<https://www.tutorialspoint.com/artificial_intelligence_with_python/index.htm>

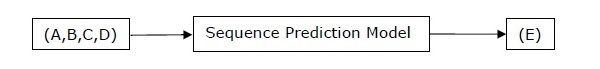
Time series data means the data that is in a series of particular time intervals. If we want to build sequence prediction in machine learning, then we have to deal with sequential data and time. Series data is an abstract of sequential data. Ordering of data is an important feature of sequential data.

Basic Concept of Sequence Analysis or Time Series Analysis

Sequence analysis or time series analysis is to predict the next in a given input sequence based on the previously observed. The prediction can be of anything that may come next: a symbol, a number, next day weather, next term in speech etc. Sequence analysis can be very handy in applications such as stock market analysis, weather forecasting, and product recommendations.

**Example**

Consider the following example to understand sequence prediction. Here **A,B,C,D** are the given values and you have to predict the value **E** using a Sequence Prediction Model.



Example

The following example shows you handling and slicing the time series data by using Pandas. Note that here we are using the Monthly Arctic Oscillation data, which can be downloaded from [monthly.ao.index.b50.current.ascii](http://www.cpc.ncep.noaa.gov/products/precip/CWlink/daily_ao_index/monthly.ao.index.b50.current.ascii) and can be converted to text format for our use.

Handling time series data

For handling time series data, you will have to perform the following steps −

The first step involves importing the following packages −

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

Next, define a function which will read the data from the input file, as shown in the code given below −

def read\_data(input\_file):

input\_data = np.loadtxt(input\_file, delimiter = None)

Now, convert this data to time series. For this, create the range of dates of our time series. In this example, we keep one month as frequency of data. Our file is having the data which starts from January 1950.

dates = pd.date\_range('1950-01', periods = input\_data.shape[0], freq = 'M')

In this step, we create the time series data with the help of Pandas Series, as shown below −

output = pd.Series(input\_data[:, index], index = dates)

return output

if \_\_name\_\_=='\_\_main\_\_':

Enter the path of the input file as shown here −

input\_file = "/Users/admin/AO.txt"

Now, convert the column to timeseries format, as shown here −

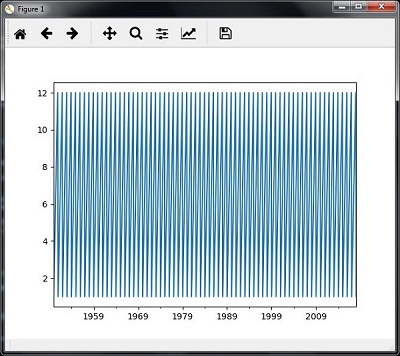
timeseries = read\_data(input\_file)

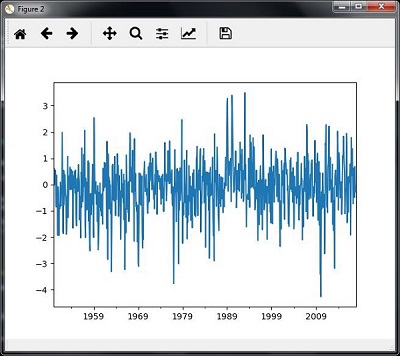
Finally, plot and visualize the data, using the commands shown −

plt.figure()

timeseries.plot()

plt.show()





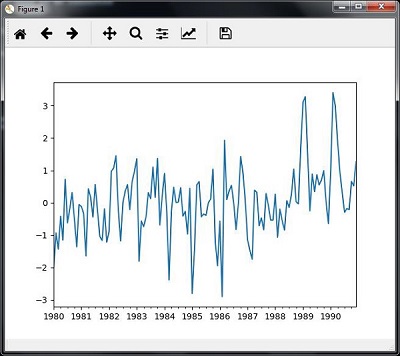
Slicing time series data

Slicing involves retrieving only some part of the time series data. As a part of the example, we are slicing the data only from 1980 to 1990. Observe the following code that performs this task −

timeseries['1980':'1990'].plot()

<matplotlib.axes.\_subplots.AxesSubplot at 0xa0e4b00>

plt.show()



Extracting Statistic from Time Series Data

timeseries.mean()

timeseries.max()

timeseries.min()

If you want to calculate all statistics at a time, you can use the **describe()** function as shown here −

timeseries.describe()

count 817.000000

mean -0.111431

std 1.003151

min -4.265700

25% -0.649430

50% -0.042744

75% 0.475720

max 3.495300

dtype: float64

Re-sampling

You can resample the data to a different time frequency. The two parameters for performing re-sampling are −

* Time period
* Method

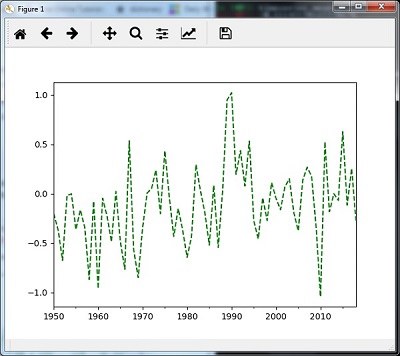
You can use the following code to resample the data with the mean()method, which is the default method −

timeseries\_mm = timeseries.resample("A").**mean()**

timeseries\_mm.plot(style = 'g--')

plt.show()

Then, you can observe the following graph as the output of resampling using mean() −



Re-sampling with median()

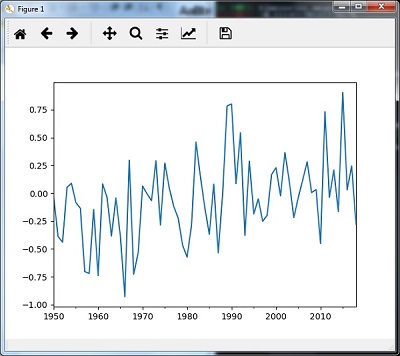
You can use the following code to resample the data using the **median()**method −

timeseries\_mm = timeseries.resample("A").median()

timeseries\_mm.plot()

plt.show()

Then, you can observe the following graph as the output of re-sampling with median() −



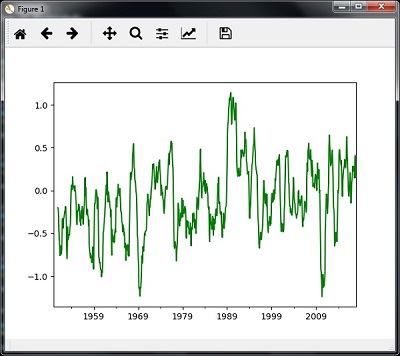
Rolling Mean

You can use the following code to calculate the rolling (moving) mean −

timeseries.rolling(window = 12, center = False).mean().plot(style = '-g')

plt.show()

Then, you can observe the following graph as the output of the rolling (moving) mean −



Analyzing Sequential Data by Hidden Markov Model (HMM)

HMM is a statistic model which is widely used for data having continuation and extensibility such as time series stock market analysis, health checkup, and speech recognition. This section deals in detail with analyzing sequential data using Hidden Markov Model (HMM).

Hidden Markov Model (HMM)

HMM is a stochastic model which is built upon the concept of Markov chain based on the assumption that probability of future stats depends only on the current process state rather any state that preceded it. For example, when tossing a coin, we cannot say that the result of the fifth toss will be a head. This is because a coin does not have any memory and the next result does not depend on the previous result.

Mathematically, HMM consists of the following variables −

States (S)

It is a set of hidden or latent states present in a HMM. It is denoted by S.

Output symbols (O)

It is a set of possible output symbols present in a HMM. It is denoted by O.

State Transition Probability Matrix (A)

It is the probability of making transition from one state to each of the other states. It is denoted by A.

Observation Emission Probability Matrix (B)

It is the probability of emitting/observing a symbol at a particular state. It is denoted by B.

Prior Probability Matrix (Π)

It is the probability of starting at a particular state from various states of the system. It is denoted by Π.

Hence, a HMM may be defined as **𝝀 = (S,O,A,B,𝝅)**,

where,

* **S = {s1,s2,…,sN}** is a set of N possible states,
* **O = {o1,o2,…,oM}** is a set of M possible observation symbols,
* A is an **N𝒙N** state Transition Probability Matrix (TPM),
* B is an **N𝒙M** observation or Emission Probability Matrix (EPM),
* π is an N dimensional initial state probability distribution vector.

Example: Analysis of Stock Market data

In this example, we are going to analyze the data of stock market, step by step, to get an idea about how the HMM works with sequential or time series data. Please note that we are implementing this example in Python.

import datetime

import warnings

import numpy as np

from matplotlib import cm, pyplot as plt

from matplotlib.dates import YearLocator, MonthLocator

try:

from matplotlib.finance import quotes\_historical\_yahoo\_och1

except ImportError:

from matplotlib.finance import (

quotes\_historical\_yahoo as quotes\_historical\_yahoo\_och1)

from hmmlearn.hmm import GaussianHMM

Load the data from a start date and end date, i.e., between two specific dates as shown here −

start\_date = datetime.date(1995, 10, 10)

end\_date = datetime.date(2015, 4, 25)

quotes = quotes\_historical\_yahoo\_och1('INTC', start\_date, end\_date)

In this step, we will extract the closing quotes every day. For this, use the following command −

closing\_quotes = np.array([quote[2] for quote in quotes])

Now, we will extract the volume of shares traded every day. For this, use the following command −

volumes = np.array([quote[5] for quote in quotes])[1:]

Here, take the percentage difference of closing stock prices, using the code shown below −

diff\_percentages = 100.0 \* np.diff(closing\_quotes) / closing\_quotes[:-]

dates = np.array([quote[0] for quote in quotes], dtype = np.int)[1:]

training\_data = np.column\_stack([diff\_percentages, volumes])

In this step, create and train the Gaussian HMM. For this, use the following code −

hmm = GaussianHMM(n\_components = 7, covariance\_type = 'diag', n\_iter = 1000)

with warnings.catch\_warnings():

warnings.simplefilter('ignore')

hmm.fit(training\_data)

Now, generate data using the HMM model, using the commands shown −

num\_samples = 300

samples, \_ = hmm.sample(num\_samples)

Finally, in this step, we plot and visualize the difference percentage and volume of shares traded as output in the form of graph.

Use the following code to plot and visualize the difference percentages −

plt.figure()

plt.title('Difference percentages')

plt.plot(np.arange(num\_samples), samples[:, 0], c = 'black')

Use the following code to plot and visualize the volume of shares traded −

plt.figure()

plt.title('Volume of shares')

plt.plot(np.arange(num\_samples), samples[:, 1], c = 'black')

plt.ylim(ymin = 0)

plt.show()