INTRODUCTION TO TIME SERIES ANALYSIS USING PYTHON

DATA SCIENCE INSTITUTE

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 SCIENCE
INSTITUTE

Data Snapshot

turnover_annual data

Variables

ons 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	329825

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

DATA SCIENCE INSTITUTE

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)
```

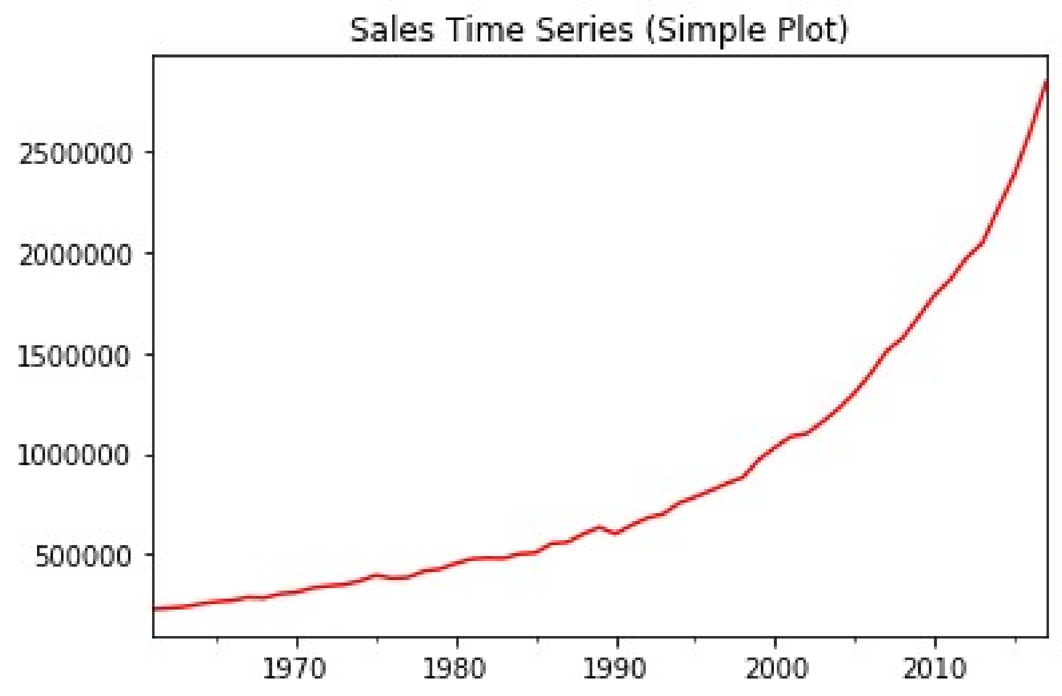
- **date_range()** creates pandas date object.
- When the time series has seasonal components, argument **freq =** can be included. It denotes number of observations per unit of time. Eg. If data is quarterly: **freq = 'Q'**, if data is monthly: **freq = 'M'**.
- **pd.Series()** combines time series 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.

DATA SCIENCE INSTITUTE

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.**

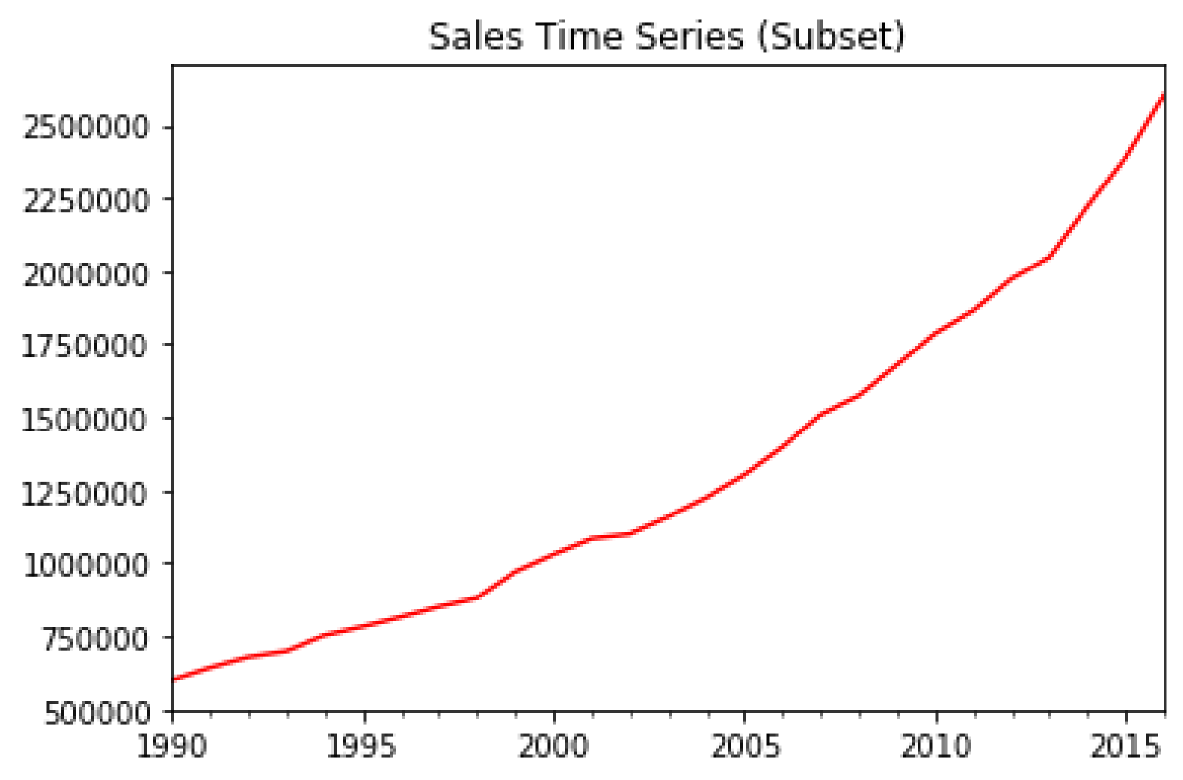
* In Pandas we can directly subset the time series object using `loc[]` function

DATA SCIENCE INSTITUTE

Subsetting Time Series in Python

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

```
# Output
```



Plot from 1990 to 2016 shows increasing trend

DATA SCIENCE INSTITUTE

What is Stationarity of Time Series ?

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

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)

A SCIENCE INSTITUTE

Stationary vs. Non-Stationary Time Series

Fig. 1: Stationary Time Series

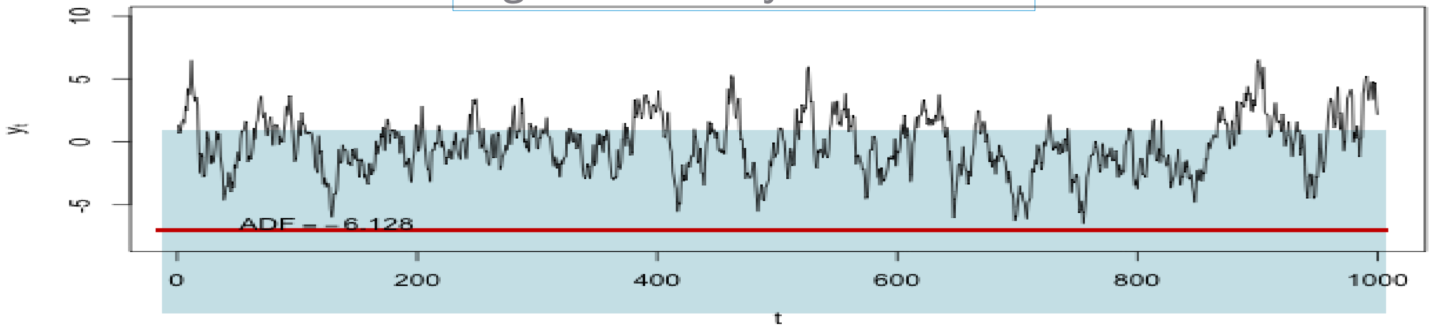
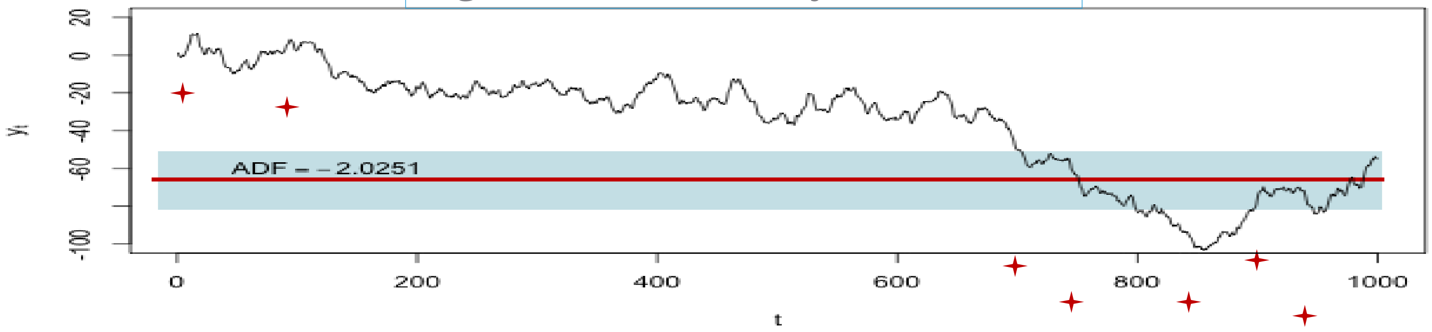


Fig. 2: Non-Stationary Time Series



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.

ATA SCIENCE
INSTITUTE

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.

DATA SCIENCE INSTITUTE

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

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

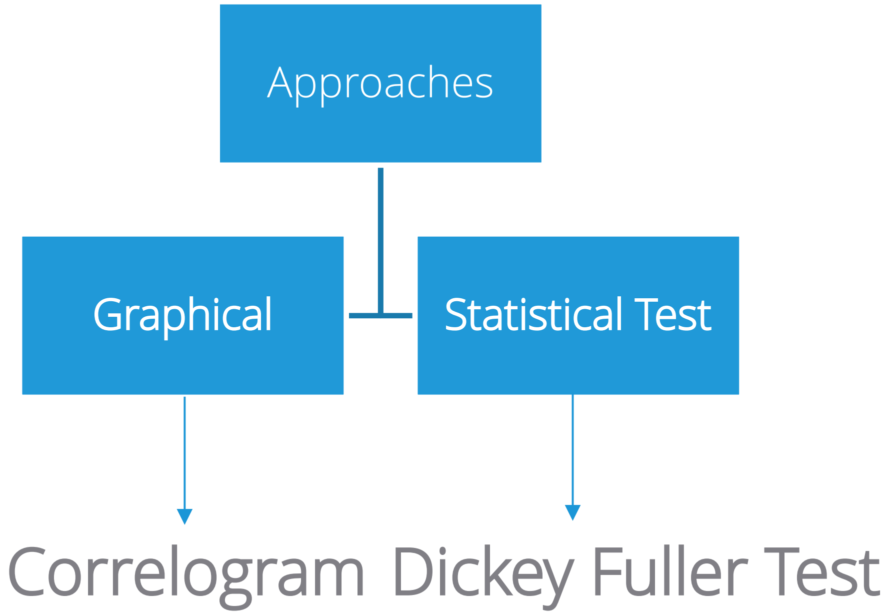
U_t is a stationary with **zero mean** and **constant variance** σ^2 . When Trend element ($\beta_1 + \beta_2 t$) is subtracted , the result is a stationary process :

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

De-trending is useful when trend is deterministic

DATA SCIENCE INSTITUTE

Identifying Stationary Time Series & Concept of Autocorrelation



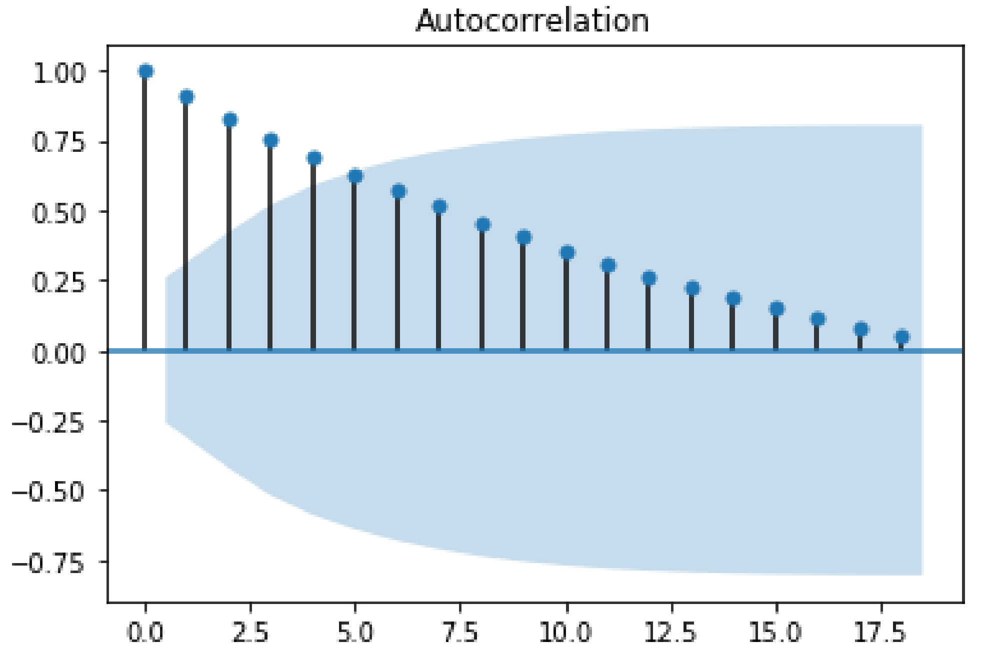
DATA SCIENCE INSTITUTE

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.

DATA SCIENCE INSTITUTE

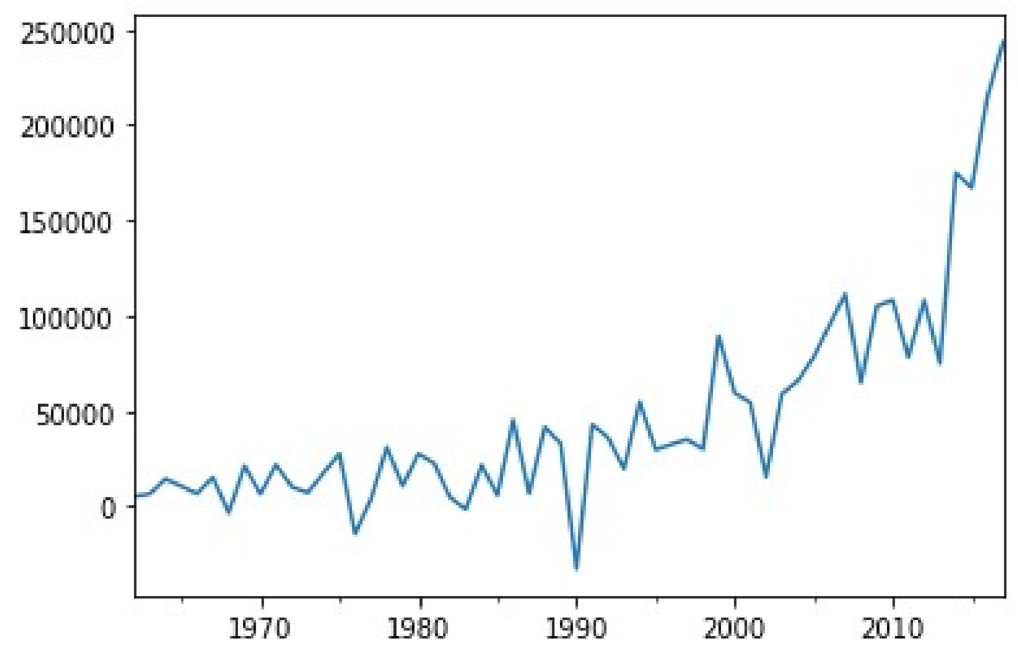
Plot of 1st Order Differenced Time Series

```
# 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
```



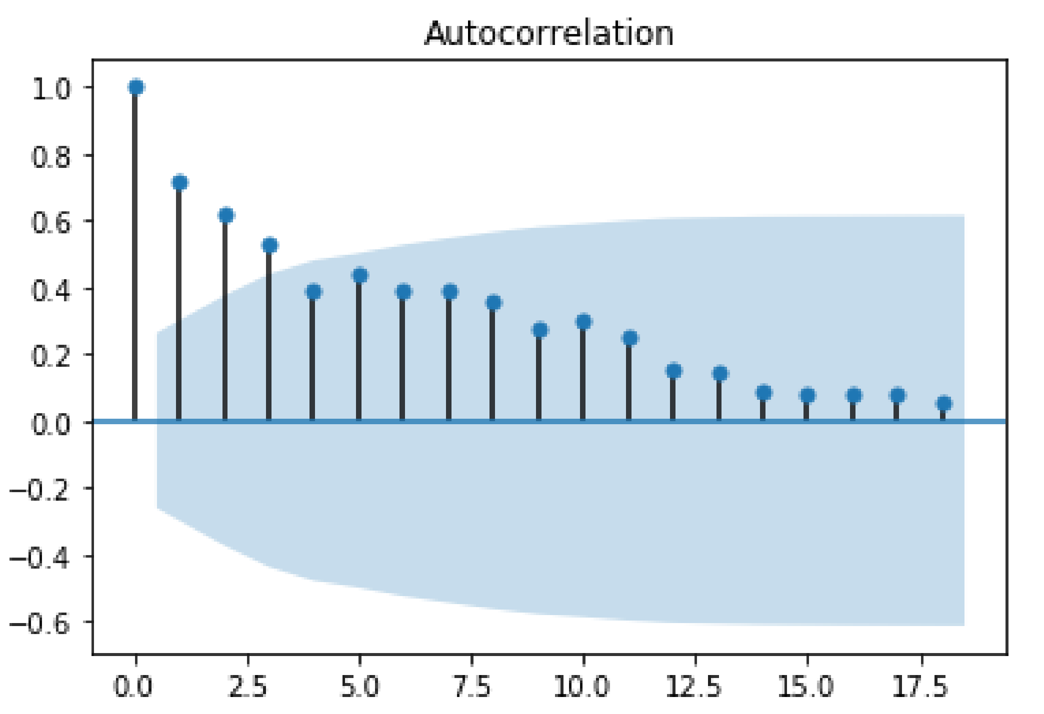
Interpretation :

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

Correlogram for 1st Order Differenced Time Series

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

```
# Output
```



Interpretation :

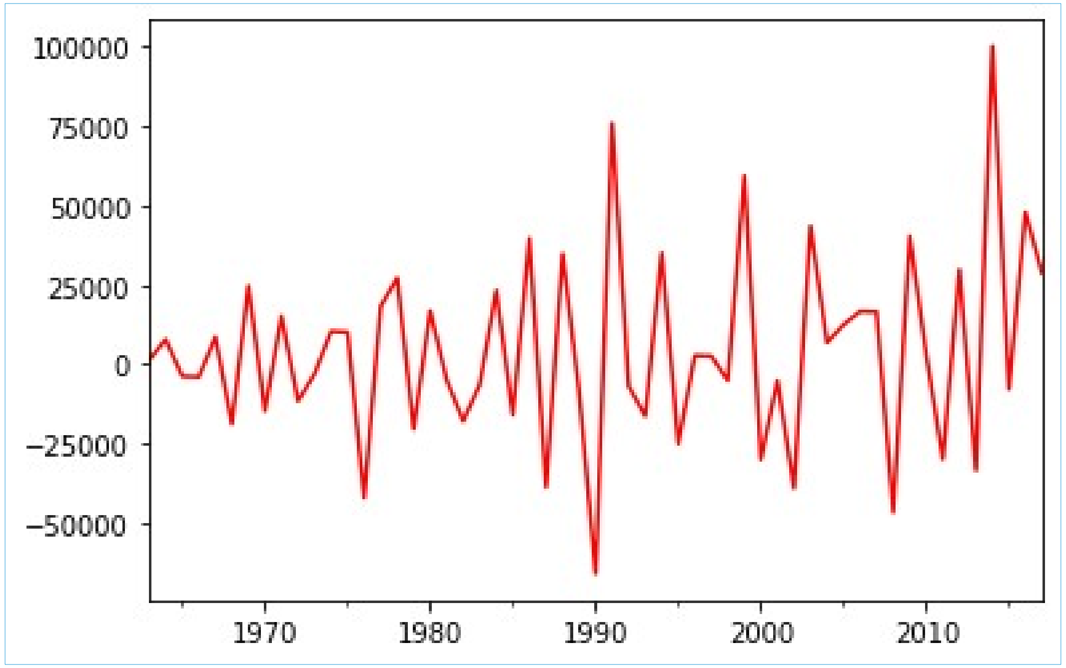
- ACF plot shows slow decay
- Stationarity is not achieved with first difference.

DATA SCIENCE INSTITUTE

Plot of 2nd Order Differenced Time Series

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

```
# Output
```



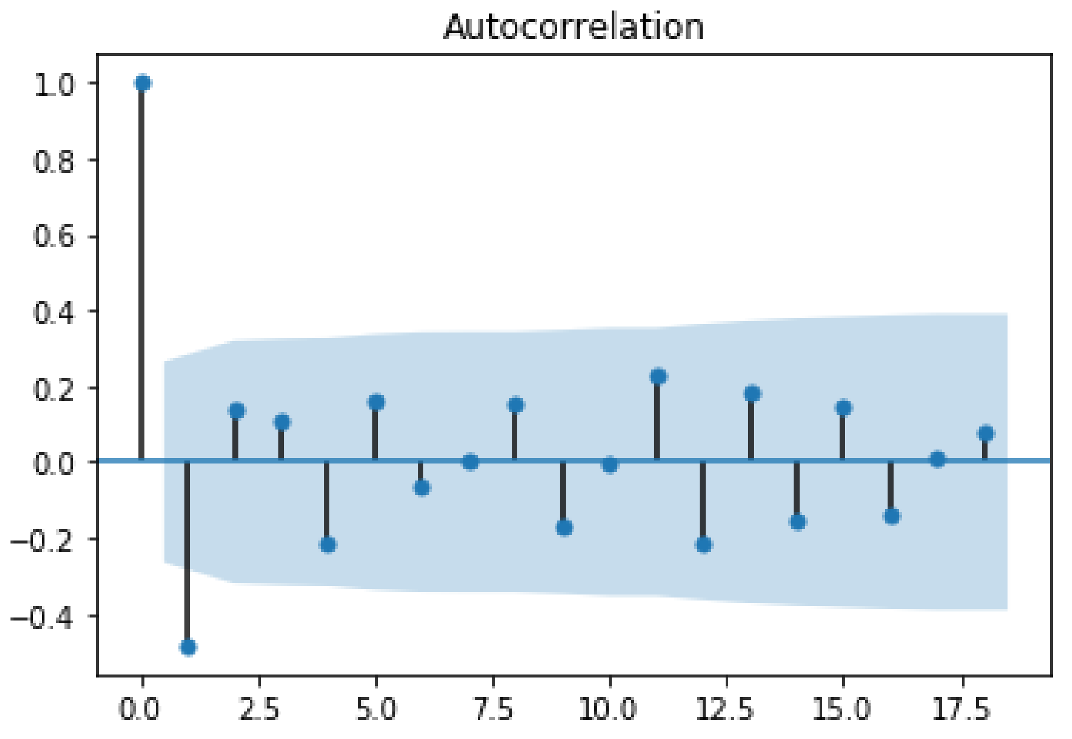
Interpretation :

- 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.

DATA SCIENCE INSTITUTE

Dickey Fuller Test

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

THANK YOU!!

DATA SCIENCE INSTITUTE