Forecasting and Modelling on Timeseries.

MAY 1

Authored by: Bishwajit Ghorai

Forecasting and Modelling in Timeseries Data

I have used the Federal Reserve’s timeseries of foreign exchange rate per dollar for this project. The data ranges from 2000 to 2019

Initially, the data looks something like this

	Unnamed: 0	Time Serie	AUSTRALIA - AUSTRALIAN DOLLAR/US$	EURO AREA - EURO/US$	NEW ZEALAND - NEW ZELAND DOLLAR/US$	UNITED KINGDOM - UNITED KINGDOM POUND/US$	BRAZIL - REAL/US$	CANADA - CANADIAN DOLLAR/US$	CHINA - YUAN/US$	HONG KONG - HONG KONG DOLLAR/US$	INDIA - INDIAN RUPEE/US$	KOREA - WON/US$
0	0	2000-01-03	1.5172	0.9847	1.9033	0.6146	1.805	1.4465	8.2798	7.7765	43.55	1128
1	1	2000-01-04	1.5239	0.97	1.9238	0.6109	1.8405	1.4518	8.2799	7.7775	43.55	1122.5
2	2	2000-01-05	1.5267	0.9676	1.9339	0.6092	1.856	1.4518	8.2798	7.778	43.55	1135
3	3	2000-01-06	1.5291	0.9686	1.9436	0.607	1.84	1.4571	8.2797	7.7785	43.55	1146.5
4	4	2000-01-07	1.5272	0.9714	1.938	0.6104	1.831	1.4505	8.2794	7.7783	43.55	1138
...	...	...	...	...	...	...	...	...	...	...	...	...
5212	5212	2019-12-25	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
5213	5213	2019-12-26	1.4411	0.9007	1.5002	0.7688	4.0602	1.3124	6.9949	7.788	71.28	1161.18
5214	5214	2019-12-27	1.4331	0.8949	1.4919	0.7639	4.0507	1.3073	6.9954	7.7874	71.45	1160.87
5215	5215	2019-12-30	1.4278	0.8915	1.4846	0.761	4.0152	1.3058	6.9864	7.7857	71.3	1155.75
5216	5216	2019-12-31	1.4225	0.8907	1.4826	0.7536	4.019	1.2962	6.9618	7.7894	71.36	1155.46

	MEXICO - MEXICAN PESO/US$	SOUTH AFRICA - RAND/US$	SINGAPORE - SINGAPORE DOLLAR/US$	DENMARK - DANISH KRONE/US$	JAPAN - YEN/US$	MALAYSIA - RINGGIT/US$	NORWAY - NORWEGIAN KRONE/US$	SWEDEN - KRONA/US$	SRI LANKA - SRI LANKAN RUPEE/US$	SWITZERLAND - FRANC/US$	TAIWAN - NEW TAIWAN DOLLAR/US$	THAILAND - BAHT/US$
0	9.4015	6.126	1.6563	7.329	101.7	3.8	7.964	8.443	72.3	1.5808	31.38	36.97
1	9.457	6.085	1.6535	7.218	103.09	3.8	7.934	8.36	72.65	1.5565	30.6	37.13
2	9.535	6.07	1.656	7.208	103.77	3.8	7.935	8.353	72.95	1.5526	30.8	37.1
3	9.567	6.08	1.6655	7.2125	105.19	3.8	7.94	8.3675	72.95	1.554	31.75	37.62
4	9.52	6.057	1.6625	7.2285	105.17	3.8	7.966	8.415	73.15	1.5623	30.85	37.3
...	...	...	...	...	...	...	...	...	...	...	...	...
5212	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
5213	18.944	14.132	1.354	6.7295	109.67	4.1337	8.8799	9.4108	181.3	0.9808	30.11	30.15
5214	18.819	14.025	1.352	6.6829	109.47	4.126	8.8291	9.3405	181.35	0.9741	30.09	30.14
5215	18.863	14.056	1.3483	6.6589	108.85	4.1053	8.7839	9.3145	181.6	0.9677	30.04	29.94
5216	18.86	13.973	1.3446	6.6554	108.67	4.0918	8.7823	9.3425	181.3	0.9677	29.91	29.75

As you above the data frame has a null values which are denoted by **ND** and an unwanted indexing which is named as **Unnamed:0.** Again, if you see above data frame the indexing for every country is given something like **'AUSTRALIA - AUSTRALIAN DOLLAR/US$'** which represents country's name followed by **‘–‘sign** and then currency of the country/U.S. dollar**.** We are required to clean the data and make it appropriate for further use.
For null values, Interpolation was done which insert the null values with new data points within the range of known data points. I dropped the **Unnamed: 0** column and replaced the **Time Serie** variable with **DATE**. After further analyzing the data it was seen that the datatype of each variable was **‘object’.** So, I changed the datatype into appropriate datatype which where **‘Float’ & ‘datetime’.**

Final dataset looked something like this,

DATE	AUSTRALIA	EURO AREA	NEW ZEALAND	UNITED KINGDOM	BRAZIL	CANADA	CHINA	HONG KONG	INDIA	KOREA	MEXICO
2000-01-03	1.517200	0.984700	1.903300	0.614600	1.805000	1.4465	8.2798	7.776500	43.55	1128.000000	9.401500
2000-01-04	1.523900	0.970000	1.923800	0.610900	1.840500	1.4518	8.2799	7.777500	43.55	1122.500000	9.457000
2000-01-05	1.526700	0.967600	1.933900	0.609200	1.856000	1.4518	8.2798	7.778000	43.55	1135.000000	9.535000
2000-01-06	1.529100	0.968600	1.943600	0.607000	1.840000	1.4571	8.2797	7.778500	43.55	1146.500000	9.567000
2000-01-07	1.527200	0.971400	1.938000	0.610400	1.831000	1.4505	8.2794	7.778300	43.55	1138.000000	9.520000
...	...	...	...	...	...	...	...	...	...	...	...
2019-12-27	1.433100	0.894900	1.491900	0.763900	4.050700	1.3073	6.9954	7.787400	71.45	1160.870000	18.819000
2019-12-28	1.431333	0.893767	1.489467	0.762933	4.038867	1.3068	6.9924	7.786833	71.40	1159.163333	18.833667
2019-12-29	1.429567	0.892633	1.487033	0.761967	4.027033	1.3063	6.9894	7.786267	71.35	1157.456667	18.848333
2019-12-30	1.427800	0.891500	1.484600	0.761000	4.015200	1.3058	6.9864	7.785700	71.30	1155.750000	18.863000
2019-12-31	1.422500	0.890700	1.482600	0.753600	4.019000	1.2962	6.9618	7.789400	71.36	1155.460000	18.860000

DATE	SOUTH AFRICA	SINGAPORE	DENMARK	JAPAN	MALAYSIA	NORWAY	SWEDEN	SRI LANKA	SWITZERLAND	TAIWAN	THAILAND
2000-01-03	6.126000	1.656300	7.3290	101.700000	3.8000	7.964000	8.443000	72.300000	1.580800	31.380000	36.970000
2000-01-04	6.085000	1.653500	7.2180	103.090000	3.8000	7.934000	8.360000	72.650000	1.556500	30.600000	37.130000
2000-01-05	6.070000	1.656000	7.2080	103.770000	3.8000	7.935000	8.353000	72.950000	1.552600	30.800000	37.100000
2000-01-06	6.080000	1.665500	7.2125	105.190000	3.8000	7.940000	8.367500	72.950000	1.554000	31.750000	37.620000
2000-01-07	6.057000	1.662500	7.2285	105.170000	3.8000	7.966000	8.415000	73.150000	1.562300	30.850000	37.300000
...	...	...	...	...	...	...	...	...	...	...	...
2019-12-27	14.025000	1.352000	6.6829	109.470000	4.1260	8.829100	9.340500	181.350000	0.974100	30.090000	30.140000
2019-12-28	14.035333	1.350767	6.6749	109.263333	4.1191	8.814033	9.331833	181.433333	0.971967	30.073333	30.073333
2019-12-29	14.045667	1.349533	6.6669	109.056667	4.1122	8.798967	9.323167	181.516667	0.969833	30.056667	30.006667
2019-12-30	14.056000	1.348300	6.6589	108.850000	4.1053	8.783900	9.314500	181.600000	0.967700	30.040000	29.940000
2019-12-31	13.973000	1.344600	6.6554	108.670000	4.0918	8.782300	9.342500	181.300000	0.967700	29.910000	29.750000

As, you see we have a multivariate timeseries data. My work deals only with stocks prices of **INDIA.** So, I dropped the rest of the variable and converted into a univariate timeseries data.

```
unidf = finaldf.drop(columns=['AUSTRALIA' ,'EURO AREA'  ,'NEW ZEALAND'  ,'UNITED KINGDOM' ,'BRAZIL' ,'CANADA' ,'CHINA'  ,'HONG KONG'  ,'KOREA'  ,'MEXICO' ,'SOUTH AFRICA' ,'SINGAPORE'  ,'DENMARK'.....
unidf
```

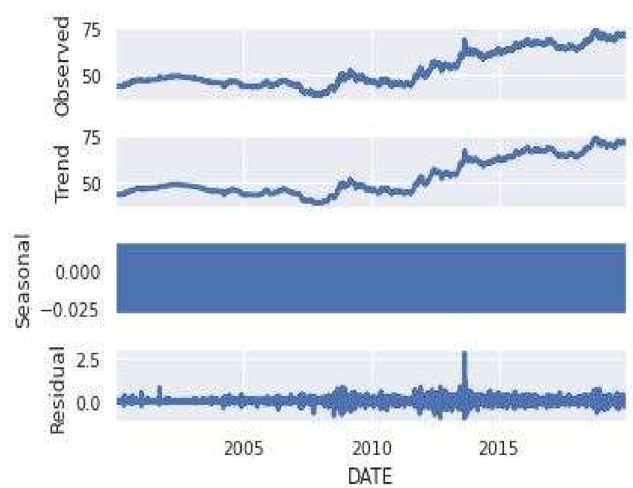
DATE	INDIA
2000-01-03	43.55
2000-01-04	43.55
2000-01-05	43.55
2000-01-06	43.55
2000-01-07	43.55
...	...
2019-12-27	71.45
2019-12-28	71.40
2019-12-29	71.35
2019-12-30	71.30
2019-12-31	71.36

Dickey Fuller Test:

we have to check whether the data is stationary or not. Time series **are** stationary if **they do not have trend or seasonal effects.**

```
Test Statistic                    -0.132362
p-value                            0.946118
Lags Used                         33.000000
Number of Observations Used     5183.000000
dtype: float64
Data is NOT stationary
```

Looks like the dataset isn't stationary. Lets look for the trends and sesonsal effects




```
DATE
2000-01-03     0.011160
2000-01-04     0.017230
2000-01-05     0.008852
2000-01-06     0.003958
2000-01-07    -0.026145
2000-01-08    -0.013735
2000-01-09    -0.001320
2000-01-10     0.011160
2000-01-11     0.017230
2000-01-12     0.008852
2000-01-13     0.003958
```

So, we can see that the trends repeats itself after every 7 days. Forecasting with this type of data can only done for a very short window as there is high seasoal change in the data.

Forecasting:

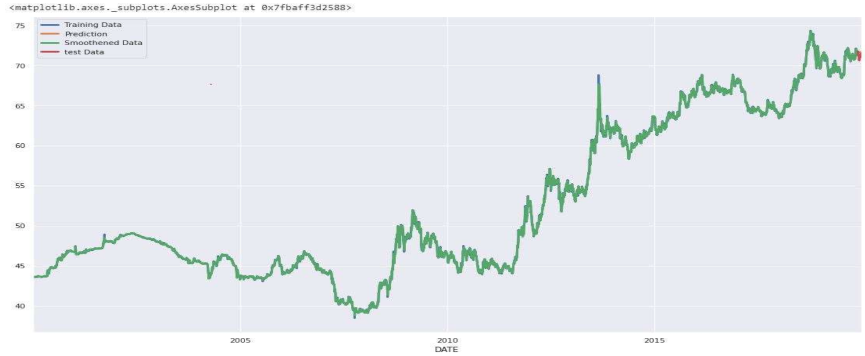
Forecasting of a time series is really important as its shows how stocks of Indian currency is doing against the US Dollar in the stock market. As the stock market has huge seasonality, we predict the upcoming month stock by simple moving average, exponential smoothing, and a few years by Autoregressive Integrated Moving Average(ARIMA).

Simple Moving Average:

Simple moving average is the simplest type of forecasting. Basically, a simple moving average is calculated by adding up the last 'n' period's values and then dividing that number by 'n'.

DATE	INDIA	sma
2019-11-26	71.510000	71.600000
2019-11-27	71.260000	71.385000
2019-11-28	71.505000	71.382500
2019-11-29	71.750000	71.627500
2019-11-30	71.716667	71.733333

<matplotlib.axes._subplots.AxesSubplot at 0x7fbaff3d2588>



From graph its impossible to determine anything as the predicted range is quite small. But if we predict for a bigger range what could happen?

As you see, the prediction are way off. So, forecasting for a very small range is much suitable for this type of dataset.

```
get_mape(test.INDIA,test['SMA_prediction'])
```

```
0.68
```

We check the mean absolute error of the moving average which is **0.68** which is quite decent. RSME was **0.5442324576503398** which was also quite good.

Exponential Smoothing:

The drawbacks of the simple moving average technique is that it gives equal weight to all the previous observations used in forecasting the future value. Exponential smoothing technique assigns differential weights to past observations.

DATE	INDIA	sma	wma
2000-01-03	43.550000	NaN	43.550000
2000-01-04	43.550000	43.550000	43.550000
2000-01-05	43.550000	43.550000	43.550000
2000-01-06	43.550000	43.550000	43.550000
2000-01-07	43.550000	43.550000	43.550000
...	...	...	...
2019-11-26	71.510000	71.600000	71.573744
2019-11-27	71.260000	71.385000	71.364581
2019-11-28	71.505000	71.382500	71.458194
2019-11-29	71.750000	71.627500	71.652731
2019-11-30	71.716667	71.733333	71.695355

DATE	INDIA	SMA_prediction	WMA_prediction
2019-12-01	71.683333	71.473611	71.473419
2019-12-02	71.650000	71.493231	71.494787
2019-12-03	71.700000	71.517506	71.518253
2019-12-04	71.470000	71.542256	71.542742
2019-12-05	71.260000	71.567498	71.567904
2019-12-06	71.260000	71.593081	71.593413
2019-12-07	71.180000	71.616684	71.616276
2019-12-08	71.100000	71.637240	71.636957
2019-12-09	71.020000	71.651981	71.650457
2019-12-10	70.840000	71.660936	71.660303

So, we did forecast with exponential smoothing and in it's plot is quite similar to moving average plot.so me need to check the mean absolute error (MAE). Its, 0.68 which is same as SMA. RSME was **0.5441636046916125,** which is just a bit better than SMA. Forecast looks pretty good with exponential smoothing.

ARIMA:

ARIMA stands for Autoregressive Integrated Moving Average and it depends on three key variables p, d, q to be successful.

p = number of lags / order of AR terms

d = order of differencing

q = number of lagged forecast errors / order of MA terms.

To get three variables, we have to do two tests to find the optimum features. We find optimum features or order of the MA process using the ACF plot, as being an M.A. process it doesn't have seasonal and trend components so we get only the residual relationship with the lags of time series in the ACF plot.

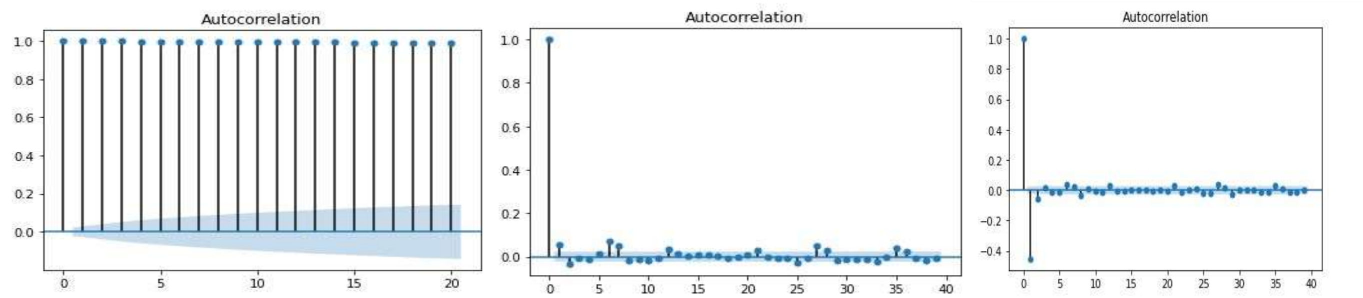
.

We did three A.C.F test with 0, 1, 2 order. From 1st order differencing we see one lag can be found above the significance level and thus q = 1. From 2nd order differencing we see timeseries is stationary at d = 1 where only the first lag is above the significance level.

```
acf_plot = plot_acf( unidf.INDIA,lags=20)
# Significant coorelation between lags
```

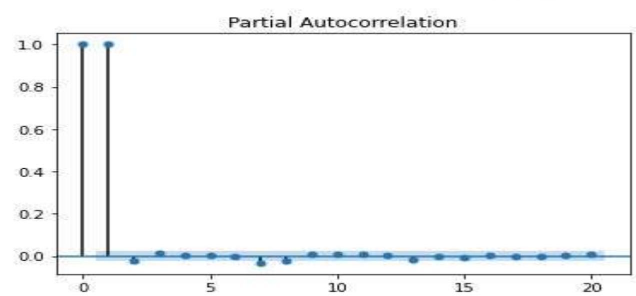
```
#1st Order Differencing
acf_plot=plot_acf(unidf.INDIA.diff().dropna())
```

```
#2nd Order Differencing
acf_plot = plot_acf(unidf.INDIA.diff().diff().dropna())
```



We find optimum features or order of the AR process using the PACF plot, as it removes variations explained by earlier lags so we get only the relevant features.

```
# No significant pattern in pacf
pacf_plot = plot_pacf( unidf.INDIA,lags=20 )
```



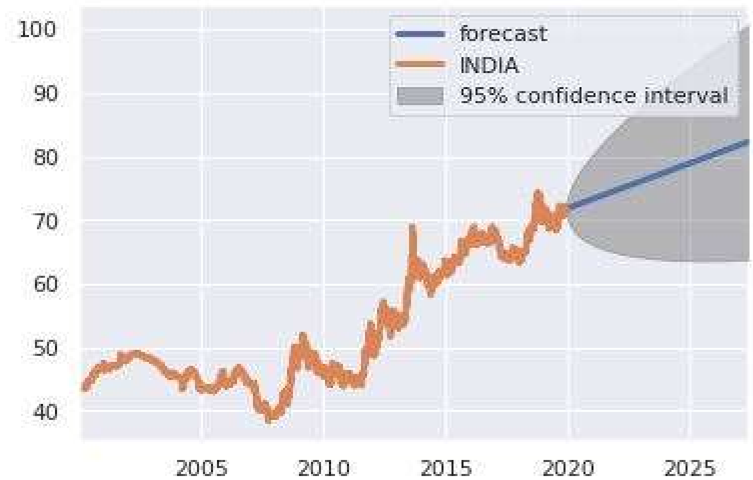
The first two lags are the one vastly above the significance level and so p = 2.

Model:	ARIMA	BIC:	-4380.9950
Dependent Variable:	D.INDIA	Log-Likelihood:	2212.7
Date:	2020-05-01 10:16	Scale:	1.0000
No. Observations:	7271	Method:	css-mle
Df Model:	4	Sample:	01-04-2000
Df Residuals:	7267		11-30-2019
Converged:	1.0000	S.D. of innovations:	0.178
No. Iterations:	6.0000	HQIC:	-4403.602
AIC:	-4415.4532		

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	0.0039	0.0021	1.8322	0.0670	-0.0003	0.0080
ar.L1.D.INDIA	0.2535	0.2328	1.0890	0.2762	-0.2027	0.7097
ar.L2.D.INDIA	-0.0474	0.0158	-2.9950	0.0028	-0.0784	-0.0164
ma.L1.D.INDIA	-0.1984	0.2329	-0.8519	0.3943	-0.6549	0.2581

	Real	Imaginary	Modulus	Frequency
AR.1	2.6731	-3.7342	4.5924	-0.1511
AR.2	2.6731	3.7342	4.5924	0.1511
MA.1	5.0400	0.0000	5.0400	0.0000

Akaike information criterion (AIC) estimates the relative amount of information lost by a given model. The less the better.



So, we forecasted the future values. The forecast values are not that accurate but its gives us a trends and general direction of the stock. It works best when your data exhibits a stable or consistent pattern over time with minimum outliers. Here we forecasted for a few year hence the prediction is just a straight line but we get range in which forecast will fluctuate with 95% confidence

Auto ARIMA:

In simple terms, Auto ARIMA tries all the possible parameters combinations and provides with the best parameters solution.

Here, the best parameters that it suggest is SARIMAX function with order(0,1,2)x(0,0,1,12)

```
Performing stepwise search to minimize aic
Fit ARIMA: (2, 1, 2)x(1, 0, 1, 12) (constant=True); AIC=-4128.959, BIC=-4074.634, Time=40.632 seconds
Fit ARIMA: (0, 1, 0)x(0, 0, 0, 12) (constant=True); AIC=-4110.799, BIC=-4097.218, Time=0.477 seconds
Fit ARIMA: (1, 1, 0)x(1, 0, 0, 12) (constant=True); AIC=-4126.666, BIC=-4099.504, Time=5.174 seconds
Fit ARIMA: (0, 1, 1)x(0, 0, 1, 12) (constant=True); AIC=-4127.657, BIC=-4100.495, Time=6.737 seconds
Fit ARIMA: (0, 1, 0)x(0, 0, 0, 12) (constant=False); AIC=-4110.824, BIC=-4104.034, Time=0.190 seconds
Fit ARIMA: (2, 1, 2)x(0, 0, 1, 12) (constant=True); AIC=-4130.177, BIC=-4082.643, Time=31.942 seconds
Fit ARIMA: (2, 1, 2)x(0, 0, 0, 12) (constant=True); AIC=-4124.122, BIC=-4083.378, Time=9.688 seconds
Fit ARIMA: (2, 1, 2)x(0, 0, 2, 12) (constant=True); AIC=-4128.274, BIC=-4073.950, Time=96.127 seconds
Fit ARIMA: (2, 1, 2)x(1, 0, 0, 12) (constant=True); AIC=-4130.098, BIC=-4082.564, Time=43.082 seconds
Fit ARIMA: (2, 1, 2)x(1, 0, 2, 12) (constant=True); AIC=-4126.819, BIC=-4065.704, Time=89.514 seconds
Fit ARIMA: (1, 1, 2)x(0, 0, 1, 12) (constant=True); AIC=-4132.136, BIC=-4091.393, Time=9.926 seconds
Fit ARIMA: (1, 1, 2)x(0, 0, 0, 12) (constant=True); AIC=-4126.081, BIC=-4092.128, Time=1.806 seconds
Fit ARIMA: (1, 1, 2)x(1, 0, 1, 12) (constant=True); AIC=-4130.949, BIC=-4083.415, Time=34.395 seconds
Fit ARIMA: (1, 1, 2)x(0, 0, 2, 12) (constant=True); AIC=-4130.308, BIC=-4082.774, Time=26.308 seconds
Fit ARIMA: (1, 1, 2)x(1, 0, 0, 12) (constant=True); AIC=-4132.056, BIC=-4091.312, Time=9.387 seconds
Fit ARIMA: (1, 1, 2)x(1, 0, 2, 12) (constant=True); AIC=-4128.422, BIC=-4074.097, Time=26.430 seconds
Fit ARIMA: (0, 1, 2)x(0, 0, 1, 12) (constant=True); AIC=-4133.901, BIC=-4099.948, Time=6.560 seconds
Fit ARIMA: (0, 1, 2)x(0, 0, 0, 12) (constant=True); AIC=-4127.859, BIC=-4100.697, Time=1.665 seconds
Fit ARIMA: (0, 1, 2)x(1, 0, 1, 12) (constant=True); AIC=-4132.761, BIC=-4092.017, Time=20.495 seconds
Fit ARIMA: (0, 1, 2)x(0, 0, 2, 12) (constant=True); AIC=-4132.094, BIC=-4091.351, Time=23.388 seconds
Fit ARIMA: (0, 1, 2)x(1, 0, 0, 12) (constant=True); AIC=-4133.815, BIC=-4099.863, Time=5.739 seconds
Fit ARIMA: (0, 1, 2)x(1, 0, 2, 12) (constant=True); AIC=-4130.214, BIC=-4082.680, Time=73.094 seconds
Fit ARIMA: (0, 1, 3)x(0, 0, 1, 12) (constant=True); AIC=-4132.088, BIC=-4091.345, Time=9.349 seconds
Fit ARIMA: (1, 1, 1)x(0, 0, 1, 12) (constant=True); AIC=-4131.340, BIC=-4097.387, Time=11.998 seconds
Fit ARIMA: (1, 1, 3)x(0, 0, 1, 12) (constant=True); AIC=-4130.639, BIC=-4083.105, Time=26.758 seconds
Total fit time: 610.921 seconds
```

SARIMAX:

Statespace Model Results

Dep. Variable:	y	**No. Observations:**	6573
Model:	SARIMAX(0, 1, 2)x(0, 0, 1, 12)	**Log Likelihood**	2071.950
Date:	Fri, 01 May 2020	**AIC**	-4133.901
Time:	08:33:31	**BIC**	-4099.948
Sample:	0 - 6573	**HQIC**	-4122.164
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0031	0.002	1.295	0.195	-0.002	0.008
ma.L1	0.0436	0.004	10.081	0.000	0.035	0.052
ma.L2	-0.0358	0.006	-5.596	0.000	-0.048	-0.023
ma.S.L12	0.0351	0.007	4.989	0.000	0.021	0.049
sigma2	0.0312	0.000	196.221	0.000	0.031	0.031

Ljung-Box (Q):	97.28	**Jarque-Bera (JB):**	157999.08
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	9.17	**Skew:**	-0.28
Prob(H) (two-sided):	0.00	**Kurtosis:**	27.01

By looking at the above table we see that AIC is less than ARIMA model, which is better.

```
get_mape(unidf.INDIA["20
```

```
0.76
```

```
print(np.sqrt(mean_square
```

```
0.6025859491776253
```

So, mean absolute error is 0.76 which is more than what we got for Exponential Smoothing and also the RMSE is 0.60258595.

Modelling:

LSTM:

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.

We need to convert the data into a proper time steps data from which a ML model can learn something (A pattern or seasonality).

After, fitting the model to the train data set with epochs of 20.

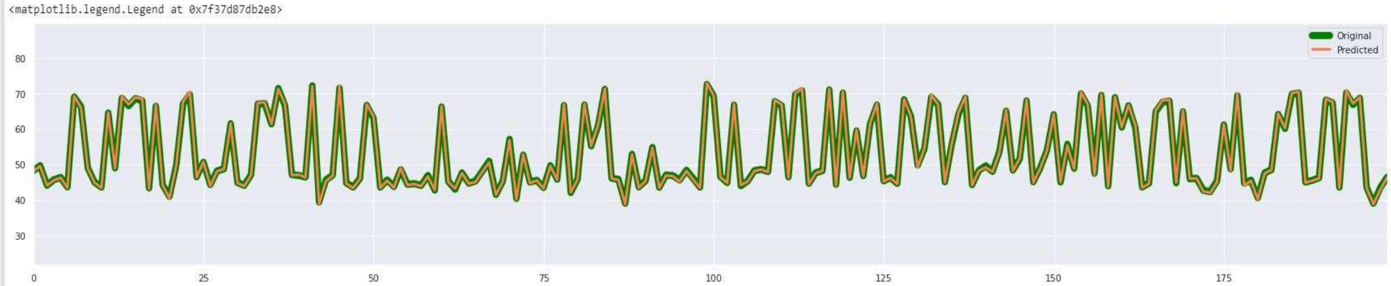
```
model = Sequential()
model.add(LSTM(7, activation='relu', input_shape=(window, num_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
history = model.fit(X_train, y_train, epochs=20, verbose=1)
```

```
Epoch 1/20
5110/5110 [==============================] - 1s 151us/step - loss: 2834.0413
Epoch 2/20
5110/5110 [==============================] - 0s 52us/step - loss: 161.5631
Epoch 3/20
5110/5110 [==============================] - 0s 51us/step - loss: 0.1313
Epoch 4/20
5110/5110 [==============================] - 0s 52us/step - loss: 0.1163
Epoch 5/20
5110/5110 [==============================] - 0s 51us/step - loss: 0.1160
Epoch 6/20
5110/5110 [==============================] - 0s 57us/step - loss: 0.1152
Epoch 7/20
5110/5110 [==============================] - 0s 59us/step - loss: 0.1143
Epoch 8/20
5110/5110 [==============================] - 0s 59us/step - loss: 0.1133
Epoch 9/20
5110/5110 [==============================] - 0s 51us/step - loss: 0.1122
Epoch 10/20
5110/5110 [==============================] - 0s 49us/step - loss: 0.1111
Epoch 11/20
5110/5110 [==============================] - 0s 52us/step - loss: 0.1098
Epoch 12/20
5110/5110 [==============================] - 0s 49us/step - loss: 0.1084
Epoch 13/20
5110/5110 [==============================] - 0s 50us/step - loss: 0.1071
Epoch 14/20
5110/5110 [==============================] - 0s 51us/step - loss: 0.1053
Epoch 15/20
5110/5110 [==============================] - 0s 49us/step - loss: 0.1035
Epoch 16/20
5110/5110 [==============================] - 0s 51us/step - loss: 0.1016
Epoch 17/20
5110/5110 [==============================] - 0s 52us/step - loss: 0.0998
Epoch 18/20
5110/5110 [==============================] - 0s 51us/step - loss: 0.0974
Epoch 19/20
5110/5110 [==============================] - 0s 49us/step - loss: 0.0954
Epoch 20/20
5110/5110 [==============================] - 0s 51us/step - loss: 0.0927
```

Lets, see how well the model works.

```
plt.margins(x=0, y=0.5)
plt.legend(["Original", "Predicted"])
```

```
<matplotlib.legend.Legend at 0x7f37d87db2e8>
```



Well its working pretty great in univariate itself in just 20 epochs. Let's check the errors

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
print(mean_absolute_error(y_test, yPred))
print(mean_squared_error(y_test, yPred))
print(np.sqrt(mean_squared_error(y_test, yPred)))
```

```
0.2297931843422424
0.0859877197781988
0.2932366276204233
```

Well, its works pretty great in univariate itself in just 20 epochs.

References:

1. https://www.babypips.com/learn/forex/simple-moving-averages
2. https://www.kaggle.com/voltvipin/indian-foreign-exchange-prediction-using-lstm
3. https://towardsdatascience.com/arima-forecasting-in-python-90d36c2246d3
4. https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/