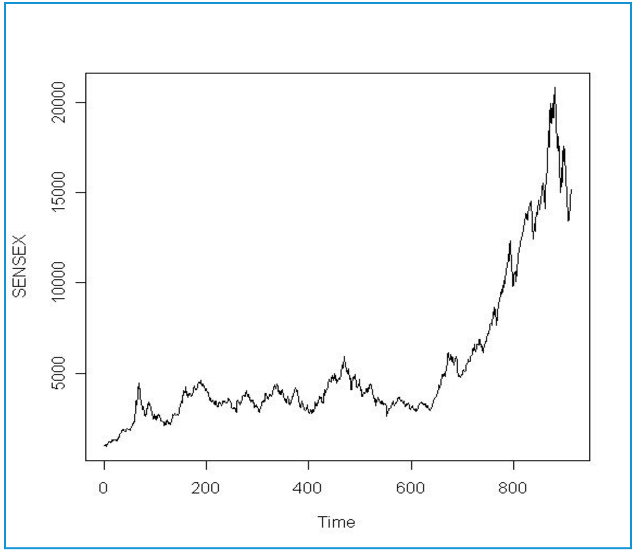
Introduction to

Time Series Analysis

Contents

What is Time Series ?

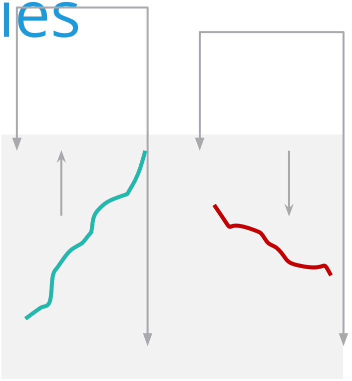
Time Series is a sequence of values observed over time

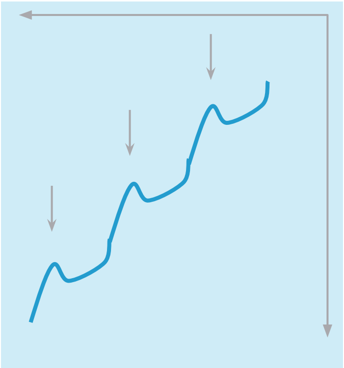


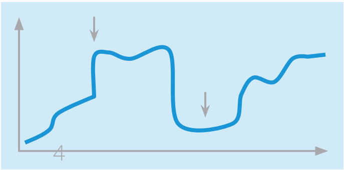
Types of Time Scale	
Discrete	Value changes after jumping from one time period to other. Example: Dow Jones Index -End of Day Values
Continuous	Value changes within an infinitely short amount of time. Example. Temperature, Dow Jones Index Tracked Real Time

Components of Time Series

Trend	Long-term increase or decrease in the time series. There may be increase/decrease in short term but overall trend in the long term can be increasing or decreasing.
Seasonality	**Predictable and recurring trends and patterns** over a period of time, normally a year. An example of a seasonal time series is retail data, which sees spikes in sales during holiday seasons like Christmas. Seasonality is reflected only when data is available for more than one year
Cyclic Pattern	Exists when data exhibit rises and falls that are not of fixed period. The duration of these fluctuations is usually of at least 2 years







Application Areas

Industry	Model/Predict	Based on Information such as:	Purpose
Finance	Price of a Stock	• Recent price movement of the stock	Forecasting
Economics	Inflation Rates	• Trend and seasonality in inflation rates	Forecasting
Retail /FMCG	Monthly Sales	• Location, marketing expenses on TV, print and online media	Predictive and Optimization

Case Study

Background

- Annual Sales for a specific company from year 1961 to 2017

Objective

- To plot a time series object

Available Information

- Number of cases: 57
- Variables: Year, sales(in 10's GBP)

Data Snapshot

turnover_annual data

Variables

Observations on Discrete Time Scale

Year	sales
1961	224786
1962	230034
1963	236562
1964	250960
1965	261615
1966	268316
1967	283589
1968	280160
1969	301422
1970	308018
1971	[illegible]
1974	364834
1975	392503

Columns	Description	Type	Measurement	Possible values
Year	Financial Year	Numeric	-	-
sales	sales(in 10's GBP)	Numeric	In British Pound	Positive values

Time Series in Python

```
# Import turnover_annual Data
```

```
import pandas as pd
salesdata = pd.read_csv('turnover_annual.csv')
```

```
# Creating a Time Series Object
```

```
rng = pd.date_range('01-01-1961','31-12-2017',freq='Y')
s = salesdata.sales.values
salesseries = pd.Series(s, rng)
```

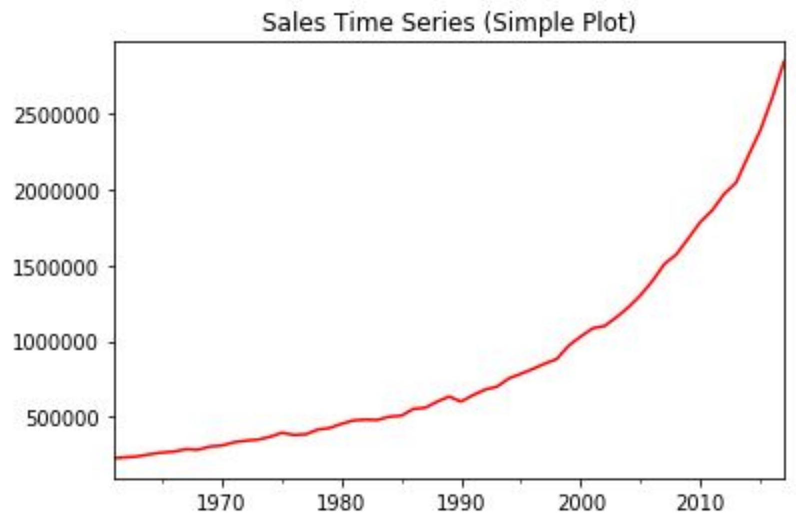
- **date_range()** creates pandas date object.
- When the time series has seasonal components, argument **freq =** can be included. It denotes number of observations per unit of time. Eg. If data is quarterly: **freq = 'Q'**, if data is monthly: **freq = 'M'**.
- **pd.Series()** combines time series variable object "s" and date object "rng".
  The new object salesseries will be used for further analysis.

Plotting Time Series in Python

```
# Plotting a Time Series Object
salesseries.plot(color='red', title ="Sales Time Series (Simple Plot)")
```

plot() generates a simple line chart.

```
# Output
```



Interpretation :

- The time-series clearly shows upward trend.

Subsetting Time Series in Python

- Large volumes of data are required for most real world analytics, time series is no exception.
- Subsetting is an important tool as it facilitates partitioning the data within Python for micro-level specific analysis.

```
# Subsetting a Time Series Object
salesseries2 = salesseries.loc['1990-12-31':'2016-12-31']
```

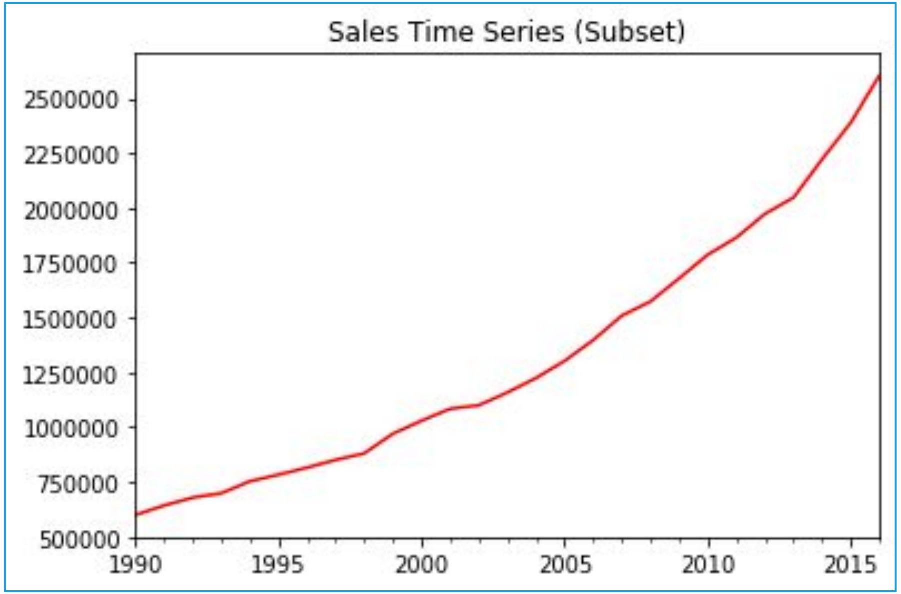
loc[ ] is a generic function which extracts the subset of the object x observed between the times **specified within the range.**

* In Pandas we can directly subset the time series object using `loc[]` function

Subsetting Time Series in Python

```
salesseries2.plot(color='red', title ="Sales Time Series (Subset)")
```

```
# Output
```



Plot from 1990 to 2016 shows increasing trend

Quick Recap

Time Series	• Sequence of values measured over time • Time scale can be discrete or continuous
Time Series in Analysis	• Analyze and forecast time series values
Time Series in Python	• pd,date_range created date object. • pd.Series creates time series object by combining date object and time series variable

Time Series Decomposition

Contents

Components of Time Series

- As we know, Trend and Seasonality are main components of Time Series.
- If we assume an additive model, we can write

where,

$$Y_t = S_t + T_t + R_t$$

Y_t : Time series value at period t

S_t : Seasonal component at period t

T_t : Trend-cycle component at period t

R_t : Remainder (or irregular or error) component at period t

- Alternatively, a multiplicative model would be written as

$$Y_t = S_t * T_t * R_t$$

* Additive model is most appropriate if the magnitude of the seasonal fluctuations or the variation around the trend-cycle does not vary with the level of the time series.

Understanding Moving Averages

- Moving Averages are **averages calculated for consecutive data from overlapping subgroups of fixed length**
- Moving averages smoothen a time series by filtering out random fluctuations

Day	End of Day Sales
1	1500
2	2100
3	1750
4	1900
5	1650

Moving Average of Period 3
NA
NA
(1500+2100+1750)/3=1783.33
(2100+1750+1900)/3=1916.67
(1750+1900+1650)/3=1766.67

The first 2 MA values for length 3 are not calculated

- Period of the moving average depends on type of data
- **Non-seasonal data**: Shorter length (Typically 3 period or 5 period MA is considered)
- **Seasonal data**: Typical period is 12 for monthly data and 4 for quarterly data

Time Series Decomposition – Simple Method

Decomposition is a statistical method that deconstructs a time series.

Steps to follow :

Find Trend	Obtain moving averages covering one season – This provides trend component of the time series
Eliminate Trend	Eliminate trend component from original time series. Calculate $Y_t - T_t$
Estimate Seasonality	To estimate the seasonal component for a given time period, simply average the de-trended values for that time period. These seasonal indexes are then adjusted to ensure that they add to zero

The remainder component is calculated by subtracting the estimated seasonal and trend-cycle components

Time Series Decomposition – Simple Method

Suppose we have monthly time series data, for three years 2014, 2015 and 2016:

Step 1 □ Calculate Moving Averages

…	Dec 14	Jan 15	Feb 15	Mar 15	Apr 15	May 15	Jun 15	Jul 15	Aug 15	Sep 15	Oct 15	Nov 15	Dec 15	Jan 16	…

(Consider moving average period of 13 - previous 6 months, next 6 months and current month to calculate moving average of current month)

This gives the trend component T_t

Step 2 □ Eliminating Trend

Remove T_t from the original time series Y_t

Step 3 □ Estimate Seasonal Component

The seasonal index for July is the average of all the de-trended July values in the data i.e. Average of De-trended July 2014, July 2015 and July 2016

Case Study

Background

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

Objective

- To decompose time series into its components and study each component separately.

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

Time Series Decomposition in Python

```
# Simple Decomposition
import pandas as pd
salesdata = pd.read_csv("Sales Data for 3 Years.csv")
rng = pd.date_range('01-01-2013','31-12-2015',freq='M')
s = salesdata.Sales.values
salesseries = pd.Series(s, rng)
```

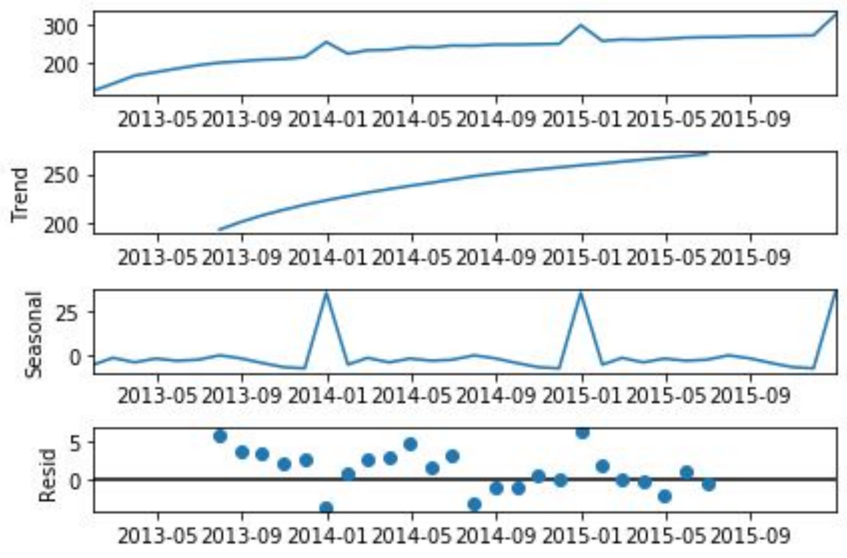
- **freq =** “M” indicates monthly date data.
- **pd.Series()** creates time series object using date object as index

```
import statsmodels.api as sm
decomp = sm.tsa.seasonal_decompose(salesseries.interpolate())
decomp.plot()
```

- **tsa.seasonal_decompose()** performs a classical seasonal decomposition through moving averages.
- **plot()** of decompose object gives a 4-level visual representation.

Time Series Decomposition in Python

```
# Output
```



Trend is not estimated for first/last few values
Seasonal Component repeats from year to year

* The **`tsa.seasonal_decompose()`** function takes time series as additive by default. If you wish to create a multiplicative time series, simply add **`model="multiplicative"`** in the function.

Decomposition in Python – Seasonal Component

```
#Analyzing the decomp object. Each component can be separately viewed
```

```
decomp.seasonal
```

2013-01-31	-5.006944
2013-02-28	-1.181944
2013-03-31	-3.736111
2013-04-30	-1.588194
2013-05-31	-2.831944
2013-06-30	-2.231944
2013-07-31	0.234722
2013-08-31	-1.525694
2013-09-30	-4.094444
2013-10-31	-6.446528
2013-11-30	-7.127778
2013-12-31	35.536806
2014-01-31	-5.006944
2014-02-28	-1.181944
2014-03-31	-3.736111
2014-04-30	-1.588194
2014-05-31	-2.831944
2014-06-30	-2.231944
2014-07-31	0.234722
2014-08-31	-1.525694
2014-09-30	-4.094444
2014-10-31	-6.446528
2014-11-30	-7.127778
2014-12-31	35.536806
2015-01-31	-5.006944
2015-02-28	-1.181944
2015-03-31	-3.736111
2015-04-30	-1.588194
2015-05-31	-2.831944
2015-06-30	-2.231944
2015-07-31	0.234722
2015-08-31	-1.525694
2015-09-30	-4.094444
2015-10-31	-6.446528
2015-11-30	-7.127778

Interpretation :

- This table shows seasonal component of time series

Decomposition in Python – Trend Component

```
decomp.trend
```

2013-01-31	NaN
2013-02-28	NaN
2013-03-31	NaN
2013-04-30	NaN
2013-05-31	NaN
2013-06-30	NaN
2013-07-31	192.833333
2013-08-31	200.770833
2013-09-30	207.450000
2013-10-31	213.195833
2013-11-30	218.408333
2013-12-31	223.016667
2014-01-31	227.166667
2014-02-28	230.962500
2014-03-31	234.550000
2014-04-30	237.929167
2014-05-31	241.100000
2014-06-30	244.516667
2014-07-31	247.891667
2014-08-31	250.575000
2014-09-30	252.933333
2014-10-31	254.991667
2014-11-30	257.041667
2014-12-31	259.104167
2015-01-31	261.041667
2015-02-28	262.995833
2015-03-31	264.916667
2015-04-30	266.841667
2015-05-31	268.758333
2015-06-30	270.841667
2015-07-31	NaN
2015-08-31	NaN
2015-09-30	NaN

Interpretation :

- This table shows trend component of time series

?

Why are NAs getting generated?

This is because trend not estimated for first/last few values. Consequently, the same will be reflected in random component as well.

Decomposition in Python – Random Component

```
decomp.resid
```

```
2013-01-31         NaN
2013-02-28         NaN
2013-03-31         NaN
2013-04-30         NaN
2013-05-31         NaN
2013-06-30         NaN
2013-07-31    5.931944
2013-08-31    3.754861
2013-09-30    3.644444
2013-10-31    2.250694
2013-11-30    2.719444
2013-12-31   -3.553472
2014-01-31    0.840278
2014-02-28    2.719444
2014-03-31    2.986111
2014-04-30    4.759028
2014-05-31    1.731944
2014-06-30    3.315278
2014-07-31   -3.126389
2014-08-31   -0.949306
2014-09-30   -0.838889
2014-10-31    0.554861
2014-11-30    0.086111
2014-12-31    6.359028
2015-01-31    1.965278
2015-02-28    0.086111
2015-03-31   -0.180556
2015-04-30   -1.953472
2015-05-31    1.073611
2015-06-30   -0.509722
2015-07-31         NaN
2015-08-31         NaN
2015-09-30         NaN
```

Interpretation :

- This table shows random component of time series

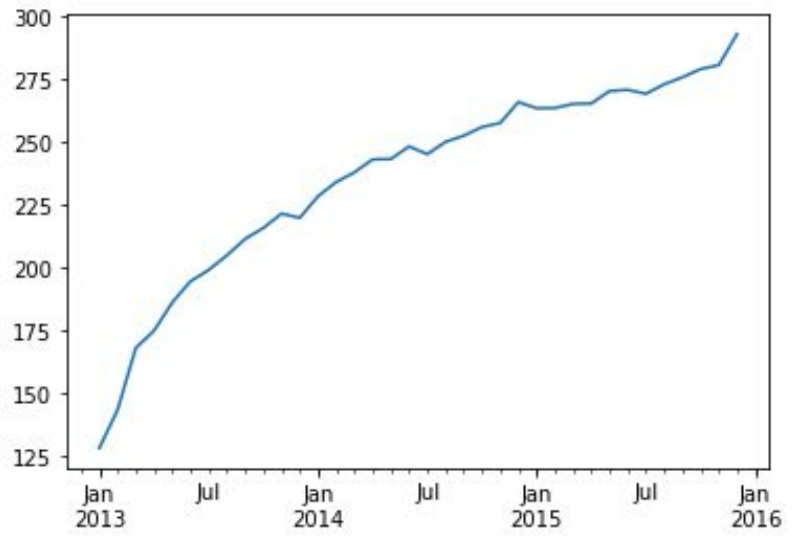
?

Why are NAs getting generated?

This is because trend not estimated for first/last few values. Consequently, the same will be reflected in random component as well.

Seasonally Adjusted Time Series

```
# Doing Seasonal Adjustment
salesadj = salesseries - decomp.seasonal
salesadj.plot()
# Output
```



Interpretation :

- This plot shows seasonally adjusted time series

Quick Recap

In this session, we learnt about **time series decomposition and exponential smoothing**:

What is Decomposition	• A time series is made up of multiple components such as seasonality, trend, randomness • Sometimes, studying these components separately provides a more comprehensive insight about the series
Decomposition in Python	• `sm.tsa.seasonal_decompose()` carries out simple seasonal decomposition

Time Series Analysis

Stationarity of Time Series - I

Contents

What is Stationarity of Time Series ?

Time series process is called **Stationary if statistical properties of the process remain unchanged over**

If Y_t is a **stationary** time series where t=1,2,3,... then,

$E(Y_t) = \mu_t = \mu$ (constant)

$Var(Y_t) = \sigma_t^2 = \sigma^2$ (constant)

$cov(Y_t, Y_{t+s})$ depends only on **s** (lag), and is independent of **t** (time)

Stationary vs. Non-Stationary Time Series

Fig. 1: Stationary Time Series

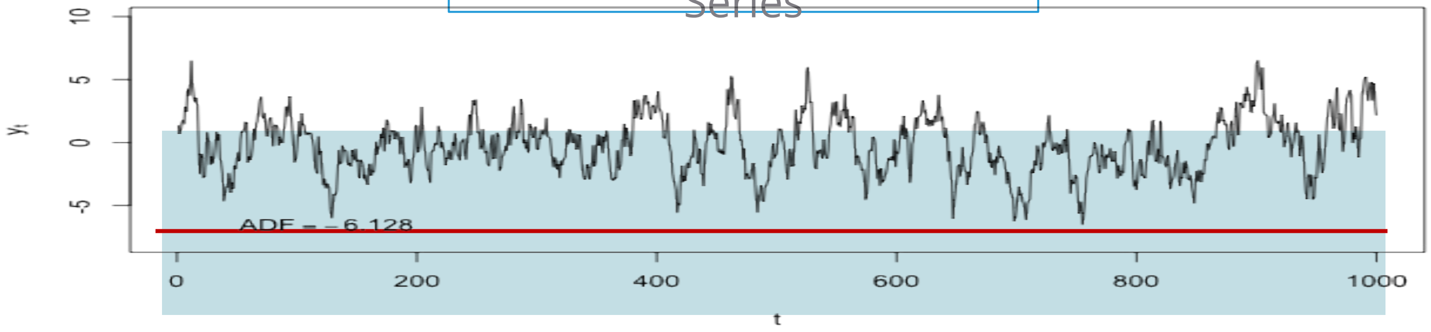
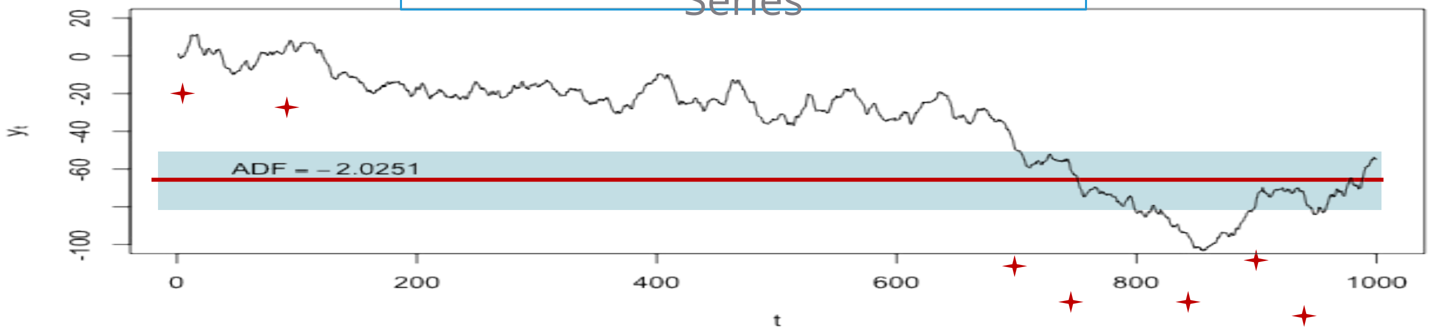


Fig. 2: Non-Stationary Time Series



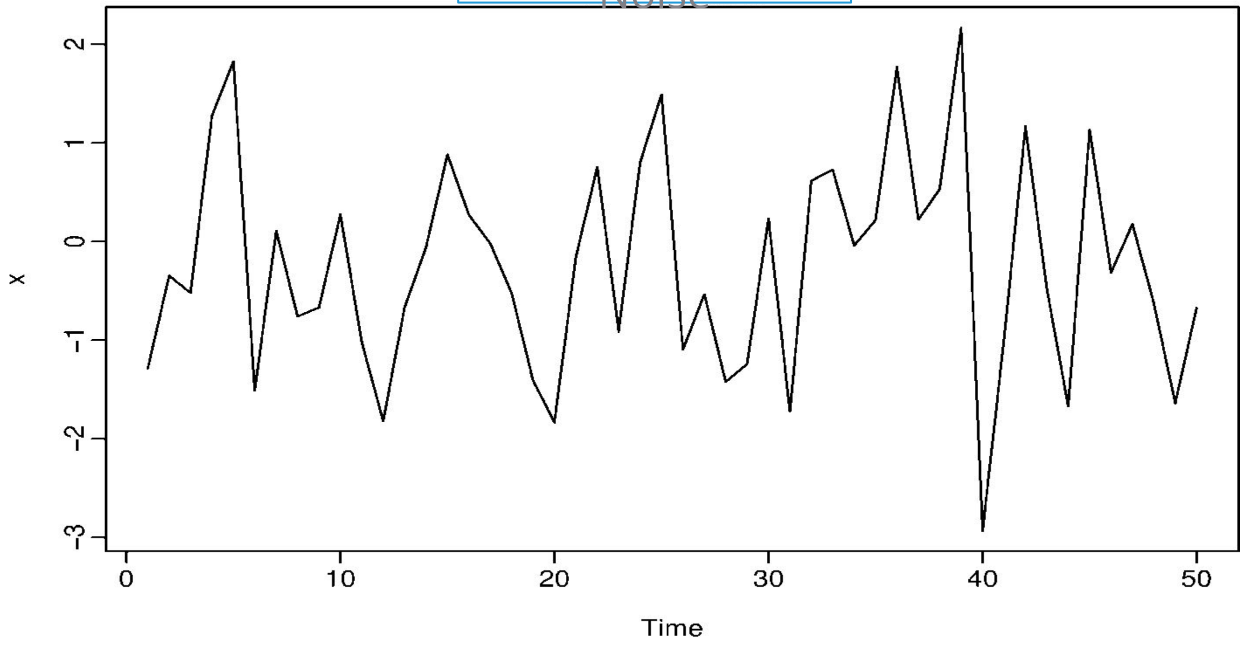
Interpretation :

- A stationary time series has a constant long term mean and variance.
- The first diagram shows a stationary time series whereas the second shows a non-stationary series.

Stationary Time Series - White Noise Process

- **White noise** is the simplest example of stationary time series.
- White Noise time series has **zero mean, constant variance and zero covariance with lagged time series.**

Fig. 3: White Noise



Non Stationary Time Series - Random Walk

- Random Walk is the simplest case of Non-stationary time series. It is of the form

$$Y_t = Y_{t-1} + U_t$$

$$t=1,2,3.....$$

- We assume that U_t is a random series with,
  - Constant mean μ
  - Constant variance σ^2
  - Serially uncorrelated

Value of Y at time t is equal to its value at time (t-1) plus a random shock

Why Random Walk is Non Stationary

$$\text{Let } Y_t = 0 \text{ at time } t = 0$$

$$Y_1 = U_1$$

$$Y_2 = Y_1 + U_2 = U_1 + U_2$$

$$Y_3 = Y_2 + U_3 = U_1 + U_2 + U_3$$

$$\text{So, } Y_t = \sum U_t \qquad t = 1, 2, 3 \ldots$$

$Yt = \sum Ut$

$E(Yt) = E(\sum Ut) = t \, . \, \mu$

i.e. Mean is not constant

$Var(Yt) = Var(\sum Ut) = t \, .$

i.e. Variance is not constant

Therefore, Random Walk is a non-stationary time series

Importance of Stationary Time Series

- **Calibration** (estimation of model parameters using historical data) is an important concept in the **forecasting** of time series values.
- **In the calibration of time series models we need a stationary time series.**
- With a non stationary time series we get into **spurious** regression which badly affects forecasting.

How to Make a Non Stationary Time Series Stationary?

Two Methods for Making Time Series Stationary

Differencing

$$Y_t = Y_{t-1} + U_t \quad ;t=1,2,3.....$$

U_t is a random series with **Constant mean** μ, **Constant variance** σ^2 , and is **serially uncorrelated i.e** (Ut is stationary).

Hence, Y_t is differenced:

$$Y_t - Y_{t-1} = \Delta Y = U_t$$

Differencing can be well applied in case of stochastic time series

De-trending

$$Y_t = \beta_1 + \beta_2 t + U_t \quad ;t=1,2,3.....$$

U_t is a stationary with **zero mean** and **constant variance** σ^2 . When Trend element ($\beta_1 + \beta_2 t$) is subtracted , the result is a stationary process :

$$Y_t - (\beta_1 + \beta_2 t) = U_t$$

De-trending is useful when trend is deterministic

Identifying Stationary Time Series & Concept of Autocorrelation

Approaches

Graphical

Statistical Test

Correlogra
m

Dickey Fuller
Test

Autocorrelation of Lag 1

Data

Year	Sales(m)
Jan	110
Feb	113
Mar	121
Apr	123
May	126
Jun	120
Jul	119
Aug	127
Sep	129
Oct	131
Nov	130
Dec	132

First Order Lag

Year	Yt	Yt-1
Jan	110	
Feb	113	110
Mar	121	113
Apr	123	121
May	126	123
Jun	120	126
Jul	119	120
Aug	127	119
Sep	129	127
Oct	131	129
Nov	130	131
Dec	132	130

Autocorrelation of Lag 1,
where $\overline{Y} = \text{Avearge of sales}$

$$\sum_{t=2}^{n} \frac{(Y_t - \overline{Y})(Y_{t-1} - \overline{Y})}{\sum_{t=1}^{n}(Y_t - \overline{Y})^2}$$

- Autocorrelation is a correlation between a time series (Y_t) and another time series representing lagged values of the same time series.(Y_{t-k})

Autocorrelation Function & Correlogram

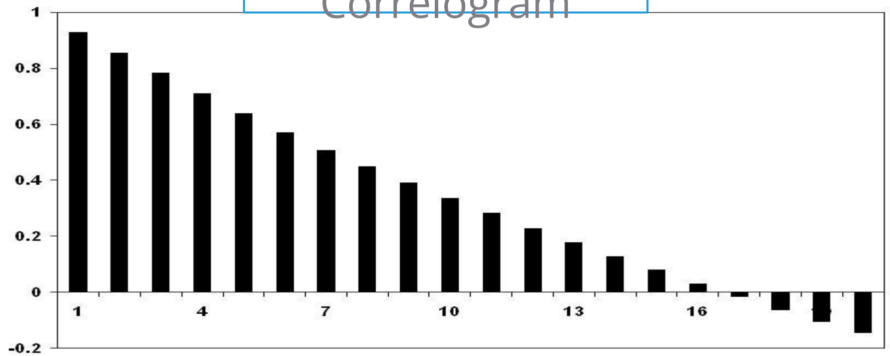
- Plot of the **sample autocorrelation function against lag is called Correlogram.**
- ACF (autocorrelation function) is a general expression for lag k autocorrelation.
- Correlogram is mainly used in checking stationarity of a series.

$$r_K \text{ (Sample autocorrelation of lag K)} = \sum_{t=k+1}^{n} \frac{(Y_t - \bar{Y})(Y_{t-k} - \bar{Y})}{\sum_{t=1}^{n}(Y_t - \bar{Y})^2}$$

Correlogram for Checking Stationarity

- For a non-stationary time series showing trend, **slow decay pattern will be observed.**
- If a time series is characterized by seasonal fluctuations, then the **correlogram would also exhibit oscillations** at the same frequency.

Fig. 4: Correlogram



Sample correlogram of *LGDPI*

This figure presents the sample correlogram.

21

Partial Autocorrelation Function

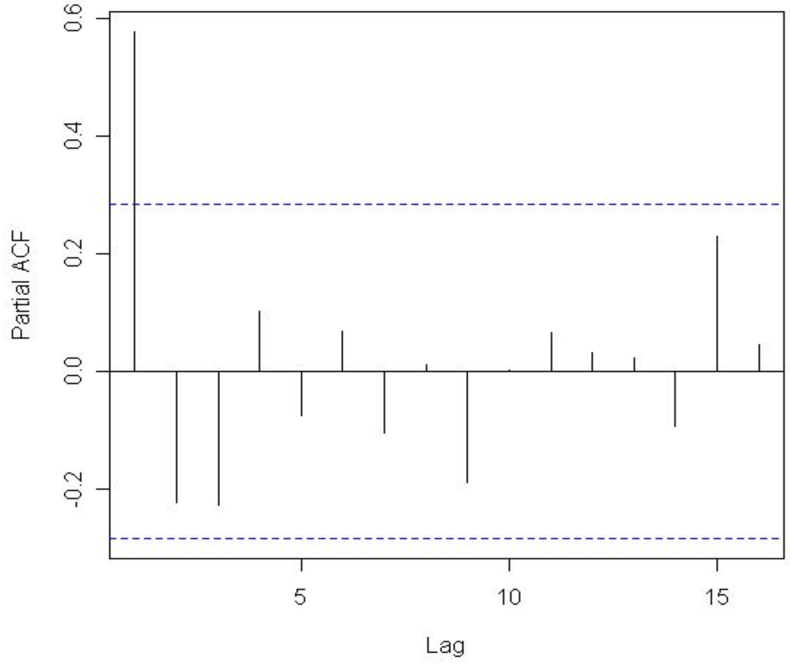
- Partial autocorrelation (PACF) is the

> Autocorrelation between y_t and y_{t-h} after removing any linear dependence on $y_1, y_2, \ldots, y_{t-h+1}$

- PACF gives **partial autocorrelations at various lags.**
- PACF is **mainly used to identify order of moving average present in the process.**

Partial Correlogram

Fig. 5: Partial ACF Plot



- Y-Axis = Partial Autocorrelation Function
- X-Axis = Lag
- Here we see that, after lag 1, the PACF drops dramatically and all PACFs after lag 1 are statistically insignificant.

Quick Recap

What is Stationarity	• Time series is stationary if the statistical properties of the process remain unchanged over time • Time series which has zero mean, constant variance and zero covariance with lagged time series is known as **"White Noise"**
Random Walk	• Is the simplest form of non-stationary time series where $Y_t - Y_{t-1} = U_t$ (U_t is a random error term)
Making a Non-Stationary Time Series Stationary	• Two approaches: **Differencing** (when time series is stochastic) and **De-trending** (when trend is deterministic)
Identifying Stationarity of Time Series	• **Graphical method:** ACF and PACF Correlograms • **Analytical method:** Dickey Fuller Test

Time Series Modeling

Seasonal ARIMA Model

Contents

Case Study

Background

- Annual Sales for a specific company from year 1961 to 2017

Objective

- To assess stationarity of time series

Available Information

- Number of cases: 57
- Variables: Year, sales(in 10's GBP)

Data Snapshot

turnover_annual data

Variables

Observations on Discrete Time Scale

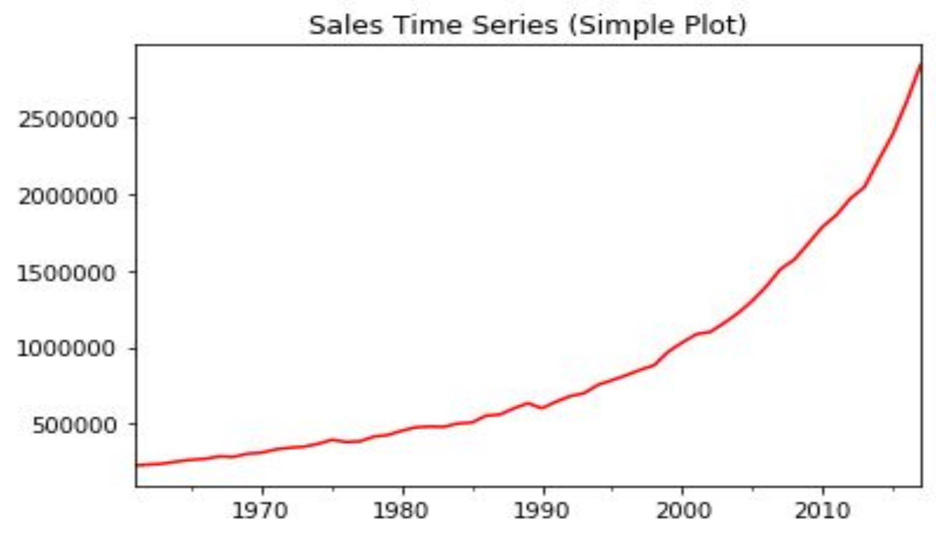
Year	sales
1961	224786
1962	230034
1963	236562
1964	250960
1965	261615
1966	268316
1967	283589
1968	280160
1969	301422
1970	308018
[illegible]	[illegible]
1974	364834
1975	392503

Columns	Description	Type	Measurement	Possible values
Year	Financial Year	Numeric	-	-
sales	sales(in 10’s GBP)	Numeric	In British Pound	Positive values

Creating and Plotting Time Series in Python

```
#Importing turnover_annual data
import pandas as pd
salesdata=pd.read_csv('turnover_annual.csv')
#Creating and Plotting a Time Series Object
rng = pd.date_range('01-01-1961','31-12-2017',freq='Y')
s = salesdata.sales.values
salesseries = pd.Series(s, rng)

salesseries.plot(color='red', title ="Sales Time Series
(Simple Plot)")
```



- **date_range()** creates pandas date object.
- **freq ='Y' indicates yealy data**
- **pd.Series()** creates time series object
- Plot function gives line chart

Interpretation :

- The time-series clearly shows a positive trend.

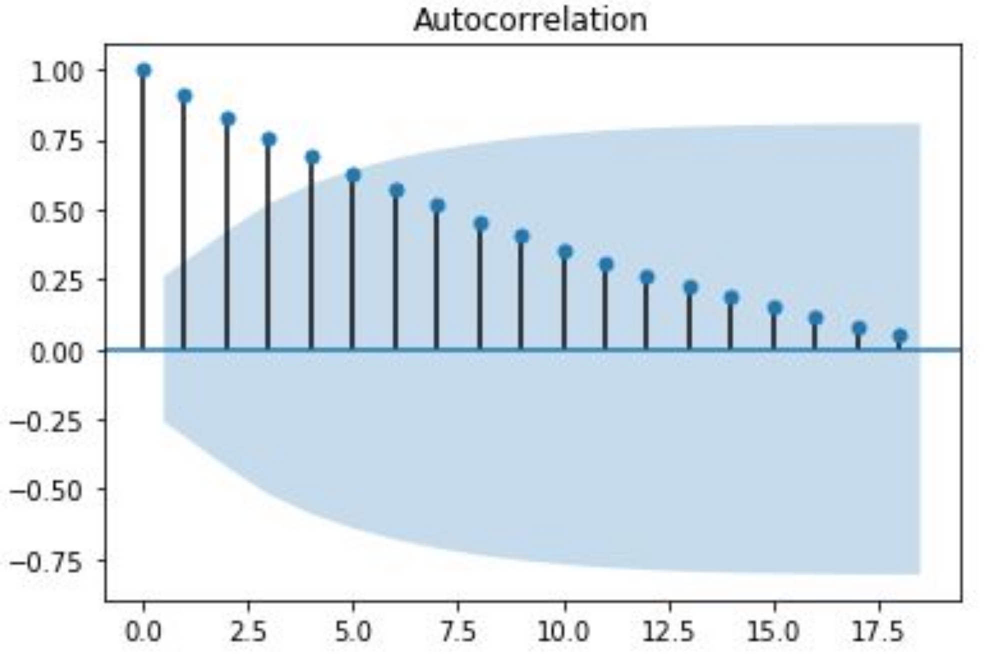
Checking Stationarity – Correlogram

```
# ACF Plot
```

```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(salesseries)
```

- **plot_acf()** returns an ACF (Auto Correlation Function) plot.

```
# Output
```



Interpretation :

- We can observe that there is a very slow decay which is a sign of Non-stationarity.

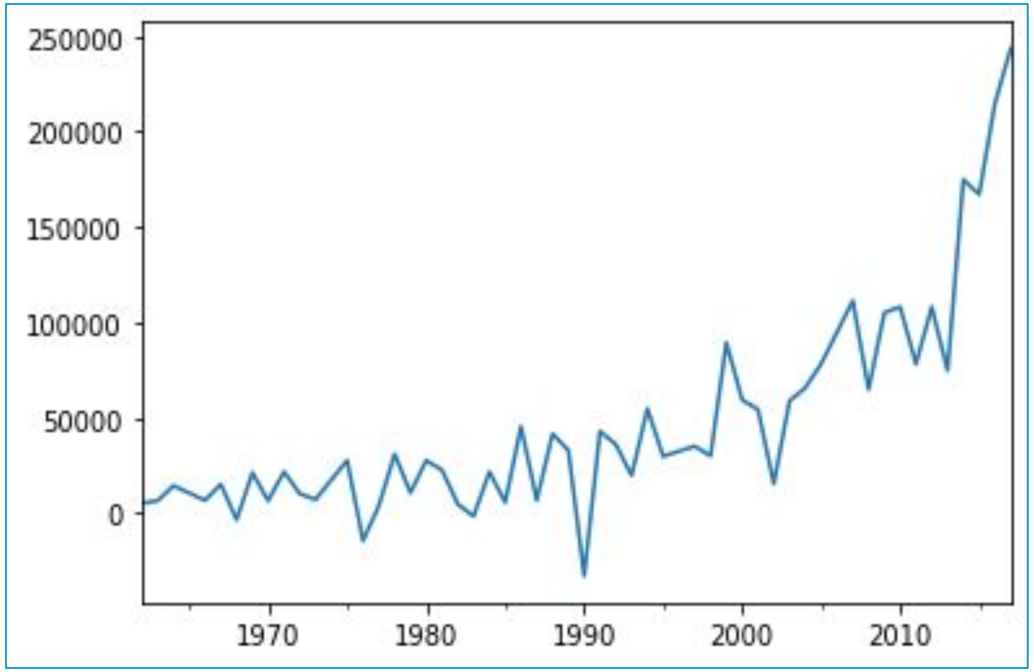
Plot of 1st Order Differenced Time Series

```
# Creating and Plotting a Difference Series
```

```
from statsmodels.tsa.statespace.tools import diff
salesdiff = diff(salesseries)
salesdiff.plot()
```

- diff() gives 1st order differences
- plot function gives line chart for differenced series

```
# Output
```



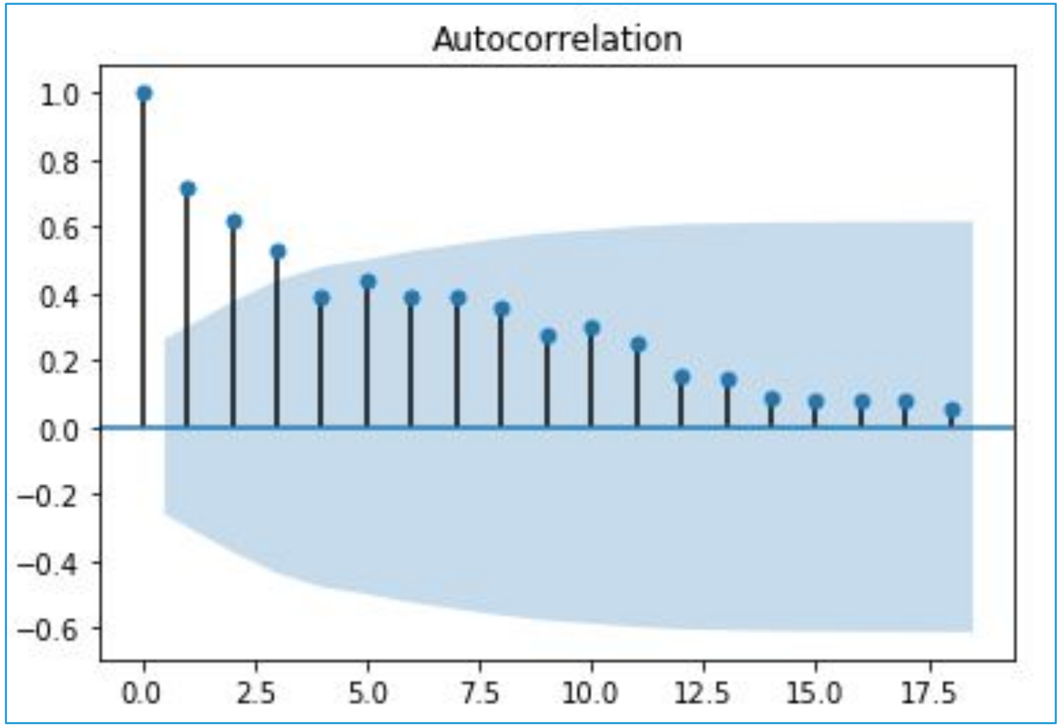
Interpretation :

- Even after first order differencing, the series looks non-stationary.

Correlogram for 1st Order Differenced Time Series

```
# ACF Plot
plot_acf(salesdiff)
```

```
# Output
```



Interpretation :

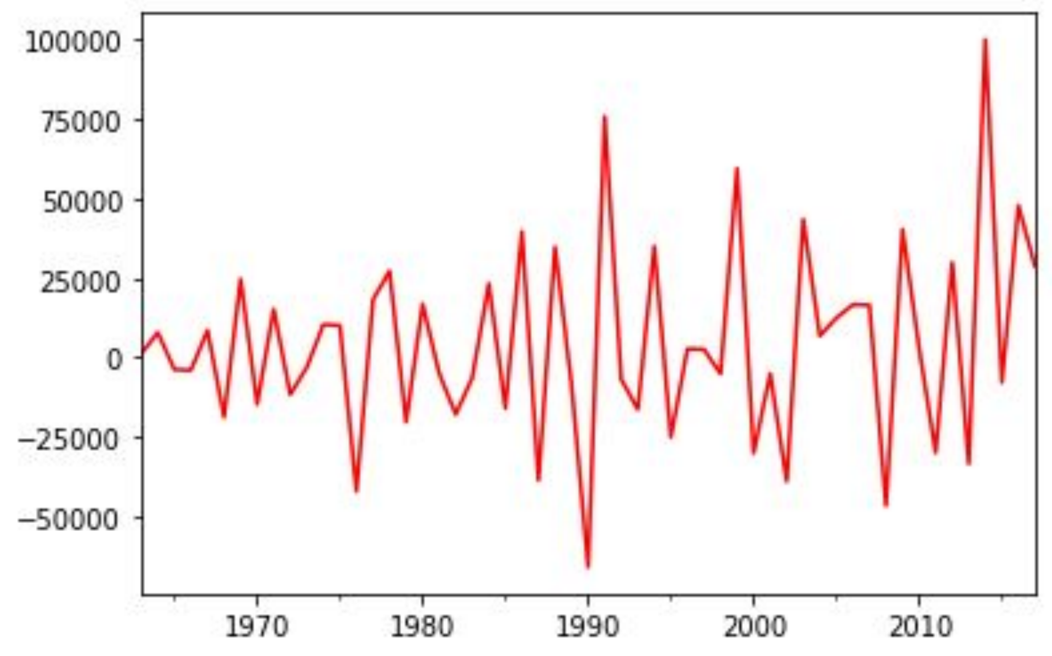
- ACF plot shows slow decay
- Stationarity is not achieved with first difference.

Plot of 2nd Order Differenced Time Series

#Creating and Plotting 2nd Difference Series

```
salesdiff2 = diff(salesdiff)
salesdiff2.plot(color='red')
```

Output

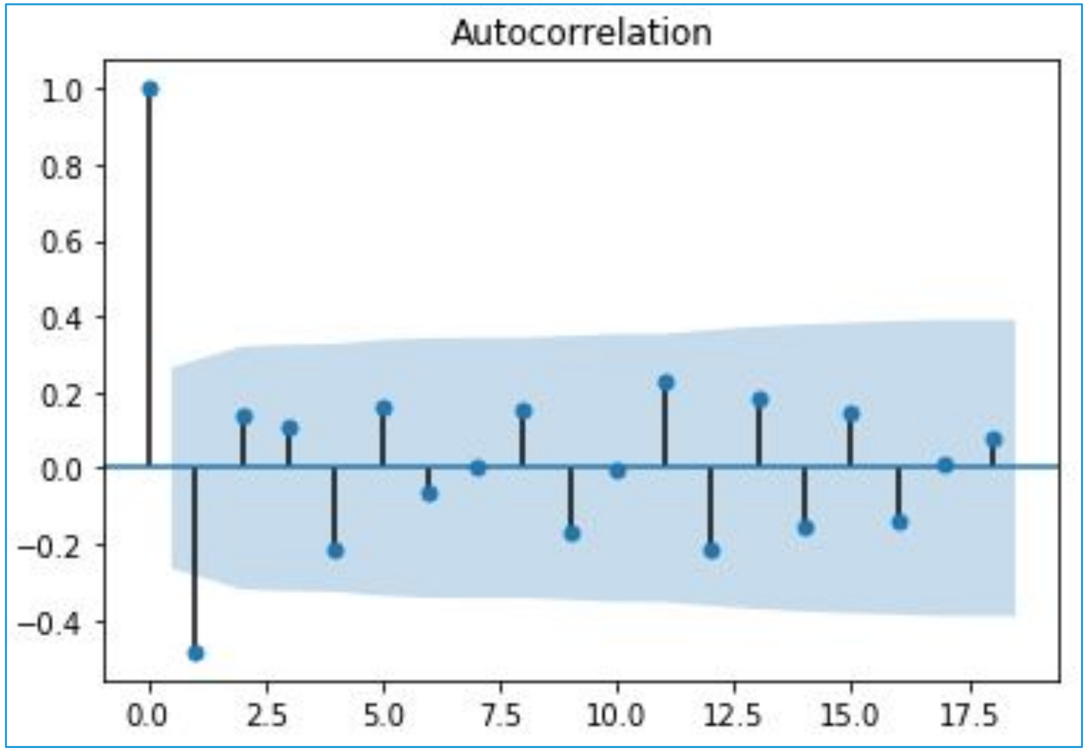


Interpretation :

- After 2nd order differencing, the series looks **stationary**.

Correlogram for 2nd Order Differenced Time Series

```
# ACF Plot
plot_acf(salesdiff2)
# Output
```



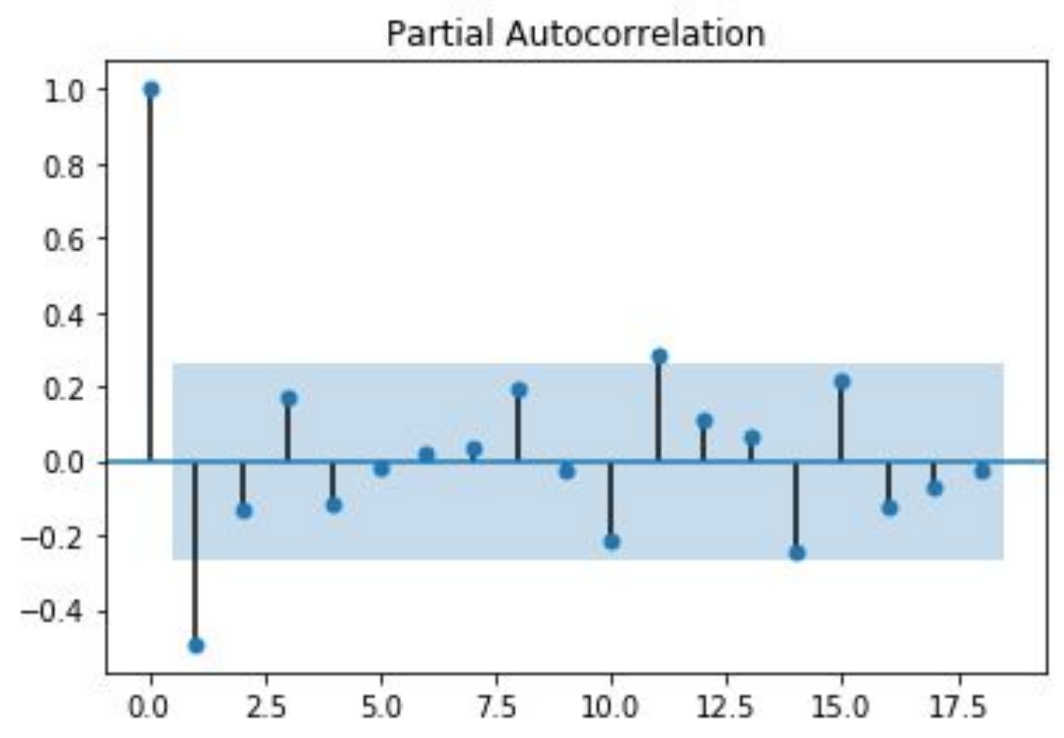
Interpretation :

- Stationarity is achieved with 2nd order difference.

Partial Autocorrelations for 2nd Order Differenced Time Series

```
# PACF Plot
plot_pacf(salesdiff2)
# Output
```

- **plot_pacf()** returns an PACF (Partial Auto Correlation Function) plot.



Interpretation :

- Stationarity is achieved with 2nd order difference.

Analytical Method – Dickey Fuller (DF) Test

- A linear stochastic process has a unit root if 1 is the root of the process's characteristic equation. Such a process is non-stationary.
- Dickey and Fuller pioneered idea of testing for unit roots for stationarity checking.

Consider X_t (t=1,2,3,...) is a time series of the form

$$X_t=\rho X_{t-1}+U_t \quad \text{..................}(1)$$

If $\rho=1$ then X_t becomes a random walk

- We assume that $U_t \sim IID(0, \sigma^2)$, i.e U_t is a white noise
- Therefore, we are interested in testing for $\rho=1$

* IID means independent and identically distributed

Dickey Fuller (DF) Unit Root Test

Objective	To test the **null hypothesis** that **time series is not stationary**

Null Hypothesis H_0: $\rho=1$

Alternate Hypothesis H_1: $\rho< 1$

$(X_t-X_{t-1}) = \Delta X_t=(\rho-1)\, X_{t-1}+U_t$..............from (1)

H_0: $\rho^*=0$, H_1: $\rho^*<0$, $\rho^*=(\rho-1)$

Test Statistic	$(\rho^*/SE(\rho^*))$ Test statistic follows DF distribution under null
Decision Criteria	Reject the null hypothesis **tcal < DF table value**

Dickey Fuller Test

```
# Install “arch”
pip install arch
# Import “ADF” from library “arch”
from arch.unitroot import ADF

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

- **ADF()** performs a Dickey Fuller unit root test on time series data.
- **lags=** allows to mention the number of lags to use in the ADF regression. We have used zero.
- **trend=‘nc’** specifies no trend and constant in regression

```
# Output
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                 19.275
P-value                         1.000
Lags                                0
-------------------------------------

Trend: No Trend
Critical Values: -2.61 (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 as value of test statistic is greater than 5% critical value.

Dickey Fuller Test

```
# Checking stationarity for series with difference of order 2
adf = ADF(salesdiff2,lags=0,trend='nc')
adf.summary()
```

```
# Output
```

```
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                -11.908
P-value                         0.000
Lags                                0
-------------------------------------

Trend: No Trend
Critical Values: -2.61 (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 as value of test statistic is less than 5% critical value.

Quick Recap

Correlograms	• **`plot_acf() & plot_pacf()`** function in Python generate Correlograms
Differencing a Time Series	• Simple numeric function **`diff()`** can be used to difference a series
Dickey Fuller Test	• **`ADF()`** function from the package **`arch`** performs a Dickey Fuller test • The output gives test statistic and critical values for the test statistic

Time Series Modeling

ARIMA Model

Contents

Box-Jenkins (ARIMA) Models

- ARIMA (**A**uto **R**egressive **I**ntegrated **M**oving **A**verage) models are **Regression models that use lagged values of the dependent variable and/or random disturbance term as explanatory variables.**

- ARIMA models **rely heavily on the autocorrelation pattern** in the data.

- ARIMA models can also be developed in the presence of seasonality in the time series.

- ARIMA models thus **essentially ignore domain theory (by ignoring "traditional" explanatory variables)**

When to Use ARIMA Models

Little or nothing is known about the dependent variable being forecasted

The independent variables known to be important cannot be forecasted effectively

Objective is to obtain short term forecasts

Basic ARIMA Models

1. Autoregressive model of order p (AR(p)):

$$y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

Where y_t depends on its *p* previous values

2. Moving Average model of order q (MA(q))

$$y_t = \delta + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q}$$

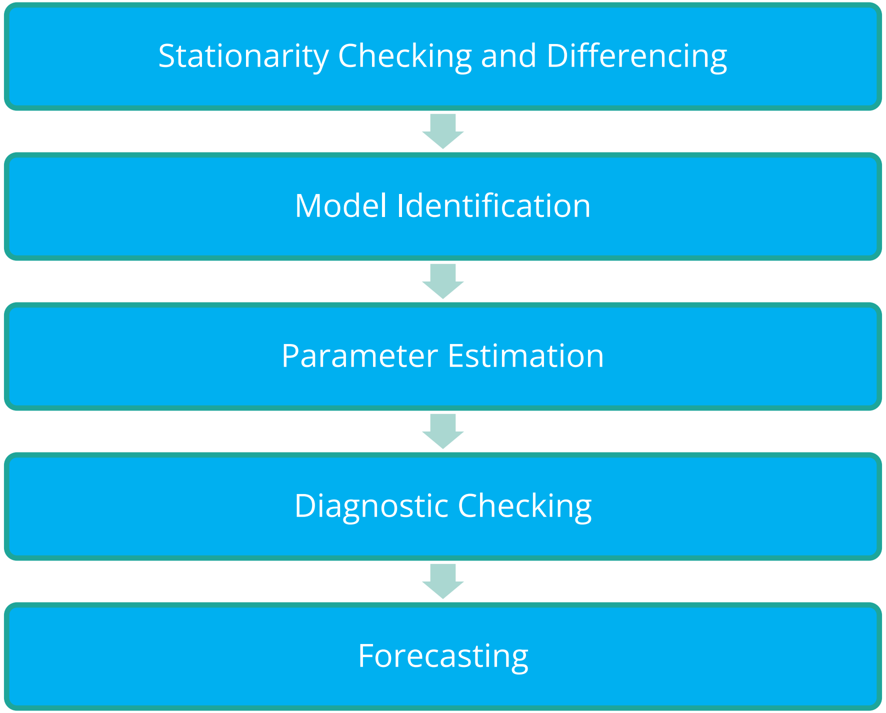
y_t depends on *q* previous random error terms

3. Autoregressive-Moving Average model of order p and q (ARMA(p,q))

$$\begin{aligned} y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} \\ + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q} \end{aligned}$$

y_t depends on its *p* previous values and *q* previous random error terms

Five-Step Iterative Procedure



Step 1: Stationarity Checking

Differencing

- Differencing continues until stationarity is achieved

$$\boxed{\Delta y_t = y_t - y_{t-1}}$$

$$\boxed{\Delta^2 y_t = \Delta(\Delta y_t) = \Delta(y_t - y_{t-1}) = y_t - 2y_{t-1} + y_{t-2}}$$

- The differenced series has n-1 values after taking the first-difference, n-2 values after taking the second difference, and so on

- The number of times that the original series must be differenced in order to achieve stationarity is called the order of integration, denoted by d ?

? How many times should a series be differenced?
In practice, it is not required to go beyond second difference.

Case Study

Background

- Annual Sales for a specific company from year 1961 to 2017

Objective

- To develop time series model and forecast sales for next 3 years

Available Information

- Number of cases: 57
- Variables: Year, sales(in 10's GBP)

Data Snapshot

turnover_annual data

Variables

Observations on Discrete Time Scale

Year	sales
1961	224786
1962	230034
1963	236562
1964	250960
1965	261615
1966	268316
1967	283589
1968	280160
1969	301422
1970	308018
1974	364834
1975	392503

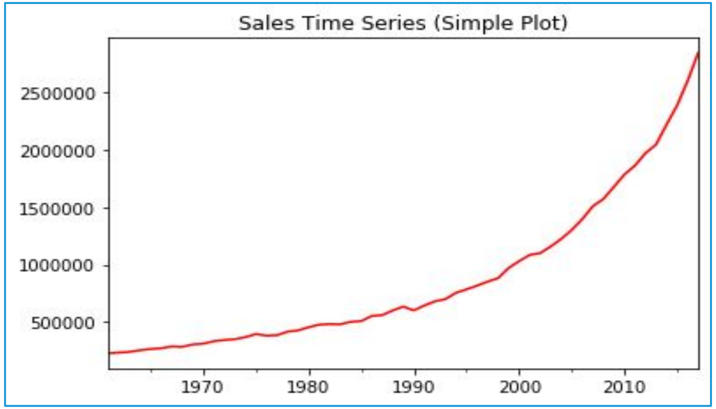
Columns	Description	Type	Measurement	Possible values
Year	Financial Year	Numeric	-	-
sales	sales(in 10’s GBP)	Numeric	In British Pound	Positive values

Creating and Plotting Time Series in Python

```
#Importing turnover_annual data
import pandas as pd
salesdata=pd.read_csv('turnover_annual.csv')
#Creating and Plotting a Time Series Object
rng = pd.date_range('01-01-1961','31-12-2017',freq='Y')
s = salesdata.sales.values
salesseries = pd.Series(s, rng)

salesseries.plot(color='red', title ="Sales Time Series (Simple Plot)")
```

- **date_range()** creates pandas date object.
- **freq ='Y' indicates yearly data**
- **pd.Series()** creates time series object
- Plot function gives line chart



Interpretation :

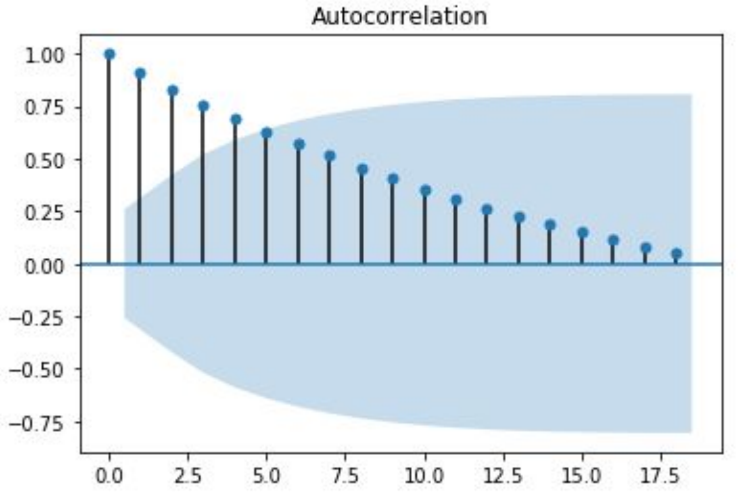
- The time-series clearly shows a positive trend.

Checking Stationarity – Correlogram

```
# ACF Plot
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(salesseries)
```

- **plot_acf()** returns an ACF (Auto Correlation Function) plot.

```
# Output
```



Interpretation :

- We can observe that there is a very slow decay which is a sign of Non-stationarity.

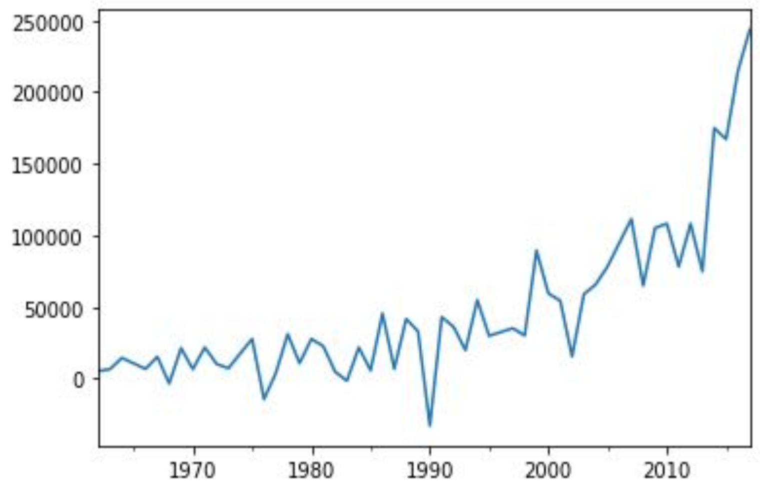
Plot of 1st Order Differenced Time Series

```
# Creating and Plotting a Difference Series
```

```
from statsmodels.tsa.statespace.tools import diff
salesdiff = diff(salesseries)
salesdiff.plot()
```

- diff() gives 1st order differences
- plot function gives line chart for differenced series

```
# Output
```



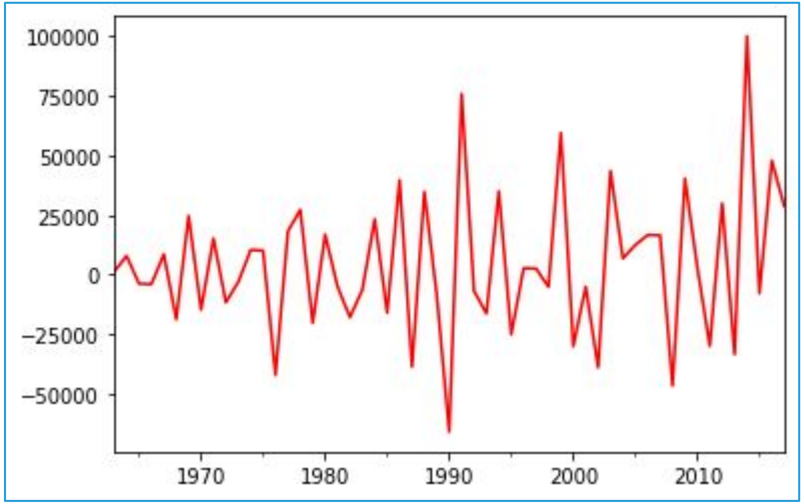
Interpretation :

- Even after first order differencing, the series looks non-stationary.

Plot of 2nd Order Differenced Time Series

```
#Creating and Plotting 2nd Difference Series
salesdiff2 = diff(salesdiff)
salesdiff2.plot(color='red')
```

```
# Output
```

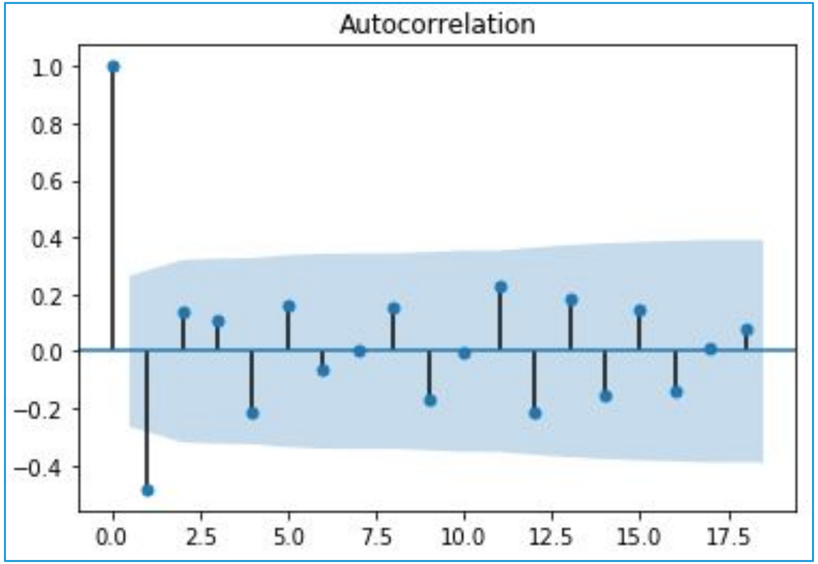


Interpretation :

- After 2nd order differencing, the series looks **stationary**.

Correlogram for 2nd Order Differenced Time Series

```
# ACF Plot
plot_acf(salesdiff2)
# Output
```



Interpretation :

- Stationarity is achieved with 2nd order difference.

Dickey Fuller Test

```
# Install "arch"
pip install arch
# Import "ADF" from library "arch"
from arch.unitroot import ADF

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

- **ADF()** performs a Dickey Fuller unit root test on time series data.
- **lags=** allows to mention the number of lags to use in the ADF regression. We have used zero.
- **trend='nc'** specifies no trend and constant in regression

```
# Output
```

```
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                 19.275
P-value                         1.000
Lags                                0
-------------------------------------

Trend: No Trend
Critical Values: -2.61 (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 as value of test statistic is greater than 5% critical value.

Dickey Fuller Test

```
# Checking stationarity for series with difference of order 2
adf = ADF(salesdiff2,lags=0,trend='nc')
adf.summary()
```

```
# Output
```

```
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                -11.908
P-value                         0.000
Lags                                0
-------------------------------------

Trend: No Trend
Critical Values: -2.61 (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 as value of test statistic is less than 5% critical value.

Step 2: Model Identification

Model Identification

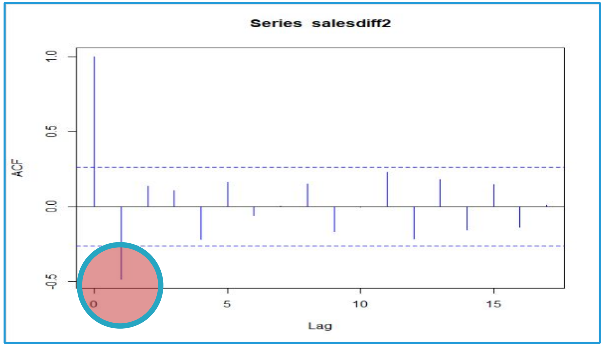
- When the data are confirmed stationary, proceed to tentative identification of models through **visual inspection of correlogram and partial correlogram**

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

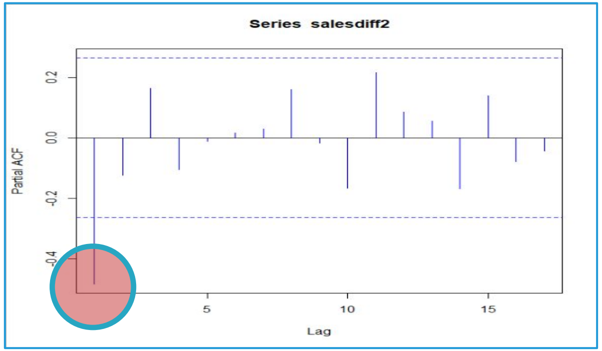
Model Identification

- ARIMA model is expressed as arima (p,d,q) where
  - p = no. of autoregressive terms
  - d = order of differencing
  - q = no. of moving average terms

ACF Plot



PACF Plot



- ACF and PACF correlograms will help in determining the MA and AR values respectively.

Indicative Model : arima(2,2,2)

Step 3: Parameter Estimation

Parameter Estimation in Python

```
# Simple Estimation
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(salesseries, order=(2, 2, 2)).fit(trend='nc')
```

- **ARIMA()** fits a model to a univariate time series.
- **order=** argument gives the model (p,d,q) order.

```
model.params
model.aic
```

params() and **aic()** return the model coefficients and AIC value.

```
# Output
```

```
ar.L1.D2.None    -1.235413
ar.L2.D2.None    -0.670320
ma.L1.D2.None     0.785072
ma.L2.D2.None     0.330062
dtype: float64
```

```
# Output
```

```
1285.9836066562698
```

Interpretation :

- Smaller the AIC value, better is the model. We need to try out various combinations of AR and MA terms to arrive at final model.

Automatic Estimation of Model Parameters

```
# Automatic Model Identification and Parameter Estimation
```

```
import pmdarima as pm
model = pm.auto_arima(salesseries,max_p=2, max_q=2, d=2,
                      seasonal=False, trace=True)
```

- **auto_arima()** generates the best order arima model. The function conducts a search over possible model within the order constraints provided.
- **trace=** True returns the list of all models considered.
- **max_p and max_q** gives maximum values of p and q respectively.
- **seasonal**= allows you to specify whether to fit a seasonal ARIMA or not.

Automatic Estimation of Model Parameters

```
# Output
```

```
Fit ARIMA: order=(2, 2, 2) seasonal_order=(0, 0, 0, 0); AIC=1294.586, BIC=1306.630, Fit
time=0.056 seconds
Fit ARIMA: order=(0, 2, 0) seasonal_order=(0, 0, 0, 0); AIC=1295.387, BIC=1299.402, Fit
time=0.006 seconds
Fit ARIMA: order=(1, 2, 0) seasonal_order=(0, 0, 0, 0); AIC=1291.568, BIC=1297.590, Fit
time=0.019 seconds
Fit ARIMA: order=(0, 2, 1) seasonal_order=(0, 0, 0, 0); AIC=1292.152, BIC=1298.174, Fit
time=0.018 seconds
Fit ARIMA: order=(0, 2, 0) seasonal_order=(0, 0, 0, 0); AIC=1294.532, BIC=1296.540, Fit
time=0.006 seconds
Fit ARIMA: order=(2, 2, 0) seasonal_order=(0, 0, 0, 0); AIC=1291.358, BIC=1299.387, Fit
time=0.022 seconds
Fit ARIMA: order=(2, 2, 1) seasonal_order=(0, 0, 0, 0); AIC=1292.717, BIC=1302.754, Fit
time=0.044 seconds
Fit ARIMA: order=(1, 2, 1) seasonal_order=(0, 0, 0, 0); AIC=1293.092, BIC=1301.122, Fit
time=0.029 seconds
Total fit time: 0.203 seconds
```

```
ARIMA(order=(2, 2, 0))
```

Lowest AIC

Interpretation :

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

ARIMA Model Using BEST Order

```
# Run arima() for cross checking parameters based on model suggested
# by auto.arima
```

```python
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(salesseries, order=(2, 2, 0)).fit(trend='nc')

model.params
model.aic
```

```
# Output
```

```
ar.L1.D2.None    -0.506320
ar.L2.D2.None    -0.101797
dtype: float64
```

```
# Output
```

```
1285.114088844849
```

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 commonly used method for checking if the residuals are white noise process.

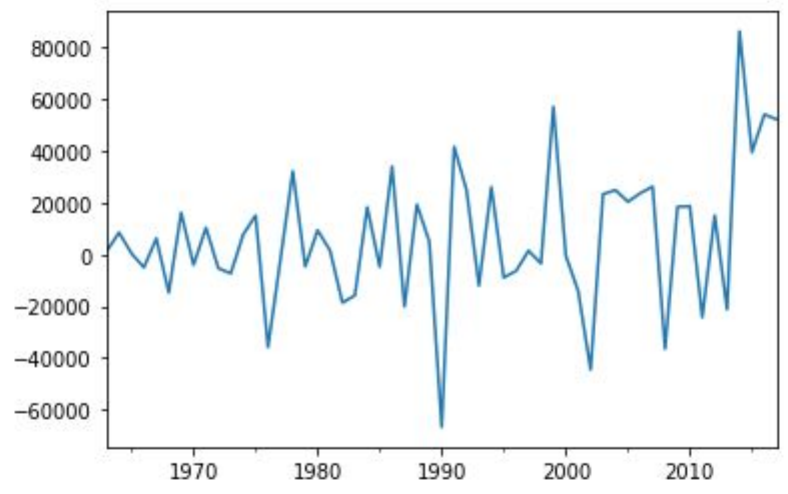
Residual Plot In Python

```
resi = model.resid

resi.plot()
```

resid() calculates residual values.

```
# Output
```



Interpretation :

- Errors follow white noise process.

Step 5: Forecasting

Forecasting

```
# Forecast for next 3years
model.forecast(steps=3)
```

forecast() function here gives predicted values for 3 years

```
# Output
(array([3072357.58495709, 3303466.18670477, 3533054.4598124 ]),
 array([27092.85937931, 48699.96239955, 75757.91735836]),
 array([[3019256.55633543, 3125458.61357874],
        [3208016.0143532 , 3398916.35905634],
        [3384571.67024626, 3681537.24937854]]))
```

Interpretation :

- **forecast()** returns three arrays:
- array of three forecasts
- array of these standard error of the forecasts
- array of the confidence interval for the forecast

Quick Recap

Stationarity Checking	• Plot correlogram using plot_`acf()` and ADF() for Dickey-Fuller Test
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 • 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

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

1. Stationarity Checking and Seasonal Differencing
2. Model Identification
3. Parameter Estimation
4. Diagnostic Checking
5. Forecasting

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
[illegible]	[illegible]	[illegible]
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

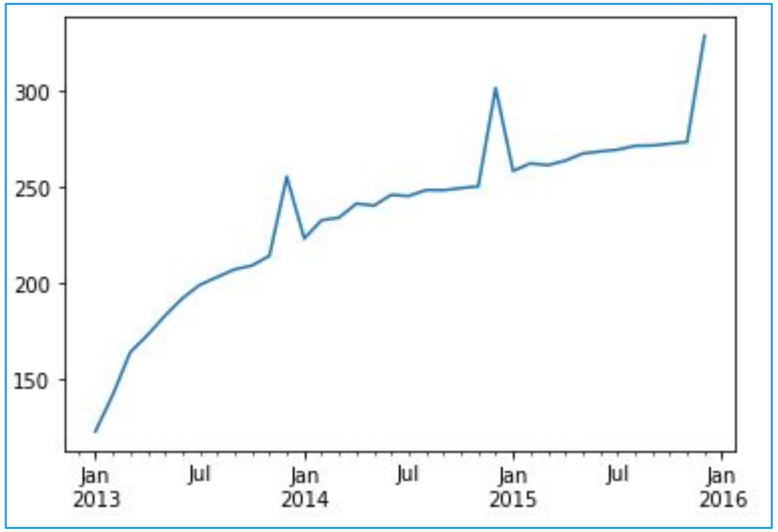
Plotting a Time Series in Python

```
# 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



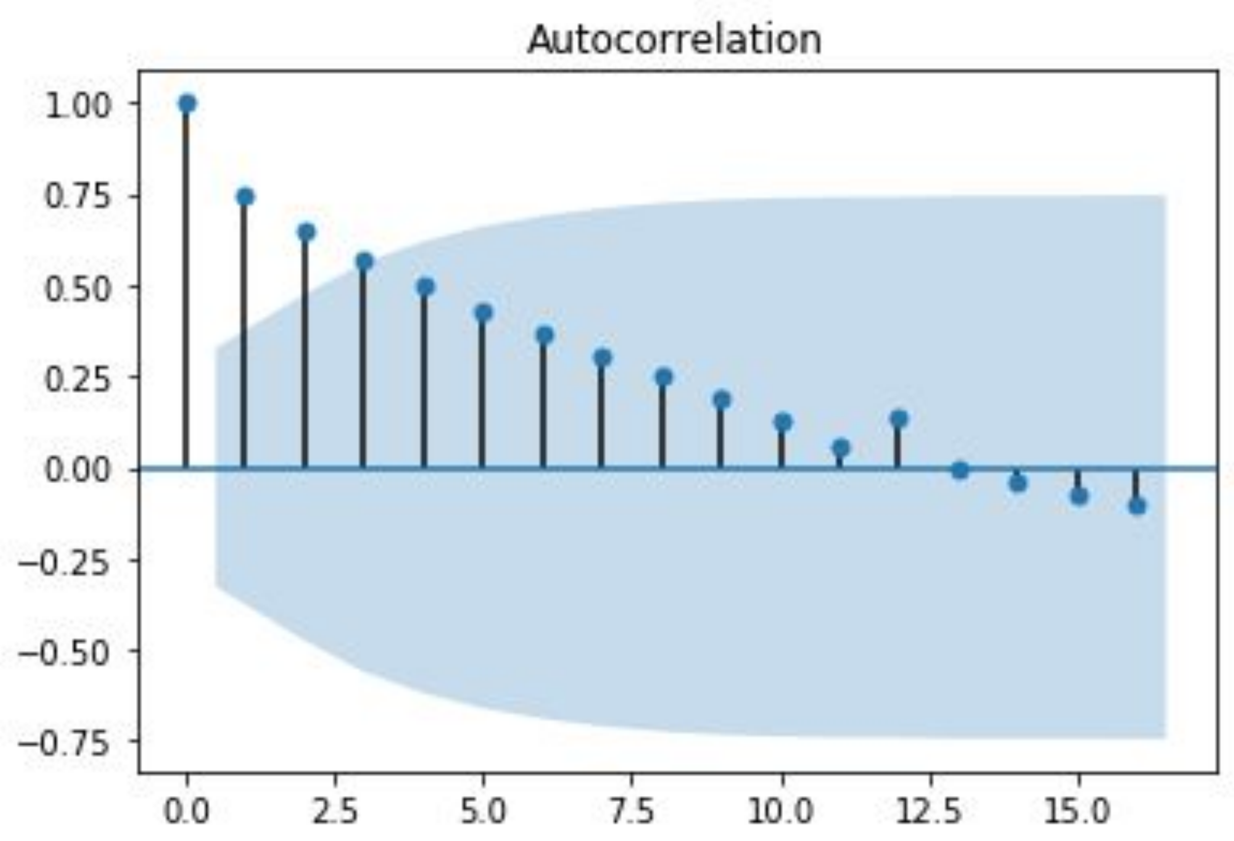
Interpretation :

- 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

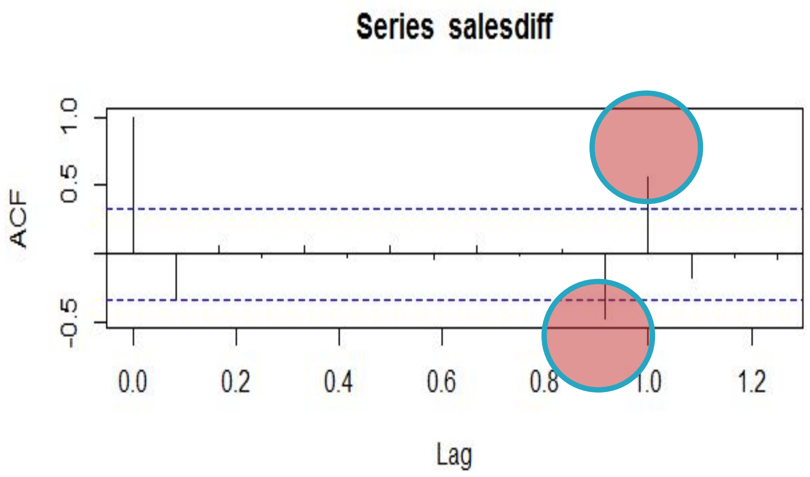
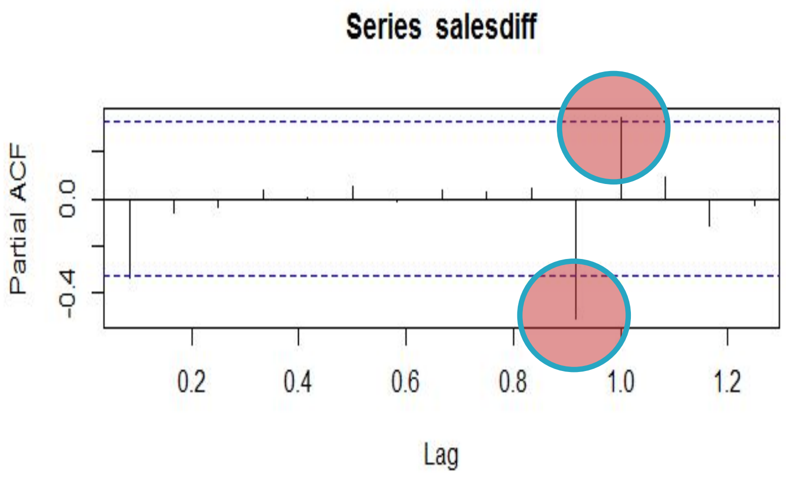


Fig. 4: PACF Plot



<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, …, 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
```

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

Interpretation :

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

* 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'

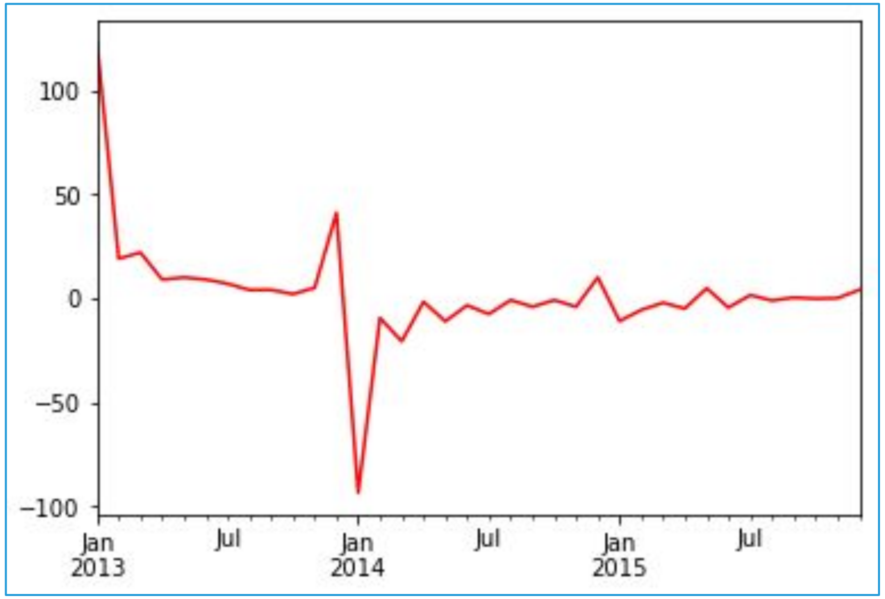
- White Noise time series has **zero mean, constant variance and zero covariance with lagged time series.**

- **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

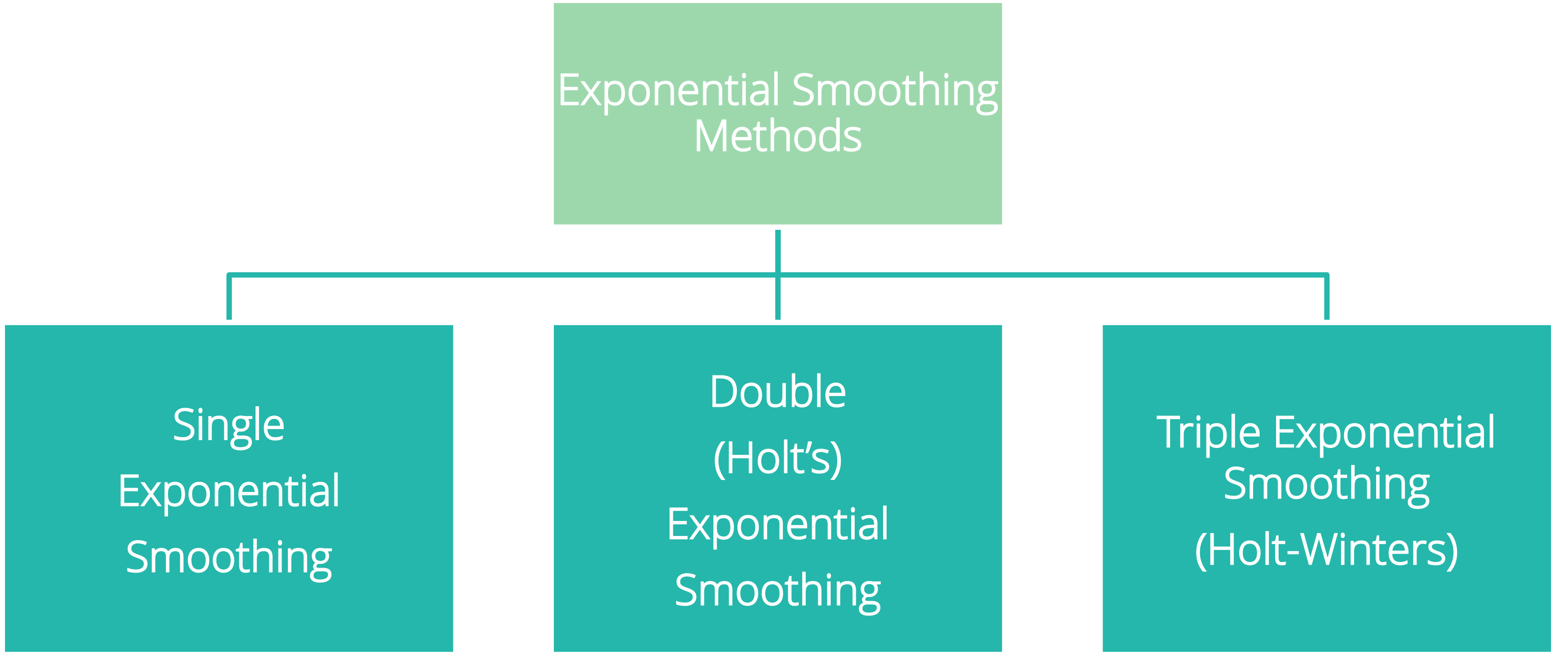
Time Series Analysis –Exponential Smoothing Methods for Forecasting

DATA SCIENCE
INSTITUTE

Contents

Forecasting Using Smoothing Methods

- Random, unexplained variation in a time series can have an undesirable impact on forecasts
- Smoothing can cancel or reduce such impacts
- Smoothing can either be Simple (using Moving Averages) or Exponential



DATA SCIENCE INSTITUTE

Single Exponential Smoothing Model

Mathematical Model :

$$F_{t+1} = \alpha Y_t + (1-\alpha) F_t$$

Where,

F_{t+1}	:	Forecast value for period t + 1
F_t	:	Forecast value for period t
Y_t	:	Actual value for period t
α	:	Alpha (Smoothing constant)

Single Exponential Smoothing Model

Assume α=0.8

t	yt	Ft	
1	23	-	
2	24	23	
3	26	23.80	=0.8*24+0.2*23
4	23.5	25.56	=0.8*26+0.2*23.8
5	27	23.91	
6	26.1	26.38	
7	28	26.16	
8	27	27.63	
9	29	27.13	
10	29.3	28.63	
11	28.2	29.17	
12	27	28.39	
		27.28	

*

1st future value will always be the previous value & then for rest future values exponential smoothing mathematical formula is applied.

Single Exponential Smoothing Model - Smoothing Constant α

Values of α

close to one have less of a smoothing effect and give greater weight to recent changes in the data

closer to zero have a greater smoothing effect and are less responsive to recent changes

- There is no formally correct procedure for choosing α. Sometimes the statistician's judgment is used to choose an appropriate factor.
- Alternatively, α can be decided based on statistical measure such as Root Mean Squared Error.

Get an Edge!

Why the Name “Exponential”?

- This method gives weights to past observation in exponentially decreasing manner.

$$F_{t+1} = \alpha y_t + \alpha(1-\alpha) y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \alpha(1-\alpha)^3 y_{t-3} -----$$

$$= \alpha y_t + (1-\alpha) [\alpha y_{t-1} + \alpha(1-\alpha) y_{t-2} + \alpha(1-\alpha)^2 y_{t-3} ----]$$

$$= \alpha y_t + (1-\alpha) F_t$$

- Larger alpha gives more weight to recent values.

DATA SCIENCE INSTITUTE

Case Study

Background

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

Objective

- To apply Decomposition & Exponential Smoothing to Time Series data using different methods.

Available Information

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

Data Snapshot

Sales Data for 3 Years

Variables

nly 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

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

Simple Exponential Smoothing in Python

```
# Import data

import pandas as pd
salesdata = pd.read_csv("Sales Data for 3 Years.csv")
rng = pd.date_range('2013','2016',freq='M')
s = salesdata.Sales.values
salesseries = pd.Series(s, rng)
```

- **freq =** tells Python the frequency of time period in the data, 'M' for monthly data.
- **pd.Series()** converts a column from a data frame to a simple time series object.

Simple Exponential Smoothing in Python

```
#Single Exponential Smoothing

from statsmodels.tsa.holtwinters import SimpleExpSmoothing
model = SimpleExpSmoothing(salesseries)
fit1 = model.fit()

fit1.predict()

fit1.summary()
```

SimpleExpSmoothing() from holtwinters in statsmodels undertakes exponential smoothing.

DATA SCIENCE INSTITUTE

Simple Exponential Smoothing in Python

```
# Output

2016-01-31     313.35045
Freq: M, dtype: float64
```

Predicted value

SimpleExpSmoothing Model Results

Dep. Variable:	None	**No. Observations:**	36
Model:	SimpleExpSmoothing	**SSE**	10897.844
Optimized:	True	**AIC**	209.661
Trend:	None	**BIC**	212.828
Seasonal:	None	**AICC**	210.951
Seasonal Periods:	None	**Date:**	Tue, 17 Oct 2023
Box-Cox:	False	**Time:**	14:05:33
Box-Cox Coeff.:	None		

	coeff	**code**	**optimized**
smoothing_level	0.7351707	alpha	True
initial_level	129.80567	l.0	True

alpha

Interpretation :
It returns predicted future value & value of alpha.

DATA SCIENCE INSTITUTE

Double (Holt) and Triple(Holt-Winters) Exponential Smoothing Methods

Double exponential smoothing has two equations

First equation is similar to single exponential smoothing method

Second equation updates trend using constant beta.

Double exponential smoothing method is used when there is a trend in the time series.

Triple exponential smoothing has three equations

First 2 equations are similar to double exponential smoothing method

Third equation updates seasonal component using constant gamma.

Triple exponential smoothing method is used when there is trend + seasonality in the time series.

Double Exponential Smoothing Model

Mathematical Model :

Where,

F_{t+1}	:	Forecast value for period t +1
F_t	:	Forecast value for period t
T_t	:	Trend component for period t
T_{t+1}	:	Trend component for period t +1
Y_t	:	Actual value for period t
α	:	Alpha (Smoothing constant)
β	:	Beta (Second smoothing constant)

DATA SCIENCE INSTITUTE

Double Exponential Smoothing in Python

```
#Double Exponential Smoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
model = ExponentialSmoothing(salesseries, trend='add',seasonal = None)
fit2 = model.fit()
print(fit2.predict())
fit2.summary()
```

```
# Output
2016-01-31     295.970308
Freq: M, dtype: float64
```

Predicted value

ExponentialSmoothing Model Results

Dep. Variable:	None	No. Observations:	
Model:	ExponentialSmoothing	SSE	8649.636
Optimized:	True	AIC	205.343
Trend:	Additive	BIC	211.677
Seasonal:	None	AICC	208.240
Seasonal Periods:	None	Date:	Tue, 17 Oct 2023
Box-Cox:	False	Time:	14:15:55
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.3039444	alpha	True
smoothing_trend	0.3039444	beta	True
initial_level	127.50907	l.0	True
initial_trend	11.565240	b.0	True

alpha, b

Interpretation :
It returns predicted future value, value of alpha and beta.

Triple Exponential Smoothing Model

Mathematical Model :
$$F_{t+1} = \alpha \frac{Y_t}{S_{t+1-k}} + (1-\alpha) F_t - T_t$$

where,

S_{t+1-k}	:	Seasonal smoothing value for period t +1
F_t	:	Forecast value for period t
F_{t+1}	:	Forecast value for period t +1
F_t	:	Forecast value for period t
T_t	:	Trend component for period t
T_{t+1}	:	Trend component for period t +1
Y_t	:	Actual value for period t

α : Alpha (Smoothing constant) β : Beta (Second smoothing constant)
: Gamma (Third smoothing constant)

DATA SCIENCE INSTITUTE

Triple Exponential Smoothing in Python

```
#Triple Exponential Smoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
model = ExponentialSmoothing(salesseries,seasonal_periods=12,
trend='add', seasonal='add')
fit3 = model.fit()
print(fit3.predict())
fit3.summary()
```

Interpretation :
It returns predicted future value & value of alpha, beta and gamma.

Output

```
2016-01-31    293.447983
Freq: M, dtype: float64
```

Predicted value

ExponentialSmoothing Model Results

Dep. Variable:	None	**No. Observations:**	
Model:	ExponentialSmoothing	**SSE**	539.668
Optimized:	True	**AIC**	129.468
Trend:	Additive	**BIC**	154.804
Seasonal:	Additive	**AICC**	169.703
Seasonal Periods:	12	**Date:**	Tue, 17 Oct 2023
Box-Cox:	False	**Time:**	14:06:06
Box-Cox Coeff.:	None		

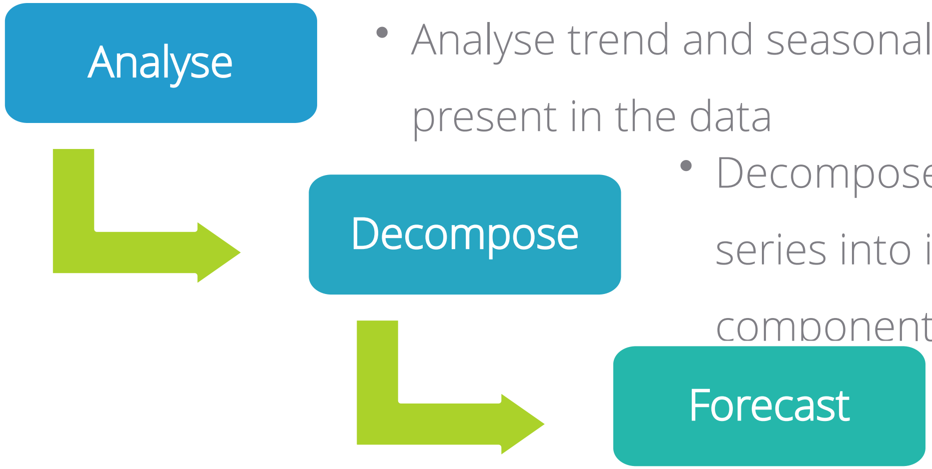
	coeff	code	optimized
smoothing_level	1.0000000	alpha	True
smoothing_trend	0.3703030	beta	True
smoothing_seasonal	1.9375e-08	gamma	True
initial_level	121.18636	l.0	True
initial_trend	13.392973	b.0	True

alpha, beta, gamma

DATA SCIENCE INSTITUTE

Get an Edge!

Always approach time series analysis in a systematic manner



- Analyse trend and seasonality present in the data
- Decompose time series into its components
- Forecast future values of the time series

- For vector time series, investigate connections between two or more time series with the aim of using values of some of the processes to predict those of the others. (Eg. Pairs trading in stock market)

DATA SCIENCE INSTITUTE

Quick Recap

In this session, we learnt about **exponential smoothing**:

Smoothing	• Smoothing gives weights to past observations, in order to give more significance to seasonality and trend components of a time series
Smoothing in Python	• From **`statsmodels.tsa.holtwinters :`** • Use **`SimpleExpSmoothing()`** to carry out simple exponential smoothing • Use **`ExponentialSmoothing()`** to carry out double and triple exponential smoothing

The Concept of Co-integration

DATA SCIENCE INSTITUTE

Contents

Concept of Co-integration

- The stock prices are believed to be non-stationary in nature.

- If there exists a linear combination of 2 stock prices , such that errors of the long term relationship are stationary (mean reverting) in nature, then the 2 stocks are said to be co-integrated.

- Engle - Granger test is used to identify the co-integrated pairs in a particular sector.

Co-Integration

Examples of Co-integrated Series

- Income and Consumption
- Money, National Incomes, Price Rates, Interest Rates
- Price of a Commodity and Taxes Levied on that Commodity, Inflation Rate
- Temperatures, Electricity Consumption
- Prices of Two Stocks

Co-Integration-Formal Definition

- **"Order of integration"** tells you the **minimum number of differences** needed to get a stationary series.
- **A series of successive differences, d,** can transform the time series into one with stationarity. The differences are denoted by **I(d)**, where **d is the order of integration**.

- Suppose Y_t and X_t are two time series integrated of order d, then
  - Any linear combination of such two series will also be integrated of order d (denoted as I(d)). This is called 'Integration'
- However,

> If there exists a vector β such that $u_t = Y_t - \beta X_t$ is of a lower order of integration (I(d – b), where b > 0) then Y_t and X_t are defined as Cointegrated of order (d,b)

- The idea of cointegration was introduced by **Engle and Granger** in 1987

* Usually, the order of integration is either I(0) or I(1); It's rare to see values for d that are 2 or more.

What is Pairs Trading

- Pairs trading finds its roots in the area of securities trading

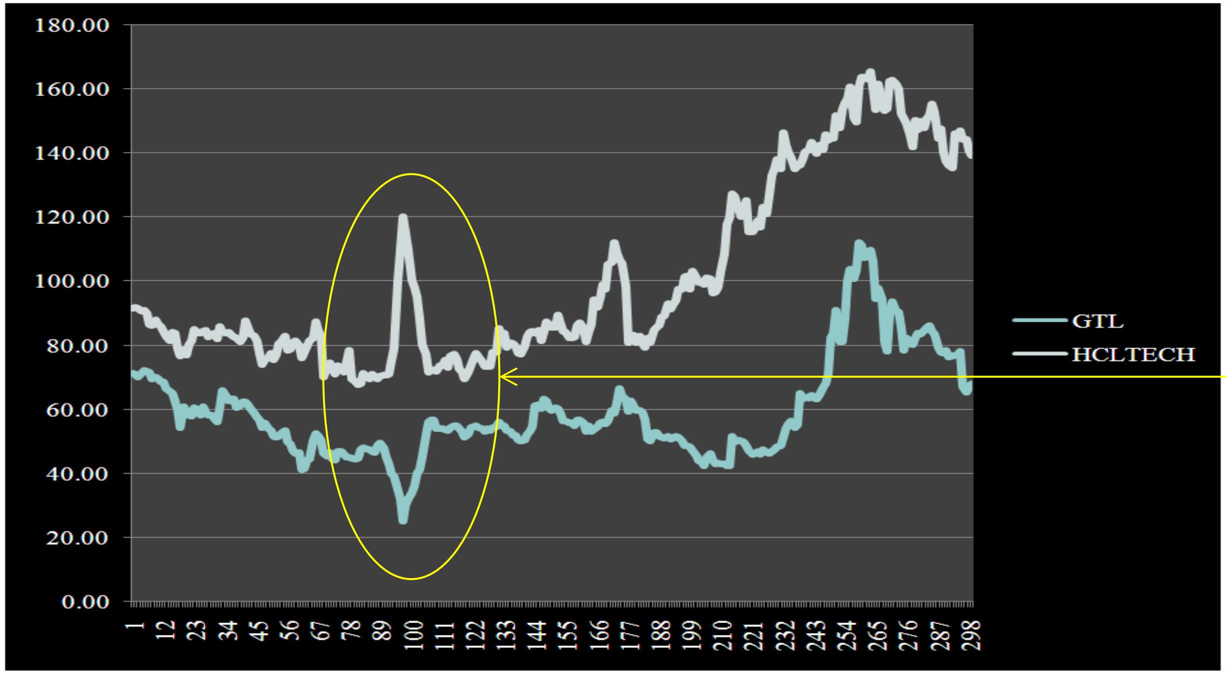
- A quest to generate returns, irrespective of market behavior, has led to the evolution of this strategy

- Certain securities, often competitors in the same sector, are associated in their day-to-day price movements. When the association breaks down, Traders take advantage of short term over pricing or under pricing of securities & invest accordingly, betting that the "spread" between the two would eventually converge

- Pairs Trading strategy works best in volatile market conditions

Simple Example of Pairs Trading

Fig. 1: Pairs Trading



This is the right opportunity for Pairs Trade

TRADER WILL GAIN PROFIT IF HE SELLS HCLTECH AND BUYS GTL

DATA SCIENCE INSTITUTE

Co-Integration and Pairs Trading

- The **stock prices** are believed to be **non-stationary** in nature.

- **If there exists a linear combination of 2 stock prices ,** such that errors of the long term relationship are stationary (mean reverting) in nature, then the **2 stocks are said to be co-integrated.**

- Pairs trading is in fact an extended application of the theory of co-integration.

- **Engle -Granger test** is used to identify the co-integrated pairs in a particular sector.

Engle Granger Test for Stock Prices

Let Y_t : TCS stock price at time t

Let X_t : IBM stock price at time t

Both the stock prices form non-stationary time series over time t.

If there exists a linear combination, such that the errors form stationary time series then TCS and IBM stocks are said to be co-integrated.

$$Y_t = \alpha + \beta X_t + \varepsilon_t$$

α = model intercept

β = regression coefficient

ε_t = error term

Engle Granger Test for Stock Prices

Regression coefficient β is estimated using ordinary least square (OLS) method

If $\beta < 0$ then the pair is discarded

Selection of regression equation is done by comparing β

$$Y_t = \alpha_1 + \beta_1 X_t + \varepsilon_t \quad \text{..........1}$$

$$X_t = \alpha_2 + \beta_2 Y_t + \varepsilon_t \quad \text{..........2}$$

The equation with higher β is selected

In case of TCS and IBM stocks $\beta 1 > \beta 2$ hence equation 1 was selected

Engle Granger Test

Engle Granger test is used to check stationarity of error time series.

Consider,

$$\Delta \varepsilon_t = \gamma^* \varepsilon_{t-1} + W_t$$

$\Delta \varepsilon_t = \varepsilon_t - \varepsilon_{t-1}$

ε_{t-1} = errors with lag 1

Objective	To test the **null hypothesis** that **time series is not stationary**

Null Hypothesis H0: Two stocks are not co-integrated

Alternate Hypothesis H1: Two stocks are co-integrated

Test Statistic	(γ*/SE(γ*)) Test statistic follows DF distribution under null
Decision Criteria	Reject the null hypothesis **if tcal < c(p)**

Engle Granger Test in Python

```
# Installing and Loading Required Libraries
import yfinance as yf
import pandas as pd
```

- **yfinance** is a library that allows import of data directly from yahoo finance.

```
#EG Test

# Define the stock symbols for Wipro and Infosys
tickers = ["WIPRO.NS", "INFY.NS"]

start_date = "2022-01-01"
end_date = "2022-12-31“

# Download the historical data
data = yf.download(tickers, start=start_date, end=end_date)
adj_close_data = data['Adj Close']
adj_close_data.columns = ['INFY','WIPRO']

infy = adj_close_data.INFY
wipro = adj_close_data.WIPRO
```

- **tickers** is a set symbols used to extract data.
- **start_date, end_date** is the start date and end date of the date range

DATA SCIENCE
INSTITUTE

Engle Granger Test in Python

```
# Import "coint" from "statsmodels"
from statsmodels.tsa.stattools import coint
coint(infy,wipro)
```

- **coint()** is a function that runs the engel garner test and returns t-statistic, p-value, Critical values for the test statistic at the 1 %, 5 %, and 10 % levels based on regression curve.

```
# Output

(-2.154422825452808,
 0.44797584761041354,
 array([-3.94132922, -3.36097908, -3.06166543]))
```

Interpretation :

- As tcal > c(p) at 5%, we accept null Hypothesis.
- Stocks INFY & WIPRO are not cointegrated.

?

What if we want to get data from other sources?

There are different packages using database APIs for data from other indices, eg. nsepy for India's National Stock Exchange (NSE) prices

DATA SCIENCE INSTITUTE

Get an Edge!

There are a number of alternative approaches for testing co-integration, apart from the Engle Granger test, such as

- CRDW (Cointegrating Regression Durbin Watson) test ✉ Simple regression of one variable on the other, and the standard Durbin-Watson test on the residuals.
- Error Correction Test ✉ The tendency of cointegrated variables to revert to common stochastic trends is expressed in terms of error-correction
- Johansen's Multivariate VAR Approach ✉ Examines the number of independent linear combinations (k) for an m time series variables set that yields a stationary process

Quick Recap

In this session, we learnt about **co-integration in time series:**

Co-integration	• When the time series is integrated and linear combination of variables in that series is also integrated, but having an order lower than the whole series, then the variables are said to be co-integrated.
Pairs Trading	• When the association between two cointegrated stocks breaks down, Traders take advantage of short term over pricing or under pricing of securities & invest accordingly, betting that the "spread" between the two would eventually converge.
EG Test in Python	• `coint()` is used to perform Engle Granger test in Python, to check if a pair of time series is co-integrated.

DATA SCIENCE INSTITUTE