An ARIMA is a class of statistical models for analyzing and forecasting time series data. ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. Briefly they are-

**AR**: *Autoregression*. A model that shows dependent relationship between an observation and some number of lagged observation

**I**: *Integrated*. It uses 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.

Each of these components are explicitly specified in the model as a parameter. 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.

An ARIMA model consists of coordinates (p, d, q):

* **p** stands for the number of autoregressive terms, i.e. the number of observations from past time values used to forecast future values. e.g. if the value of p is **4**, then this means that four previous time observations in the series are being used to forecast the future trend.
* **d** denotes the number of differences needed to make the time series stationary (i.e. one with a constant mean, variance, and autocorrelation). For instance, if d = 1, then it means that a first-difference of the series must be obtained to transform it into a stationary one.
* **q** represents the moving average of the previous forecast errors in our model, or the lagged values of the error term. As an example, if q has a value of 1, then this means that we have one lagged value of the error term in the model.

**Implementing ARIMA with statsmodels in Python**

For this model we have to do as below

* **1. Load Libraries**

Firstly, we load our libraries .

import pandas

import matplotlib.mlab as mlab

import matplotlib.pyplot as plt

import numpy as np

import math

from statsmodels.tsa.stattools import acf, pacf

import statsmodels.tsa.stattools as ts

from statsmodels.tsa.arima\_model import ARIMA

* **2. Import csv and define “Close” as closing price variable using pandas**

variables = pandas.read\_csv('jnj.csv')

close = variables['close']

* **3. Autocorrelation and Partial Autocorrelation Plots**

We use the **ACF and PACF (Autocorrelation and Partial Autocorrelation)**readings to determine whether our data is stationary upon differencing. The autocorrelation and partial autocorrelation function both measure, to varying degrees, the correlation coefficient between a series and lags of the variable over time. An autoregressive process is when a time series follows a particular pattern in that its present value is in some way correlated to its past value(s). For instance, if we are able to use regression analysis to discern the present value of a variable from using its past value, then we refer to this as an AR(1) process:

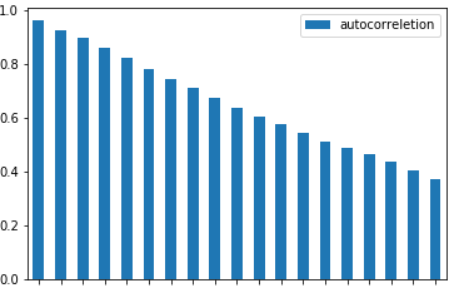
* Xt = ß0 + ß1X(t-1) + et

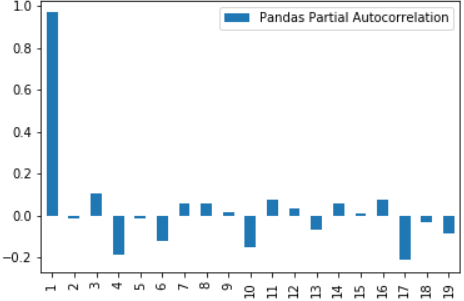
However, there are some instances in which the present value of a variable can be determined from the past two or three values, which would incorporate an AR(2) or AR(3) process respectively:

* Xt = ß0 + ß1X(t-1) + ß2X(t-2) + et
* Xt = ß0 + ß1X(t-1) + ß2X(t-2) + ß3X(t-3) + et

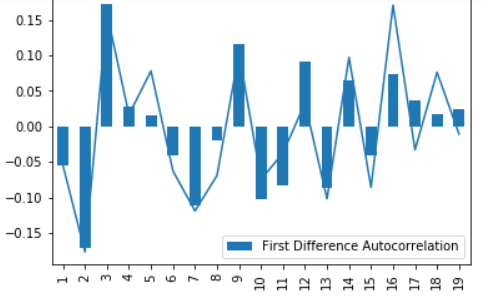
Now we can generate the acf and pacf plots:

We see that statsmodels produces the autocorrelation and partial autocorrelation plots:

* 

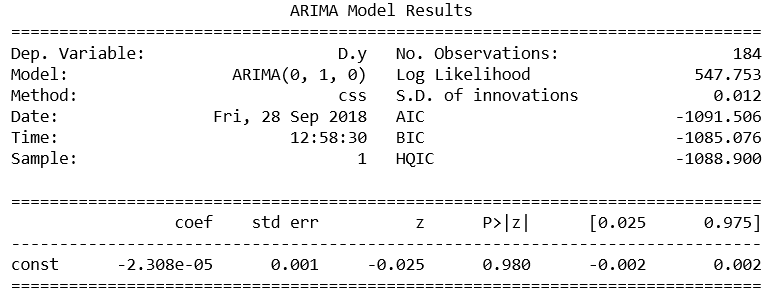


* Moreover, we have confirmation that our data follows an AR(1) stationary process (one with a constant mean, variance, and autocorrelation), and we see that the price plot now shows a stationary process:



**4. ARIMA Model Generation**

* price\_matrix=lnprice.as\_matrix()
* model = ARIMA(price\_matrix, order=(0,1,0))
* model\_fit = model.fit(disp=0)
* print(model\_fit.summary())

Using the (0,1,0) configuration, our ARIMA model is generated:  
   


As previously mentioned, our data is in logarithmic format. Since we are analysing stock price, this format is necessary to account for compounding returns. However, once we have obtained the forecasts (for seven periods out in this case), then we can obtain the real price forecast by converting the logarithmic figure to an exponent:

* predictions=model\_fit.predict(122, 127, typ='levels')
* predictions
* predictionsadjusted=np.exp(predictions)
* predictionsadjusted

