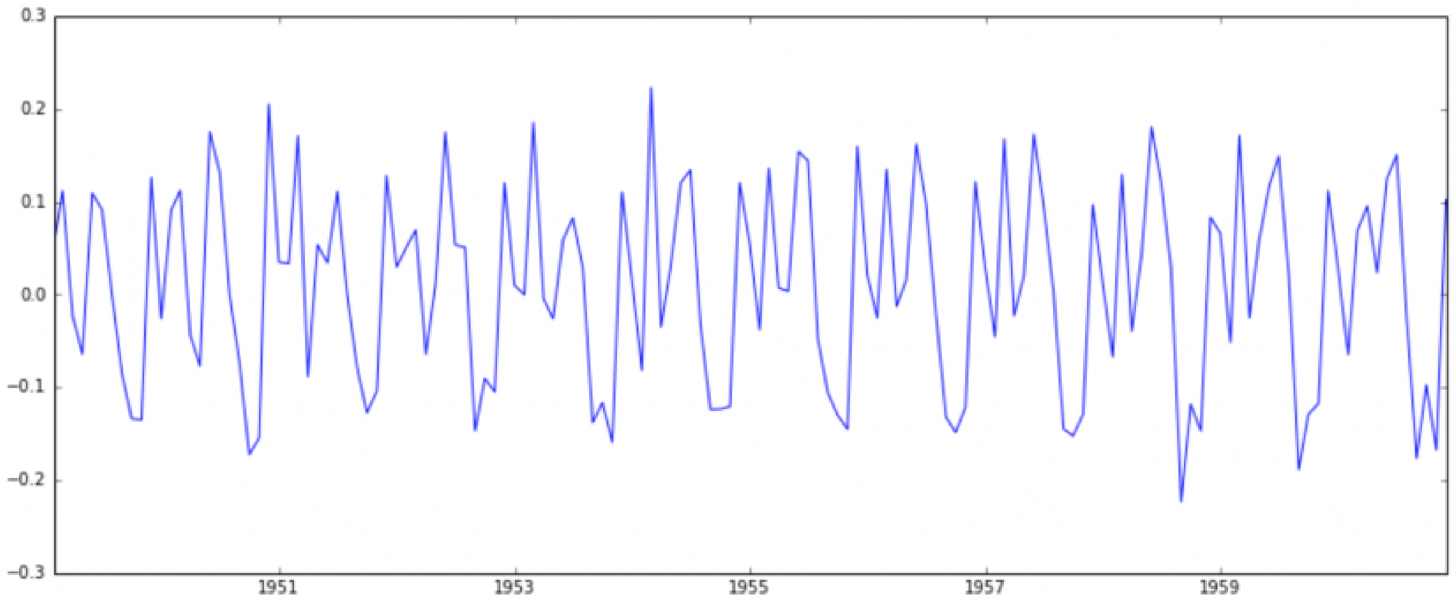
The simple trend reduction techniques discussed before don't work in all cases, particularly the ones with high seasonality. Lets discuss two ways of removing trend and seasonality:

1. **Differencing** – taking the differece with a particular time lag
2. **Decomposition** – modeling both trend and seasonality and removing them from the model.

Differencing

One of the most common methods of dealing with both trend and seasonality is differencing. In this technique, we take the difference of the observation at a particular instant with that at the previous instant. This mostly works well in improving stationarity. First order differencing can be done in Pandas as:

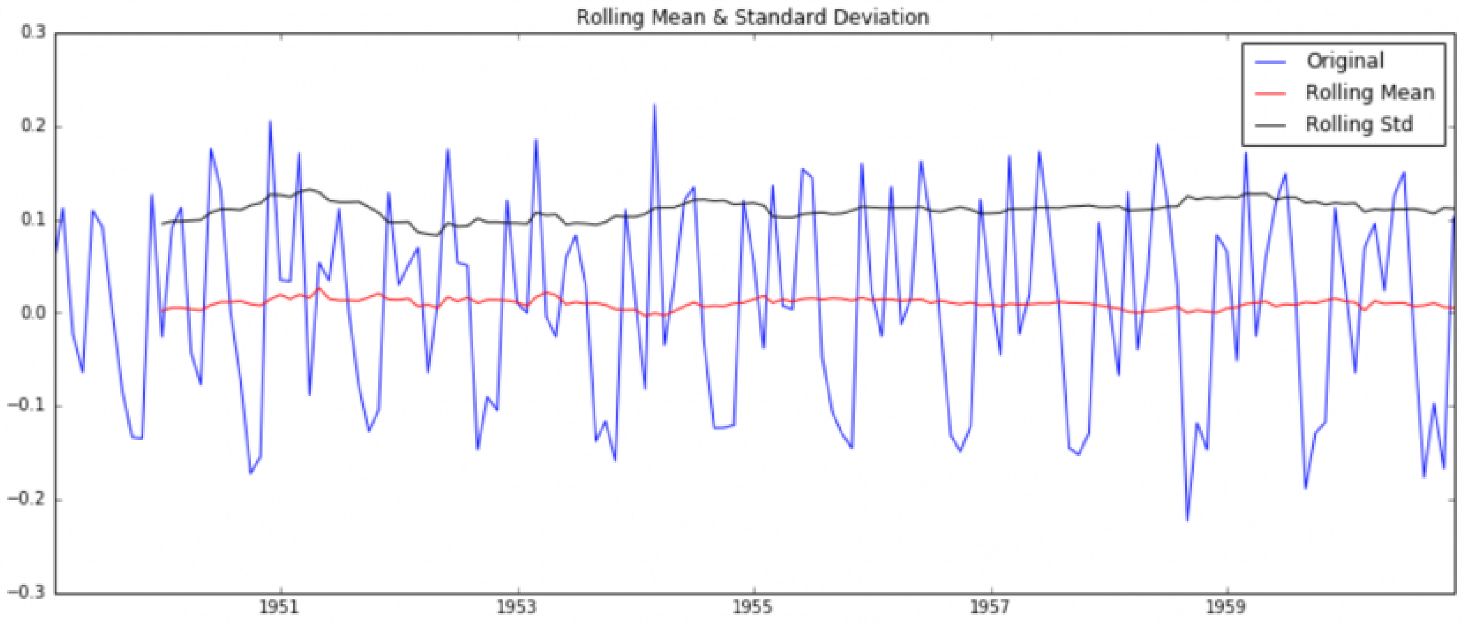
```
ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff)
```

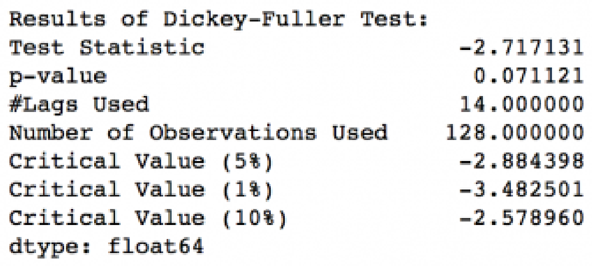


(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/14.-ts-diff.png)

This appears to have reduced trend considerably. Lets verify using our plots:

```
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
```

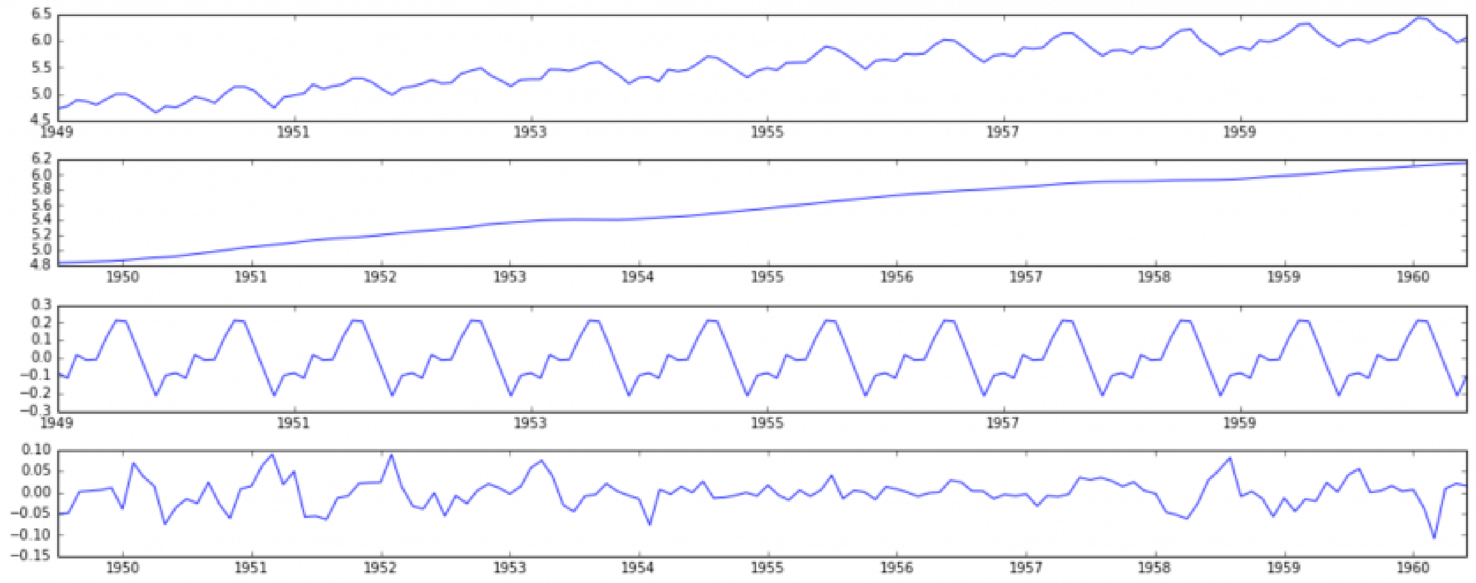
(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/4.-dfuller-diff.png)

We can see that the mean and std variations have small variations with time. Also, the Dickey-Fuller test statistic is **less than the 10% critical value**, thus the TS is stationary with 90% confidence. We can also take second or third order differences which might get even better results in certain applications. I leave it to you to try them out.

Decomposing

In this approach, both trend and seasonality are modeled separately and the remaining part of the series is returned. I'll skip the statistics and come to the results:

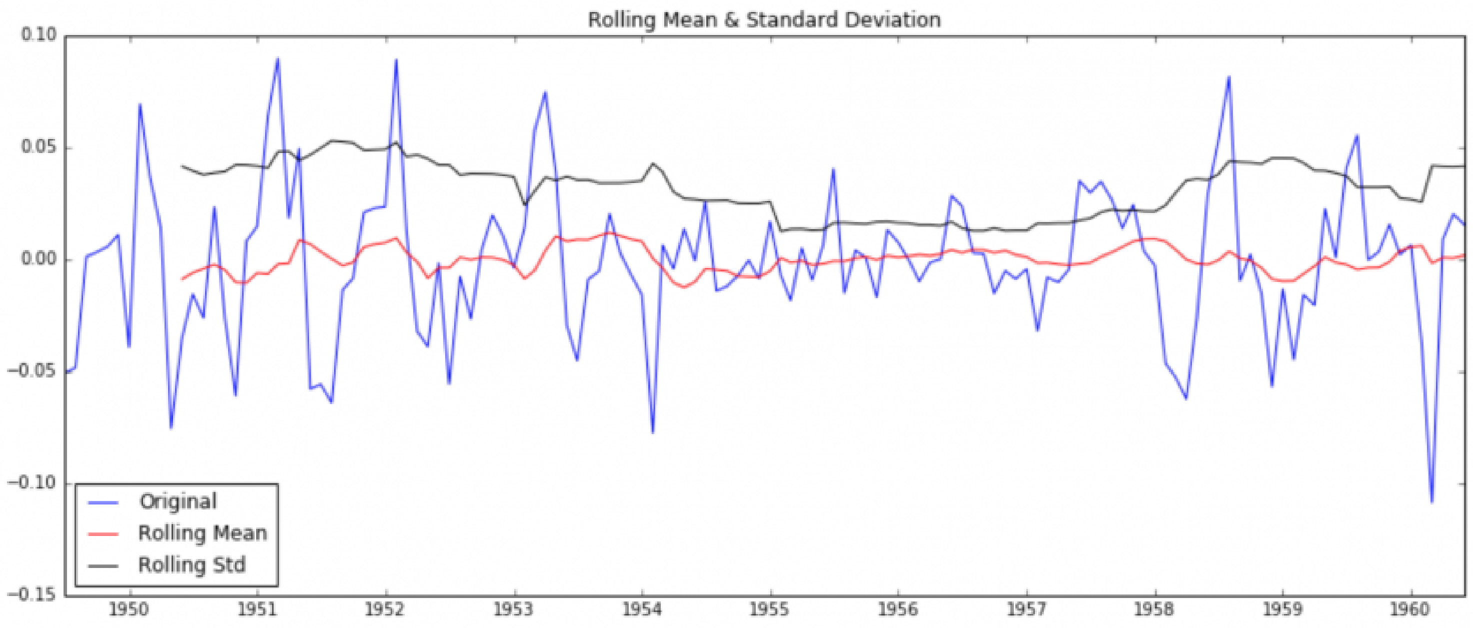
```
from statsmodels.tsa.seasonal import seasonal_decompose

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()
```



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/16.-decompose.png)

Here we can see that the trend, seasonality are separated out from data and we can model the residuals. Lets check stationarity of residuals:

```
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
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/5.-dfuller-decompose.png)

The Dickey-Fuller test statistic is significantly **lower than the 1% critical value**. So this TS is very close to stationary. You can try advanced decomposition techniques as well which can generate better results. Also, you should note that converting the residuals into original values for future data in not very intuitive in this case.

5. Forecasting a Time Series

We saw different techniques and all of them worked reasonably well for making the TS stationary. Lets make model on the TS after differencing as it is a very popular technique. Also, its relatively easier to add noise and seasonality back into predicted residuals in this case. Having performed the trend and seasonality estimation techniques, there can be two situations:

1. A **strictly stationary series** with no dependence among the values. This is the easy case wherein we can model the residuals as white noise. But this is very rare.
2. A series with significant **dependence among values**. In this case we need to use some statistical models like ARIMA to forecast the data.

Let me give you a brief introduction to **ARIMA**. I won't go into the technical details but you should understand these concepts in detail if you wish to apply them more effectively. ARIMA stands for **Auto-Regressive Integrated Moving Averages**. The ARIMA forecasting for a stationary time series is nothing but a linear (like a linear regression) equation. The predictors depend on the parameters (p,d,q) of the ARIMA model:

1. **Number of AR (Auto-Regressive) terms (p):** AR terms are just lags of dependent variable. For instance if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).
2. **Number of MA (Moving Average) terms (q):** MA terms are lagged forecast errors in prediction equation. For instance if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at i^{th} instant and actual value.
3. **Number of Differences (d):** These are the number of nonseasonal differences, i.e. in this case we took the first order difference. So either we can pass that variable and put d=0 or pass the original variable and put d=1. Both will generate same results.

An importance concern here is how to determine the value of 'p' and 'q'. We use two plots to determine these numbers. Lets discuss them first.

1. **Autocorrelation Function (ACF):** It is a measure of the correlation between the the TS with a lagged version of itself. For instance at lag 5, ACF would compare series at time instant 't1'…'t2' with series at instant 't1-5'…'t2-5' (t1-5 and t2 being end points).
2. **Partial Autocorrelation Function (PACF):** This measures the correlation between the TS with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. Eg at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.

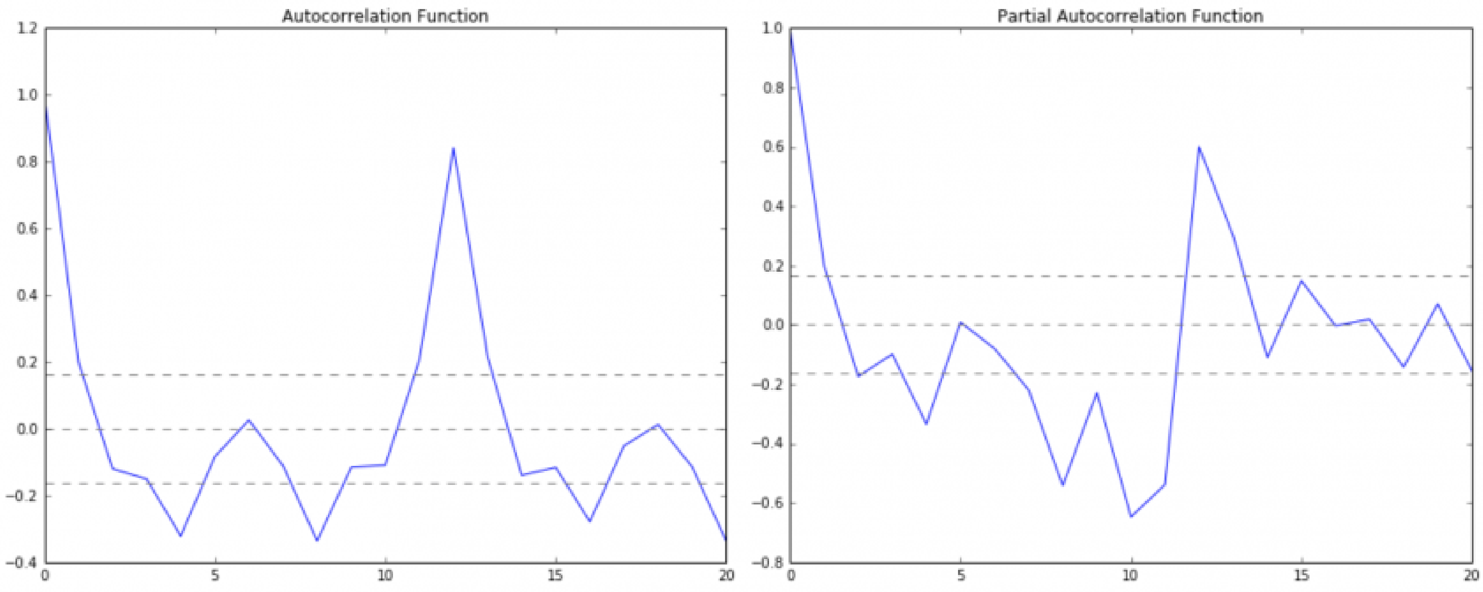
The ACF and PACF plots for the TS after differencing can be plotted as:

```
#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()
```



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/6.-acf-pcf-final.png)

In this plot, the two dotted lines on either sides of 0 are the confidence interevals. These can be used to determine the 'p' and 'q' values as:

1. **p** – The lag value where the **PACF** chart crosses the upper confidence interval for the first time. If you notice closely, in this case p=2.
2. **q** – The lag value where the **ACF** chart crosses the upper confidence interval for the first time. If you notice closely, in this case q=2.

Now, lets make 3 different ARIMA models considering individual as well as combined effects. I will also print the RSS for each. Please note that here RSS is for the values of residuals and not actual series.

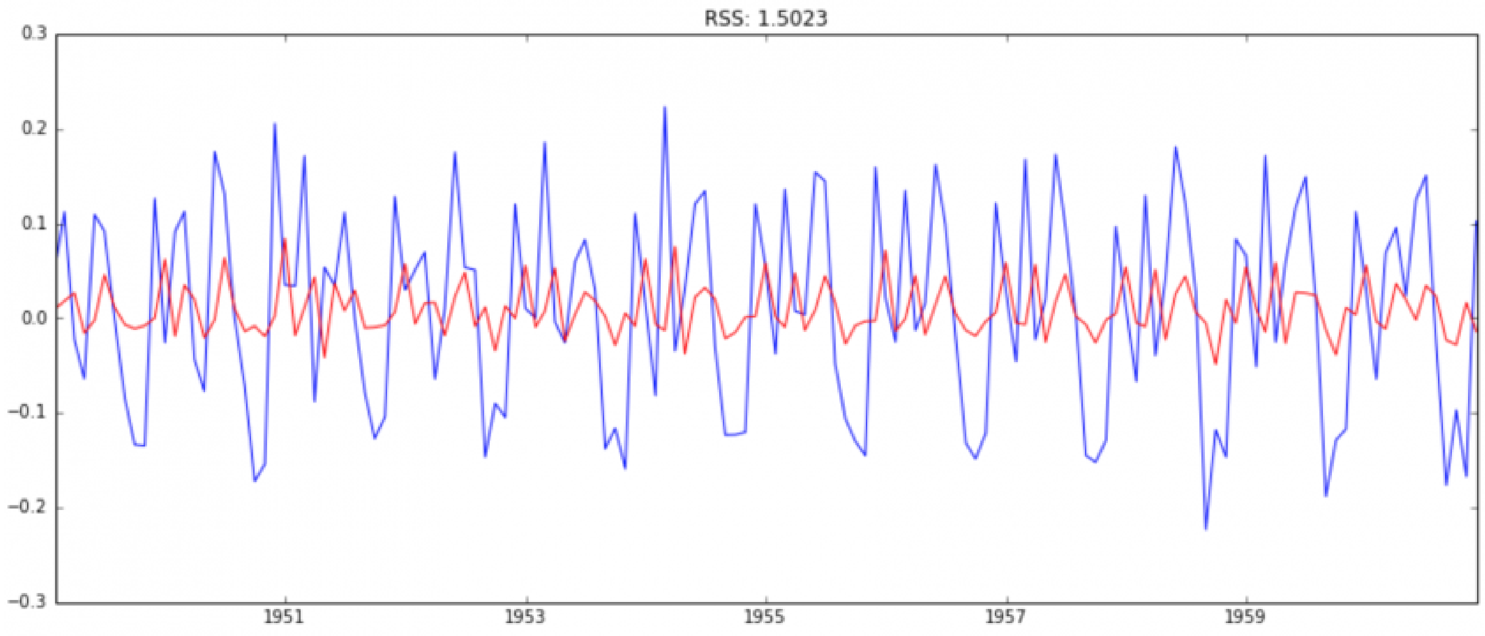
We need to load the ARIMA model first:

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

The p,d,q values can be specified using the order argument of ARIMA which take a tuple (p,d,q). Let model the 3 cases:

AR Model

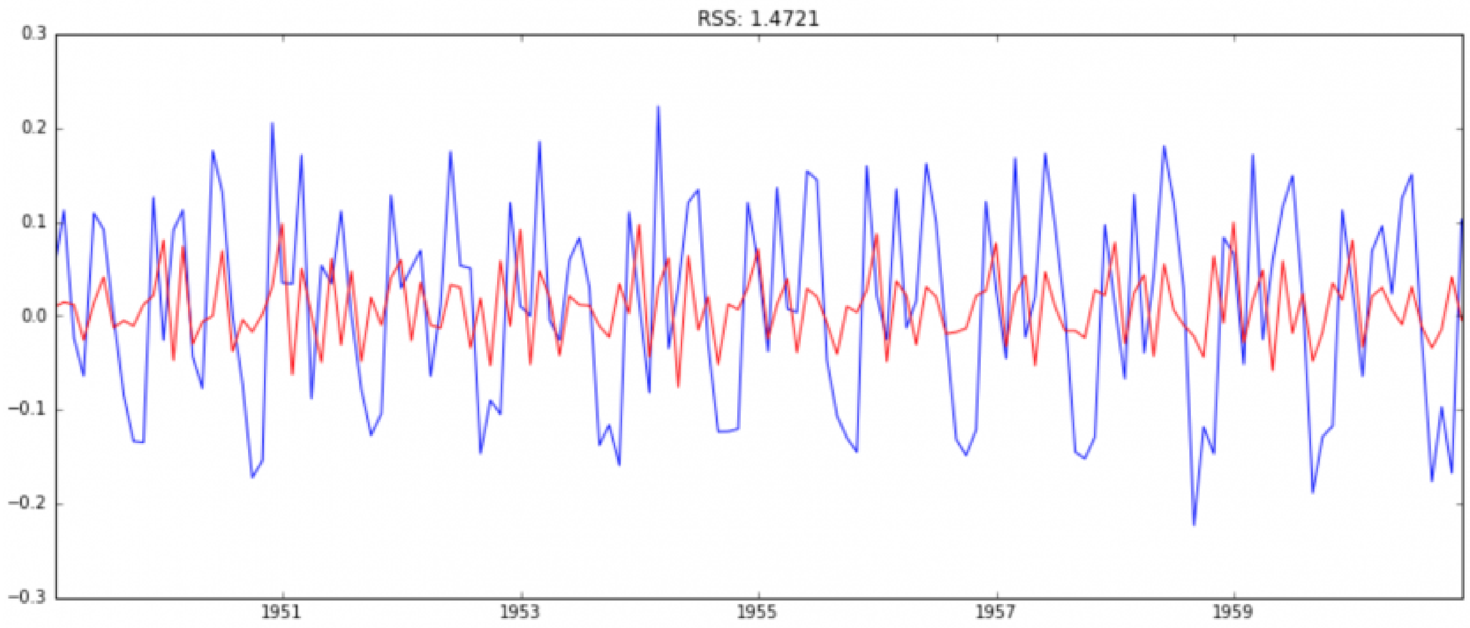
```
model = ARIMA(ts_log, order=(2, 1, 0))
results_AR = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_AR.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_AR.fittedvalues-ts_log_diff)**2))
```



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/18.-model-AR.png)

MA Model

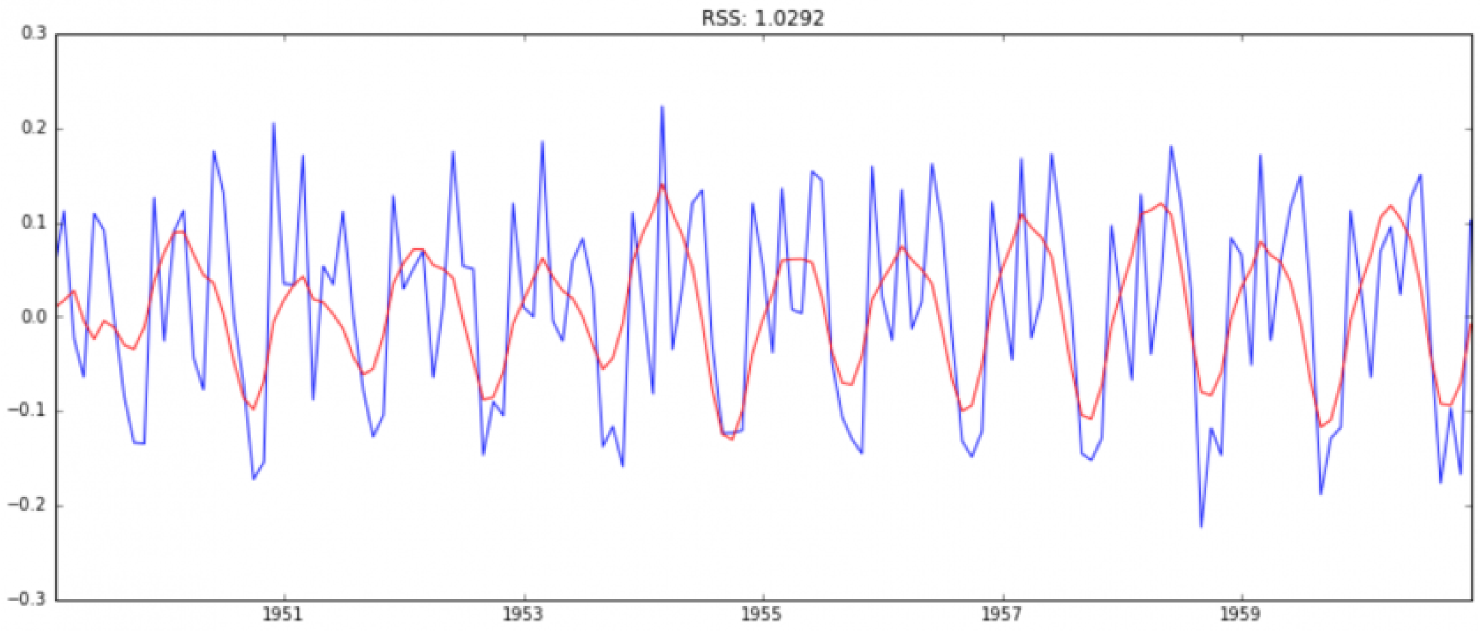
```
model = ARIMA(ts_log, order=(0, 1, 2))
results_MA = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_MA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_MA.fittedvalues-ts_log_diff)**2))
```



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/19.-model-MA.png)

Combined Model

```
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))
```



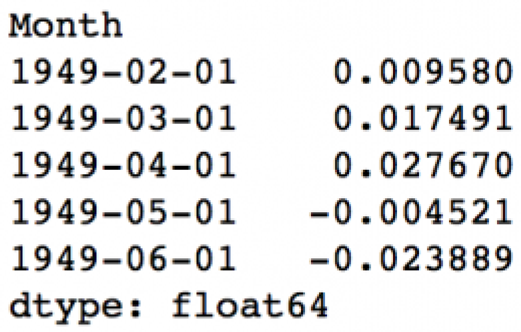
(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/20.-model-both.png)

Here we can see that the AR and MA models have almost the same RSS but combined is significantly better. Now, we are left with 1 last step, i.e. taking these values back to the original scale.

Taking it back to original scale

Since the combined model gave best result, lets scale it back to the original values and see how well it performs there. First step would be to store the predicted results as a separate series and observe it.

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

print predictions_ARIMA_diff.head()
```

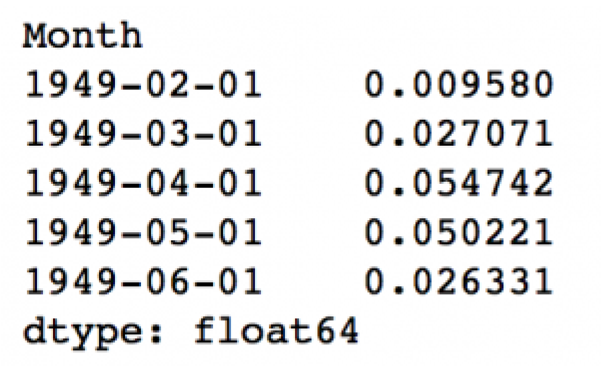


```
Month
1949-02-01     0.009580
1949-03-01     0.017491
1949-04-01     0.027670
1949-05-01    -0.004521
1949-06-01    -0.023889
dtype: float64
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/21.-check-output.png)

Notice that these start from ‘1949-02-01’ and not the first month. Why? This is because we took a lag by 1 and first element doesn’t have anything before it to subtract from. The way to convert the differencing to log scale is to add these differences consecutively to the base number. An easy way to do it is to first determine the cumulative sum at index and then add it to the base number. The cumulative sum can be found as:

```
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()

print predictions_ARIMA_diff_cumsum.head()
```

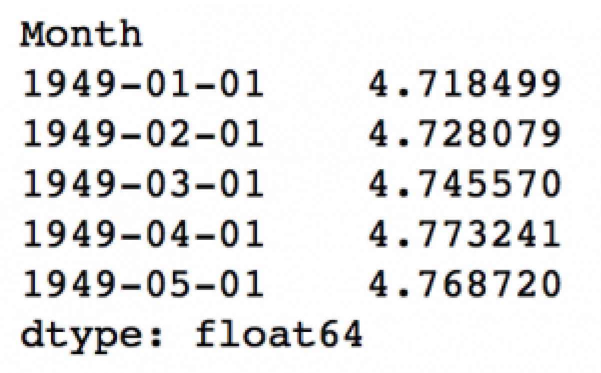


```
Month
1949-02-01    0.009580
1949-03-01    0.027071
1949-04-01    0.054742
1949-05-01    0.050221
1949-06-01    0.026331
dtype: float64
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/22.-cumsum.png)

You can quickly do some back of mind calculations using previous output to check if these are correct. Next we’ve to add them to base number. For this lets create a series with all values as base number and add the differences to it. This can be done as:

```
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()
```

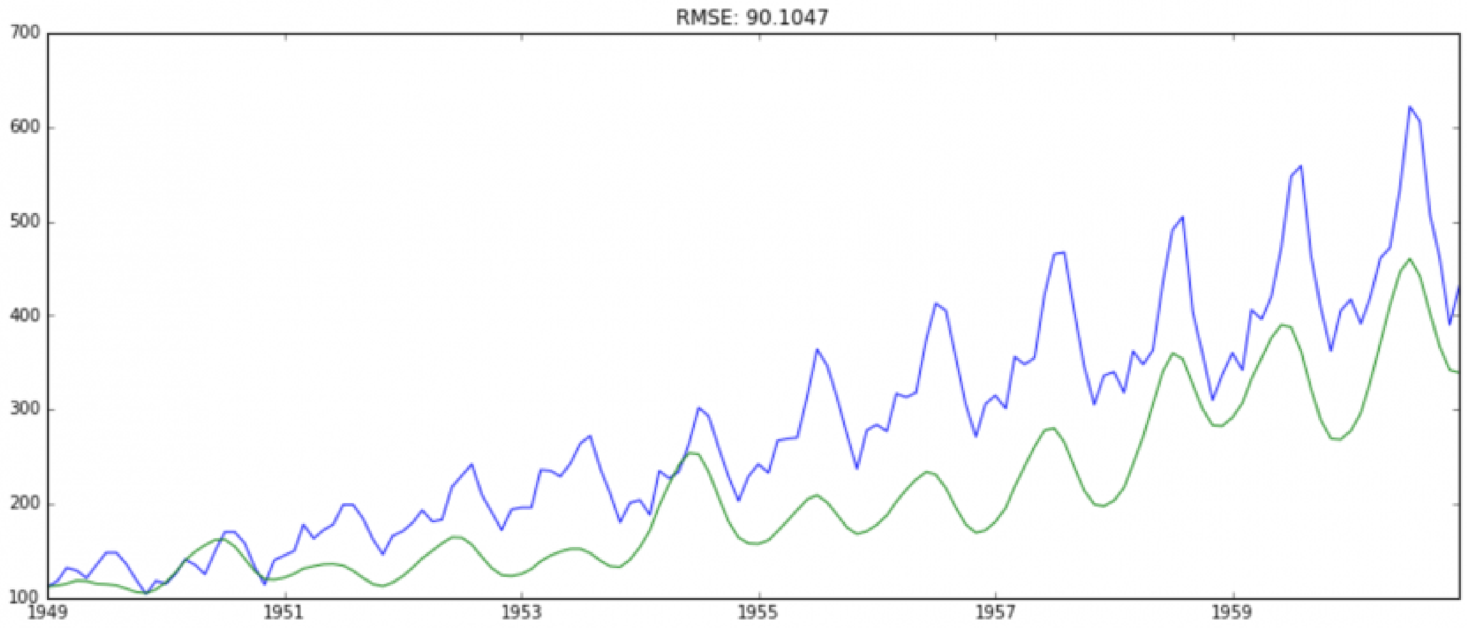


```
Month
1949-01-01    4.718499
1949-02-01    4.728079
1949-03-01    4.745570
1949-04-01    4.773241
1949-05-01    4.768720
dtype: float64
```

(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/23.-add-cumsum.png)

Here the first element is base number itself and from thereon the values cumulatively added. Last step is to take the exponent and compare with the original series.

```
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)))
```



(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/24.-final-plot.png)

Finally we have a forecast at the original scale. Not a very good forecast I would say but you got the idea right? Now, I leave it upto you to refine the methodology further and make a better solution.

End Notes

Through this article I have tried to give you a standard approach for solving time series problem. This couldn't have come at a better time as today is our Mini DataHack (http://datahack.analyticsvidhya.com/contest/mini-datahack) which will challenge you to solve a similar problem. We've covered concepts of stationarity, how to take a time series closer to stationarity and finally forecasting the residuals. It was a long journey and I skipped some statistical details which I encourage you to refer using the suggested material. If you don't want to copy-paste, you can download the iPython notebook with all the codes from my GitHub (https://github.com/aarshayj/Analytics_Vidhya/tree/master/Articles) repository.

I hope this article will help you achieve a good first solution today. All the best guys!

Did you like the article? How helpful was it in the hackathon today? Somethings bothering you which you wish to discuss further? Please feel free to post a comment and I'll be more than happy to discuss.

Note – The discussions of this article are going on at AV's Discuss portal. Join here (https://discuss.analyticsvidhya.com/t/discussions-for-article-a-comprehensive-beginners-guide-to-create-a-time-series-forecast-with-codes-in-python/65783?u=jalfaizy)!

TAGS : ARIMA (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/ARIMA/), FORECASTING ANALYTICS (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/FORECASTING-ANALYTICS/), PANDAS (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PANDAS/), TIME SERIES (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TIME-SERIES/), TIME SERIES ANALYSIS (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TIME-SERIES-ANALYSIS/), TIME SERIES FORECASTING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TIME-SERIES-FORECASTING/)

NEXT ARTICLE

What I learnt about Time Series Analysis in 3 hour Mini DataHack?
(https://www.analyticsvidhya.com/blog/2016/02/hand-learn-time-series-3-hours-mini-datahack/)

•••

PREVIOUS ARTICLE

Mini DataHack and the tactics of the three “Last Man Standing”!
(https://www.analyticsvidhya.com/blog/2016/02/secrets-winners-signature-hackathon-last-man-standing/)

(https://www.analyticsvidhya.com/blog/author/aarshay/)

Aarshay Jain (Https://Www.Analyticsvidhya.Com/Blog/Author/Aarshay/)

Aarshay is a ML enthusiast, pursuing MS in Data Science at Columbia University, graduating in Dec 2017. He is currently exploring the various ML techniques and writes articles for AV to share his knowledge with the community.

(mailto:aarshayjain@gmail.com) (https://in.linkedin.com/in/aarshayjain)

(https://github.com/aarshayj) (aarshay)

RELATED ARTICLES

GUEST BLOG (HTTPS://WWW.ANALYTIC…

(https://www.analyticsvidhya.com/b insight-fighting-cyber-fraud-with-analytics/)

Industry Insight – Fighting Cyber Fraud with Analytics (https://www.analyticsvidhya.com/blog/2016/08/industry-insight-fighting-cyber-fraud-with-analytics/)

TAVISH SRIVASTAVA (HTTPS://WWW.AN…

(https://www.analyticsvidhya.com/b cheering-2014-fifa-wc-winner-twitter/)

Who is the world cheering for? 2014 FIFA WC winner predicted using Twitter feed (in R) (https://www.analyticsvidhya.com/blog/2014/07/world-cheering-2014-

PRANAV DAR (HTTPS://WWW.ANALYTIC…

(https://www.analyticsvidhya.com/b ai-ml-developments-this-week-230418/)

AVBytes: AI & ML Developments this week – Pandas to end Python 2 Support, Intel's Framework-Neutral Library, Google's Cancer Detection Algo, etc. (https://www.analyticsvidhya.com/blog/2018/04/avbytes-ai-ml-developments-this-week-230418/)

KUNAL JAIN (HTTPS://WWW.ANALYTICS…

(https://www.analyticsvidhya.com/b 10-articles-2014/)

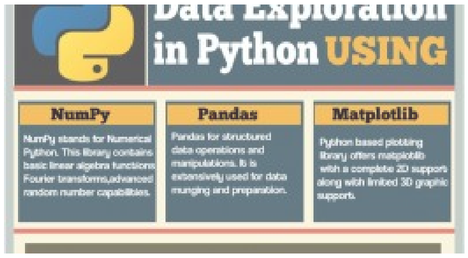
Our top 10 Data Science articles in 2014 (https://www.analyticsvidhya.com/blog/2014/12/top-10-articles-2014/)

KUNAL JAIN (HTTPS://WWW.ANALYTICS…

(https://www.analyticsvidhya.com/b big-data-loss-privacy/)

Boon from big data or loss of privacy? (https://www.analyticsvidhya.com/blog/2014/01/boon-big-data-loss-privacy/)

ANALYTICS VIDHYA CONTENT TEAM (HT…



(https://www.analyticsvidhya.com/b cheat-sheet-data-exploration-python/)

Cheat Sheet for Exploratory Data Analysis in Python (https://www.analyticsvidhya.com/blog/2015/06/infographic-cheat-sheet-data-exploration-python/)

This article is quite old and you might not get a prompt response from the author. We request you to post this comment on Analytics Vidhya's **Discussion portal** (https://discuss.analyticsvidhya.com/) to get your queries resolved

72 COMMENTS

DR.D.K.SAMUEL

February 6, 2016 at 10:02 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105271)

Real thanks for a post which met my need

AARSHAY JAIN

February 6, 2016 at 10:04 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105272)

I'm glad you liked it 🙂

SATISH

February 7, 2016 at 3:17 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105327)

Thanks for great explanation related to timeseries. What is difference between holtwinters and arima forcast?

SHAN

February 8, 2016 at 6:09 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105357)

Holtwinters is double exponential smoothening method. ARIMA, forecasts by identifying p,d,q component of a series. Hope it helps.

AARSHAY JAIN

February 8, 2016 at 6:51 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105358)

To add to Shan, Holtwinters uses a weighted average of past values while ARIMA uses both past values and past errors. You can find more details here:
https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwiUgKXCx-fKAhXIc44KHTTpDyUQFggjMAE&url=http%3A%2F%2Fwww.ons.gov.uk%2Fons%2Fguide-method%2Fukcemga%2Fukcemga-publications%2Fpublications%2Farchive%2Ffrom-holt-winters-to-arima-modelling–measuring-the-impact-on-forecasting-errors-for-components-of-quarterly-estimates-of-public-service-

output.pdf&usg=AFQjCNGmYzfVB-_gdss4LKTGw4VVZgBC_w&sig2=9pnseABiC_4oxC2KnWmHNw&cad=rja (https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwiUgKXCx-fKAhXIc44KHTTpDyUQFggjMAE&url=http%3A%2F%2Fwww.ons.gov.uk%2Fons%2Fguide-method%2Fukcemga%2Fukcemga-publications%2Fpublications%2Farchive%2Ffrom-holt-winters-to-arima-modelling--measuring-the-impact-on-forecasting-errors-for-components-of-quarterly-estimates-of-public-service-output.pdf&usg=AFQjCNGmYzfVB-_gdss4LKTGw4VVZgBC_w&sig2=9pnseABiC_4oxC2KnWmHNw&cad=rja)

TL

April 11, 2016 at 12:59 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109273)

Holt winters (at least the additive model) is a special case of arima model (a seasonal arima model). That would be an arima(p,d,q)(P,D,Q) where the second parentheses contains the seasonal effects. I would additionally recommend checking out any of Rob Hyndman’s work on arima modeling, I find it to be very accessible.

SHAN

February 7, 2016 at 6:12 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105332)

Hi..
Thanks. for an informative article.
I am eager to know on followings :

a) how can we identify what should be nlags value to test with
lag_acf = acf(ts_log_diff, nlags=20)
lag_pacf = pacf(ts_log_diff, nlags=20, method=’ols’)

b) How can we forecast for future time points (say 12 time points ahead).
Can we use followings still ?
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)
ts_log is not available for future points.

c) In one of the article (A Complete Tutorial on Time Series Modeling in R,) referred by you ,

while performing adf says
adf.test(diff(log(AirPassengers)), alternative=”stationary”, k=0)
What is k , and how can we identify the value of k while performing the test..

while performing ARIMA says :
fit <- arima(log(AirPassengers), c(0, 1, 1),seasonal = list(order = c(0, 1, 1), period = 12)))

We can identify the (p,d,q) from ACF PACF plots .
Please explain parameter seasonal = list(order = c(0, 1, 1)
What values should we pass in seasonal parameter and how to identify it.

It will be helpful if you guide on above.. Thanks in anticipation.

AARSHAY JAIN
February 8, 2016 at 6:52 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105359)

Hi Shan,

Thanks for reaching out. Please find my responses below:

a) So the 'nlags' doesn't affect the output values. I just specifies how many values to display. So you can start with a small number and if you don't find the crossing point within that, you can increase maximum upto the number of observations in data.

b) ARIMA has a specific function for forecasting values. The 'results_ARIMA' variable here is of the type 'ARIMAresults' which has a 'predict' function. You can check the details as –
http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARMAResults.predict.html#stat
(http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARMAResults.predict.html#sta
Please feel free to get back to me in case you face challenges in implementing this. You can also start a thread in the discussion forum which will allow more freedom of expression while discussing 🙂

c) I'm not much experienced with R so let me read the code syntax. I'll get back to you on this.

Cheers!

CHIRAG BHATIA
July 15, 2016 at 5:48 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-113494)

Hi Aarshay
I am trying to predict future values on same AirPassenger data but i am not getting correct results. I may miss some parameters while predicting. Please help me. I am stuck from past 2 days. My code is:

```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima_model import ARIMA
import matplotlib.pylab as plt

data_1 = pd.read_csv(‘AirPassengers.csv’)
avg= data_1[‘#Passengers’]
avg=list(avg)
res = pd.Series(avg, index=pd.to_datetime(data_1[‘Month’],format=’%Y-%m’))

ts=np.log(res)
ts_diff = ts – ts.shift()
ts_diff.dropna(inplace=True)
r = ARIMA(ts,(2,1,2))
r = r.fit(disp=-1)
pred = r.predict(start=’1961-01′,end=’1970-01′)
dates = pd.date_range(‘1961-01′,’1970-01′,freq=’M’)
# print dates
predictions_ARIMA_diff = pd.Series(pred, copy=True)
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
predictions_ARIMA_log = pd.Series(ts.ix[0])
predictions_ARIMA_log=predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill_value=0)
predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(res)
plt.plot(predictions_ARIMA)
# plt.title(‘RMSE: %.4f’% np.sqrt(sum((predictions_ARIMA-ts1)**2)/len(ts)))
plt.show()

print predictions_ARIMA.head()
print ts.head()
```

AARSHAY JAIN

February 8, 2016 at 12:32 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105373)

Hi Shan,

I guess you have started a separate discussion thread for your query ‘c’. Lets continue the discussion there. For others who’re reading this and interested in exploring further, please check out this link:

http://discuss.analyticsvidhya.com/t/seasonal-parameter-in-arima-and-adf-test/7385/1 (http://discuss.analyticsvidhya.com/t/seasonal-parameter-in-arima-and-adf-test/7385/1)

Cheers!

AMITSETHIA

February 13, 2016 at 12:23 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105619)

Thanks Aarshay for this write up. It is also recommended to not to go for combined models as p & q used together will nullify their impact on the model, hence, it is either a moving average or auto correlation along with differences, but here combined model has given the best results. Can you please correct my understanding around combined models.

AARSHAY JAIN

February 13, 2016 at 12:50 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-105623)

I haven’t read that p & q should not be combined. It’s actually appears counter intuitive because if this was the case then ARIMA should not exist in the first place. Can you throw some light on why do you believe that they cancel out the effect of one another?

AAYUSH KUMAR SINGHA

February 29, 2016 at 10:58 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-106400)

Hi!
The article is the best available on Time Series with Python with great external links too for those who want to understand the stat behind also.
I would like to request to please extend this article to predict out-of-sample data range also with different models to depict the better ones as you did for eliminating trend (taking rolling average and ewma).
That will make it all fully fledged time-series article.
Thanks in advance.

AARSHAY JAIN

March 1, 2016 at 8:46 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-106425)

Hi Ayush!
Thanks for your valuable feedback. Yes I think that component is necessary. But instead of extending this article, I'll probably write a separate post taking another case study. I'm a bit crunched for bandwidth but you can expect it sometime in this month. Stay tuned!

MICHAEL
March 13, 2016 at 5:49 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-107267)

Thanks for the excellent article. I have 2 clarifications
1) In the Estimating & Eliminating Trend step, I have negative numbers. Could you please tell me what transformations could I apply. Log and Sqrt returns NAN?
2) Also, test_stationarity(ts_log_decompose,nlags=10) while executing specifies nlags not defined.

Thanks in advance.

AARSHAY JAIN
March 13, 2016 at 6:12 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-107270)

Hi Michael,

Thanks for reaching out. Regarding your queries:
1. You can try scaling up your values and then applying transformations. Also, you might want to check if log transformation is actually required in your case. You can try a cube root as well.
2. Please remove the nlags argument and then run the code. I've updated the code above as well.

WILL WELCH
March 14, 2016 at 1:34 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-107299)

Nice article, you rarely see this range of models discussed in one place
and in such a hands-on way.

For anyone doing seasonal decomposition in Python, I'd like to shamelessly
plug my `seasonal` package (PyPI or https://github.com/welch/seasonal (https://github.com/welch/seasonal)) in
addition to statsmodels seasonal_decompose. `seasonal` offers some richer
and more robust detrending possibilities, and will also estimate your model's

periodicity for you (convenient in a dev-ops setting with thousands of streams at hand). It also includes a robust periodogram for visualizing the periodicities in your data.

AARSHAY JAIN

March 14, 2016 at 5:58 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-107313)

Thanks Will for sharing your library. It'll be helpful for everyone.

ALON STAR

March 24, 2016 at 7:38 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-108176)

Can you please explain what is the dftest[0:4]?

AARSHAY JAIN

March 25, 2016 at 6:26 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-108224)

the adfuller function returns a list with many values. I'm picking the first 4 using [0:4]. I've used the 5th value separately. You might want to print the dftest variable and you'll know.

LBERT

April 10, 2016 at 5:53 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109243)

Can we use this method for decimal data? Why the program gave me an error of "ValueError: You must specify a freq or x must be a pandas object with a timeseries index"?

AARSHAY JAIN

April 11, 2016 at 6:59 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109280)

I don't think it is a decimal error. Please check whether your index is a timeseries object.

CLOGA (HTTP://WWW.CLOGA,INFO)
April 14, 2016 at 9:54 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109445)

Hi Aarshay Jain,

One more question,
When you fit model , you use ts_log as sampel, ie. model = ARIMA(ts_log, order=(2, 1, 2)) , but when you predict you use predict value as diff value : predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True), is results_ARIMA.fittedvalues return log value or diff value?

Thank you for your time.

AARSHAY JAIN
April 14, 2016 at 10:53 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109454)

actually while calling ARIMA I have set order = (2,1,2). Here the middle argument 1 means that ARIMA will automatically take a difference of 1 while making predictions.

CLOGA (HTTP://WWW.CLOGA,INFO)
April 15, 2016 at 2:31 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109489)

Got it thank you!

AYODEJI OLUFEMI AYOTUNDE
April 19, 2016 at 12:44 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109660)

What an excellent article on Time Series, more grease to your elbow. But the question is, is this method a package of analyzing Time Series related data or what? And can’t we do the same on SPSS and have the same simple method as this? However, I have to commend you a lot for this wonderful presentation. God will continue to increase your knowledge.

AARSHAY JAIN

April 19, 2016 at 5:09 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109664)

Thanks Ayodeji!
I'm not sure about SPSS and sorry I didn't get your questions – what do you mean "is this method a package of analyzing Time Series related data"? Please elaborate.

ANDREW
April 22, 2016 at 4:47 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109830)

Hi Aarshay Jain

I've tried the dickey-fuller test code with different dataset and then an error shown up like this:

ValueError: too many values to unpack

please give an advice

thank you

AARSHAY JAIN
April 25, 2016 at 5:50 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-110007)

please share the code..

DENNIS
May 27, 2016 at 1:41 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111506)

Andrew, your are probably passing in a dataframe instead of a series, in the code Aarshay wrote up for the dftest.
Specifically here: dftest = adfuller(timeseries.unstack(), autolag='AIC')
note the .unstack() that I added — transforming the df into a series — when I also encountered the same error.

TANVIR

April 23, 2016 at 8:10 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-109922)

Hello,
I am struggling on a question.
Here, forecast is sowing upto 1960-12-01. Now based on the current measure, I want to forecast the upcoming years for example 1961-01-01 to 1965-12-01.
How can I do this ?

ANIRBAN DHAR
May 3, 2016 at 6:45 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-110390)

Thanks Aarshay for this detailed and illustrative posts.

one concern, the decomposition is not working for 12 months [of a single year] data.
eg. in AirPassengers.csv if I take only the records of 1949 it fails to decompose giving below error:

```
File “C:\anirban\install\Anaconda3\lib\site-packages\statsmodels\tsa\seasonal.
py”, line 88, in seasonal_decompose
trend = convolution_filter(x, filt)
File “C:\anirban\install\Anaconda3\lib\site-packages\statsmodels\tsa\filters\f
iltertools.py”, line 289, in convolution_filter
result = signal.convolve(x, filt, mode=’valid’)
File “C:\anirban\install\Anaconda3\lib\site-packages\scipy\signal\signaltools.
py”, line 470, in convolve
return correlate(volume, kernel[slice_obj], mode)
File “C:\anirban\install\Anaconda3\lib\site-packages\scipy\signal\signaltools.
py”, line 160, in correlate
_check_valid_mode_shapes(in1.shape, in2.shape)
File “C:\anirban\install\Anaconda3\lib\site-packages\scipy\signal\signaltools.
py”, line 72, in _check_valid_mode_shapes
“in1 should have at least as many items as in2 in ”
ValueError: in1 should have at least as many items as in2 in every dimension for
‘valid’ mode.
```

I think this is somehow related to the filter but not able to nail it since I am too novice.
Please note – the code is exact replica of yours

Any help will be appreciated, Thanks again 🙂

BAWASIR KI DAWA (HTTP://RAJHERBALS.COM/ARSHOHER-BAWASIR-KI-DAWA.HTML)
May 6, 2016 at 12:47 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-110515)

Great blog post, thanks for sharing this post.
bawaseer ka ayurvedic ilaj (http://rajherbals.com/arshoher-bawasir-ki-dawa.html)

AARSHAY JAIN
May 6, 2016 at 5:56 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-110533)

You are welcome 🙂

SK
May 16, 2016 at 7:06 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111065)

How to do the prediction for the future?

start = len(ts)
end = len(ts)+14
y_forecast = np.exp(results_ARIMA.predict(start, end))
This does not provide good results.
Would you please expand your code and description to include one month ahead forecast?

PRAKHA SHRIVASTAVA
May 19, 2016 at 10:59 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111179)

Hi
Thank You for sharing this post. i have one question: time series in pandas does only work with csv file because i want to forecast my database values for next 6 months. I did connect the python with mySQl database. i.e i have data in python with dataset not in csv file.So how can i used time series forecasting method. If you provide me code it will be huge help for me.

PRAKHAR SHRIVASTAVA

May 25, 2016 at 12:45 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111437)

this problem is done…. by using data = pd.read_sql_query(cur,con).

PRAKHAR SHRIVASTAVA

May 25, 2016 at 12:32 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111435)

In Dicket-fuller test my came Results of Dickey-Fuller Test:
Test Statistic -2.287864
p-value 0.175912
#Lags Used 11.000000
Number of Observations Used 215.000000
Critical Value (1%) -3.461136
Critical Value (10%) -2.573986
Critical Value (5%) -2.875079
dtype: float64
My p value is so less? Its means my data is not normal ox not not suited to this model?

PRAKHAR SHRIVASTAVA

May 27, 2016 at 9:24 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111519)

This problem is also done.

SATYA CHANDU

July 1, 2016 at 12:31 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-112881)

If test static (-2.287864) is greater than critical value (-3.46, -2.57, -2.87) then we can't reject the null hypothesis, the series is stationary. That said it is still non-stationary. If you increase the i value in ARIMA model, perhaps above condition may meet and you may get the good forecast values.

DENNIS

May 27, 2016 at 1:43 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111507)

Hi Aarshay,

I really enjoyed this but running on python 3, I've encountered a couple errors on the last portion.
"plt.title('RSS: %.4f'% sum((results_MA.fittedvalues-ts_log_diff)**2))

pandas\tslib.pyx in pandas.tslib.Timestamp.__radd__ (pandas\tslib.c:14048)()

pandas\tslib.pyx in pandas.tslib._Timestamp.__add__ (pandas\tslib.c:19022)()

ValueError: Cannot add integral value to Timestamp without offset.'
Googling around it seems that its a bug in statsmodel but if I was wondering perhaps if you or someone else ported it to python 3?

Thanks

PRAKHAR SHRIVASTAVA
May 27, 2016 at 9:23 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111518)

Thank you for this post. I find error in model = ARIMA(ts_log, order=(2, 1, 0))
and i unable to find the error.Please me

PRAKHAR SHRIVASTAVA
June 1, 2016 at 2:29 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111696)

thank you for this example. i have one problem. When i tried to put model in my program its said
ValueError: Given a pandas object and the index does not contain dates but in my dataset date is there.
data = pd.read_sql_query(cur,con, index_col='datum', coerce_float=True, params=None)
I dont know what is problem with this?

SEAN STANISLAW
June 3, 2016 at 5:08 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-111794)

I still prefer gretl for building time series and econometric models easy to use its an open source just download and go

---

YUER

June 10, 2016 at 1:09 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-112070)

Hi, thank you for your sharing.
I am using this model to predict play number of a song.
When I input data with 'data = pd.read_csv('00play.csv', parse_dates='data', index_col='data',date_parser=dateparse)'
There are some error 'TypeError: Only booleans, lists, and dictionaries are accepted for the 'parse_dates' parameter'
if I delete this parameter parse_dates, is there any influence?
Using data without parameter parse_dates, when making seasonal_decompose, another error is 'ValueError: freq D not understood. Please report if you think this in error.'
please give an advice
thank you

JITENDRA

June 12, 2016 at 4:00 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-112152)

hi can you give me an idea in case of multiple time series forecasting

---

JIE

June 19, 2016 at 3:59 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-112372)

really liked this post. Thank you very much for sharing.

BOM

June 23, 2016 at 5:09 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-112604)

How to deal with the tendency of irregular time series data(data with different time interval)?

SATYA CHANDU

June 27, 2016 at 10:58 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-112764)

Hi,

This is a very good blog and very useful. I could follow the entire process. But I did not understand how to forecast for next 12 months from the last value. In the current case the last value is 1960-12, I need to forecast till 1961-12 (12 values). How can I do that in the following code? It would be great if you kindly add that process and update this article.

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)

AADITYA

June 28, 2016 at 7:23 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-112783)

Hi Aarshay,

First of all thanks for this brilliant post.

I am following a similar approach for forecasting a minute's data using previous hours data. I am using forecast function in statsmodels along with ARIMA model. Calculating the P, Q and D using the approach you mentioned in your post.

However, I am facing a few problems:

1. At times the ARIMA throws an error for AR or MA parameter .
2. ARIMA in python takes a lot of time. Similar code in R takes less than 30 minutes for forecasting a months data. Am I missing something or ARIMA in python is inherently slow?
3. I get MLE not converging Warning almost every-time, why is that so.
4. ARIMA does not allow D value more than two, however, at times adfuller results in d value more than two. What should be done in this case.

Looking forward to your suggestions.

Thanks,
Aaditya

SATYA CHANDU

June 28, 2016 at 4:21 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-112798)

Hi Arshay,

This is very useful article. I got a small doubt about forecasting values. How can I get the forecast values from the following code? Suppose I want to print for next 1 year, how can I do that?

Thanks,
Satya

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)

SHANTANU SAHA

July 5, 2016 at 7:07 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-113125)

Thank you for such a detailed post. I was wondering what if the data was in Country-level? How can we deal with such time series data then?

EVELYN

July 8, 2016 at 5:50 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-113228)

For this code line:
data = pd.read_csv('AirPassengers.csv', parse_dates='Month', index_col='Month',date_parser=dateparse)
Does it work? I got error message in my Anaconda Python 2.7 because Python can't identify 'Month' as a list of Month column value for parameter parse_dates, so I changed to ['Month'], It works.
Could anyone confirm it in Python 3? Thanks.

MICHAEL FRANCIS

July 11, 2016 at 4:44 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-113333)

“ValueError: Cannot add integral value to Timestamp without offset.” i keep getting this error whenever I use the ARIMA function and I was wondering if you could tell me what this means and how i could fix it. Im using the same data and steps as the example above.

FLORENT

July 18, 2016 at 4:58 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-113624)

Thanks for this great article, it greatly helped me get started with time series forecasting.

What would be the additional steps if you wanted to make a more accurate forecast?

ABS

July 20, 2016 at 9:22 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-113719)

Really nice post,

My question is how would different would the second part of the problem be if you were to use decomposing instead of differecing for forecasting the time-series?

SHREYAK TIWARI

July 20, 2016 at 11:53 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-113723)

Can someone please explain while creating ARIMA models you are using ts_log (just a log time series) but while calculating RSS you are using ts_log_diff . Am i missing something here ?

MAYANK SATNALIKA

July 26, 2016 at 7:03 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-114036)

Hey I’m a newbie in machine learning and wanted sort out an issue: What type of problems can be classified under time forecasting problem. Many tutorials begin with predicting stock prices for next few days, so is it a time forecast problem. Also is the Bike sharing Demand question from Kaggle a part of time forecasting question as we are given the demand for some dates and we need to predict demand for upcoming days.

BHUVANESHWARAN
August 17, 2016 at 7:22 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-114897)

How to select which model is better one for our data? Is there any parameters in data to select models ?

JARAD
August 25, 2016 at 4:15 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-115111)

This is literally the BEST article I've ever seen on time-series analysis with Python. Very well explained. I wish the statsmodels documentation was this good (they give you the tools but don't show you how to use them!).

I am very confused about ACF and PACF and how to read the charts to determine the proper P an Q. You concluded that p and q are both 2 and you mention "upper confidence level". I don't see the lines crossing the upper-confidence level dashed line at point 2 in ACF or PACF. Is this a typo?

If not a typo, can you explain?

JARAD
August 26, 2016 at 6:22 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-115149)

I wonder what your thoughts are on doing a decomposition, then performing ARIMA forecasting on each component (trend, seasonality, residual), then re-scaling back. Is this a sound method/approach? I did this and the prediction line looks like what I'd expect. I'm just wondering if this is a common practice.

ANIL
August 29, 2016 at 7:55 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-115299)

Hi Aarshey.. great article. I have tested the code and working fine, however, I am not getting the years in X axis, I tried different date parse methods, but no luck. How did you get year values in X axis where as parse method converting Month column as string in %Y-%m-%d format?

YASSIR

September 2, 2016 at 3:00 pm (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-115462)

I got confused on many points: 1- we do many transformations to get stationarity data and every transformation we get data with good stationarity and on the example, you got the best stationary after applying the Decomposing, then why did you use the ts_log_diff and ts_log data with ACF,PACF and ARIMA instead of using the Decomposing data !? 2- I did see many styles for ACF and PACF one like continuous graph and another one like pins, which one I should go for it? 3- what is the best and easiest way to detect AR and MA by ACF and PACF? some tutorials mention about every ARIMA model has a special ACF and PACF pattern and others mention about the intersection between the lags and the confidence upper line! 4-is there any way to automate the step of getting the AR and MA instead of trying to investigate the ACF and PACF plots?

ALEX DEBIE

September 21, 2016 at 12:59 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-116262)

thanks alot for the information, i learned a ton. Im just a little confused now that i have this model how to use it to predict the next point in time

DWITI BASU

September 26, 2016 at 6:31 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-116483)

Hi I am getting this error when I am writing the following codes, can anyone help?

date1= lambda dates: pd.datetime.strptime(dates, ‘%Y-%m’)
dataset= pd.read_csv(‘AirPassangers.csv’, parse_dates=’Month’, index_col=’Month’,date_parser=date1)

This is what I am getting:
========
date1= lambda dates: pd.datetime.strptime(dates, ‘%Y-%m-%d’)

dataset= pd.read_csv(‘AirPassangers.csv’, parse_dates=’Month’, index_col=’Month’,date_parser=date1)

Traceback (most recent call last):

File “”, line 1, in
dataset= pd.read_csv(‘AirPassangers.csv’, parse_dates=’Month’, index_col=’Month’,date_parser=date1)

```
File "C:\Users\dwiti.b\AppData\Local\Continuum\Anaconda2\lib\site-packages\pandas\io\parsers.py", line 562, in parser_f
return _read(filepath_or_buffer, kwds)

File "C:\Users\dwiti.b\AppData\Local\Continuum\Anaconda2\lib\site-packages\pandas\io\parsers.py", line 315, in _read
parser = TextFileReader(filepath_or_buffer, **kwds)

File "C:\Users\dwiti.b\AppData\Local\Continuum\Anaconda2\lib\site-packages\pandas\io\parsers.py", line 645, in __init__
self._make_engine(self.engine)

File "C:\Users\dwiti.b\AppData\Local\Continuum\Anaconda2\lib\site-packages\pandas\io\parsers.py", line 799, in _make_engine
self._engine = CParserWrapper(self.f, **self.options)

File "C:\Users\dwiti.b\AppData\Local\Continuum\Anaconda2\lib\site-packages\pandas\io\parsers.py", line 1202, in __init__
ParserBase.__init__(self, kwds)

File "C:\Users\dwiti.b\AppData\Local\Continuum\Anaconda2\lib\site-packages\pandas\io\parsers.py", line 893, in __init__
kwds.pop('parse_dates', False))

File "C:\Users\dwiti.b\AppData\Local\Continuum\Anaconda2\lib\site-packages\pandas\io\parsers.py", line 873, in _validate_parse_dates_arg
raise TypeError(msg)

TypeError: Only booleans, lists, and dictionaries are accepted for the 'parse_dates' parameter
```

DENNIS

October 3, 2016 at 10:42 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-116715)

thank you for this post. however do you have any tutorials on stock price prediction using artificial neural networks?

LEO

October 4, 2016 at 5:09 am (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#comment-116749)

This is not a complete guide. This can be something to get you started. Time series analysis is not that limited.

DATAHACK SUMMIT 2018

(https://www.analyticsvidhya.com/datahack-summit-2018/?utm_source=AVbannerdhslong)

POPULAR POSTS

24 Ultimate Data Science Projects To Boost Your Knowledge and Skills (& can be accessed freely) (https://www.analyticsvidhya.com/blog/2018/05/24-ultimate-data-science-projects-to-boost-your-knowledge-and-skills/)

Essentials of Machine Learning Algorithms (with Python and R Codes) (https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/)

A Complete Tutorial to Learn Data Science with Python from Scratch (https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/)

7 Types of Regression Techniques you should know! (https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/)

Understanding Support Vector Machine algorithm from examples (along with code) (https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)

6 Easy Steps to Learn Naive Bayes Algorithm (with codes in Python and R) (https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/)

A comprehensive beginner’s guide to create a Time Series Forecast (with Codes in Python) (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/)

20 Challenging Job Interview Puzzles which every analyst should solve atleast once (https://www.analyticsvidhya.com/blog/2016/07/20-challenging-job-interview-puzzles-which-every-analyst-solve-atleast/)

RECENT POSTS

An Intuitive Guide to Interpret a Random Forest Model using fastai library (Machine Learning for Programmers – Part 2) (https://www.analyticsvidhya.com/blog/2018/10/interpret-random-forest-model-machine-learning-programmers/)

OCTOBER 29, 2018

MADRaS: A Multi-Agent DRiving Simulator for Autonomous Driving Research (https://www.analyticsvidhya.com/blog/2018/10/madras-multi-agent-driving-simulator/)

OCTOBER 29, 2018

A Computer Vision Approach to Hand Gesture Recognition (https://www.analyticsvidhya.com/blog/2018/10/computer-vision-approach-hand-gesture-recognition/)

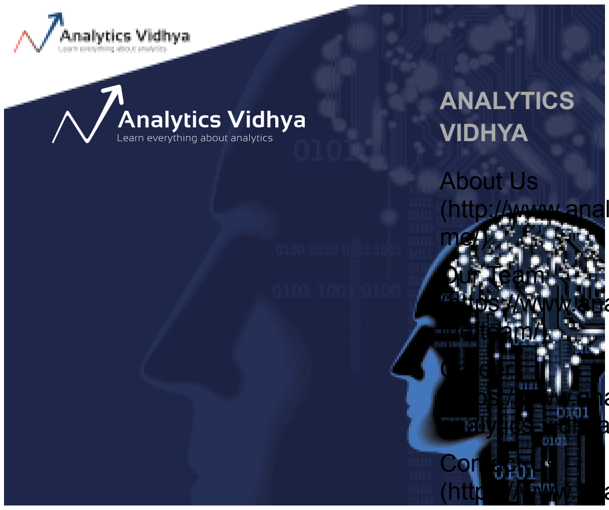
OCTOBER 29, 2018

Stock Prices Prediction Using Machine Learning and Deep Learning Techniques (with Python codes) (https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/)

OCTOBER 25, 2018

(http://www.edvancer.in/certified-data-scientist-with-python-course?utm_source=AV&utm_medium=AVads&utm_campaign=AVadsnonfc&utm_content=pythonavad)

(https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+DS101+2018T2/about?utm_source=AVStickybanner1)



v1:AnalyticsVidhya+CVDL101+CVDL101_T1/about?utm_source=CV101AVBlogBanner&utm_medium=Stickybanner2utm_campaign=CV101banner)

ANALYTICS VIDHYA

About Us (http://www.analyticsvidhya.com/about-me/)

Our Team (https://www.analyticsvidhya.com/about-me/team/)

Careers (https://www.analyticsvidhya.com/careers/analytics-vidhya/)

Contact us (https://www.analyticsvidhya.com/contact/)

Write for us (https://www.analyticsvidhya.com/about-me/write/)

DATA SCIENTISTS

Blog (https://www.analyticsvidhya.com/blog)

Hackathon (https://datahack.analyticsvidhya.com/)

Discussions (https://discuss.analyticsvidhya.com/)

Apply Jobs (https://www.analyticsvidhya.com/jobs/)

Leaderboard (https://datahack.analyticsvidhya.com/users/)

COMPANIES

Post Jobs (https://www.analyticsvidhya.com/corporate/)

Trainings (https://trainings.analyticsvidhya.com)

Hiring Hackathons (https://datahack.analyticsvidhya.com/)

Advertising (https://www.analyticsvidhya.com/contact/)

Reach Us (https://www.analyticsvidhya.com/contact/)

(https://trainings.analyticsvidhya.com/courses/course-

JOIN OUR COMMUNITY :

(https://www.fac (https://twitter.com/analyticsv

(https://www.fac (https://twitter.com/analyticsv

18469 Followers

(https://www.fac (https://twitter.com/analyticsv

(https://plus.goo (https://www.linkedin

3008 Followers

(https://plus.goo (https://www.linkedin

7513 Followers

(https://plus.goo (https://www.linkedin

Subscribe to emailer >

Privacy Policy (https://www.analyticsvidhya.com/privacy-policy/)

Terms of Use (https://www.analyticsvidhya.com/terms/)

Refund Policy (https://www.analyticsvidhya.com/refund-policy/)

Don't have an account? Sign up (https: