Course name: **Python programming and analytics by Rahul Sir**

Topic name: **topic 29**

Video name: **python by Rahul sir TSA & TSF**

Video length: **1 hour 32 minutes 40 seconds**

Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. Time series forecasting is the use of a model to predict future values based on previously observed values. Time series are widely used for non-stationary data, like economic, weather, stock price, and retail sales

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.

**Time series model that is very popular among the Data scientists is ARIMA. It stands for Autoregressive Integrated Moving average. ARIMA models aim to describe the correlations in the data with each other. An improvement over ARIMA is Seasonal ARIMA. It takes into account the seasonality of dataset**

**What is a Time Series?**

Time series is a sequence of observations recorded at regular time intervals.

Depending on the frequency of observations, a time series may typically be hourly, daily, weekly, monthly, quarterly and annual. Sometimes, you might have seconds and minute-wise time series as well, like, number of clicks and user visits every minute etc.

**Why even analyze a time series?**

Because it is the preparatory step before you develop a forecast of the series.

Besides, time series forecasting has enormous commercial significance because stuff that is important to a business like demand and sales, number of visitors to a website, stock price etc are essentially time series data.

**So, what does analyze a time series involve?**

Time series analysis involves understanding various aspects about the inherent nature of the series so that you are better informed to create meaningful and accurate forecasts.

# Properties and types of series

***Trend***: A long-term increase or decrease in the data. This can be seen as a slope (doesn't have to be linear) roughly going through the data.

***Seasonality***: A time series is said to be seasonal when it is affected by seasonal factors (hour of day, week, month, year, etc.). Seasonality can be observed with nice cyclical patterns of fixed frequency.

***Cyclicity***: A cycle occurs when the data exhibits rises and falls that are not of a fixed frequency. These fluctuations are usually due to economic conditions, and are often related to the "business cycle". The duration of these fluctuations is usually at least 2 years.

***Residuals***: Each time series can be decomposed in two parts:
- A forecast, made up of one or several *forecasted* values
- Residuals. They are the difference between an observation and its predicted value at each time step. Remember that

***Value of series at time t = Predicted value at time t + Residual at time t***

*A time series is a data sequence ordered (or indexed) by time. It is discrete, and the interval between each point is constant.*

***Decomposition of a time series***

*Each time series can be thought as a mix between several parts:*

- *A trend (upward or downwards movement of the curve on the long term)*
- *A seasonal component*
- *Residuals*

# Import packages:

In [1]:
```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
```

To start with we will make use of NumPy, pandas, matplotlib and seaborn. Additionally, if you want the images to be plotted in the Jupyter Notebook, you can make use of the IPython magic by adding %matplotlib inline to your code. Alternatively, you can also switch to the Seaborn defaults with sns.set()

# Load the data:

In [2]:
```
df = pd.read_csv('C:/Users/Sahibjot/Desktop/multiTimeline.csv')
df.head()
#load data
```

Out[2]:

| | Month | diet: (Worldwide) | gym: (Worldwide) | finance: (Worldwide) |
|---|---|---|---|---|
| 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 |

Pandas dataframe.info() function is used to get a concise summary of the data frame. It comes really handy when doing exploratory analysis of the data. To get a quick overview of the dataset we use the dataframe.info() function.

```
In [3]: df.info() #to check data types, number of rows etc
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 168 entries, 0 to 167
Data columns (total 4 columns):
Month                    168 non-null object
diet: (Worldwide)        168 non-null int64
gym: (Worldwide)         168 non-null int64
finance: (Worldwide)     168 non-null int64
dtypes: int64(3), object(1)
memory usage: 5.3+ KB
```

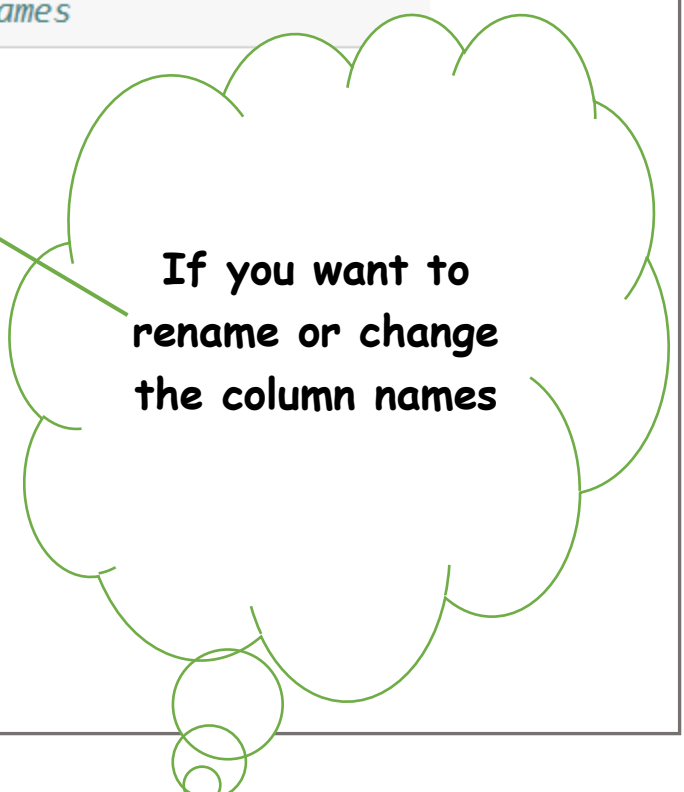
As you can see in the output, the summary includes list of all columns with their data types and the number of non-null values in each column.

This method prints information about a Data Frame including the index dtype and column dtypes, non-null values and memory usage.

```
In [5]: df.columns = ["month", "diet", "gym", "finance"]
        df.head() #rename column names
```

Out[5]:

| | 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 |



```
In [6]: #converting month data type which is "obj" to "datetime" and using it as an index for time series analysis
        df.month = pd.to_datetime(df.month)
        df.set_index('month', inplace=True)  # inplace=true is used to eliminate the default index(0,1,2,3,4)

In [7]: df.head()
```

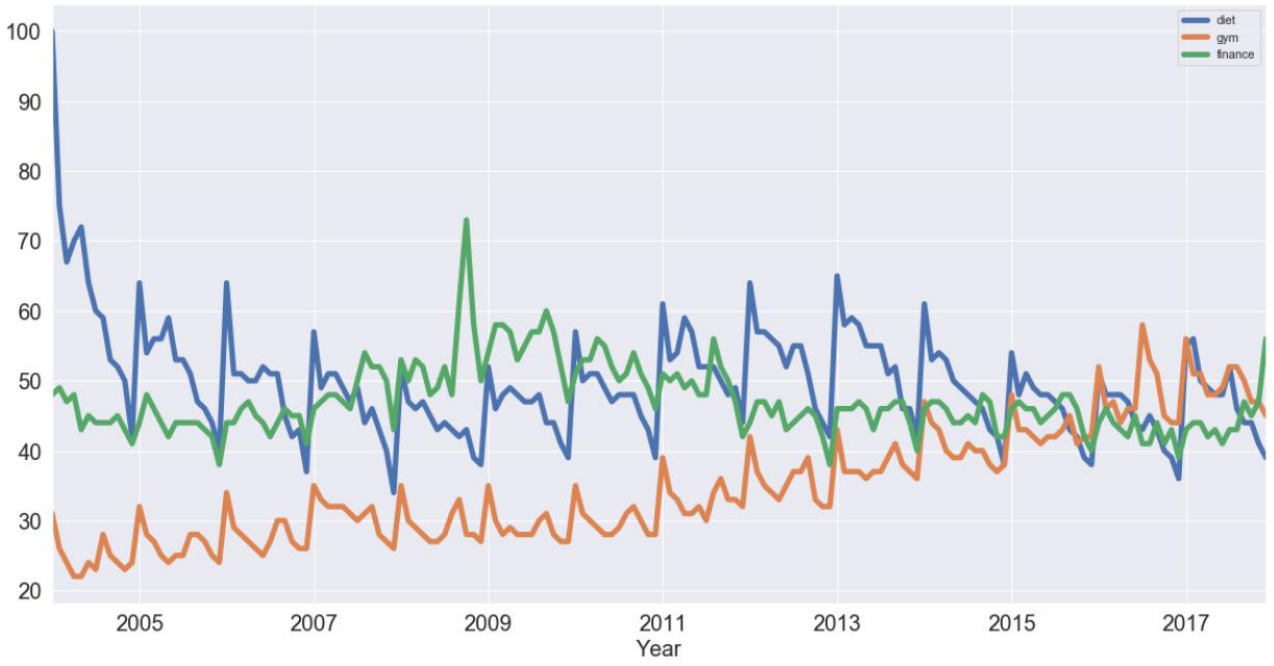
Out[7]:

| month | diet | gym | finance |
|---|---|---|---|
| 2004-01-01 | 100 | 31 | 48 |
| 2004-02-01 | 75 | 26 | 49 |
| 2004-03-01 | 67 | 24 | 47 |
| 2004-04-01 | 70 | 22 | 48 |
| 2004-05-01 | 72 | 22 | 43 |

You can see in the output that the default index is removed and in its place date time denoted as month is represented as the index

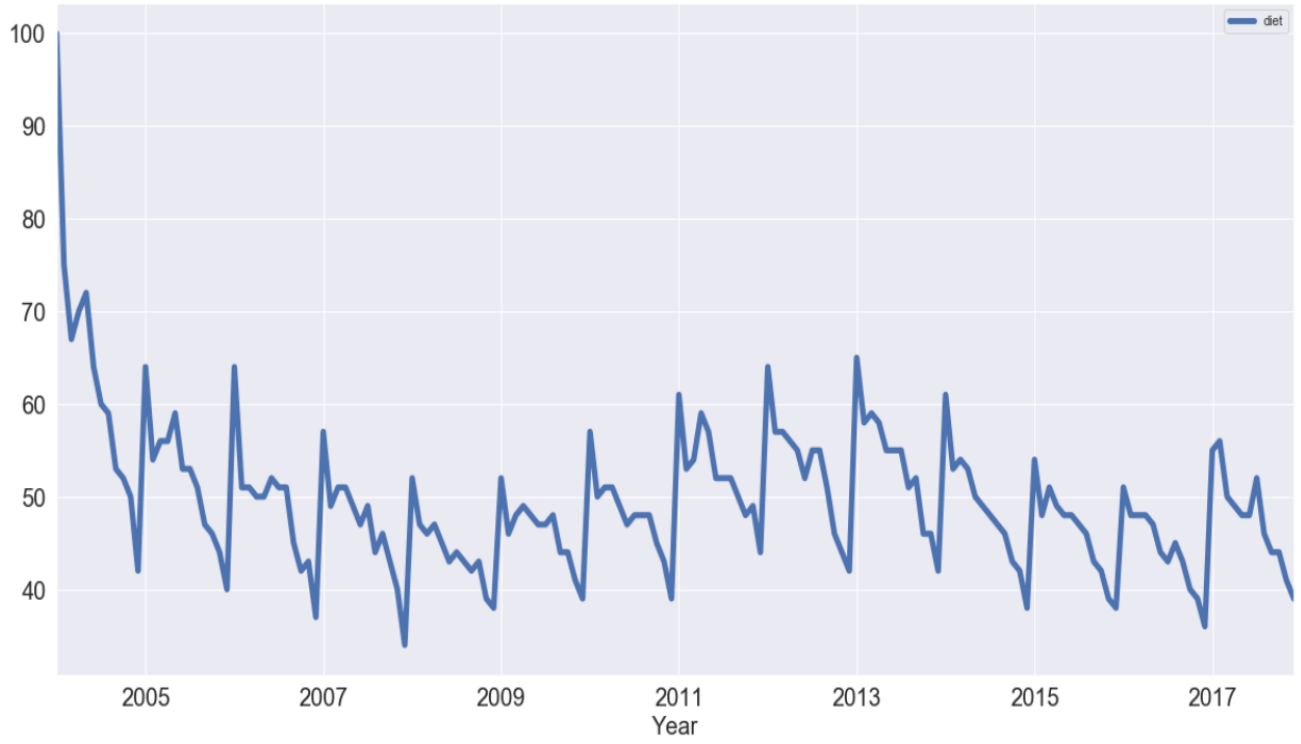
# *Plotting the data:*

```
In [8]: #exploratory data analysis
        df.plot(figsize=(20,10), linewidth=5, fontsize=20)
        plt.xlabel('Year', fontsize=20);
```



if you want to see the single category trend:

```
In [9]: df[['diet']].plot(figsize=(20,10), linewidth=5, fontsize=20)
        plt.xlabel('Year', fontsize=20); #plot single category trend
```

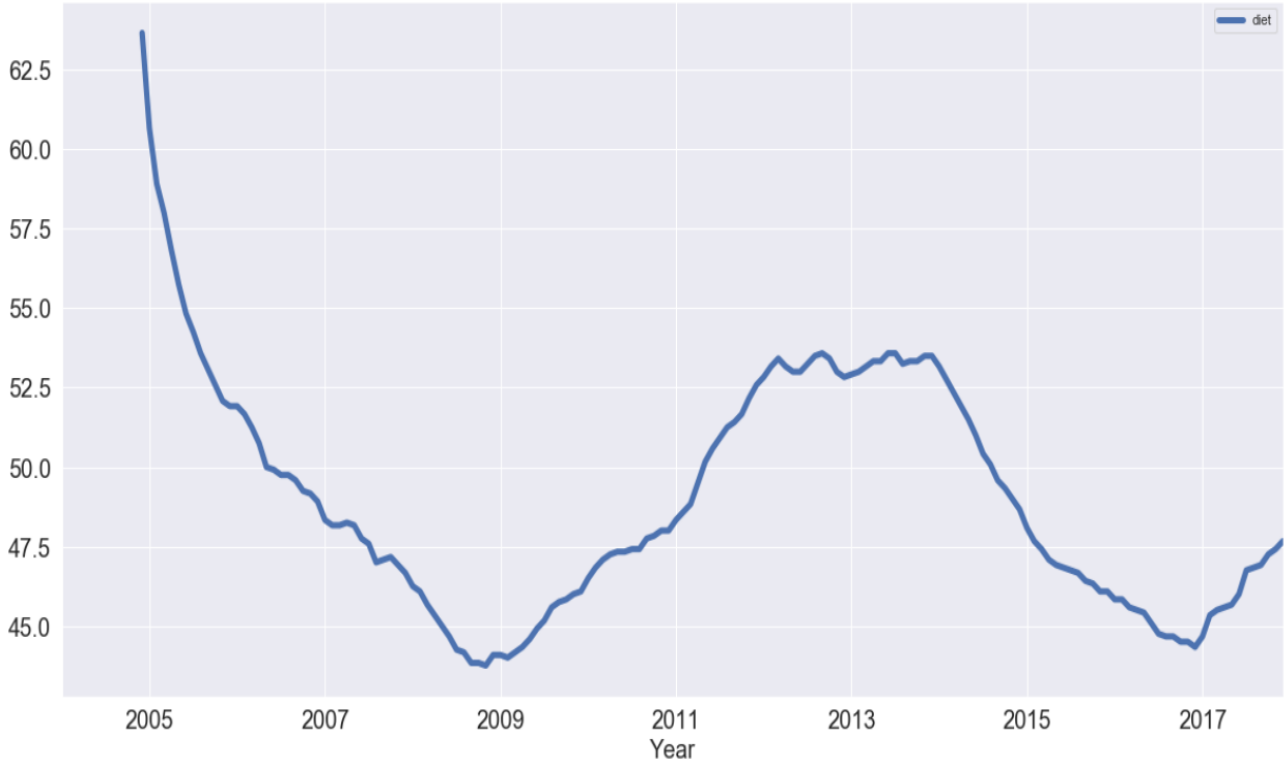



# The golden rule: Stationarity

Before going any further into our analysis, our series has to be made ***stationary.***
***Stationarity*** is the property of exhibiting constant statistical properties (mean, variance, autocorrelation, etc.). If the mean of a time-series increases over time, then it's not stationary.

Rolling means (or moving averages) are generally used to smooth out short-term fluctuations in time series data and highlight long-term trends

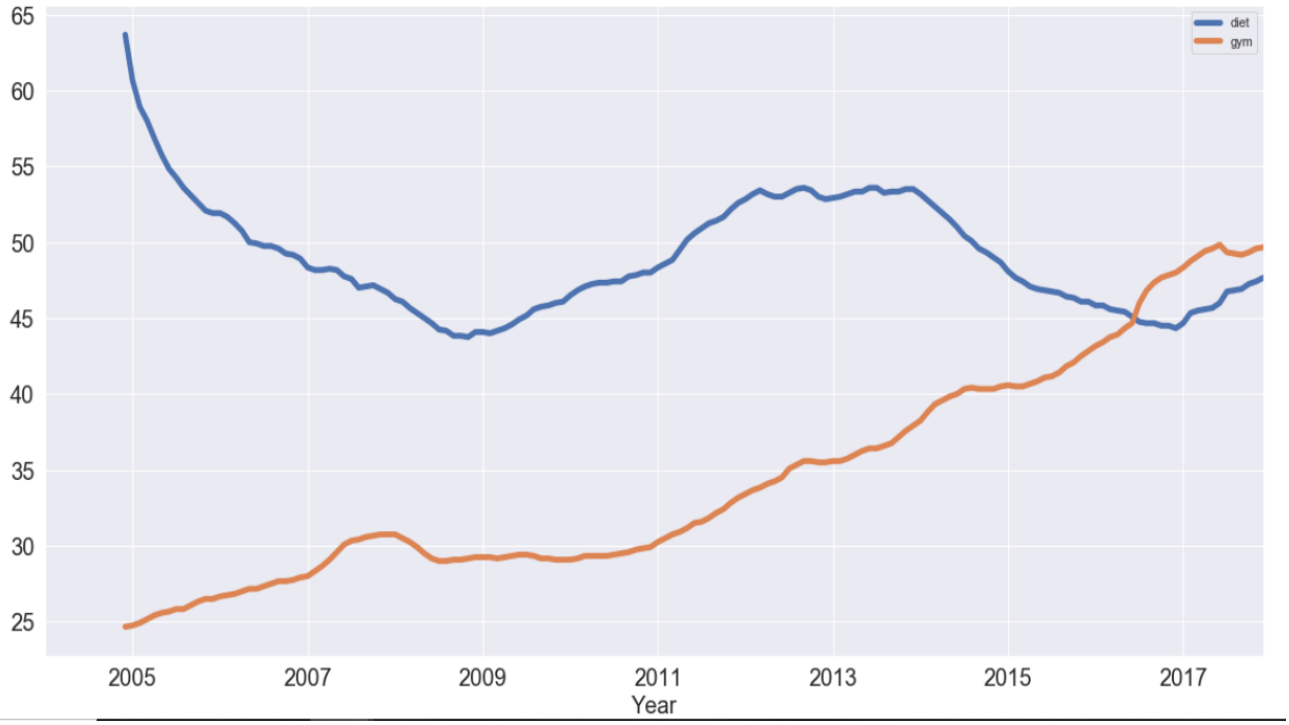
```
In [10]: diet = df[['diet']]
         diet.rolling(12).mean().plot(figsize=(20,10), linewidth=5, fontsize=20)
         plt.xlabel('Year', fontsize=20); #smoothning on year basis
```




Plotting rolling means and variances is a first good way to visually inspect our series. If the rolling statistics exhibit a clear trend (upwards or downwards) and show varying variance (increasing or decreasing amplitude), then you might conclude that the series is very likely not to be stationary.

***Concatenating two trends in one single plot:***

```
In [12]: df_rm = pd.concat([diet.rolling(12).mean(), gym.rolling(12).mean()], axis=1)
         df_rm.plot(figsize=(20,10), linewidth=5, fontsize=20)
         plt.xlabel('Year', fontsize=20);
```




Pandas dataframe.corr() is used to find the pairwise correlation of all columns in the data frame. Any NA values are automatically excluded. For any non-numeric data type columns in the data frame it is ignored.

```
In [13]: df.corr()
```

Out[13]:

| | diet | gym | finance |
|---|---|---|---|
| **diet** | 1.000000 | -0.100764 | -0.034639 |
| **gym** | -0.100764 | 1.000000 | -0.284279 |
| **finance** | -0.034639 | -0.284279 | 1.000000 |

***What we have to do now is plot the first-order differences of these time series and then compute the correlation of those because that will be the correlation of the seasonal components, approximately. Remember that removing the trend may reveal correlation in seasonality.***

```
In [15]: df.diff().plot(figsize=(20,10), linewidth=5, fontsize=20)
         plt.xlabel('Year', fontsize=20);
```


Calculates the difference of a Data Frame element compared with another element in the Data Frame

Pandas **dataframe.diff()** is used to find the first discrete difference of objects over the given axis

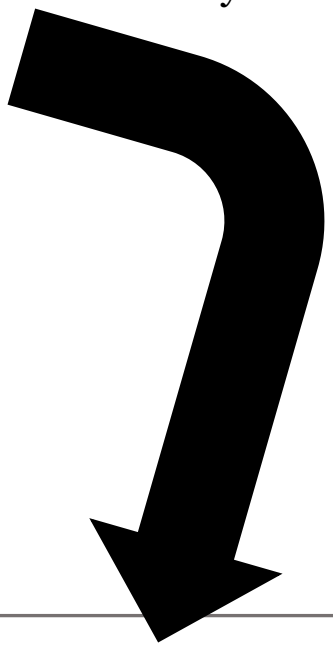
# *Periodicity and Autocorrelation*

A time series is periodic if it repeats itself at equally spaced intervals, say, every 12 months.

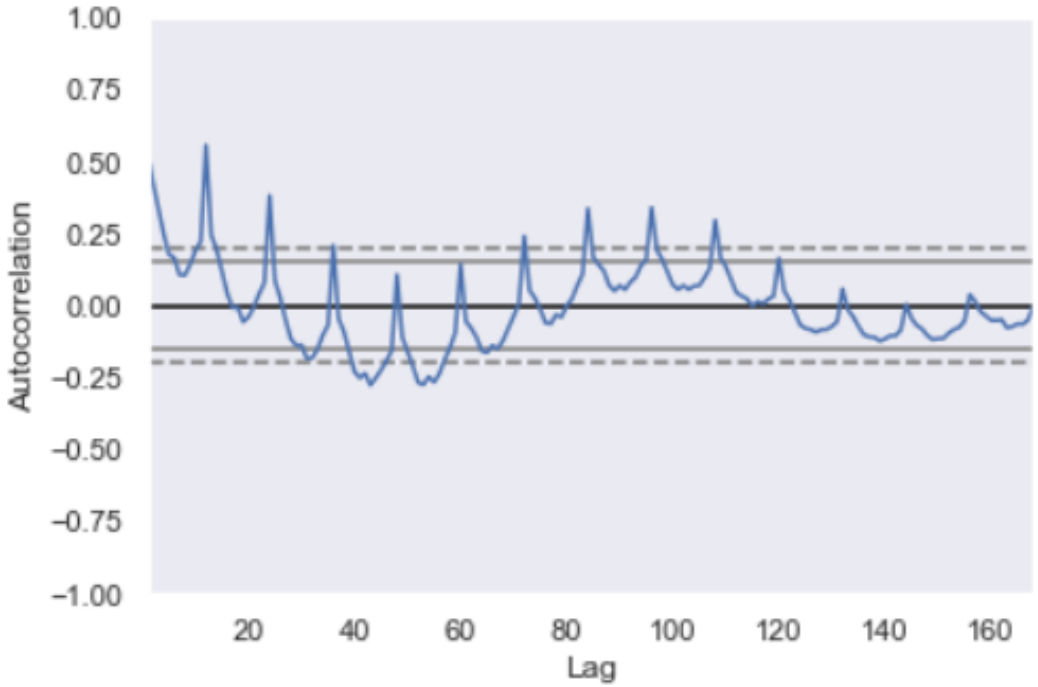
An *autocorrelation* (ACF) plot represents the autocorrelation of the series with lags of itself.
A *partial autocorrelation* (PACF) plot represents the amount of correlation between a series and a lag of itself that is not explained by correlations at all lower-order lags.

These are plots that graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps.

Auto correlation measures a set of current values against a set of past values and finds whether they correlate. Auto correlation is the correlation of one time series data to another time series data which has a time lag. Auto correlation varies from +1 to -1. An auto correlation of +1 indicates that if the time series one increases in value the time series 2 also increases in proportion to the change in time series 1. An auto correlation of -1 indicates that if the time series one increases in value the time series 2 decreases in proportion to the change in time series 1. Auto correlation has its applications in signal processing, technical analysis of stocks.



```
In [17]: pd.plotting.autocorrelation_plot(diet); #auto correlation
```




The dotted lines in the above plot actually tell you about the statistical significance of the correlation.

## Time series forecasting ARIMA

Autoregressive Integrated Moving Average Model

An ARIMA model is a class of statistical models for forecasting time series data.

AR: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations.

I: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

MA: Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

*A standard notation is used of ARIMA(p, d, q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.*

*p: The number of lag observations included in the model, also called the lag order.*

*d: The number of times that the raw observations are differenced, also called the degree of differencing.*

*q: The size of the moving average window, also called the order of moving average.*

## Loading the data:

```
In [19]: series = pd.read_csv("C:/Users/sahibjot/Desktop/shampoo.csv")
```

```
In [20]: series.head()
```

Out[20]:

| | Month | Sales |
|---|---|---|
| 0 | 1-01 | 266.0 |
| 1 | 1-02 | 145.9 |
| 2 | 1-03 | 183.1 |
| 3 | 1-04 | 119.3 |
| 4 | 1-05 | 180.3 |

```
In [21]: series.shape
```

```
Out[21]: (36, 2)
```

In [24]:
```
def parser(x):
    return datetime.strptime('200'+x, '%Y-%m')

series = pd.read_csv("C:/Users/sahibjot/Desktop/shampoo.csv", parse_dates=[0], index_col=0, date_parser=parser)
series.head()
```

Out[24]:

| Month | Sales |
|---|---|
| 2001-01-01 | 266.0 |
| 2001-02-01 | 145.9 |
| 2001-03-01 | 183.1 |
| 2001-04-01 | 119.3 |
| 2001-05-01 | 180.3 |

In [25]:
```
series.shape
```

Out[25]: (36, 1)

# Plot:

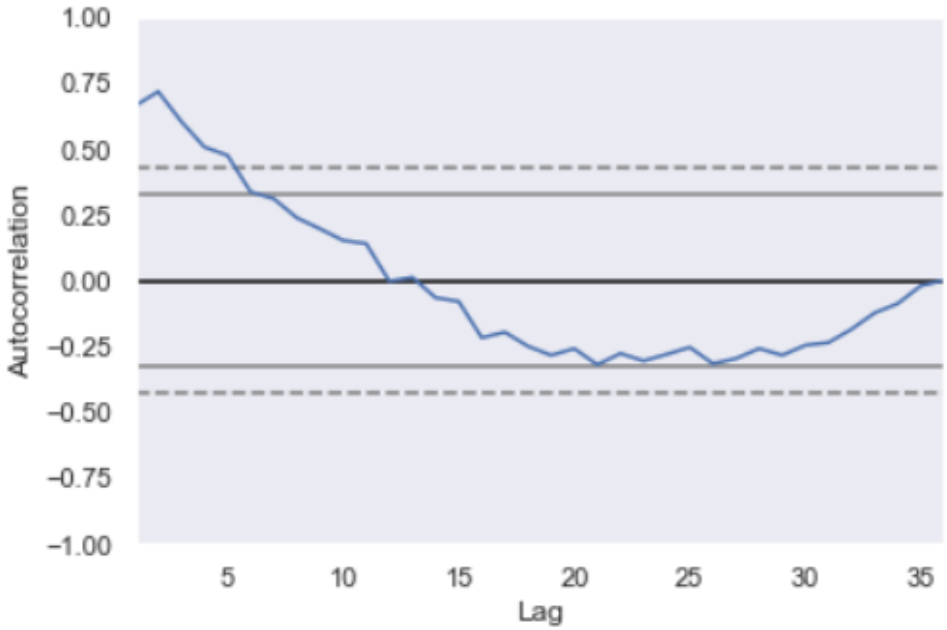
In [26]:
```
series.plot()
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1f738cb8b70>

## Autocorrelation plot:

```
In [27]: pd.plotting.autocorrelation_plot(series)
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1f738298b38>
```




## Arima modelling:

```
In [28]: #ARIMA modelling
         X = series.values
```

```
In [29]: size = int(len(X) * 0.66)
         train, test = X[0:size], X[size:len(X)]
```

```
In [30]: history = [x for x in train]
```

```
In [34]: history
```

```
Out[34]: [array([266.]),
          array([145.9]),
          array([183.1]),
          array([119.3]),
          array([180.3]),
          array([168.5]),
          array([231.8]),
          array([224.5]),
          array([192.8]),
```

The statsmodels library provides the capability to fit an ARIMA model.

An ARIMA model can be created using the statsmodels library as follows:

1. Define the model by calling ARIMA() and passing in the *p*, *d*, and *q* parameters.
2. The model is prepared on the training data by calling the fit() function.
3. Predictions can be made by calling the predict() function and specifying the index of the time or times to be predicted.

In [35]:
```
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
```

In [36]:
```
predictions = list()
for t in range(len(test)):
    model = ARIMA(history, order=(5,1,0))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
```

```
predicted=564.213536, expected=342.300000

predicted=798.107324, expected=339.700000
predicted=439.191945, expected=440.400000
predicted=411.660284, expected=315.900000
predicted=394.859912, expected=439.300000
predicted=388.826092, expected=401.300000
predicted=403.105102, expected=437.400000
predicted=434.223386, expected=575.500000
predicted=463.590894, expected=407.600000
predicted=521.815170, expected=682.000000
predicted=512.730097, expected=475.300000
predicted=587.993285, expected=581.300000
predicted=581.042919, expected=646.900000
```

# *Plotting the predictions:*

```
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
# plot
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```

```
Test MSE: 6958.324
```


A line plot is created showing the expected values (blue) compared to the rolling forecast predictions (red).