Indian Trade Data Analysis and Forecasting

*By*

Akarsh Somani (162) & Gaurav Misra (172)



*Bachelor Thesis submitted to*

Indian Institute of Information Technology Kalyani

*For the partial fulfillment of the degree of*

**Bachelor of Technology**

**in**

**Computer Science and Engineering**

**June, 2020**

Certificate

This is to certify that the thesis entitled “Indian Trade Data Analysis and Forecasting” being submitted by Akarsh Somani and Gaurav Misra, undergraduate students, Reg. No 162 and 172, Roll No. 39/CSE/16005 and 39/CSE/16015, respectively, in the Department of Computer Science and Engineering, Indian Institute of Information Technology Kalyani, West Bengal 741235, India, for the award of Bachelors of Technology in Computer Science and Engineering is an original research work carried by them under my supervision and guidance. The thesis has fulfilled all the requirements as per the regulations of Indian Institute of Information Technology Kalyani and in my opinion, has reached the standards needed for submission. The work, techniques and the results presented have not been submitted to any other University or Institute for the award of any other degree or diploma.

Dalia Nandi

Assistant Professor

Electronics and Communication Engineering Department

Indian Institute of Information Technology Kalyani

Declaration

I hereby declare that the work being presented in this thesis entitled, “Indian Trade Data Analysis and Forecasting”, submitted to Indian Institute of Information Technology Kalyani in partial fulfillment for the award of the degree of **Bachelor of Technology** in Computer Science and Engineering during the period from January, 2020 to May, 2020 under the supervision of Dr. Dalia Nandi, Department of Electronics and Communication Engineering, Indian Institute of Information Technology Kalyani, West Bengal 741235, India, does not contain any classified information.

Akarsh Somani and Gaurav Misra

Reg No/Roll No: 39/CSE/16005/162 and 39/CSE/16015/172

Department: Computer Science and Engineering

Institute Name: Indian Institute of Information Technology Kalyani

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

..................................

Dr. Dalia Nandi

Position: Assistant Professor

Departmental address: Electronics and Communication Engineering

Place: Kalyani

Date: May 1, 2020

Acknowledgments

Firstly, we would like to thank our supervisor Dr. Dalia Nandi for her support and guidance to complete this project work without whom we would have not been able to make out this far. We would also like to thank our friends who supported us greatly and were always willing to help us. We are very grateful to Department of Computer Science and Engineering, Indian Institute of Information Technology Kalyani, West Bengal - 741235, India, for providing us this wonderful opportunity. Lastly, we would like to thank our Parents a God for their never-ending grace.

Akarsh Somani and Gaurav Misra

Reg No/Roll No: 39/CSE/16005/162 and 39/CSE/16015/172

Department: Computer Science and Engineering

Institute Name: Indian Institute of Information Technology Kalyani

Place: Kalyani

Date: June 16, 2020

Abstract

Trade is an economic concept which involves Buying and Selling of the commodities, or exchanging goods and services between needy people. Trade is important in a way that it increases competition and decreases overall world wise cost of a product.

In this thesis, we will be dealing with the **Indian trade related Data Analysis and Forecasting** with respect to other countries and try to interpret the impact of the trade. Trade is one of the important factors for the economy of any country and that underlines the purpose of this thesis.

India is only second largest market place (after China) in terms of human resource. Hence many foreign countries exploit this fact, especially China, to sell their products. Hence our Import Trade matters a lot from economic point of view.

On the other hand, we, being second largest population, also produce a vast range of products, all thanks to diversity present in India, while in North India, we have huge production of almost all types of grains and also fruits; in South India, we produce every kind of spices; in East India, we have a huge amount of tea-production. All of this is just a chunk of what we Export to other countries.

So to balance our economy with feasible amount of Import and Export trade, we need to analyze which area of production needs more attention for Exports and which product needs to be encouraged for production to decrease the amount of Imports. So our Data Analysis part points at this area of our thesis.

Import and Export both are very dynamic in nature since every year we evolve in what we want as an Import and what we produce in order to Export. If we can forecast the trade amount we might know the areas, which are putting our economy into deficit and also reinforce those areas where we’re improving in terms of trade profit.

Our Motivations for this thesis are listed as following -

1. Which HS Code involves the most import/export?
2. From which country we import/export the most?
3. Forecasting the Indian import/export data to predict what should be our country’s next move?
4. How our trades are evolving across the years?
5. Also analyzing impact of few trade related decisions taken by Indian govt. recently (Case Study)?

Contents

1. [Indian Trade Forecasting 1](#_Toc42376331)

[1.1 Significance of Forecasting 1](#_Toc42376332)

[1.2 Data 1](#_Toc42376333)

[1.3 Preprocessing of Dataset 3](#_Toc42376334)

[1.4 Visualization 4](#_Toc42376335)

[1.5 Insights about Data 4](#_Toc42376336)

2. [Overview of Forecasting Models 7](#_Toc42376337)

[2.1 Which one to use 8](#_Toc42376338)

[2.2 Types of Timeseries 8](#_Toc42376339)

3. [Timeseries Forecasting Models 10](#_Toc42376340)

[3.1 Exponential Smoothing(Exponential Averaging) 10](#_Toc42376341)

[3.2 Auto Regressive Model 11](#_Toc42376342)

[3.3 Moving Average Model 13](#_Toc42376343)

[3.4 Holt-Winters Method 15](#_Toc42376344)

[3.5 ARIMA Multiplicative 16](#_Toc42376345)

[3.6 ARIMA Additive 18](#_Toc42376346)

[3.7 Seasonal ARIMA 20](#_Toc42376347)

[3.8 RNN (Recurrent Neural Networks) 21](#_Toc42376348)

[3.8.1 LSTM Networks 22](#_Toc42376349)

4. [Comparation 27](#_Toc42376350)

5. [Conclusion 28](#_Toc42376351)

[5.1 Technologies Used 28](#_Toc42376352)

[5.2 Future Prospects 28](#_Toc42376353)

[Bibliography 29](#_Toc42376354)

**Chapter 1**

# Indian Trade Forecasting

## Significance of Forecasting

What if we can predict the next month’s total import or export amount? The question depends on how much accuracy can be achieved while forecasting future data points. Assuming we get fairly well accuracy, we can surely say that there’s a lot of use cases to get returns with forecasting total trade amount.

We purchase things regularly out of our need. Sometimes more and sometimes less, but if we know that next month’s import is going to be lower, we can say that prices are also going to get cheaper. The reason behind lowering of price is that either demand is low or we are producing that product on a larger scale in our country, hence the decrement in import.

The most specific use-case appear to be for industries, since they buy very precious machineries for functionality and production purpose, hence if they can somehow know that imports are going to decrease and hence prices of machines can be cheaper they can plan to buy machines earlier than usual and same for delaying the purchase in case of increment in imports and decrement in exports.

Government can plan to handle demands of next month using forecasted imported value similarly for exports as well. Government can also control prices based on whether imports and exports are going to go up or down.

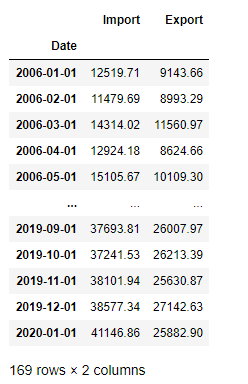
## Data

Dataset has been scraped from Department of Commerce, Govt. of India. Data is available from January, 2006 to January, 2020. We have total trade amount (Import/Export) for each month which lies in expressed in million US dollars.

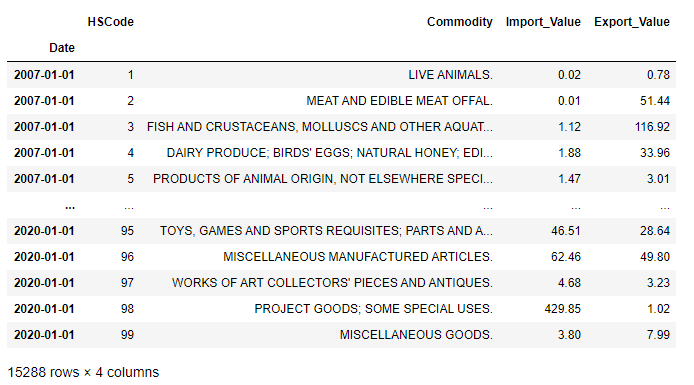
HS Code: - Harmonized System (HS) of tariff nomenclature is an internationally standardized system of names and numbers to classify traded products.

**Sample Data: -**

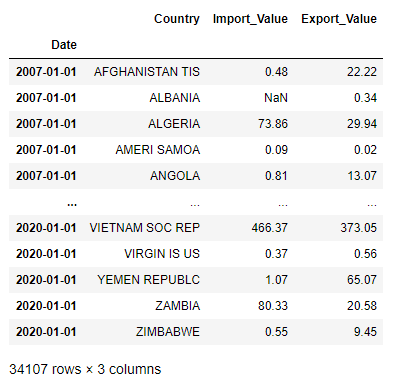
Monthly Import Export Data Total: -



Monthly Import Export Data HSCode Wise: -



Monthly Import Export Data Country Wise: -



## Preprocessing of Dataset

We have total 169 data points for both kinds of trade Import and Export. Now we break the dataset into two sets – training and validation such that training set includes 156 data points and validation set includes 13 data points to forecast upon.

We have no missing values hence we don’t need to impute any data point. We have all numerical continuous columns hence we don’t need to one hot encode as well.

## Visualization



## Insights about Data

We analyzed the countries and products which India trades the most. China tops the list with around $4379 million Import followed by United Arab Emirates, Saudi Arab and USA. While USA is top most country to which India exports the most $3013 million followed by UAE, China, Hong Kong and Singapore. As we know that Trade is the key to normalize costs across the world. India is a developing country with rich resources and population thus attracts the business from all over the world.



Products corresponding to HS code 27 are the highest contributor for Imports made by India, which includes Minerals Fuels and Mineral Oils specially. India is a country of 130 Crore people, it explains this consumption. For every kind of fuel we’re dependent on other countries especially Arab countries, from where we get fuels like Petrol, Diesel, Kerosene etc. If we are to get rid of this vulnerability, Indian government should start investing in identification of new fuel resources and encourage the citizens to use it e.g. electric vehicles, solar panels, windmill etc.

It’s quite surprising to see the HS code 27 on top of list in Exports too. The major reason for that is exchange deals with various countries especially Singapore, to whom India does most export of Mineral and Crude Oils.

Also being one the biggest population who likes to wear jewelries, the Import and Export is second most for HS code 71.



The Description of the mentioned HS codes in the graph is as follows



**Chapter 2**

# Overview of Forecasting Models

* **Exponential Smoothing** (Sandip Roy, Sankar Prasad Biswas, Subhajyoti Mahata, Rajesh Bose 2018)**:** It is the technique for smoothing(Averaging) the timeseries using exponential window function. In simple moving average the past observation are equally weighted where as here it exponentially decreases over the time.
* **Auto Regressive Model (AR)** (Nimisha Tomar, Durga Patel, Akshat Jain 2020)**:** It is used when a value from the time series has dependency on previous values e.g. Xt = f(Xt-1). The order of an auto-regression is the number of immediately preceding values in the series that are used to forecast the current value e.g. order of 2 denotes that the value Xt is dependent on Xt-1 and Xt-2.
* **Moving Average Model** (S. Vemuri, R. Balasubramanian, E.F. Hill 1974)**:** In time series analysis, the Moving-Average model (MA model) specifies that the output variable depends linearly on the current and various past values of a stochastic (imperfectly predictable) term.
* **Holt-Winters Model** (Chatfield 1978)**:** It is an extension over the simple exponential smoothing method. Here we use triple smoothing with the factor - seasonal period, trend type and seasonal type. Here seasonal and trend type means *Multiplicative* or *Additive*.
* **ARIMA Model** (Lyashenko 2020)**:** Auto Regressive Integrated Moving Average (ARIMA) models are applied in some cases where data show evidence of non-stationarity, where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity. We’ll discuss Additive ARIMA, Multiplicative ARIMA and Seasonal ARIMA.
* **CNN Model:** In deep learning, a Convolutional Neural Network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery.
* **RNN Model** (Yi-Ting Tsai, Yu-Ren Zeng, Yue-Shan Chang 2018)**:** A Recurrent Neural Network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior.
* **LSTM Model:** Long Short-Term Memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections.

## 2.1 Which one to use

* Exponential/Moving Average models may perform moderately in short term forecasting but are not reliable since there’s no weightage of past values taken.
* Autoregressive models forecast only on past information; they implicitly assume that the fundamental forces, that influenced the past prices, will not change over time. This can lead to surprising and inaccurate predictions.
* ARIMA method is appropriate only for a time series that is stationary with signs of trend and seasonality, whereas our data contains some randomness.
* CNNs are inappropriate for sequential data as they don’t use past values to make predictions.
* RNNs are used for processing sequential data but perform badly in case of long-term dependency, this happens because of vanishing gradient problem.
* LSTMs store past information and know what to forget and what not to forget as a modification over RNNs.

**Now we’ll try each model one by one and See which one provides us with the best forecasting results.**

## 2.2 Types of Timeseries

There are basic two kind of time-series’ – Multiplicative and Additive. Time series data with varying amplitude is called Multiplicative otherwise Additive.

**Multiplicative Time Series = Trend \* Seasonality \* Randomness**



**Additive Time Series = Trend + Seasonality + Randomness**



**Chapter 3**

# Timeseries Forecasting Models

## 3.1 Exponential Smoothing(Exponential Averaging)

It is the technique for smoothing(Averaging) the timeseries using exponential window function. In simple moving average the past observation are equally weighted where as here it exponentially decreases over the time. In this method we use



Where S0 is the value at time t = 0 and the forecast at time t is given as St

The term smoothing factor applied to α here is something of a misnomer, as larger values of α actually reduce the level of smoothing, and in the limiting case with α = 1 the output series is just the current observation.

As mentioned, it uses the exponential window function; we substitute the value of the above equation back to itself.



In other words, it forms a Geometric Progression (GP) which is a discrete version of an exponential function.

**Code Snippet**:

from statsmodels.tsa.holtwinters import SimpleExpSmoothing

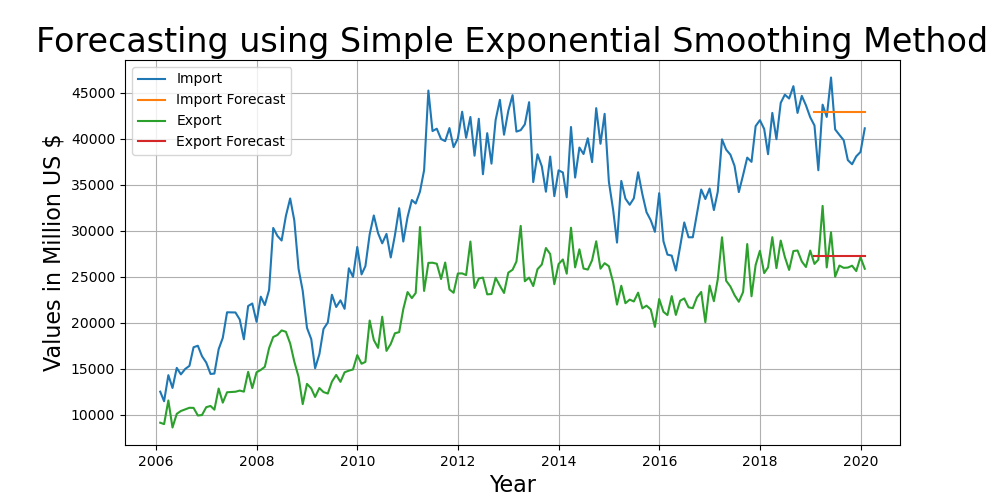
model\_import = SimpleExpSmoothing(df\_train['Import']).fit(smoothing\_level=0.6,optimized=False)

yhat\_import = model\_import.forecast(test\_count)

model\_export = SimpleExpSmoothing(df\_train['Export']).fit(smoothing\_level=0.6,optimized=False)

yhat\_export = model\_export.forecast(test\_count)

**Plot**:



**Result**:

RMSE Import/Export - 3771.36, 2042.23

## 3.2 Auto Regressive Model

It is used when a value from the time series has dependency on previous values e.g. Xt = f(Xt-1). The order of an auto-regression is the number of immediately preceding values in the series that are used to forecast the current value e.g. order of 2 denotes that the value Xt is dependent on Xt-1 and Xt-2.

Terminologies - **Autocorrelation Function** (**ACF**) and **Partial Autocorrelation Function** (**PACF**).

ACF = CORR (Yt , Yt-k) ,

PACF can be calculated as

* Remove the linear dependency from the timeseries
* Calculate the correlation.

PACF is used to find the order of the autoregressive model.

**Code Snippet**:

from statsmodels.tsa.ar\_model import AR

from random import random

# Import

# fit model

model = AR(df\_train['Import'])

model\_fit = model.fit()

# make prediction

yhat\_import = model\_fit.predict(len(df\_train), len(df\_train)+test\_count-1)

# Export

# fit model

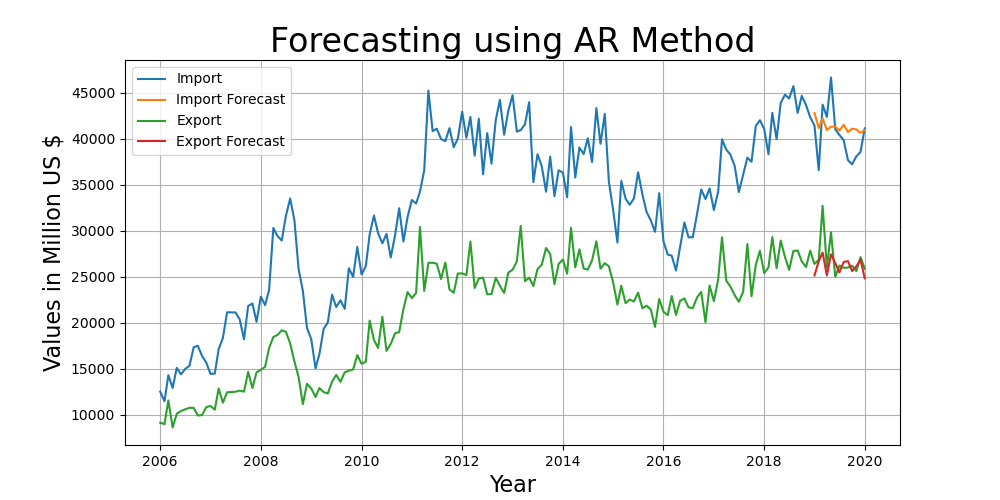
model2 = AR(df\_train['Export'])

model2\_fit = model2.fit()

# make prediction

yhat\_export = model2\_fit.predict(len(df\_train), len(df\_train)+test\_count-1)

**Plot**:



**Result**:

RMSE Import/Export - 2718.07, 1738.07

## 3.3 Moving Average Model

Moving Average is a technique that calculates the overall trend in the dataset. As the name suggest that we go by taking the Average over a fixed rolling size window. The MAt is calculated by taking the unweighted mean of the previous window\_size (here 4) data.

The moving average is thus calculated as –



MA4 = X4 + X3 + X2 + X1

And the successive value can be calculated as –

MAn = MAn-1 + (Xn – Xn-window\_size)

In our model we have chosen the window of 12

**Code Snippet**:

from statsmodels.tsa.arima\_model import ARMA

from random import random

# Import

# fit model import

model = ARMA(df\_train['Import'], order=(0, 10))

model\_fit = model.fit(disp=False)

# make prediction

yhat\_import = model\_fit.predict(len(df\_train), len(df\_train)+ test\_size-1)

# Export

# fit model export

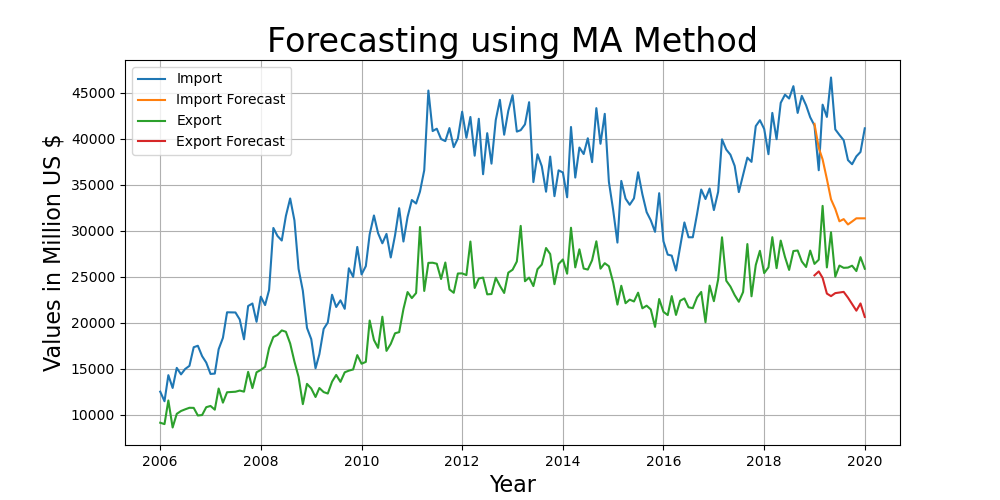
model2 = ARMA(df\_train['Export'], order=(0, 12))

model2\_fit = model2.fit(disp=False)

# make prediction

yhat\_export = model2\_fit.predict(len(df\_train), len(df\_train)+ test\_size-1)

**Plot**:



**Result**:

RMSE Import/Export - 7743.73, 4284.66

## 3.4 Holt-Winters Method

It is the extension over the simple exponential smoothing method. Here we use triple smoothing with the factor - seasonal period, trend type and seasonal type. Here seasonal and trend type means *Multiplicative* or *Additive*.

st represents the smoothed value of the constant part for time t. bt represents the sequence of best estimates of the linear trend that are superimposed on the seasonal changes. ct is the sequence of seasonal correction factors. ct is the expected proportion of the predicted trend at any time t mod L in the cycle that the observations take on. As a rule of thumb, a minimum of two full seasons (or 2L periods) of historical data is needed to initialize a set of seasonal factors.

The output of the algorithm is again written as Ft+m, an estimate of the value of x at time t+m, m>0 based on the raw data up to time t. Triple exponential smoothing with multiplicative seasonality is given by the formulas{\displaystyle {\begin{aligned}s\_{0}&=x\_{0}\\s\_{t}&=\alpha {\frac {x\_{t}}{c\_{t-L}}}+(1-\alpha )(s\_{t-1}+b\_{t-1})\\b\_{t}&=\beta (s\_{t}-s\_{t-1})+(1-\beta )b\_{t-1}\\c\_{t}&=\gamma {\frac {x\_{t}}{s\_{t}}}+(1-\gamma )c\_{t-L}\\F\_{t+m}&=(s\_{t}+mb\_{t})c\_{t-L+1+(m-1)\mod L},\end{aligned}}}



where α is the data smoothing factor, 0 < α < 1, β is the trend smoothing factor, 0 < β < 1, and γ is the seasonal change smoothing factor, 0 < γ < 1. [Reference](https://en.wikipedia.org/wiki/Exponential_smoothing)

In our model we used seasonal period = 12 and trend type and seasonal type as multiplicative.

**Code Snippet**:

from statsmodels.tsa.holtwinters import ExponentialSmoothing

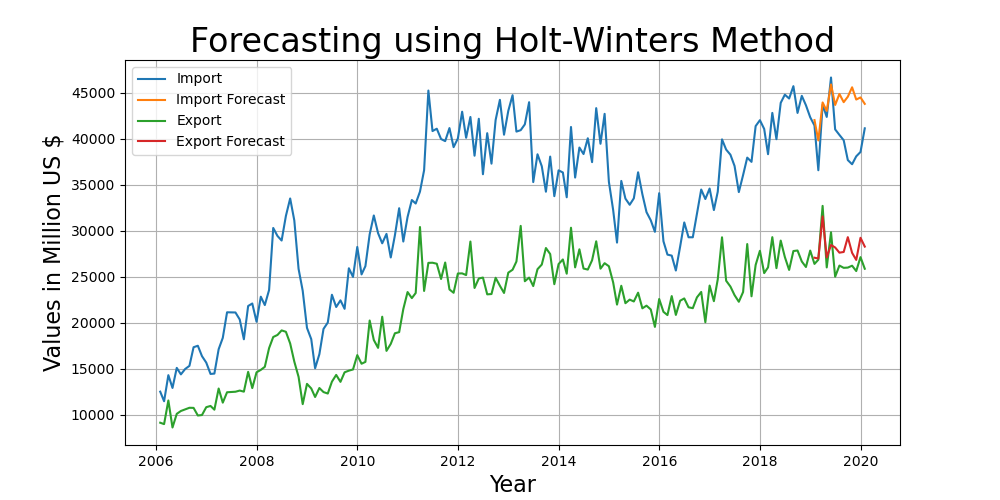
model\_import = ExponentialSmoothing(df\_train['Import'] ,seasonal\_periods=12 ,trend='mul', seasonal='mul',).fit()

yhat\_import = model\_import.forecast(test\_size)

model\_export = ExponentialSmoothing(df\_train['Export'] ,seasonal\_periods=12 ,trend='mul', seasonal='mul',).fit()

yhat\_export = model\_export.forecast(test\_size)

**Plot**:



**Result**:

RMSE Import/Export - 4417.84, 1850.49

## 

## 3.5 ARIMA Multiplicative

ARIMA stands for *Autoregressive Integrated Moving Average* which is a combination of three terms –

The AR part of ARIMA indicates the evolving variable of interest is regressed on its own lagged (i.e., prior) values. The MA part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past. The I (for "integrated") indicate that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible.

Non-seasonal ARIMA models are generally denoted ARIMA(p,d,q) where parameters p, d, and q are non-negative integers, p is the order (number of time lags) of the autoregressive model, d is the degree of differencing (the number of times the data have had past values subtracted), and q is the order of the moving-average model.

We have already seen the Autoregressive and Moving average individually.

Multiplicative Time Series = Trend \* Seasonality \* Randomness, As explained previously.

For predicting the p and q we use partial autocorrelation and autocorrelation function. We used plots to find the optimal value of p and q graphically.

Here we are taking the (p,d,q) = (1,0,5). d as 0 because we do not require the differencing as our randomness is stationary.

**Code Snippet**:

Import:

from statsmodels.tsa.arima\_model import ARIMA

# fit model

model = ARIMA(df['Randomness\_Import'][:-test\_size], order=(1, 0, 5))

model\_fit = model.fit(disp=False)

# make prediction

yhat = model\_fit.predict(len(df[:-test\_size]), len(df[:-test\_size]) + test\_size-1)

df['ARMA\_import\_forecast'] = yhat

df['ARMA\_import\_forecast'] = df['ARMA\_import\_forecast'] + df['Seasonal\_Import'] + df['MA\_Import']

Export:

from statsmodels.tsa.arima\_model import ARIMA

# fit model

model = ARIMA(df['Randomness\_Export'][:-test\_size], order=(1, 0, 5))

model\_fit = model.fit(disp=False)

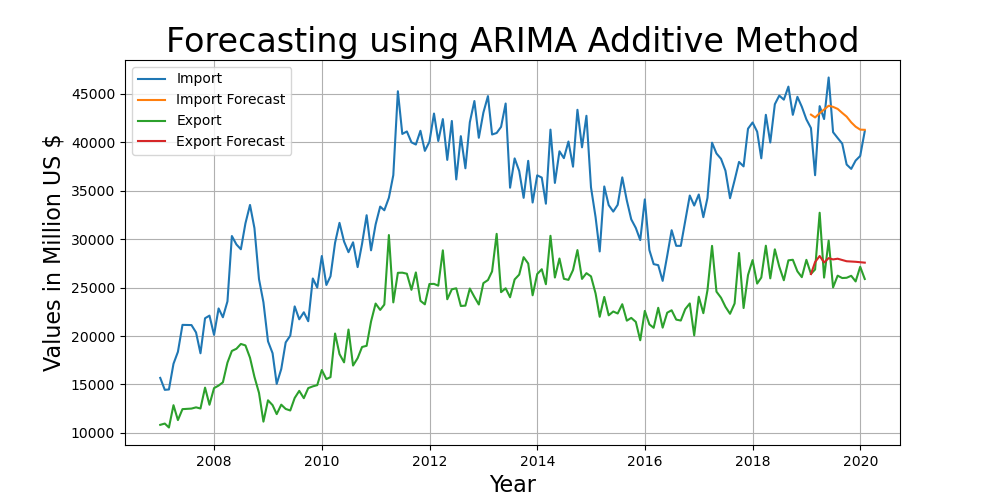
# make prediction

yhat = model\_fit.predict(len(df[:-test\_size]), len(df[:-test\_size]) + test\_size-1)

df['ARMA\_export\_forecast'] = yhat

df['ARMA\_export\_forecast'] = df['ARMA\_export\_forecast'] + df['Seasonal\_Export'] + df['MA\_Export']

**Plot**:



**Result**:

RMSE Import/Export - 3286.15, 2026.4

## 3.6 ARIMA Additive

This is similar to the above method except for the fact that we have considered our time series as Additive (just for comparison).

Additive Time Series = Trend + Seasonality + Randomness

Here we are taking the (p,d,q) = (1,0,5). d as 0 because we do not require the differencing as our randomness is stationary.

we can see that the forecast does not fits right.

**Code Snippet**:

Import:

from statsmodels.tsa.arima\_model import ARIMA

# fit model

model = ARIMA(df['Randomness\_Import'][:-test\_size], order=(2, 0, 5))

model\_fit = model.fit(disp=False)

# make prediction

yhat = model\_fit.predict(len(df[:-test\_size]), len(df[:-test\_size]) + test\_size-1)

df['ARMA\_import\_forecast'] = yhat

df['ARMA\_import\_forecast'] = df['ARMA\_import\_forecast'] \* df['Seasonal\_Import'] \* df['MA\_Import']

Export:

from statsmodels.tsa.arima\_model import ARIMA

# fit model

model = ARIMA(df['Randomness\_Export'][:-test\_size], order=(1, 0, 5))

model\_fit = model.fit(disp=False)

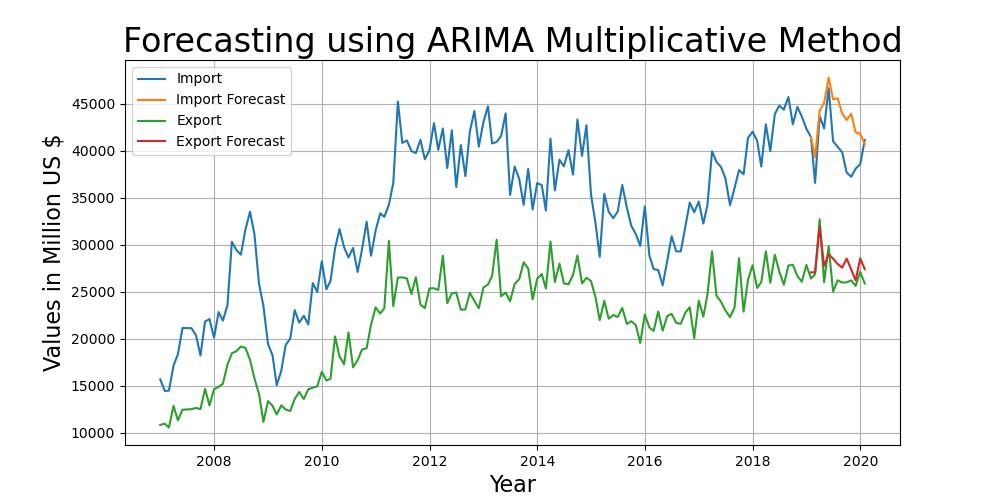
# make prediction

yhat = model\_fit.predict(len(df[:-test\_size]), len(df[:-test\_size]) + test\_size-1)

df['ARMA\_export\_forecast'] = yhat

df['ARMA\_export\_forecast'] = df['ARMA\_export\_forecast'] \* df['Seasonal\_Export'] \* df['MA\_Export']

**Plot**:



**Result**:

RMSE Import/Export - 3739.46, 1646.06

## 3.7 Seasonal ARIMA

Seasonal ARIMA here is ARIMA Multiplicative method with a seasonality factor m. Here we have chosen the multiplicative factor of 12 as ours is a monthly data.

Also we used brute force to find the best p,d,q values which reduces the RMSE. Import (1,0,0) and Export (1,0,2). This model has performed quite well and thus below is the plot.

**Code Snippet**:

import statsmodels.api as sm

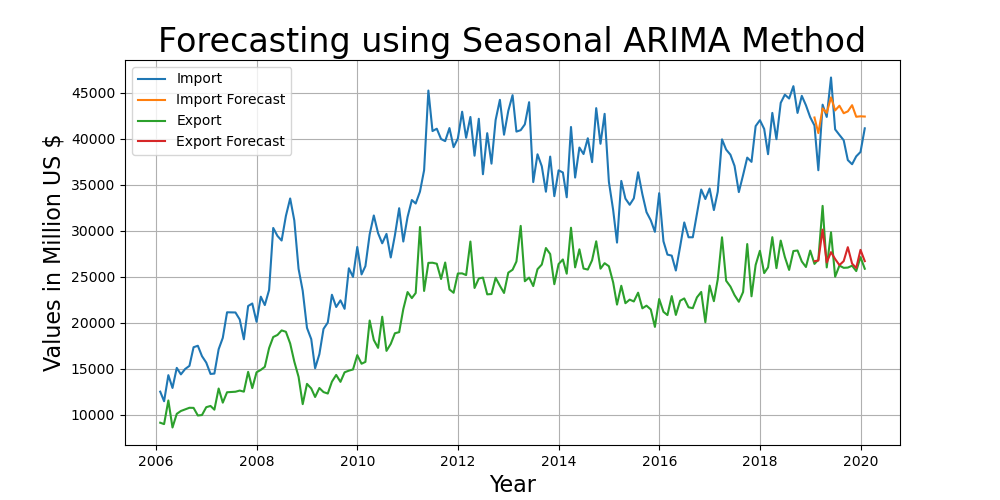
model\_import = sm.tsa.statespace.SARIMAX(df\_train['Import'], order=(1, 0, 0),seasonal\_order=(0,1,1,12), trend='n').fit()

yhat\_import = model\_import.forecast(test\_size)

model\_export = sm.tsa.statespace.SARIMAX(df\_train['Export'], order=(1, 0, 2),seasonal\_order=(0,1,1,12), trend='n').fit()

yhat\_export = model\_export.forecast(test\_size)

**Plot**:



**Result**:

RMSE Import/Export - 3391.32, 1310.58

## 3.8 RNN (Recurrent Neural Networks)

Humans don’t start their thinking from scratch every second. As we read this paragraph, we understand each word based on your understanding of previous words. We don’t throw everything away and start thinking from scratch again. Our thoughts have persistence.

Traditional neural networks can’t do this, and it seems like a major shortcoming.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



Sometimes, we only need to look at recent information to perform the present task. In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.

But there are also cases where we need more contexts. Consider trying to predict the last word in the text “I grew up in France… I speak fluent French.” It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

Thankfully, LSTMs Solves this problem!

### **3.8.1 LSTM Networks**

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

**Preparing Data for applying LSTM**

Scale data values between 0 and 1 for faster convergence using gradient descent. Prepare input for each of data point as set of previous 12 data point, e.g. for 13th data point a13, the input will be set (a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12) and the output will be a13. Store all these input sets in an input list and outputs in output list.

**LSTM Model Specification**

We have four layers model, which is arranged as following:

1. LSTM layer of 200 nodes with 0.2 dropout
2. LSTM layer of 200 nodes with 0.2 dropout
3. LSTM layer of 150 nodes with 0.2 dropout
4. Dense layer with ‘linear’ activation.

After defining our model, we compile it using mean-squared-error as loss function and Adam optimizer.

Now we fit our model for prepared input and output with batch size of 15 for 30 epochs.

Then we predict data values for next 15 months one by one, however we can do this in one go as well. These predicted values will be scaled in between 0 and 1 hence we need to inverse transform these values to get actual value for trade of that month.

**Code Snippet:**

#converting dataset into x\_train and y\_train

scaler = MinMaxScaler()

smoothing\_window\_size = 24

for di in range(0, 156, smoothing\_window\_size):

scaler.fit(train[di:di+smoothing\_window\_size, :])

train[di:di+smoothing\_window\_size, :] = scaler.transform(train[di:di+smoothing\_window\_size, :])

# Normalize test data

valid = scaler.transform(valid)

x\_train, y\_train = [], []

for i in range(no\_of\_sig\_days, len(train)):

x\_train.append(train[i-no\_of\_sig\_days: i, :])

y\_train.append(train[i, 0])

# building neural networks and adding layers

model = Sequential()

model.add(LSTM(200, input\_shape=(x\_train.shape[1], 1), return\_sequences=True))

model.add(LSTM(200, input\_shape=(x\_train.shape[1],1), return\_sequences=True))

model.add(LSTM(150, input\_shape=(x\_train.shape[1],1), return\_sequences=False))

model.add(Dense(1,activation='linear'))

# compile model

model.compile(loss='mean\_squared\_error', optimizer='Adam')

from keras.callbacks import EarlyStopping, ModelCheckpoint

early\_stop = EarlyStopping(monitor='val\_loss', patience=10, mode='min')

checkpoint = ModelCheckpoint(mode + '\_model\_best\_weight\_last\_13preds\_comp.h5', monitor='val\_loss', save\_best\_only=True, mode='min', period=1)

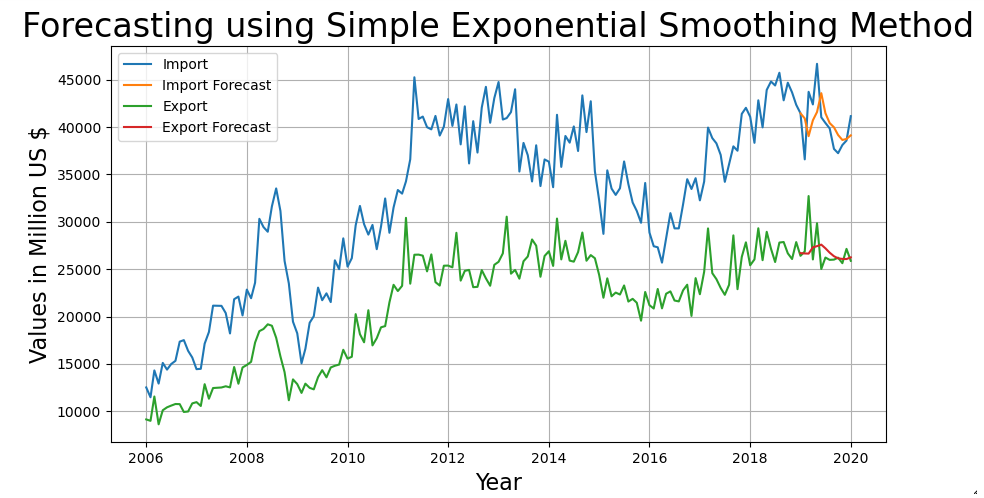
model.fit(x\_train, y\_train, epochs=30, batch\_size=4, callbacks=[early\_stop, checkpoint], verbose=1, validation\_data=(x\_test, y\_test)) # verbose=1 shows us the progress of epochs

# make predictions

pred = model.predict(x\_test, batch\_size=1)

pred = scaler.inverse\_transform(pred)

**Plot:**

****

**Result**:

RMSE Import/Export = 2637.71, 2043.39

**Chapter 4**

# Comparation

All the models perform quite well but the best observation which we have observed is of Seasonal ARIMA.

Below is the RMSE Comparisons –

|  |  |  |
| --- | --- | --- |
| **Model\Error** | **Import RMSE** | **Export RMSE** |
| **Exponential Smoothing** | 3771.36 | 2042.23 |
| **Auto Regressive (AR) model** | 2718.07 | 1738.07 |
| **Moving Average (MA) model** | 7743.73 | 4284.66 |
| **Holt-Winters Method** | 4417.84 | 1850.49 |
| **ARIMA Multiplicative** | 3739.46 | 1646.06 |
| **ARIMA Additive** | 3286.15 | 2026.4 |
| **Seasonal ARIMA** | 3391.32 | 1310.58 |
| **LSTM** | 2637.71 | 2043.39 |

The comparisons are so close because one model is the extension of the other with some advancement. Holt-winters is the advancement of the Simple exponential smoothing. ARIMA is the combination of AR and MA model, Seasonal ARIMA is the advancement of the ARIMA model.

* The mean of Error in forecasting is 3363.970323 and standard deviation is 2352.545462.
* To get more accurate forecasting we should fine tune our model’s parameter in a better way.
* We can also modify our model’s network structure.
* The brighter side is that our model is able to forecast right trend of trade import amount.
* As we have only 165 data points (150 training + 15 validations), it’s too much to ask for a very high accuracy.
* More data points will improve our model’s accuracy.

**Chapter 5**

# Conclusion

Throughout the thesis, we found answers of all of our motivational questions mentioned in **Abstract**. So we have all the insights and many forecasting models in our hands. And we have also forecasted the Near Future Trade value with our best models. The results are quite satisfactory and we see that there is a downward trend in Import forecast and upward trend in the Export forecast. The more we export, and the lesser we import, is beneficial for the Indian economy. This shows that India will be leading in terms of trade, if we modify our Foreign trade policy according to the case study.

**About Our Forecasting Models:** We used 8 time series forecasting models. Surprisingly Seasonal ARIMA outperformed LSTM quite easily, which we didn’t expect at the beginning of this project. We found out that ARIMA captures seasonality much better than any other model, even with **lesser data.** That’s what caused LSTM to underperform, for any Neural Network, we need a lot of data points to get good enough results, and this explains a lot. Now we hope that our successors will find this thesis useful and can apply our methods without trouble. We’re open-sourcing our codes used for forecasting and analysis and it’ll be available on GitHub.

**GitHub Link: https://github.com/akarshsomani/Indian-Import-Export-Data-Analysis**

## 5.1 Technologies Used

* Python
* Keras
* Pandas
* Numpy
* Matplotlib
* Seaborn

## 5.2 Future Prospects

1. Making Existing Models more robust
2. Using New models e.g. Attention Models.
3. Include some factor to encounter the randomness in data
4. Getting more data points for better forecasting
5. Using this thesis for case studies

# Bibliography

Lyashenko, Oksana. "ResearchGate." *The Application of the ARIMA-models for forecasting the Dynamics of Foreign Trade of Ukraine*, 2020.

M.S. Gusev, A.A. Shirov. "ResearchGate." *Foreign Trade forecast in the system of midterm forecasting of the Russian economy*, 2009.

Nimisha Tomar, Durga Patel, Akshat Jain. "ResearchGate." *Air Quality Index Forecasting using Auto-regression Models*, 2020.

S. Vemuri, R. Balasubramanian, E.F. Hill. "ReasearchGate." *Load Forecasting using Moving Average Stochastic Models*, 1974.

Sandip Roy, Sankar Prasad Biswas, Subhajyoti Mahata, Rajesh Bose. "ResearchGate." *Time Series Forecasting using Exponential Smoothing to Predict the Major Atmospheric Pollutants*, 2018.