Time Series Modeling

Seasonal ARIMA Model

Contents

Seasonal Box-Jenkins (ARIMA) Models

- ARIMA (Auto Regressive Integrated Moving Average) models are Regression models that use lagged values of the dependent variable and/or random disturbance term as explanatory variables.

- Seasonal ARIMA (Often abbreviated as SARIMA) Model is formed by including seasonal terms in the ARIMA model.

- Several real world time series have a seasonal component. Some examples are: Sales of woolen clothes, demand for fertilizers, electricity consumption, etc.

Seasonal Box-Jenkins (ARIMA) Models

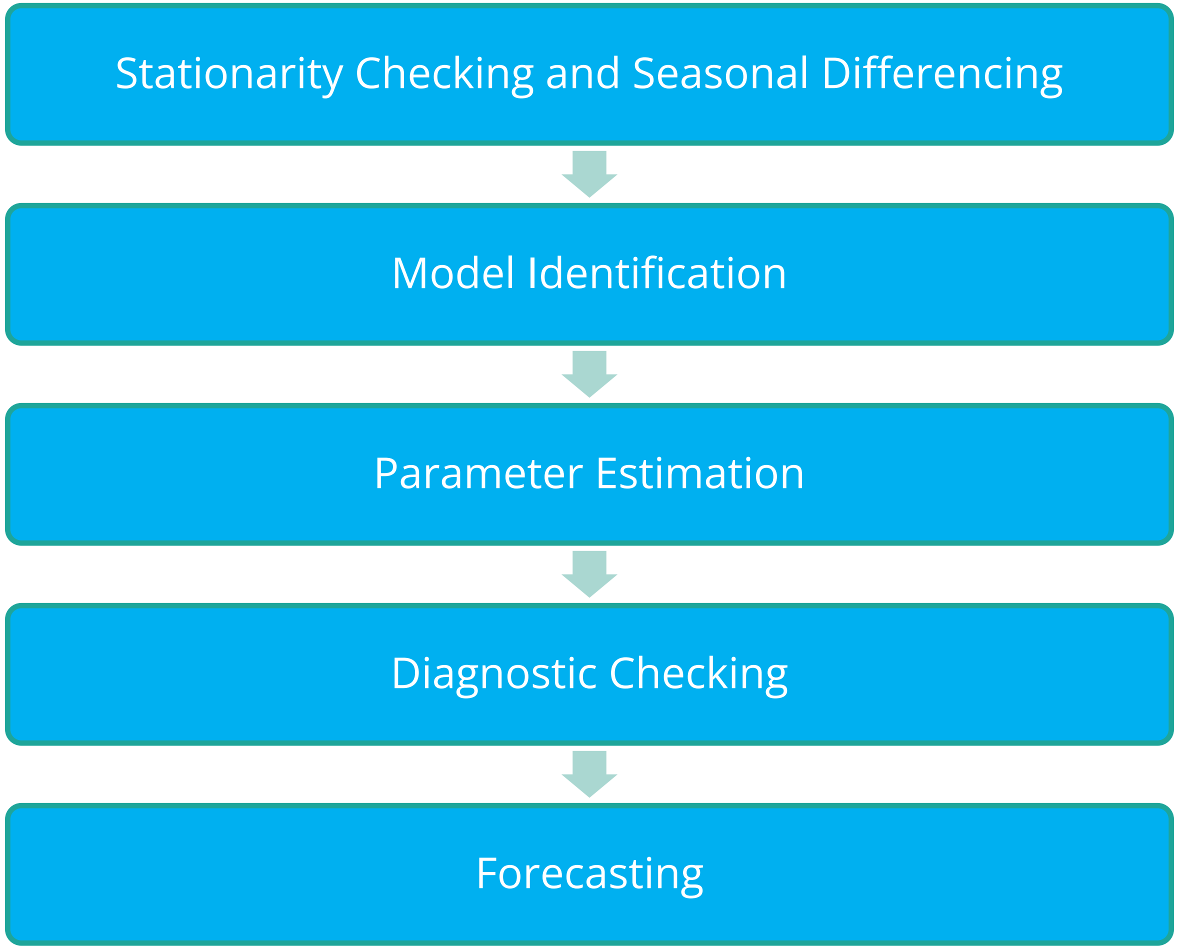
- The **seasonal ARIMA model** incorporates both **non-seasonal and seasonal** factors in a multiplicative model.

- Shorthand notation for the model is,

$$\text{ARIMA } (p, d, q) \times (P, D, Q)S,$$

with,
p = non-seasonal AR order,
d = non-seasonal differencing,
q = non-seasonal MA order,
P = seasonal AR order,
D = seasonal differencing,
Q = seasonal MA order, and
S = time span of repeating seasonal pattern.

Five-Step Iterative Procedure



Step 1: Stationarity Checking

Assessing Stationarity of Time Series

- Stationarity of a time series can be assessed using:

Time Series Plot (Time v/s Variable) OR Correlogram OR Dickey-Fuller Test

- If a time series is non-stationary then it can be converted via

Differencing

De-trending

* Concept of Stationarity is explained previously in 'Stationarity in Time Series' ppt in detail.

Seasonal Differencing

- Seasonal differencing is denoted as ,

$$\boxed{\Delta_s y_t = y_t - y_{t-s}}$$

Where,

s denotes **frequency of season**

s = 12 if data is monthly; s = 4 if data is quarterly and so on

- First and seasonal span differencing for monthly data is,

$$\boxed{\Delta_1 \Delta_s y_t = \Delta_1 (y_t - y_{t-s}) = y_t - y_{t-1} - y_{t-s} + y_{t-s-1}}$$

Case Study

Background

- Sales Data for 3 Years (2013, 2014, 2015)

Objective

- To fit a Seasonal ARIMA Model and forecast next 3 Months sales.

Available Information

- Sample size is 36
- Variables: Year, Month, Sales

Data Snapshot

Sales Data for 3 Years

Variables

Monthly Observations

Year	Month	Sales
2013	Jan	123
2013	Feb	142
2013	Mar	164
2013	Apr	173
2013	May	183
2013	Jun	192
2013	Jul	199
2013	Aug	203
2013	Sep	207
2013	Oct	209
2013	Nov	214
2014	Jul	245

Columns	Description	Type	Measurement	Possible values
Year	Year	numeric	2013, 2014, 2015	3
Month	Month	character	Jan - Dec	12
Sales	Sales in USD Million	numeric	USD Million	Positive values

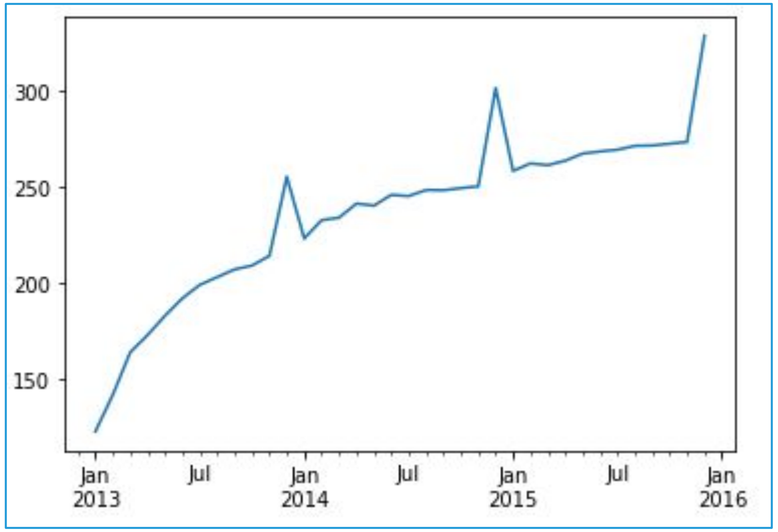
Plotting a Time Series in Python

```
# Importing the Data
import pandas as pd
salesdata = pd.read_csv('Sales Data for 3 Years.csv')
#Creating and Plotting a Time Series Object
rng = pd.date_range('01-01-2013','31-12-2015',freq='M')

s = salesdata.Sales.values
salesseries = pd.Series(s, rng)

salesseries.plot()
```

\# Output



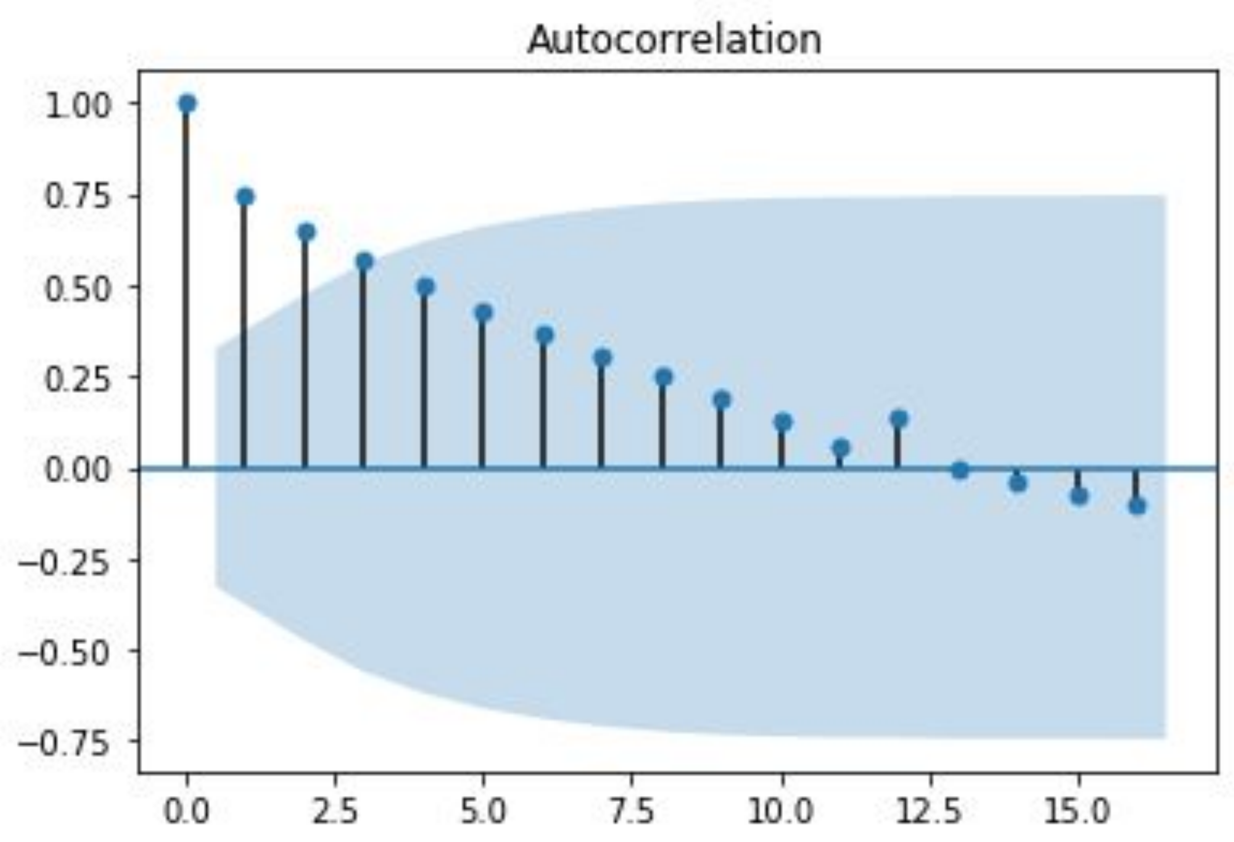
Interpretation :

- The time series shows periodic peaks, indicative of seasonality.

Correlogram

```
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(salesseries)
```

```
# Output
```



Interpretation :

- ACF plot shows a slow decay indicating non-stationarity.

Dickey Fuller Test

```
# Dickey Fuller Test
```

```
from arch.unitroot import ADF

adf = ADF(salesseries,lags=0,trend='nc')
adf.summary()
```

```
# Output
```

```
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                  1.621
P-value                         0.975
Lags                                0
-------------------------------------

Trend: No Trend
Critical Values: -2.63 (1%), -1.95 (5%), -1.61 (10%)
Null Hypothesis: The process contains a unit root.
Alternative Hypothesis: The process is weakly stationary.
```

Interpretation :

- Time series is non-stationary. Value of test statistic is greater than 5% critical value.

Dickey Fuller Test – Differenced Series

```
# Dickey Fuller Test for Difference Series
```

```
from statsmodels.tsa.statespace.tools import diff
salesdiff = diff(salesseries)
(ADF(salesdiff,lags=0,trend='nc')).summary()
```

```
# Output
```

```
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                 -6.891
P-value                         0.000
Lags                                0
-------------------------------------

Trend: No Trend
Critical Values: -2.63 (1%), -1.95 (5%), -1.61 (10%)
Null Hypothesis: The process contains a unit root.
Alternative Hypothesis: The process is weakly stationary.
```

Interpretation :

- Time series is stationary. Value of test statistic is less than 5% critical value.

Step 2: Model Identification

Model Identification

- When the data are confirmed stationary, proceed to tentative identification of models through **visual inspection of correlogram and partial correlogram**

Model	AC	PAC
	Dies down	Cuts off after lag p
	Cuts off after lag q	Dies down
	Dies down	Dies down

Model Identification

- Seasonal ARIMA model is expressed as arima(p,d,q) (P,D,Q) where
  - p = no. of autoregressive terms
  - d = order of differencing
  - q = no. of moving average terms
  - (P,D,Q) are seasonal equivalents of autoregressive, difference and moving average terms

Fig. 3: ACF Plot

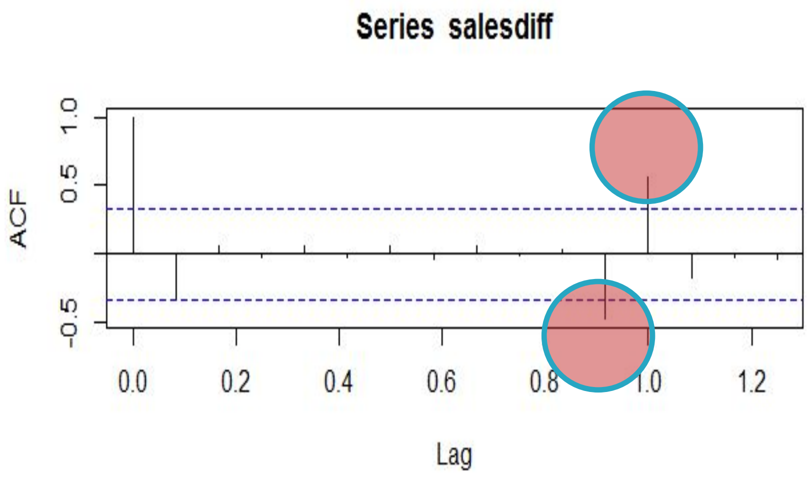
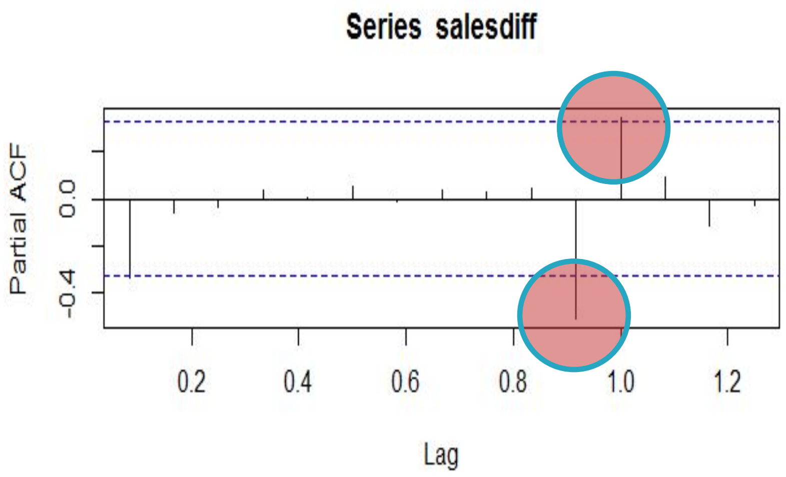


Fig. 4: PACF Plot



<u>Indicative Model :</u>
arima(2,1,2)(2,1,2)

Step 3: Parameter Estimation

Parameter Estimation

- There are two ways in which parameters of arima models can be estimated

1. Ordinary Least Squares
2. Maximum Likelihood Method – when the model involves MA component

- Given n observations $y_1, y_2, \ldots, y_n$, the likelihood function **L is defined as - the probability of obtaining the data actually observed**
- The maximum likelihood estimators **(MLE) are those values of the parameters for which the data actually observed are most likely**, that is, the values that maximize the likelihood function L.

Parameter Estimation in Python

```
# Automatic Model Identification and Parameter Estimation
import pmdarima as pm
model = pm.auto_arima(salesseries,
                      max_p=2, max_q=2,
                      max_P=2,max_Q=2,
                      d=1,m=12,
                      seasonal=True,
                      D=1, suppress_warnings=True,
                      trace=True)

model
```

- **auto_arima()** generates the best order arima model. The function conducts a search over possible model within the order constraints provided.
- Seasonal model requires **max_D,max_P** and **max_Q** arguments as well.
- **trace=** True returns the list of all models considered.
- D= gives order of seasonal differencing

Automatic Model Identification

```
# Output
```

```
Fit ARIMA: order=(2, 1, 2) seasonal_order=(1, 1, 1, 12); AIC=157.695, BIC=166.779, Fit
time=1.214 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 12); AIC=152.639, BIC=154.910, Fit
time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 12); AIC=156.618, BIC=161.160, Fit
time=0.140 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 12); AIC=156.623, BIC=161.165, Fit
time=0.118 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 12); AIC=156.110, BIC=157.245, Fit
time=0.017 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 12); AIC=154.629, BIC=158.035, Fit
time=0.089 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 12); AIC=154.629, BIC=158.035, Fit
time=0.056 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 12); AIC=156.629, BIC=161.171, Fit
time=0.127 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 12); AIC=154.633, BIC=158.040, Fit
time=0.068 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 12); AIC=154.636, BIC=158.042, Fit
time=0.067 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 12); AIC=155.485, BIC=160.027, Fit
time=0.115 seconds
Total fit time: 2.118 seconds
```

Interpretation :

- Model with the lowest AIC value is selected as the best model.

```
ARIMA(order=(0, 1, 0), seasonal_order=(0, 1, 0, 12))
```

* Do note that the model identified by visually analysing correlograms was different. This shows it is not safe to rely just on visual review.

ARIMA Model in Python

```
# Obtaining Coefficient
```

```
salesseries = pd.to_numeric(salesseries.astype(float))

from statsmodels.tsa.statespace.sarimax import SARIMAX
salesmodel = SARIMAX(salesseries, order=(0,1,0),
seasonal_order=(0,1,0,12)).fit(trend='nc')

salesmodel.params
salesmodel.aic
```

```
# Output
```

```
sigma2     47.585895
dtype: float64
```

```
# Output
```

```
156.10960236428923
```

Model Selection Criteria

- **Akaike Information Criterion (AIC)**

$$\boxed{AIC = -2\ \ln(L) + 2k}$$

- **Schwartz Bayesian Criterion**

  (SBC, also called Bayesian Information Criterion - BIC)

$$\boxed{SBC = -2\ \ln(L) + k\ \ln(n)}$$

where L = Likelihood function

k = Number of parameters to be estimated

n = Number of observations

<u>Ideally, the AIC and SBC should be as small as possible</u>

Step 4: Diagnostic Checking

Residual Analysis

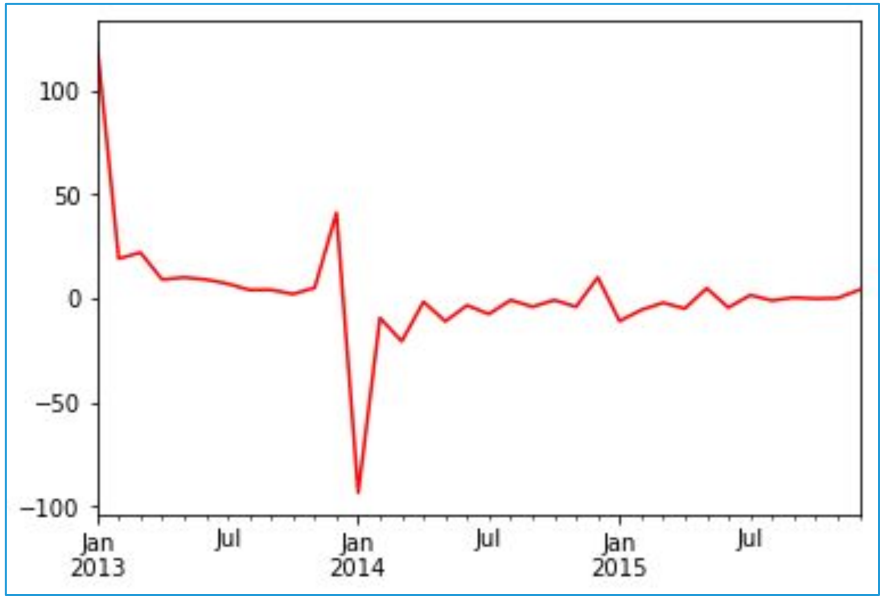
If an ARMA(p,q) model is an adequate representation of the data generating process then the residuals should be 'White Noise'

- White Noise time series has **zero mean, constant variance and zero covariance with lagged time series.**
- **Residual plot is used for checking if the residuals are white noise process.**

Residual Plot In Python

```
resi = salesmodel.resid
resi.plot(color="red")
```

Output



Step 5: Forecasting

Forecasting

```
# Forcast for next 3years
salesmodel.forecast(steps=3)
```

forecast() function is used to yield forecasts for a time series

```
# Output
```

```
2016-01-31     285.0
2016-02-29     288.9
2016-03-31     288.0
Freq: M, dtype: float64
```

Quick Recap

Stationarity Checking	• Plot correlogram using **`plot_acf()`** and validate stationarity using **`ADF()`**
Model Identification	• Tentative identification of models through visual inspection of correlogram and partial correlogram
Parameter Estimation	• **`auto_arima()`** is recommended for obtaining best ARIMA model & **`SARIMAX()`** for fitting the best model • It uses AIC as the model selection criteria
Diagnostic Checking	• **`Residual plot`** for checking whether errors follow white noise process
Forecasting	• Use **`forecast()`** to generate forecasts