# Univariate Forecasting Using Seasonal Auto Regressive Integrated Moving Average (SARIMA)

## *Introduction:*

The forecasting is structured in three phases. The dataset was provided for training and testing purpose and the forecasting is carried out in three different stages. For the 1st phase 4 values having ID stamp from 61-64 were to be forecasted and in the 2nd stage the next 4 values having ID stamp 64-68 were forecasted while adding the previous forecasted values to the historical data. In the 3rd and final stage final 4 values were to be forecasted having ID stamp 69-72 while adding the previous 1st and 2nd stage forecasted values to the historical data.

## *Dataset:*

The Dataset file is named **'exam1.csv'**. The dataset provided has 2 variable columns namely 'ID' & 'Var'. The total **number of observations given is 60** having ID stamp from 1-60. The dataset provided is a Univariate Series.

## *Methodology:*

The coding was done on Python (Jupyter Notebook). The various functions used in the code are elaborated in the Jupyter notebook using 'Markdown Feature'. The forecasting technique used is called ARIMA model.

**ARIMA** stands for **Autoregressive integrated moving average**. The model is fitted to time series data either to better understand the data or to predict future points in the series (forecasting). The same code is used for all the three phases. The performance criteria used to evaluate the model is MAE (In Python) and MAPE (In Excel).

## *Code and Theory:*

**NumPy:**
It is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

**Pandas:**
In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

**Statsmodels:**
statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.

**Matplotlib:**
Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits.

```
In [1]: import numpy as np
        import pandas as pd
        import statsmodels.api as sm
        import matplotlib.pyplot as plt
        %matplotlib inline
```

We assign the variable 'df' to our data file.

```
In [2]: df=pd.read_csv('exam1.csv')
```

We check if the data has been loaded correctly. Now in python the **indexing starts by default from 0**.

```
In [3]: df.head(10)
```

Out[3]:

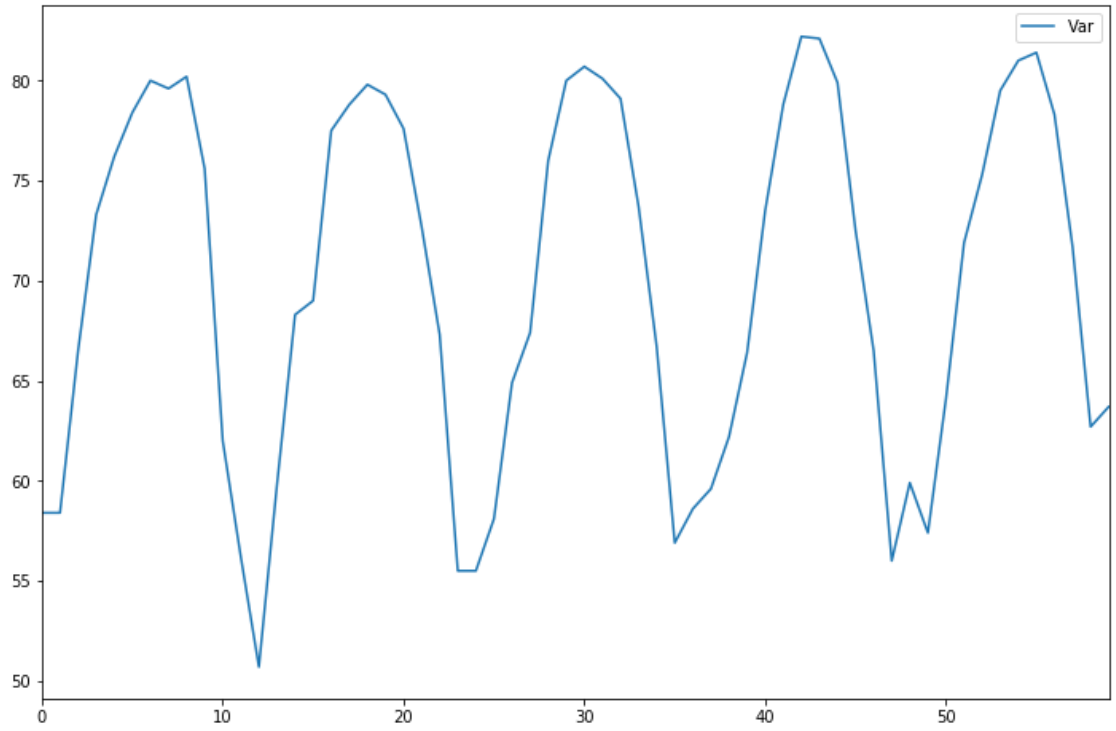
| | ID | Var |
|---|---|---|
| 0 | 1 | 58.4 |
| 1 | 2 | 58.4 |
| 2 | 3 | 66.5 |
| 3 | 4 | 73.3 |
| 4 | 5 | 76.2 |
| 5 | 6 | 78.4 |
| 6 | 7 | 80.0 |
| 7 | 8 | 79.6 |
| 8 | 9 | 80.2 |
| 9 | 10 | 75.6 |

```
In [4]: df.tail(5)
```

Out[4]:

| | ID | Var |
|---|---|---|
| 55 | 56 | 81.4 |
| 56 | 57 | 78.3 |
| 57 | 58 | 71.7 |
| 58 | 59 | 62.7 |
| 59 | 60 | 63.7 |

**Plots for understanding & Preliminary Observation:**



***We can clearly see that the data follows a trend of going upward, reaching a certain peak and again going down (Dipping in values). This seasonal trend follows after every 12 observations.***

Now we recall the **ARIMA model** from statsmodel package, which we previously loaded.

```
In [19]: model=sm.tsa.statespace.SARIMAX(df['Var'],order=(1, 0, 0),seasonal_order=(0,1,1,12))
         results=model.fit()
```

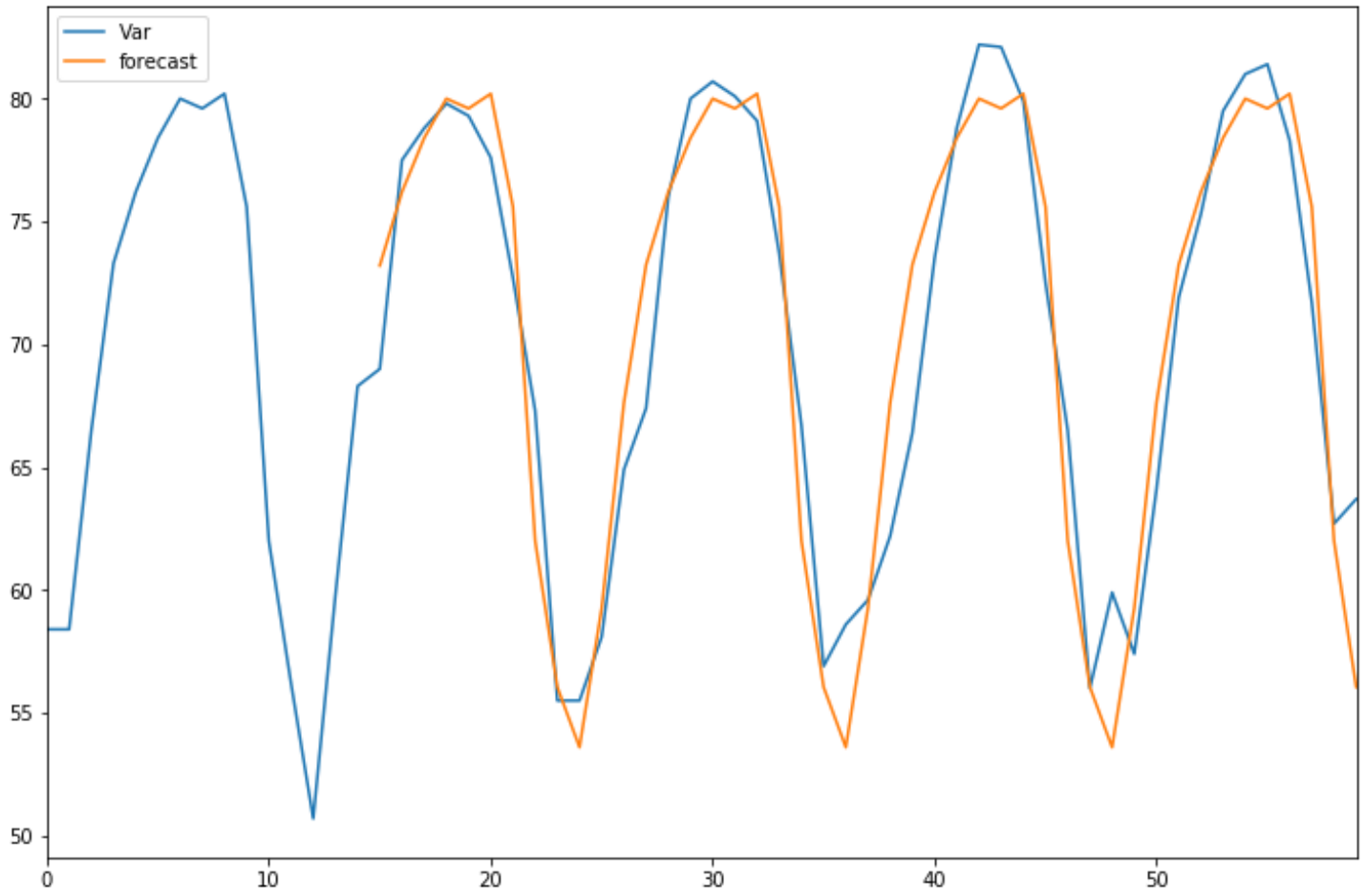
We Provide the column 'Var' to the model for training and testing purpose. Here there two parameters of the ARIMA model namely Order & Seasonal Order which we need to Discuss.

**Order:**
The (p, d, q) order of the model for the number of AR parameters, differences, and MA parameters. d must be an integer indicating the integration order of the process, while p and q may either be an integers indicating the AR and MA orders (so that all lags up to those orders are included) or else iterables giving specific AR and / or MA lags to include. Default is an AR (1) model: (1,0,0). For this particular model we decided to go ahead with default AR (1) model.

**Seasonal Order:**
The (P, D, Q, s) order of the seasonal component of the model for the AR parameters, differences, MA parameters, and periodicity. d must be an integer indicating the integration order of the process, while p and q may either be an integer indicating the AR and MA orders (so that all lags up to those orders are included) or else iterables giving specific AR and / or MA lags to include. s is an integer giving the periodicity (number of periods in season), often it is 4 for quarterly data or 12 for monthly data. Default is no seasonal effect. For this particular model we are taking value of 's' as 12, since there is a seasonal trend after every 12 observations. The P & D values are taken as 1 for reference. The Q value is taken as 1 for 1 individual observation.

We now do a forecast (for training & testing purpose) using predictor function. We forecast the values from 15th observation to 59th (last) observation. We plot a graph of actual observations and forecasted values to see the difference. We can see the forecast line is pretty good and it is somewhat parsing through the actual line.

```
In [7]: df['forecast']=results.predict(start=15,end=59,dynamic=True)
        df[['Var','forecast']].plot(figsize=(12,8))
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1586cdb4188>
```




```
In [8]: predicted=df['forecast'][15:]
```

We set a variable for predicted values as 'predicted'.

```
In [9]: true=df['Var'][15:]
```

For the actual values we set the variable 'true'.

We calculate the performance of the model using Mean Absolute Error using Machine Learning package 'sklearn.metrics'.

```
In [10]: from sklearn.metrics import mean_absolute_error
         mean_absolute_error(true,predicted)

Out[10]: 2.3608647667072047
```

We get a MAE of 2.36 which is very low and acceptable.

In [11]:
```
future_dates=[df.index[-1]+i for i in range(0,5)]
```

We create a dataoffset and convert it to a dataframe.

In [12]:
```
future_datest_df=pd.DataFrame(index=future_dates[1:],columns=df.columns)
```

We now showcase the future dataset that needs to be forecasted.

In [13]:
```
future_datest_df
```

Out[13]:

| | ID | Var | forecast |
|---|---|---|---|
| 60 | NaN | NaN | NaN |
| 61 | NaN | NaN | NaN |
| 62 | NaN | NaN | NaN |
| 63 | NaN | NaN | NaN |

We now concatenate (link values together in a chain or series) the data with the previous dataset.

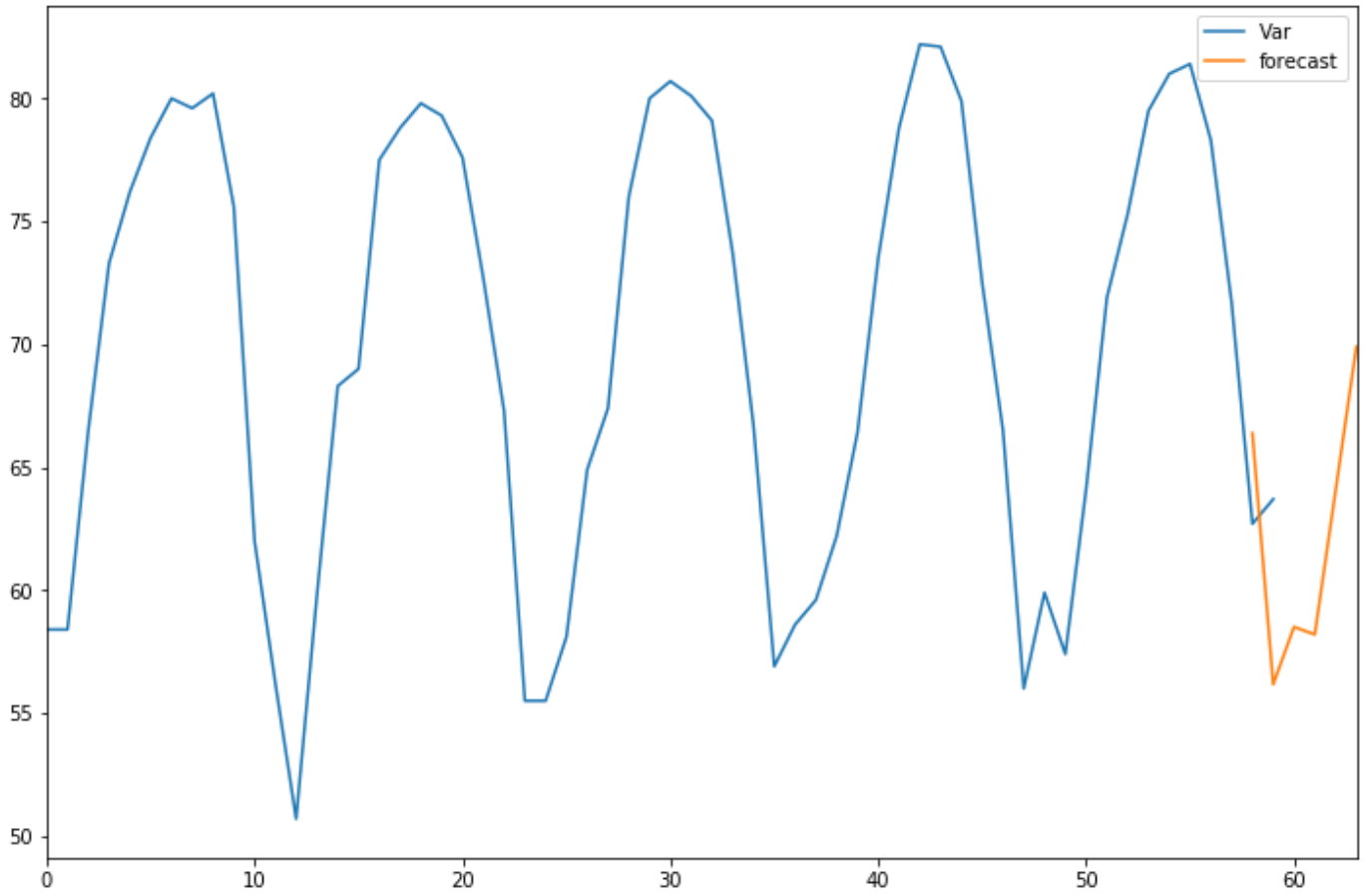
In [14]:
```
future_df=pd.concat([df,future_datest_df])
```

**Now we forecast the values from the point that we want to the wnd value.**

Note: We have forecasted the 58th and 59th value to link the forecasted line with actual data line in order to show the continuity.

In [15]:
```
future_df['forecast'] = results.predict(start = 58, end = 63, dynamic= True)
future_df[['Var', 'forecast']].plot(figsize=(12, 8))
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1586e622108>
```




```
In [16]: future_df['forecast'][60:]

Out[16]: 60    58.504228
         61    58.196701
         62    64.035552
         63    69.898070
         Name: forecast, dtype: float64
```

The above values are the forecasted values that we needed for the phase 1.

We use the same code for Phase 2 and 3.

*Conclusion:*

## Mean absolute percentage error:

The MAPE is calculated using Excel. The real values and forecasted values are showcased against each other.

| | | | | | | (Abs(Var-FCST))/Var | MAPE |
|---|---|---|---|---|---|---|---|
| 60 | 63.7 | - | | | | | |
| 61 | 60.4 | 58.50423 | | | | 0.031386954 | |
| 62 | 57.7 | 58.1967 | Phase 1 | | Phase 1 | 0.008608336 | 2.256174 |
| 63 | 65.6 | 64.03555 | | | | 0.023848293 | % |
| 64 | 68.1 | 69.89807 | | | | 0.026403377 | |
| 65 | 74.7 | 75.09507 | | | | 0.005288728 | |
| 66 | 79.3 | 79.30267 | Phase 2 | | Phase 2 | 3.36318E-05 | 0.624105 |
| 67 | 81.1 | 81.15742 | | | | 0.000708052 | % |
| 68 | 82.8 | 81.23228 | | | | 0.018933804 | |
| 69 | 80 | 79 | | | | 0.015456662 | |
| 70 | 74 | 73 | Phase 3 | | Phase 3 | 0.017823122 | 1.547587 |
| 71 | 63.2 | 64 | | | | 0.009101329 | % |
| 72 | 59.2 | 60 | | | | 0.019522348 | |

This is acceptable and proves that the forecasting model works perfectly and has good accuracy.