**TIME SERIES ANALYSIS (PANDAS, HMMLEARN, PYSTRUCT PACKAGES)**

The arrangement of data in accordance with their time of occurrence is a time series. How do people get to know that the price of a commodity has increased over a period of time? They can do so by comparing the prices of the commodity for a set of a time period. Time series are used in field like AI Data Analytics, and many more domain area

Examples of time series are in weather forecasting prediction analysis. In the prediction of the trend in market stocks or electricity consumption, time plays a major role that must now be considered in our models. For example, it would be interesting to not only know when a stock will rise in price but also when it will move up of. The figure below shows some of the applications in which data can be predicted.

**Time series has four main components**

**Trend**: Trend is the incline or decline in the series over an interval of time, it continues over a long period of time.

**Seasonality**: The structured pattern of up and down variations. It is a small-term variation occurring due to seasonal factors.

**Cyclicity:** The fluctuations in a time series which occurs themselves over a period of more than one year are the cyclic variations.

**Irregularity:** It refers to changes which occur due to random

**Benefits & Applications of Time Series:**

It helps to achieve various objectives:

1. Pictorial Analysis: determines the tendency and model of the future using graphs and other tools.

2. Forecasting: It is used broadly in economic, business forecasting based on historical trends and patterns.

3. Illustrative Analysis: to study cross-interconnection between two time-series and their dependency on one another.

Time series analysis is used for predicting and analyzing the sample model collected one useful and simple method for analyzing and predicting data is the additive model an additive model helps us to show patterns and also the trends and helps to prediction patterns based on necessary data that we have. Time series data means the data that is in a series of a specific period. 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.

### Brief overview of Time Series Analysis

Time series analysis is to estimate the next input sequence based on the previously observed. The prediction or estimation could be any sort of thing which appears next like weather, next term in speech ,etc. Sequence analysis is very useful in the stock market, weather forecasting, and product recommendations.

For time series data analysis using Python, we need to install the following packages –

### Pandas

Pandas are an open source BSD-licensed library which has high-performance, ease of data structure usage and data analysis tools for Python. Pandas can be installed using

**pip install pandas**

Pandas in python are quick, adaptable, and expressive data structures designed to make working data both simple and instinctive. And as are well suited for many different kinds of Through pandas we can perform many operations like slicing the data frame changing the index, data conversion, joining and merging concentration and changing the column headers. Pandas are a python module that makes data science easy. Some of the features of pandas are listed below:

Support for CSV JSON EXCEL, SQL, SAS, Clipboard etc.

* Pandas helps in data cleaning
* Data visualization
* Well integrated with Jupiter notebooks
* Easy handling of missing data (represented as Nan)
* Size mutability: columns can be inserted and deleted from Data Frame and higher dimensional objects
* Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, Data Frame, etc.
* Make it easy to convert ragged, Intelligent label-based slicing, fancy indexing, and sub setting of large data sets
* Intuitive merging and joining data sets
* Flexible reshaping and pivoting of data sets
* Hierarchical labeling of axes (possible to have multiple labels per tick)
* Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases.

## Hmmlearn

It is an open source BSD-licensed library which has simple algorithms and models to learn Hidden Markov Models (HMM) in Python. Hmm can be installed using

**Pip install hmmlearn**

### Hmmlearn is Hidden Markov Models (HMMs). The HMM is a generative probabilistic model, in which a sequence of observable XX variables is generated by a sequence of internal hidden states ZZ. The hidden states are not observed directly. Stock market analysis can be done with hmm

### A Important statistical tool for modeling time series data. The theory of Markov Chains is introduced in the early 20th century and a full grown Hidden Markov Model (HMM) is developed in the 1960s, its potential is recognized in the last decade only. Its application is in many domains like Signal Processing in Electronics, Brownian motions in Chemistry, Random Walks in Statistics (Time Series) in Artificial Intelligence

### PyStruct

It is a structured learning and prediction library. Learning algorithms implemented in PyStruct have names such as conditional random fields (CRF), Maximum-Margin Markov Random Networks (M3N) or structural support vector machines. Pystruct is used in structured prediction.

**What is structured prediction?**

Structured prediction is a conjecture of the standard model of supervised learning, classification, and regression. All of these can be thought of finding a function that minimizes some loss over a training set. The differences are in the kind of functions that are used and the losses.

In classification, the target domain is discrete class labels, and the loss is usually the 0-1 loss, i.e. counting the misclassifications. In regression, the target domain is the real numbers, and the loss is usually mean squared error. In structured prediction, both the target domain and the loss are more or less arbitrary. This means the goal is not to predict a label or a number, but a possibly much more complicated object like a sequence or a graph.

In structured prediction, we often deal with finite, but large output spaces. This situation could be dealt with using classification with a very large number of classes. The idea behind the structured prediction is that we can do better than this, by making use of the structure of the output space.

Pystruct can be installed with the help of the following command −

**pip install pystruct**

**Pandas Code**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

sns.set()

df = pd.read\_csv('E:/Pandas/data.csv', skiprows=1)

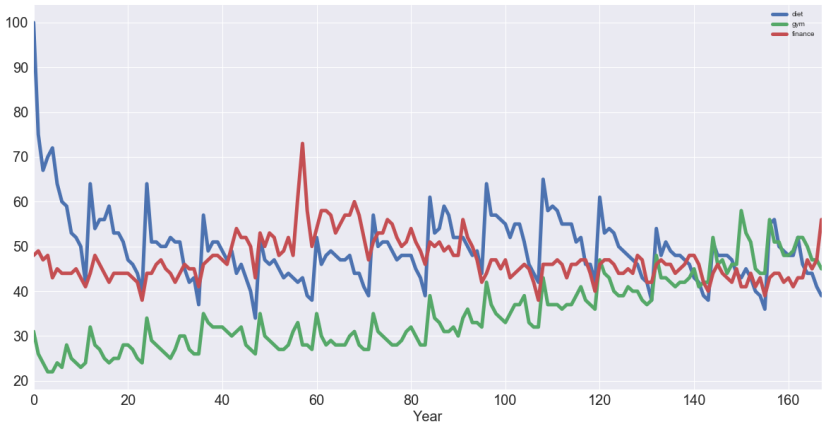
df.columns = ['month', 'diet', 'gym', 'finance']

df.head()

|  | **Month** | **diet** | **gym** | **finance** | |
| --- | --- | --- | --- | --- | --- |
| **0** | 2004-01 | 100 | 31 | 48 |
| **1** | 2004-02 | 75 | 26 | 49 |
| **2** | 2004-03 | 67 | 24 | 47 |
| **3** | 2004-04 | 70 | 22 | 48 |
| **4** | 2004-05 | 72 | 22 | 43 |

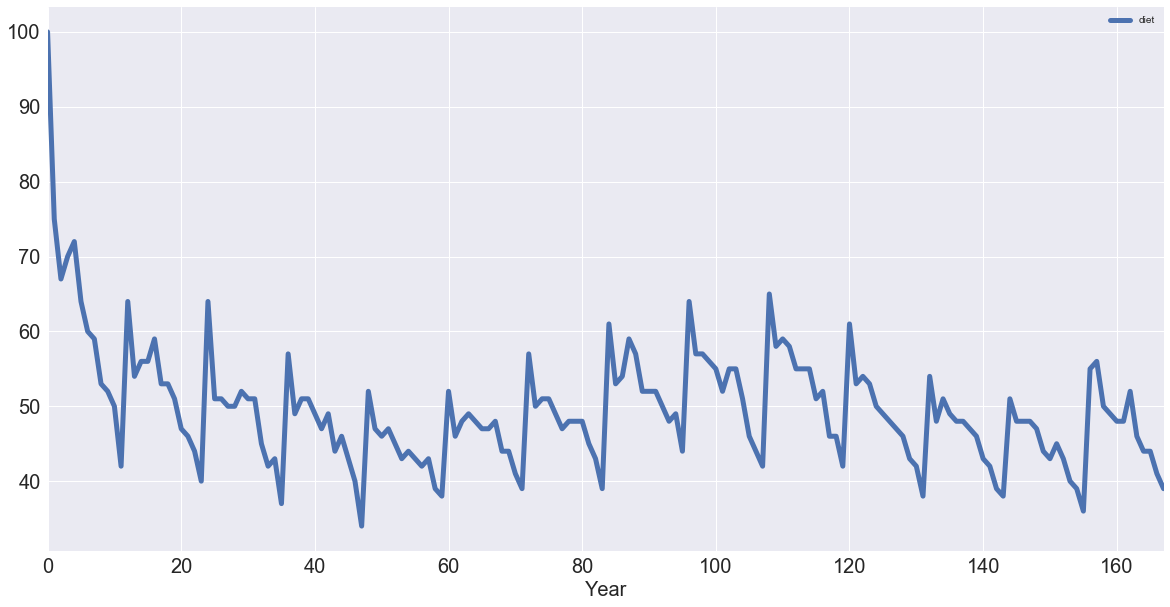
df.plot(figsize=(20,10), linewidth=5, fontsize=20)

plt.xlabel('Year', fontsize=20);



df[['diet']].plot(figsize=(20,10), linewidth=5, fontsize=20)

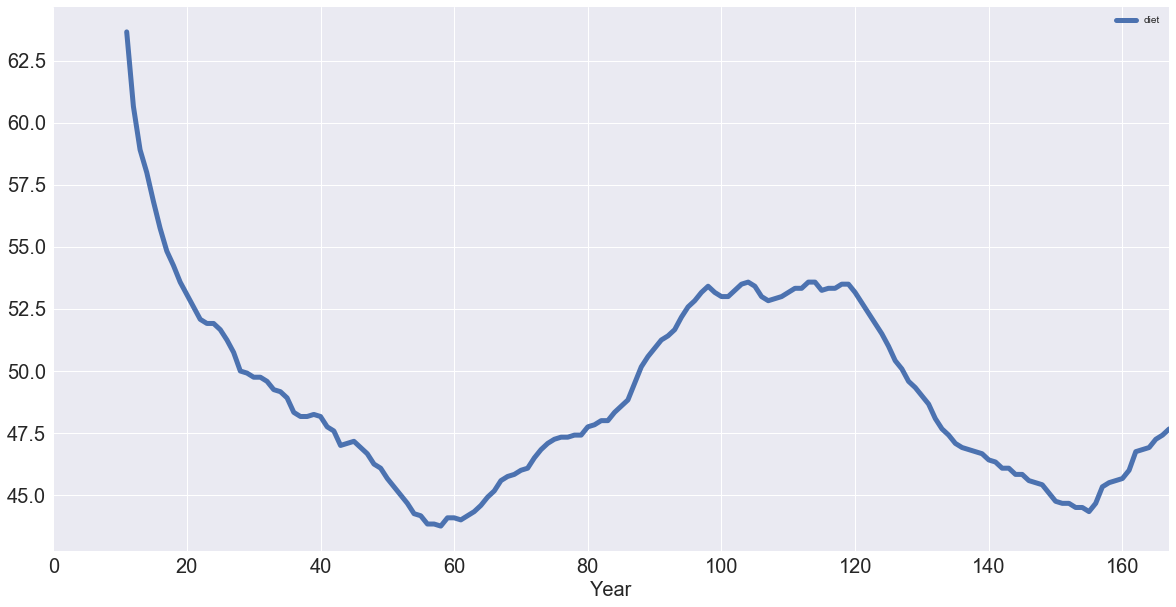
plt.xlabel('Year', fontsize=20);



diet = df[['diet']]

diet.rolling(12).mean().plot(figsize=(20,10), linewidth=5, fontsize=20)

plt.xlabel('Year', fontsize=20);



**Code pystruct**

import itertools

import numpy as np

from scipy import sparse

from sklearn.metrics import hamming\_loss

from sklearn.datasets import fetch\_mldata

from sklearn.metrics import mutual\_info\_score

from scipy.sparse.csgraph import minimum\_spanning\_tree

from pystruct.learners import OneSlackSSVM

from pystruct.models import MultiLabelClf

from pystruct.datasets import load\_scene

def chow\_liu\_tree(y\_):

# compute mutual information using sklearn

n\_labels = y\_.shape[1]

mi = np.zeros((n\_labels, n\_labels))

for i in range(n\_labels):

for j in range(n\_labels):

mi[i, j] = mutual\_info\_score(y\_[:, i], y\_[:, j])

mst = minimum\_spanning\_tree(sparse.csr\_matrix(-mi))

edges = np.vstack(mst.nonzero()).T

edges.sort(axis=1)

return edges

dataset = "scene"

# dataset = "yeast"

if dataset == "yeast":

yeast = fetch\_mldata("yeast")

X = yeast.data

X = np.hstack([X, np.ones((X.shape[0], 1))])

y = yeast.target.toarray().astype(np.int).T

X\_train, X\_test = X[:1500], X[1500:]

y\_train, y\_test = y[:1500], y[1500:]

else:

scene = load\_scene()

X\_train, X\_test = scene['X\_train'], scene['X\_test']

y\_train, y\_test = scene['y\_train'], scene['y\_test']

n\_labels = y\_train.shape[1]

full = np.vstack([x for x in itertools.combinations(range(n\_labels), 2)])

tree = chow\_liu\_tree(y\_train)

full\_model = MultiLabelClf(edges=full, inference\_method='qpbo')

independent\_model = MultiLabelClf(inference\_method='unary')

tree\_model = MultiLabelClf(edges=tree, inference\_method="max-product")

full\_ssvm = OneSlackSSVM(full\_model, inference\_cache=50, C=.1, tol=0.01)

tree\_ssvm = OneSlackSSVM(tree\_model, inference\_cache=50, C=.1, tol=0.01)

independent\_ssvm = OneSlackSSVM(independent\_model, C=.1, tol=0.01)

print("fitting independent model...")

independent\_ssvm.fit(X\_train, y\_train)

print("fitting full model...")

full\_ssvm.fit(X\_train, y\_train)

print("fitting tree model...")

tree\_ssvm.fit(X\_train, y\_train)

print("Training loss independent model: %f"

% hamming\_loss(y\_train, np.vstack(independent\_ssvm.predict(X\_train))))

print("Test loss independent model: %f"

% hamming\_loss(y\_test, np.vstack(independent\_ssvm.predict(X\_test))))

print("Training loss tree model: %f"

% hamming\_loss(y\_train, np.vstack(tree\_ssvm.predict(X\_train))))

print("Test loss tree model: %f"

% hamming\_loss(y\_test, np.vstack(tree\_ssvm.predict(X\_test))))

print("Training loss full model: %f"

% hamming\_loss(y\_train, np.vstack(full\_ssvm.predict(X\_train))))

print("Test loss full model: %f"

% hamming\_loss(y\_test, np.vstack(full\_ssvm.predict(X\_test))))s

Code hmmlearn

import numpy as np

import matplotlib.pyplot as plt

from hmmlearn import hmm

startprob = np.array([0.6, 0.3, 0.1, 0.0])

transmat = np.array([[0.7, 0.2, 0.0, 0.1],

[0.3, 0.5, 0.2, 0.0],

[0.0, 0.3, 0.5, 0.2],

[0.2, 0.0, 0.2, 0.6]])

means = np.array([[0.0, 0.0],

[0.0, 11.0],

[9.0, 10.0],

[11.0, -1.0]])

covars = .5 \* np.tile(np.identity(2), (4, 1, 1))

model = hmm.GaussianHMM(n\_components=4, covariance\_type="full")

model.startprob\_ = startprob

model.transmat\_ = transmat

model.means\_ = means

model.covars\_ = covars

X, Z = model.sample(500)

plt.plot(X[:, 0], X[:, 1], ".-", label="observations", ms=6,

mfc="orange", alpha=0.7)

for i, m in enumerate(means):

plt.text(m[0], m[1], 'Component %i' % (i + 1),

size=17, horizontalalignment='center',

bbox=dict(alpha=.7, facecolor='w'))

plt.legend(loc='best')

plt.show()