QBUS6840: Tutorial 3 – Time Series Decomposition I

Objectives

- Learn how to do moving average with Pandas
- Learn how to intuitively explore the stationarity of a time series data

了解如何使用Pandas进行移动平均线.
了解如何直观地探索时间序列数据的平稳性

In our previous tutorial, we have learnt the data manipulation in Time Series, i.e. loading the dataset from a csv file, indexing the data, and visualizing the data. In this week, we will learn how to do moving average and intuitively explore the stationarity, a pattern we will study more in the later lectures.

For today's content, we will mainly focus on moving average and analysing the stationarity.

1. Loading the dataset

Create a new Python script called "tutorial_03.py" and download the "AirPassenger.csv" file from Canvas.
Let's begin our script by importing necessary libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
```

Then read the data file into a data frame variable.

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

You can also use the following statements to check the dataset information:

```
print(data.head())
print('\n Data Types:')
print(data.dtypes)
```

The data contains a particular month and number of passengers travelling in that month. The data type here is object (month).
Let's convert it into a time series object and use the "Month" column as our index.

```
from datetime import datetime
con=data['Month']
data['Month']=pd.to_datetime(data['Month'])
data.set_index('Month', inplace=True)
#check datatype of index
print(data.index)
```

Then we extract the "Passengers" column and plot the data:

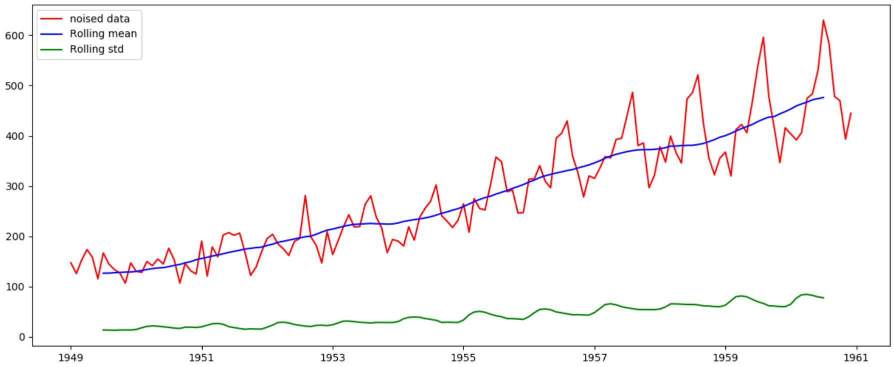
```
ts = data['Passengers']
ts.head(10)

from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6
plt.figure()
plt.plot(ts)
```

2. Rolling windows

Rolling statistics are another type of time series-specific operations implemented by Pandas. These can be accomplished via the `rolling()` attribute of Series and `DataFrame` objects (You may find out more detailed information on aggregation and grouping at here: https://jakevdp.github.io/PythonDataScienceHandbook/03.08-aggregation-and-grouping.html). This rolling view makes available a number of aggregation operations by default.

For example, here is the one-year (12 months) centered rolling mean and standard deviation of the noised data:

```
rolling_data = ts.rolling(12,center=True)
plt.figure()
plt.plot(ts,'r-',label="noised data")
plt.plot(rolling_data.mean(),'b-',label="Rolling mean")
plt.plot(rolling_data.std(),'g-',label="Rolling std")
plt.legend()
```



Answer the following questions:

(1) Why are the blue and green lines shorter than the original time series (red)?

(2) How many values are missing on each end? Can you link this with what you have studied from Lecture?

(3) Guess what if we use 13 for rolling?

3. Checking the stationarity

Stationarity is a very important concept in time series analysis. We will have about two lectures to formally introduce and discuss the concept. Here as an application of moving average, we intuitively explore the stationarity in a preliminary way.

平稳

Checking the stationarity is a very important step in Time Series Analysis. In order to apply a time series model, it is important for the time series to be stationary; in other words, all its statistical properties (mean, variance) remain constant over time. This is done basically because if you take a certain behavior over time, it is important that this behavior is same in the future in order for us to forecast the series.

In practice we can assume the series to be stationary if it has constant statistical properties over time and these properties can be:

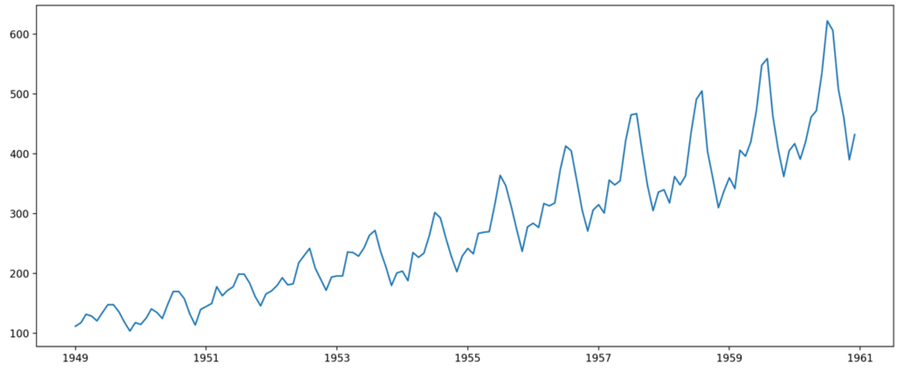
- constant mean
- constant variance
- auto co-variance that does not depend on time.

[Don't worry about these concepts for the time being. We will revisit them later on]

These details can be easily retrieved using stat commands in python.

The best way to understand the stationarity in a Time Series is by eye-balling the plot:

```
ts = data['Passengers']
plt.figure()
plt.plot(ts)
```



It’s clear from the plot that there is an overall increase in the trend, with some seasonality in it. Therefore, this means we need to do some data preprocessing to get a stationary data before we apply the forecasting techniques.

4. Making the time series stationary

There are two major factors that make a time series non-stationary. They are:

- Trend: non-constant mean
- Seasonality: Variation at specific time-frames

季节性

The basic idea is to model the trend and seasonality in this series, so we can remove it and make the series stationary. Then we can go ahead and apply statistical forecasting to the stationary series. And finally we can convert the forecasted values into original by applying the trend and seasonality constrains back to those that we previously separated.

The first step is to reduce the trend using transformation, as we can see here that there is a strong positive trend. These transformations can be log, sq-rt, cube root etc. Basically it penalizes 处罚 larger values more than the smaller. In this case we will use the logarithmic transformation.

```
ts_log= np.log(ts)
plt.figure()
plt.plot(ts_log, color='red',label='log')
```

There is some noise in realizing the forward trend here.
We will use moving average as the smoothing technique to model these trends and then remove them from the series.

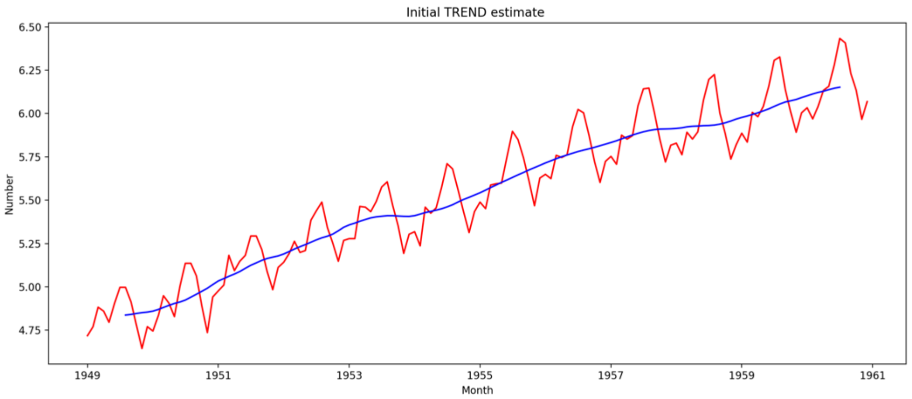
Note that, in our dataset, we can observe the data has a yearly pattern. This means the period of data m =12 months (which is visually identified).

Question: Why we choose m = 12?

To calculate the initial trend-cycle estimate, we need to do moving average smoothing 磨平 to remove seasonal fluctuation. For even periods of seasonal peak, we must do a “2 by m”-MA, where m is the period. In this case the period is m = 12, so we must do a 2x12 MA.

```
Trend = ts_log.rolling(2, center =
True).mean().rolling(12,center = True).mean()
np.random.seed(0)

plt.figure()
plt.plot(ts_log, color='red',label='log')
plt.plot(Trend, color='blue',label='MA')
plt.title('Initial TREND estimate ')
plt.xlabel('Month')
plt.ylabel('Number')
```

What if the period is an odd number?
For odd seasonal peaks we must do a m -MA.



Now subtract the rolling mean from the original series.

```
ts_res = ts_log - Trend
```

Now use variable explorer to explore ts_res variable and answer the following question:

(1) What is the size of ts_res?
(2) What is the data type of ts_res?
(3) How many “nan” elements? Why?

Remove the “nan” elements from ts_res variable:

```
ts_res.dropna(inplace = True)
```

What is the size of ts_res now?

Alternatively, if you want to replace the nan value with zeros, you can use following statement:

```
ts_res_zero = np.nan_to_num(ts_res)
```

You can now plot the ts_res:

```
rolling_data_res = pd.Series(ts_res,
dtype='float64').rolling(12,center=True)
plt.figure()
plt.plot(ts_res,'r-',label="time series data")
plt.plot(rolling_data_res.mean(),'b-',label="Rolling
mean")
plt.plot(rolling_data_res.std(),'g-',label="Rolling std")
plt.legend()
```

From the plot, you can see the data is stationary as the rolling mean and rolling std is almost a constant value.

What is the purpose of statement “pd.Series(ts_res, dtype='float64')”?

5. Checking the stationarity (advanced)

In step 3, we have learnt how to check the stationarity by observing the plot. However, for some complicated situations, you may not be able to easily examine if the data is stationary. Therefore, many people prefer to use **Dickey-fuller Test** to check the stationarity. First we consider the null hypothesis: the time series is non-stationary. The result from the rest will contain the test statistic and critical value for different confidence levels. The idea is to have Test statistics less than critical value, in this case we can reject the null hypothesis and say that this Time series is indeed stationary (the force is strong with this one !!) 首先我们考虑零假设：时间序列是非平稳的。其余结果将包含不同置信水平的检验统计量和临界值。
我们的想法是让测试统计数据低于临界值，在这种情况下我们可以拒绝零假设，
并说这个时间序列确实是静止的（这个力量很强!!）

You can find a more detailed explanation on Dickey-fuller Test at here: http://www.real-statistics.com/time-series-analysis/stochastic-processes/dickey-fuller-test/

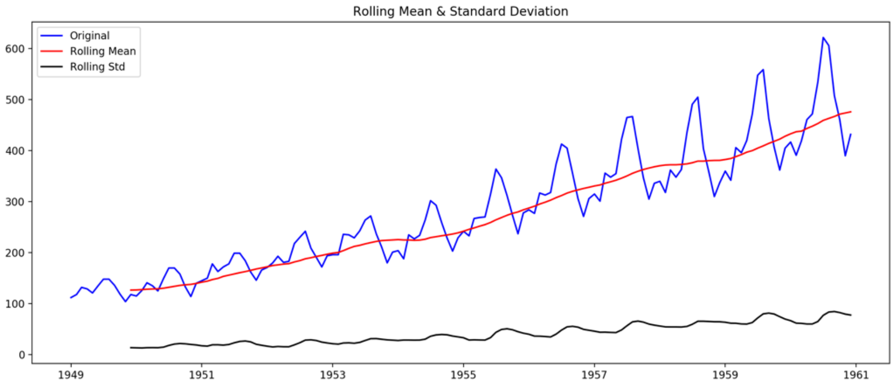
Here, we write a test_stationarity() function. This test_stationarity() will call the Dickey-fuller Test and print out the test results. For convenience, we also create a plot_curve() function to plot for visualization. This plot_curve() function will input the time series data and generate the rolling mean and rolling standard deviation curve:

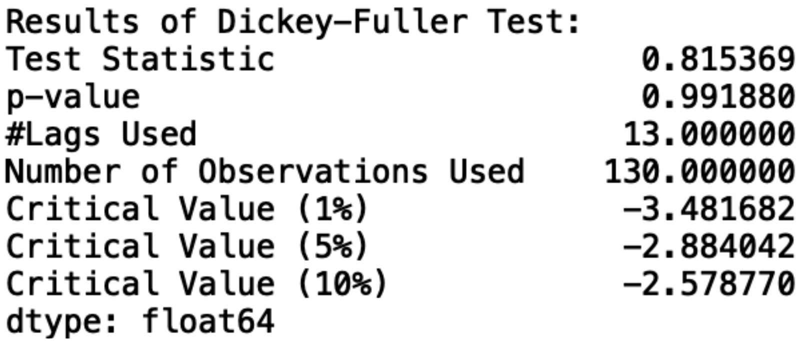
```
from statsmodels.tsa.stattools import adfuller

def test_stationarity(timeseries):
    #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)

def plot_curve(timeseries):
    #Determing rolling statistics
#    rolmean = pd.rolling_mean(pd.rolling_mean(timeseries,
window=12),window=2)
#    rolstd = pd.rolling_std(pd.rolling_std(timeseries,
window=12),window=2)
    rolmean = pd.rolling_mean(timeseries, window=12)
    rolstd = pd.rolling_std(timeseries, window=12)
    #Plot rolling statistics:
    plt.figure()
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()
```

When you finish defining these functions, please remember to call these functions with following statement (otherwise, the above statements will never be executed!!!). Now, let’s use the Dickey-fuller Test method to check the stationarity of the original data:

```
test_stationarity(ts)
plot_curve(ts)
```

Then, you should able to see something like this:





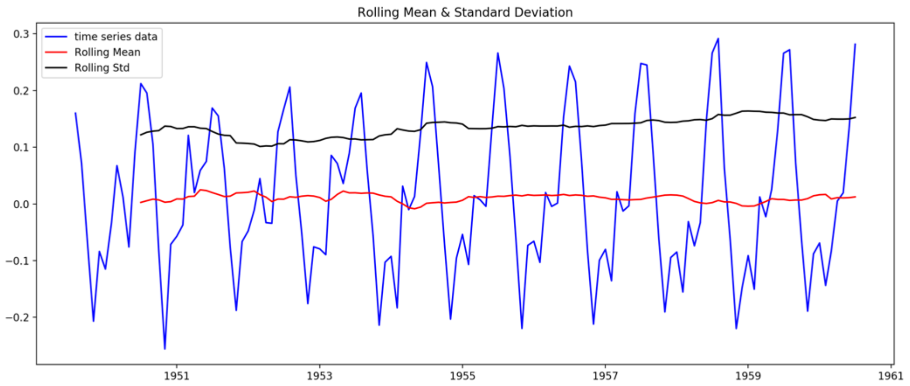
```
Results of Dickey-Fuller Test:
Test Statistic                   0.815369
p-value                          0.991880
#Lags Used                      13.000000
Number of Observations Used    130.000000
Critical Value (1%)             -3.481682
Critical Value (5%)             -2.884042
Critical Value (10%)            -2.578770
dtype: float64
```

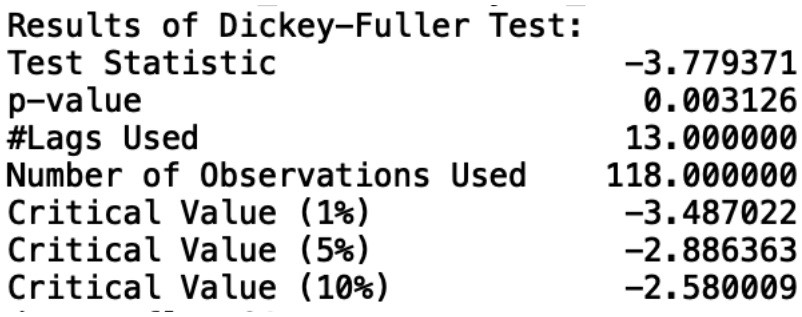
Please pay attention to the above plot and results. Clearly, this is not stationary because:

- Rolling mean is increasing even though the std is small.
- Test statistic (0.81) is larger than critical value (-2.57) at 10%.
  ** 10% here means 10% of the data. If the test statistic is smaller than the critical value at 10%, means at least 90% of the data is stationary. However, in our case, the number is larger than critical value at 10%, which means the data is non-stationary (at least for 90% of the data).

Now you could call the `test_stationarity()` and `plot_curve()` function to test the stationarity of `ts_res` in task 3.

```
test_stationarity(ts_res)
plot_curve(ts_res)
```





```
Results of Dickey-Fuller Test:
Test Statistic                  -3.779371
p-value                          0.003126
#Lags Used                      13.000000
Number of Observations Used    118.000000
Critical Value (1%)             -3.487022
Critical Value (5%)             -2.886363
Critical Value (10%)            -2.580009
```

From the above outputs, you can see the current time series data is stationary because:

- The rolling values are varying slightly but there is no specific trend
- The test statistics (-3.77) is smaller than the 1% critical values (-3.48). That tells us that we are 99% confident that this series is stationary.

If you are interested in topic and wish to learn more, please go to https://medium.com/@stallonejacob/time-series-forecast-a-basic-introduction-using-python-414fcb963000

6. (Optional) More about Dickey-Fuller Test:

The (Augmented) Dickey-Fuller test (ADF) is a type of statistical test called a unit root test. The intuition behind a unit root test is that it determines how strongly a time series is defined by a trend.

它使用自回归模型并优化跨多个不同滞后值的信息标准。

There are a number of unit root tests and the ADF may be one of the more widely used. It uses an autoregressive model and optimizes an information criterion across multiple different lag values.

Null Hypothesis (H0): If failed to be rejected, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure.
Alternate Hypothesis (H1): The null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. It does not have time-dependent structure.

Sometimes people also interpret the result using the p-value from the test. **A**

p-value below a threshold (such as 5% or 1%) suggests we reject the null hypothesis (stationary), otherwise a p-value above the threshold suggests we fail to reject the null hypothesis (non-stationary).

For example, if we set the threshold as 5%, if:

- **p-value > 0.05:** Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.
- **p-value <= 0.05:** Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

A higher positive test statistic value means lower chance to reject the Null Hypothesis, whereas the **more negative this statistic, the given time series data are more likely to reject the Null Hypothesis** (which means the data is stationary).

In the first example of task 5, the program returns a test statistic value of 0.85, means the data is more likely to be a non-stationary data. As part of the output, we can also see further statistic results, which is 0.85 is higher than the value of -2.578 at 10%. This suggest we fail to reject the Null Hypothesis and does have time-dependent structure.

For more information, please visit:
https://en.wikipedia.org/wiki/Dickey%E2%80%93Fuller_test