Simple Moving Average

Case Study

In [1]:
```
import pandas as pd
```

In [2]:
```
data = pd.read_excel(r'C:\Users\Admin\Desktop\Manufacturing Data.xlsx')
data.head()
```

Out[2]:

	DATE	Product
0	1972-01-01	59.9622
1	1972-02-01	67.0605
2	1972-03-01	74.2350
3	1972-04-01	78.1120
4	1972-05-01	84.7636

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 577 entries, 0 to 576
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   DATE     577 non-null    datetime64[ns]
 1   Product  577 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 9.1 KB
```

In [4]:
```
data.rename({'DATE': "Date","Product":"Production"},axis=1, inplace= True)
```

In [5]:
```
data.head()
```

Out[5]:

	Date	Production
0	1972-01-01	59.9622
1	1972-02-01	67.0605
2	1972-03-01	74.2350
3	1972-04-01	78.1120
4	1972-05-01	84.7636

In [6]:
```
data.set_index('Date', inplace = True)
```

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

Out[7]:

	Production
Date	
1972-01-01	59.9622
1972-02-01	67.0605
1972-03-01	74.2350
1972-04-01	78.1120
1972-05-01	84.7636

Let us use the rolling mean in Python to build a model for time series.

In [9]:
```
# 3 Simple moving Average

data['Production_3SMA'] = data['Production'].rolling(window=3).mean()
```

In [10]:
```
data.head()
```

Out[10]:

	Production	Production_3SMA
Date		
1972-01-01	59.9622	NaN
1972-02-01	67.0605	NaN
1972-03-01	74.2350	67.085900
1972-04-01	78.1120	73.135833
1972-05-01	84.7636	79.036867

In [11]:
```
# Evaluating the model
# Let us use MAPE as the metric

from sklearn.metrics import mean_absolute_percentage_error
```

In [12]:
```
# The syntax is mean_absolute_percentage_error(actual,predicted)

df = data.dropna()
```

In [13]:
```
df
```

Out[13]:

	Production	Production_3SMA
Date		
1972-03-01	74.2350	67.085900
1972-04-01	78.1120	73.135833
1972-05-01	84.7636	79.036867
1972-06-01	100.5960	87.823867
1972-07-01	100.1263	95.161967
...	...	...
2019-09-01	100.1741	104.348600
2019-10-01	90.1684	97.650333
2019-11-01	79.7223	90.021600
2019-12-01	75.7094	81.866700
2020-01-01	83.6290	79.686900

575 rows × 2 columns

In [14]:
```
print(mean_absolute_percentage_error(df['Production'],df['Production_3SMA']))
```

```
0.08664685538087853
```

Time Series Forecasting

Aim:- Using the past data related to predict the future data points.

Assumption:- Past behaviour will have an influence on the future behaviour.

Steps in Forecating

1) Import the required packages and then import the data

2) Identify the date column, convert it into proper datetime64 format.

3) Convert the date column into index.

4) check for any missing values in the data.

5) Replace the missing value(Ffill, Bfill).

6) Identify stationarity in the series

```
* Visualizataion(least recommended)

* Statistical test ( ADF,KPSS) based on p-value we would conclude
```

7) Converting non stationary data into stationary data

```
* Differencing

* Transformation - log, sqrt, cbrt
```

ARIMA model

ARIMA is a model which is used for time series forecasting. There are 3 major components in ARIMA.

1) AR - Auto Regressive Component

2) I - Integerated (How many time we are differencing to make the data stationary.

3) MA - Moving Average Component

A time series data using ARIMA model can be forecasted in two ways. Regression and Moving Average.

Regression - It is forecating the value based on previous input values.

```
Y = mX + C + e
```

Moving Average - It is forecasting the values based on previous residuals (Error).

In [15]:
```
data = pd.read_excel(r'C:\Users\Admin\Desktop\Manufacturing Data.xlsx')
data.head()
```

Out[15]:

	DATE	Product
0	1972-01-01	59.9622
1	1972-02-01	67.0605
2	1972-03-01	74.2350
3	1972-04-01	78.1120
4	1972-05-01	84.7636

In [16]:
```
data.rename({'DATE': "Date","Product":"Production"},axis=1, inplace= True)
```

In [17]:
```
data.set_index('Date', inplace = True)
```

In [18]:
```
data.isnull().sum()
```

Out[18]:
```
Production    0
dtype: int64
```

In [24]:
```
from statsmodels.tsa.stattools import adfuller
```

In [25]:
```
result = adfuller(data['Production'])
```

In [27]:
```
print('Test Statistics:',result[0])
print('p value:', result[1])
print('Critical Values:\n',result[4])

p_value = result[1]
if p_value < 0.05:
    print('The series is stationary')
else:
    print('The series is non stationary')
```

```
Test Statistics: -1.75800877551054
p value: 0.401499289940763
Critical Values:
 {'1%': -3.4421447800270673, '5%': -2.8667429272780858, '10%': -2.5695409929766093}
The series is non stationary
```

In [28]:
```
# Since the series is not stationary , we would use differencing technique to make series stationary

df1 = data['Production'] - data['Production'].shift(1)
```

In [29]:
```
df1
```

Out[29]:
```
Date
1972-01-01        NaN
1972-02-01     7.0983
1972-03-01     7.1745
1972-04-01     3.8770
1972-05-01     6.6516
                ...
2019-09-01    -2.4344
2019-10-01   -10.0057
2019-11-01   -10.4461
2019-12-01    -4.0129
2020-01-01     7.9196
Name: Production, Length: 577, dtype: float64
```

In [32]:
```
result= adfuller(df1.dropna())
```

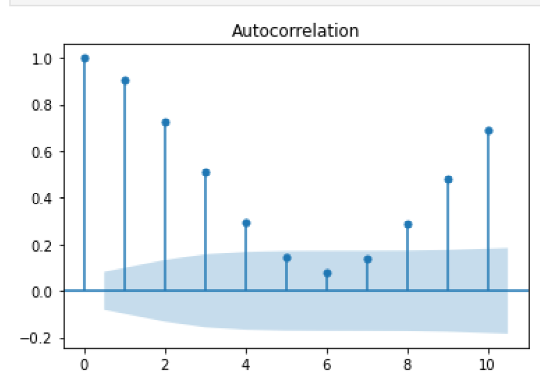
In [34]:
```
print('Test Statistics:',result[0])
print('p value:', result[1])
print('Critical Values:\n',result[4])

p_value = result[1]
if p_value < 0.05:
    print('The series is stationary')
else:
    print('The series is non stationary')
```

```
Test Statistics: -6.627957542548401
p value: 5.811621562056256e-09
Critical Values:
 {'1%': -3.4421660928041633, '5%': -2.8667523104859627, '10%': -2.56954599309042}
The series is stationary
```

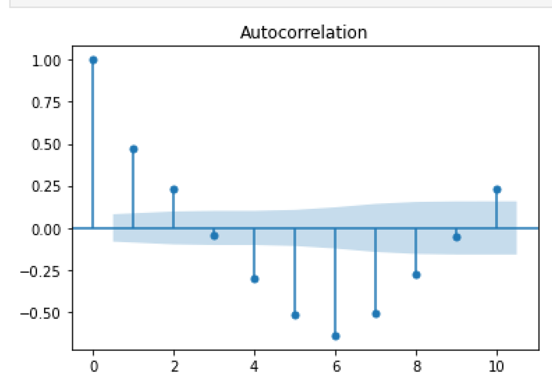
In [35]:
```
# To identify p and q we would plot acg graph
# When we get positive autocorrelation and significant autocorrelation we would use Auto regressive
## When we get negative autocorrelation and significant autocorrelation we would use moving Average.
```

In [37]:
```
import matplotlib.pyplot as plt
```

In [39]:
```
from statsmodels.graphics.tsaplots import plot_acf
ACF = plot_acf(data['Production'],lags=10)
```



In [40]:
```
from statsmodels.graphics.tsaplots import plot_acf
ACF = plot_acf(df1.dropna(),lags=10)
```



In [ ]: