Hourly Energy Consumption Forecasting Using Time Series Models

**1-May-2020**

**Vaibhav Kumar**

**OVERVIEW**

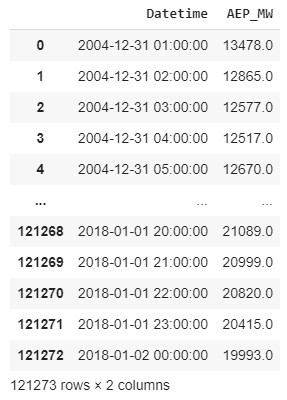
We are given the hourly energy consumption data of a region of the US from 2004-2018. And we are trying to build a time series model to accurately forecast the energy consumption rate.

**GOAL**

* Split the last year into a test set
* Find trends in energy consumption around hours of the day
* Understand how daily trends change depending of the time of year
* And we compare different Forecasting Models like Exponential Smoothening and ARMA.

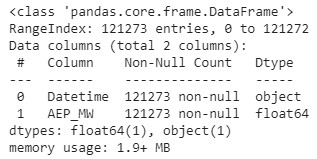
**DATASET**

Let's look at the Dataset

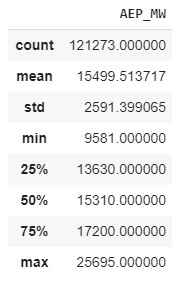


Above Dataset contains two features Datetime and AEP\_MW, Datetime is the Time stamp of the data and AEP\_MW is the energy consumed in Mw. Now lets see data information.

* df.info()



* df.describe()

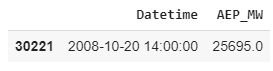


* Checking for null value

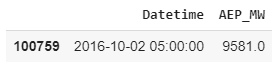


**Checking for Highest and Lowest consumption & Which Year :**

* Highest



* Lowest



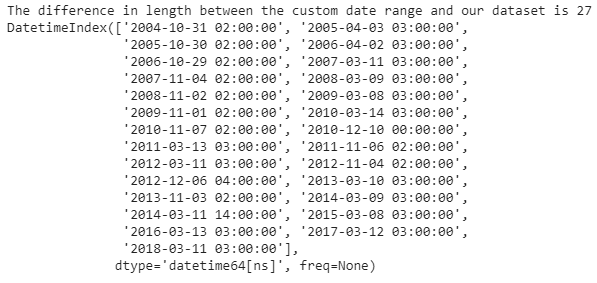
**Conclusion:** The Maximum Energy Consumed was 25695mW on September 20, 2008 at 14:00:00 and Minimum was 9581mW on September 02, 2016 at 05:00:00.

**DATA CLEANING**

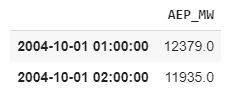
* We will change the df[‘Datetime’] column to datetime format using pandas pd.datetime and sort it in ascending order.
* Deduplicating, after formatting Datetime we will deduplicate the datetime keeping only the last measurement per date.
* Checking for the dataset is continuous or not. By checking the index frequency.



The fact our datetime index's frequency is set to None is an indication there are some missing data points somewhere (otherwise Python could deduce it). Let's compare it to an uninterrupted custom date range. We will create a date range of min(df.index) to max(df.index) i.e. 2004-10-31 02:00:00 to 2018-03-11 03:00:00.

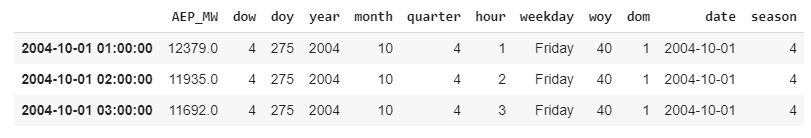


* Now we will reindex the dataset with the date range we just created
* And Using this will append missing datatime and create null values in the target variable. After this using interpolate we will fill the null values with values that lies on a linear curve between existing data points
* Let's check the dataset for continuous or not again.

Let’s look at the cleaned dataset of two rows now

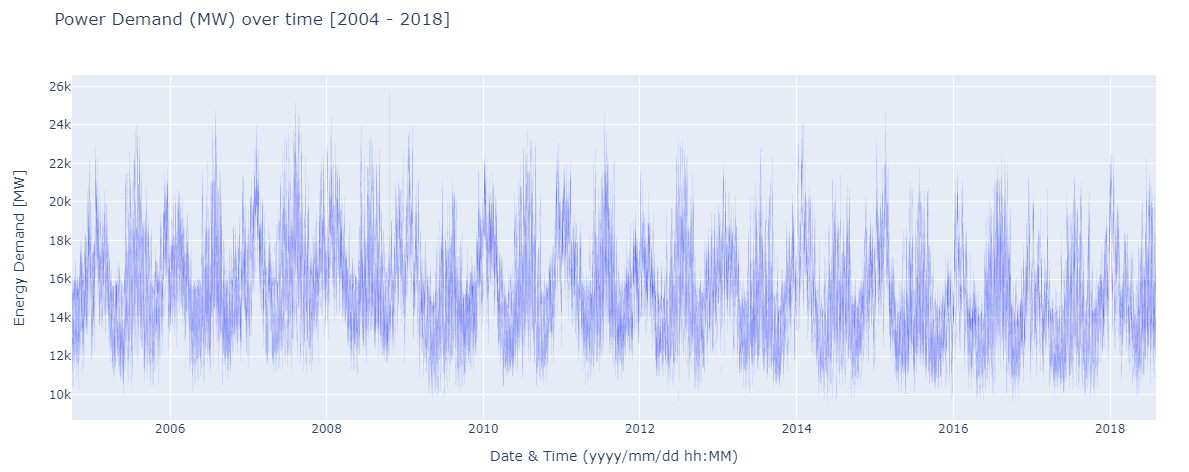
* Extracting Time Features

Extracting Day of the week, Day of the year, Year, Month, Quarter, Hour, Weekday, Week of year, Day of month, Date of Month, Date and Season from Datetime column in dataset.



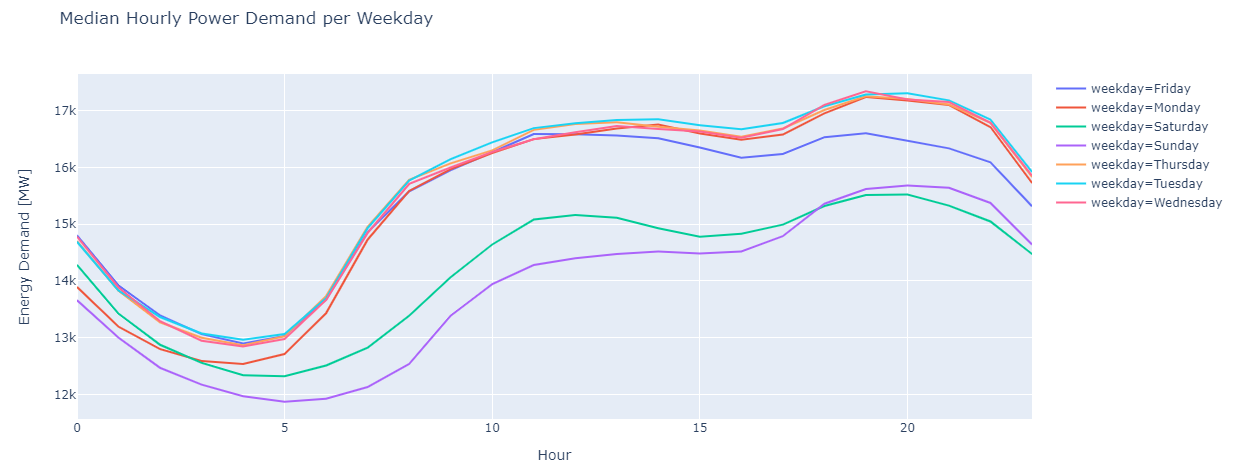
**VISUALISITAION**

* Power Demand Over Time Plot



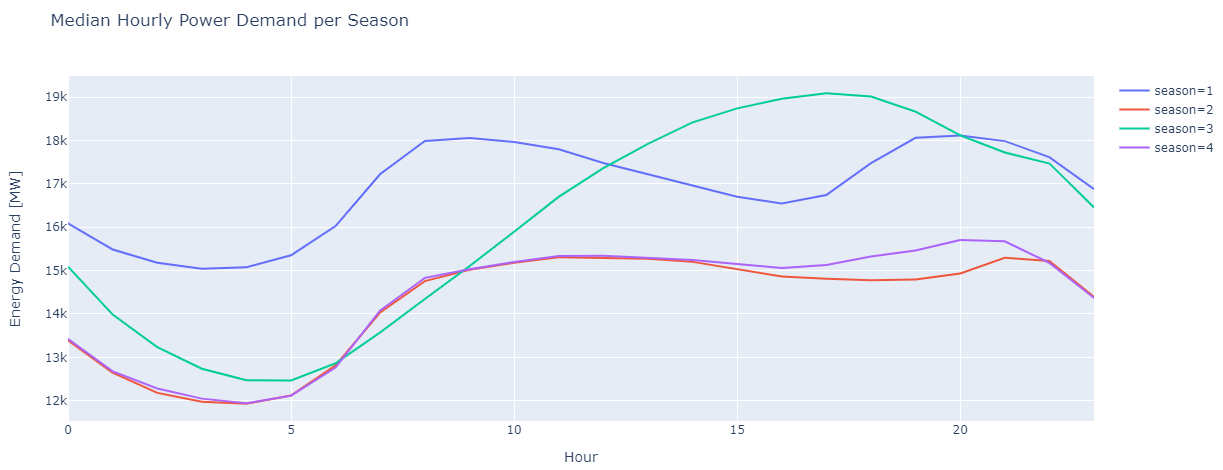
We can clearly identify a seasonal pattern in the above plot.

* Median Hourly power Demand per Weekday



From the above plot we can tell that energy demand is comparatively low on weekends and dip a little sooner on Friday afternoon.

* Median Hourly Power Demand Per Season

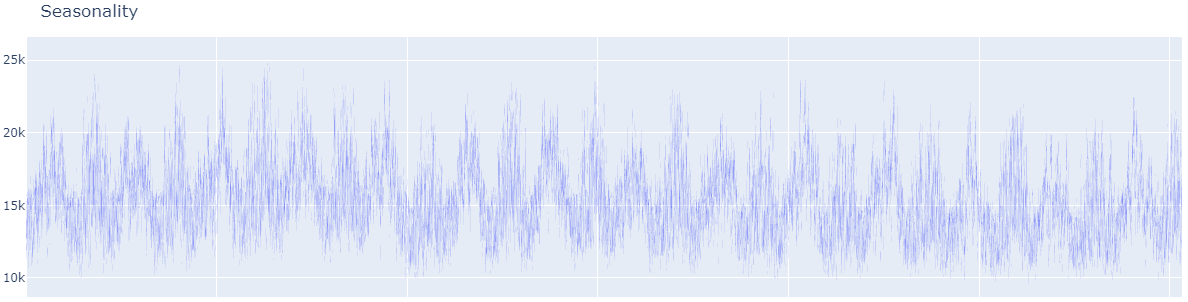


From the above plot we can say that Season 3 has the highest demand for power.

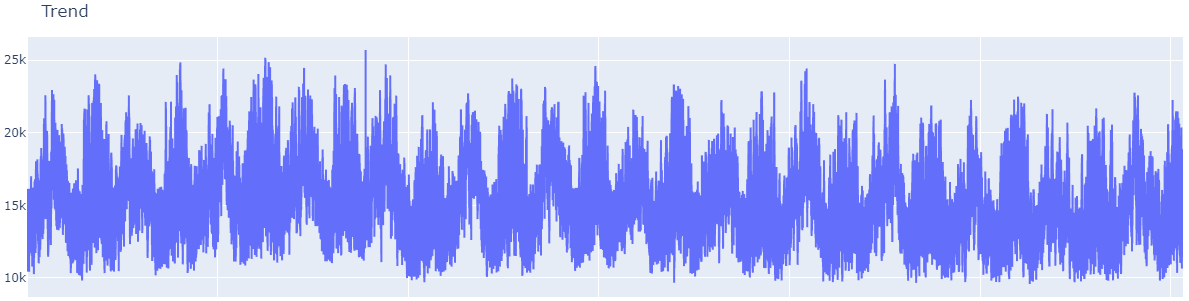
**DECOMPOSING THE TIME SERIES**

For a clear view of Seasonality, Trend and Noise/Residuals we will decompose the time series data using statsmodel seasonal\_decompose and see the plots.

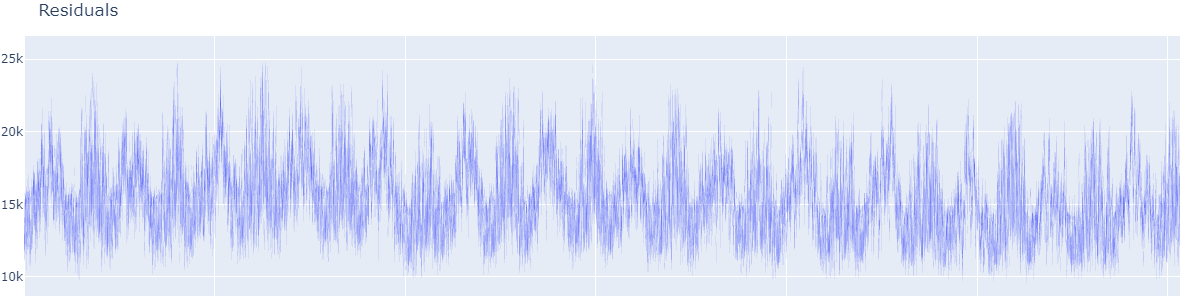
* Seasonality:



* Trend:



* Noise/Residual:



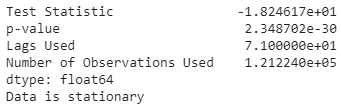
**TEST FOR STATIONARITY**

Now we're gonna perform an **Augmented Dickie-Fuller test** to check the stationarity of the data set time series. The Augmented Dickie-Fuller test is a type of statistical test called a unit root test.

The intuition behind a root test is that it determines how strongly a time series is defined by a trend.

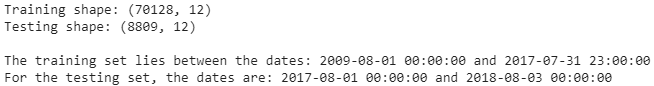
* Null Hypothesis (H0): If failed to be rejected, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure.
* Alternate Null Hypothesis (H1): The null hypothesis is rejected; it suggests the time series doesn't have a unit root, meaning it is stationary. It does not have a time dependent structure.

The result after the Dickie Fuller Stationarity Test:



It says that our Time series Data is Stationary.

**TRAIN/TEST**

After cleaning the data and Visualising and performing Stationarity test we will now split our data into training and testing for further use in model performance. We will use last 1 year data as test data.

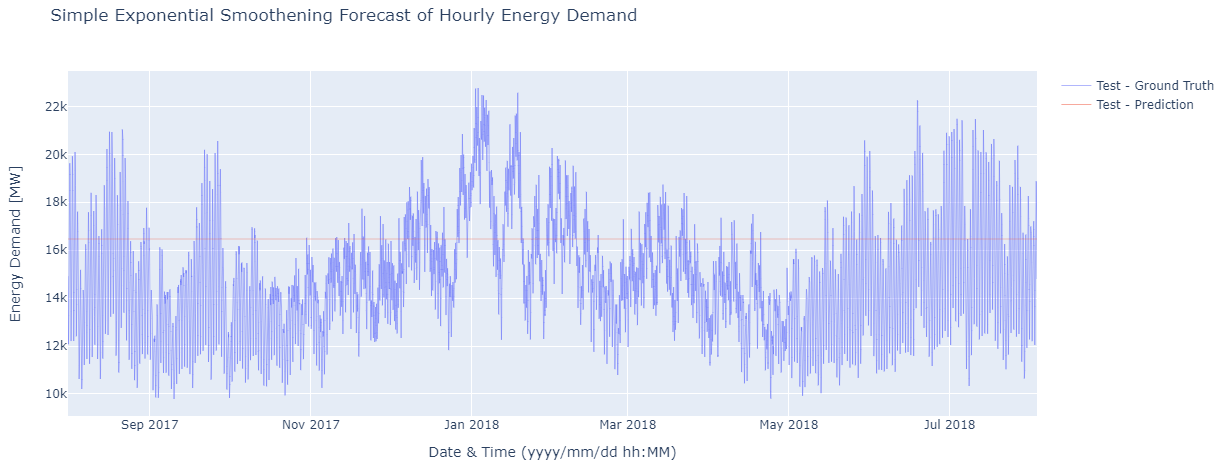
**FORECASTING MODELS**

* Simple Exponential Smoothing

Simple Exponential Smoothing, is a time series forecasting method for univariate data without a trend or seasonality.It requires a single parameter, called *alpha* (*a*), also called the smoothing factor or smoothing coefficient.

This parameter controls the rate at which the influence of the observations at prior time steps decay exponentially. Alpha is often set to a value between 0 and 1. Large values mean that the model pays attention mainly to the most recent past observations, whereas smaller values mean more of the history is taken into account when making a prediction.

Plot of Simple Exponential Smoothing of our time series.



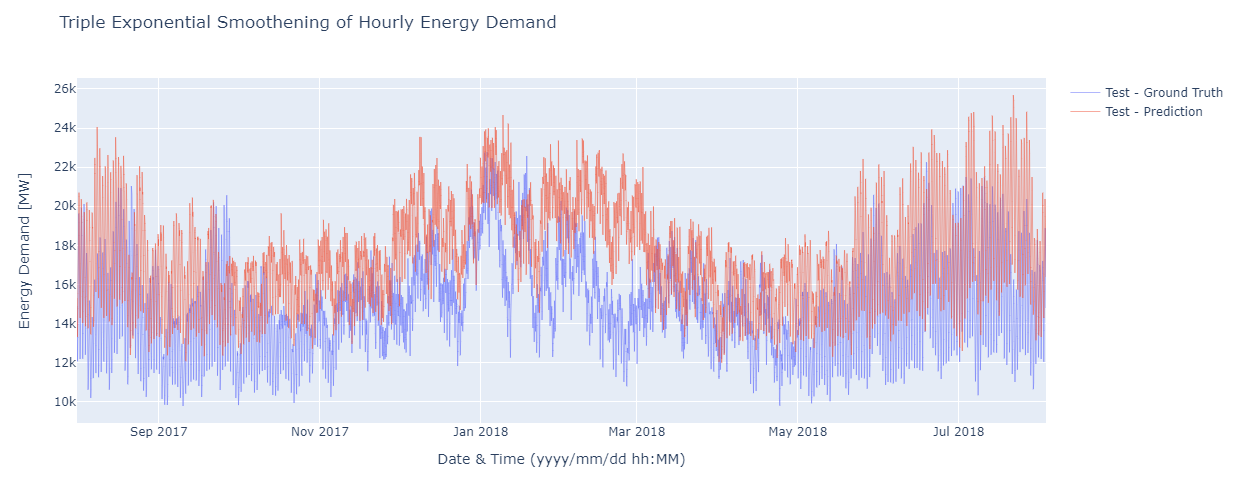
Let's look at the model evaluation results

### **TRIPLE EXPONENTIAL SMOOTHING**

Triple Exponential Smoothing is an extension of Exponential Smoothing that explicitly adds support for seasonality to the univariate time series.

This method is sometimes called Holt-Winters Exponential Smoothing, named for two contributors to the method: Charles Holt and Peter Winters.

Triple Exponential Smoothing or Holt-Winter Exponential Smoothing performed on our time series gives below plot

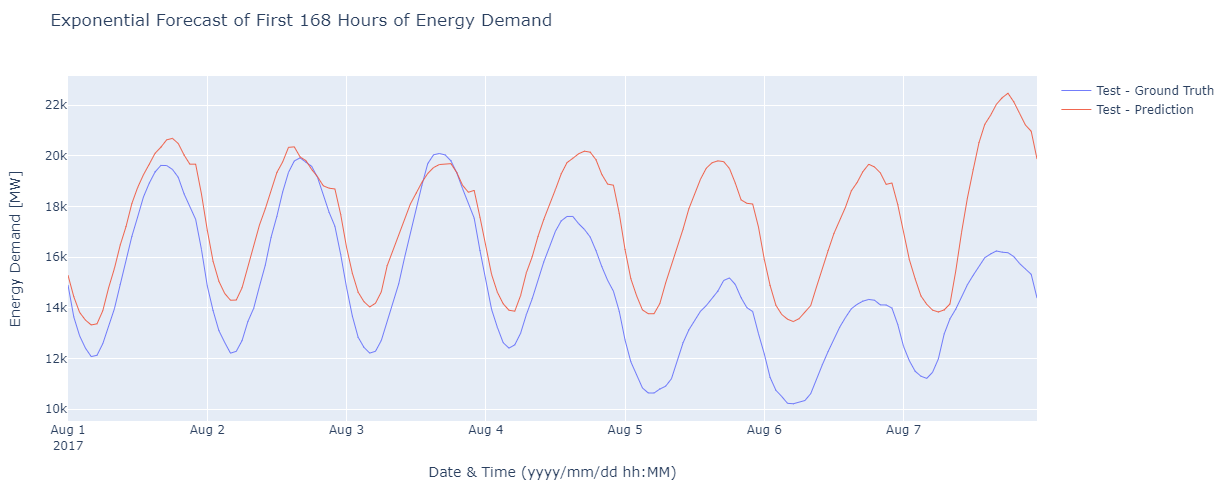


Let's look at the model evaluation results



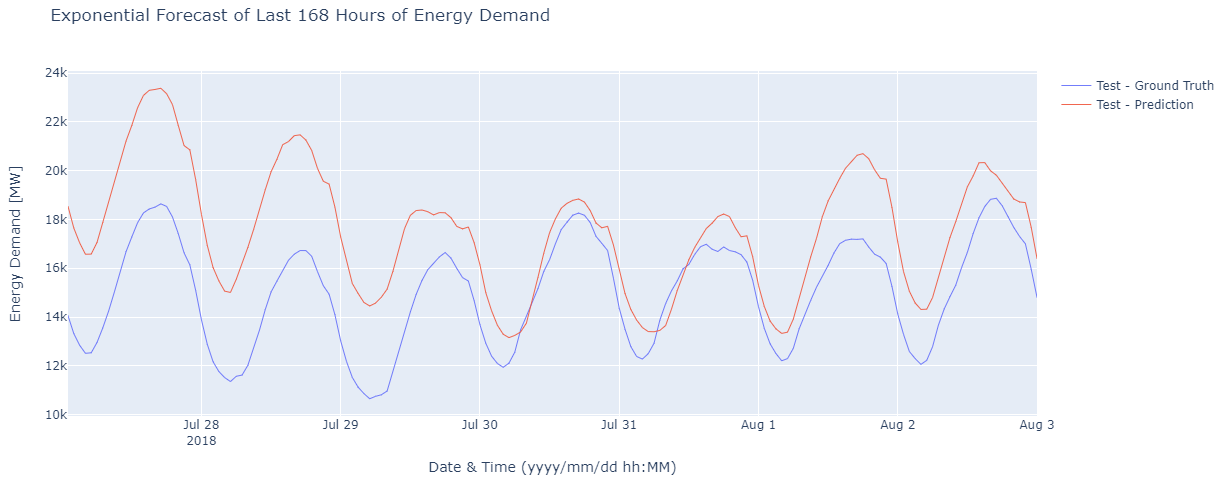
Now lets see a day wise prediction at the beginning and end of the forecast

* Plot of first 168 hours of Energy Demand



And



* Plot of last 168 hours of Energy Demand

And



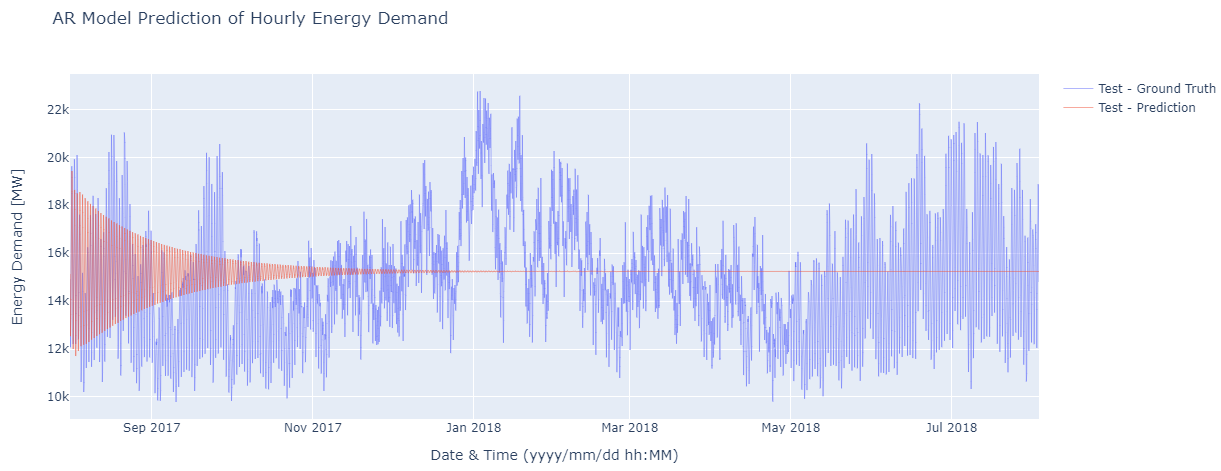
Till now Simple Exponential Smoothing gives less MAPE than Triple Exponential Smoothing.

Now we will try some other Forecasting Model,

* **AUTO-REGRESSIVE MODEL**

Autoregression is a time series model that uses observations from previous time steps as input to a regression equation to predict the value at the next time step.

It is a very simple idea that can result in accurate forecasts on a range of time series problems.

Fitting our data in Auto-Reg model and plotting the forecast

Let's look at the model evaluation results



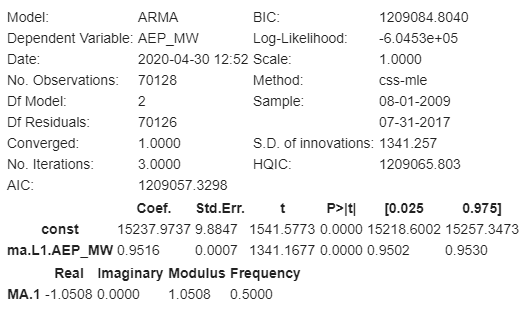


* **ARIMA (Autoregressive Integrated Moving Average)**

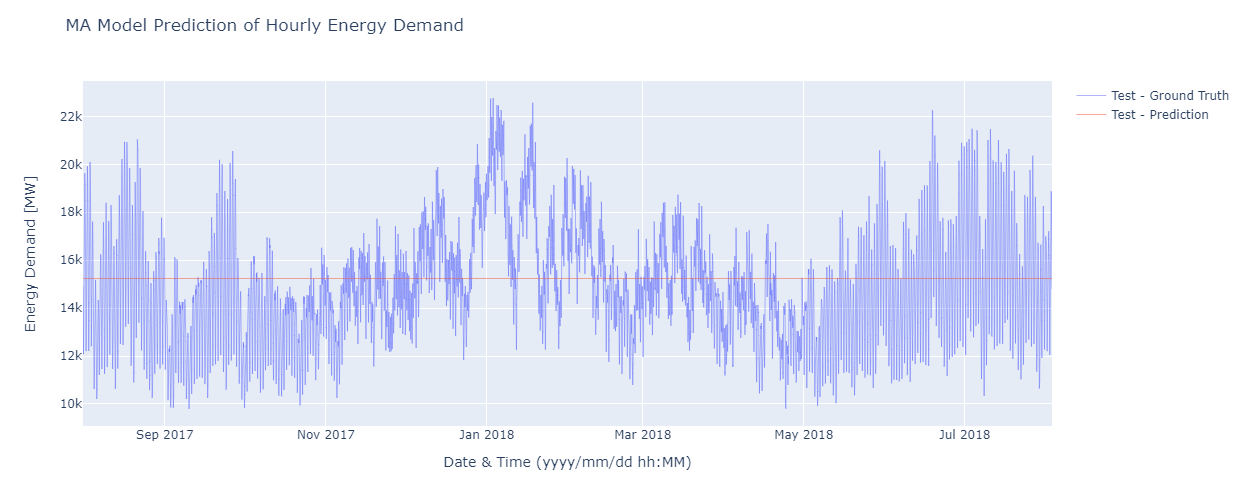
ARIMA is a model which is used for predicting future trends on time series data. It is a model that forms a regression analysis.

* **AR (Autoregression) :** Model that shows a changing variable that regresses on its own lagged/prior values.
* **I (Integrated) :** Differencing of raw observations to allow for the time series to become stationary
* **MA (Moving average) :** Dependency between an observation and a residual error from a moving average model
* Moving Average using ARIMA model

Giving ARIMA model order=(0,0,1) we can make ARIMA model predict only moving average and fitting it our time series we get model summary as



Plotting the prediction forecast:



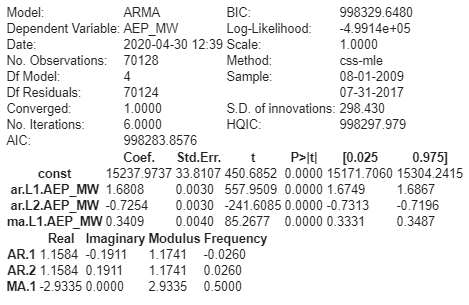
Let's look at the model evaluation results



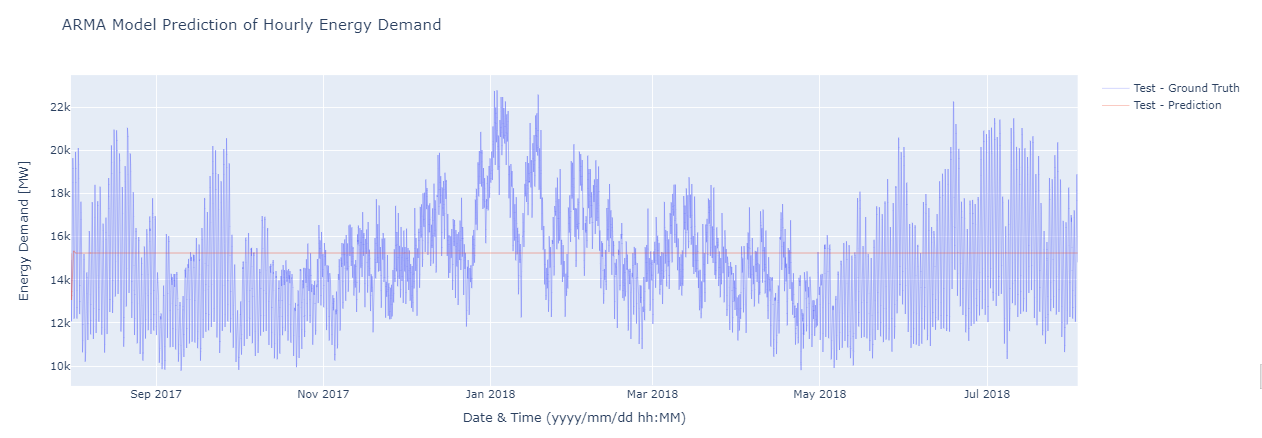


* ARMA using ARIMA model

Since our Time Series is Stationary as we checked this using Dickie Fuller Test for stationarity, we will use the ARMA model using ARIMA with order(2,0,1) and fit it our time series we get model summary as



Plotting Prediction Forecast of ARMA Model



Let's look at the model evaluation results





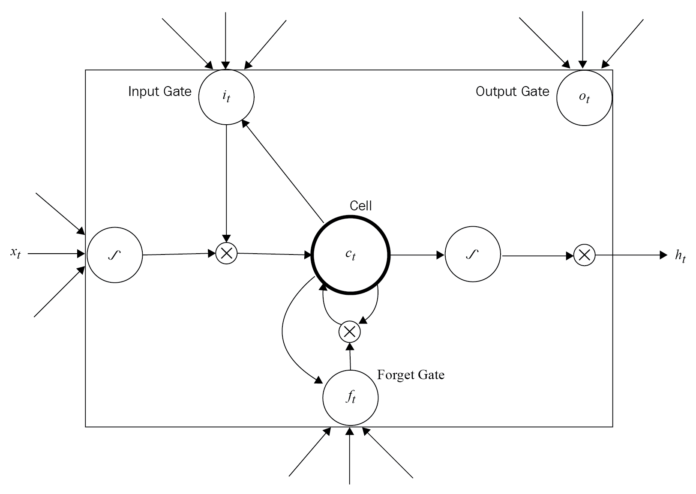
Comparing all the models performed above we can conclude that mean percentage error of all the respective models are

* MAPE of **Simple Exponential Smoothing** is **17.64**
* MAPE of **Triple Exponential Smoothing** is **19.69**
* MAPE of **Auto-Regressive Model** is **13.03**
* MAPE of **Moving Average using ARIMA** is **13.76**
* MAPE of **ARMA model using ARIMA** is **13.75**

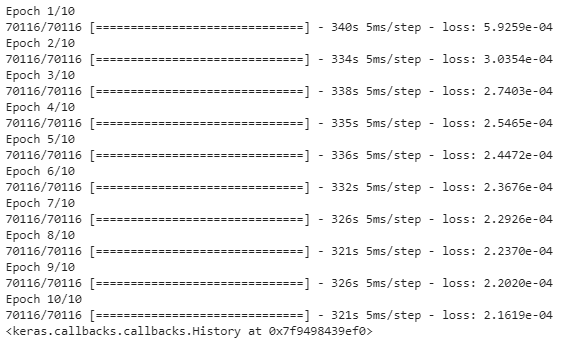
We can clearly say that **Auto-Reg Model** gives the least mean average percent error i.e. **13.03**

* **LSTM NEURAL NETWORK**

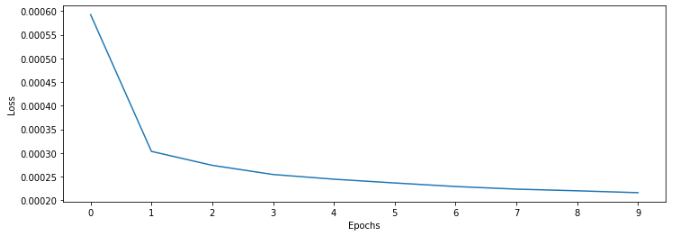
LSTM stands for long short term memory. It is a model or architecture that extends the memory of recurrent neural networks. Typically, recurrent neural networks have ‘short term memory’ in that they use persistent previous information to be used in the current neural network. Essentially, the previous information is used in the present task. That means we do not have a list of all of the previous information available for the neural node. LSTM introduces long-term memory into recurrent neural networks. It mitigates the vanishing gradient problem, which is where the neural network stops learning because the updates to the various weights within a given neural network become smaller and smaller. It does this by using a series of ‘gates’. These are contained in memory blocks which are connected through layer, like this:



Fitting LSTM Model on our time series with 10 epochs we get



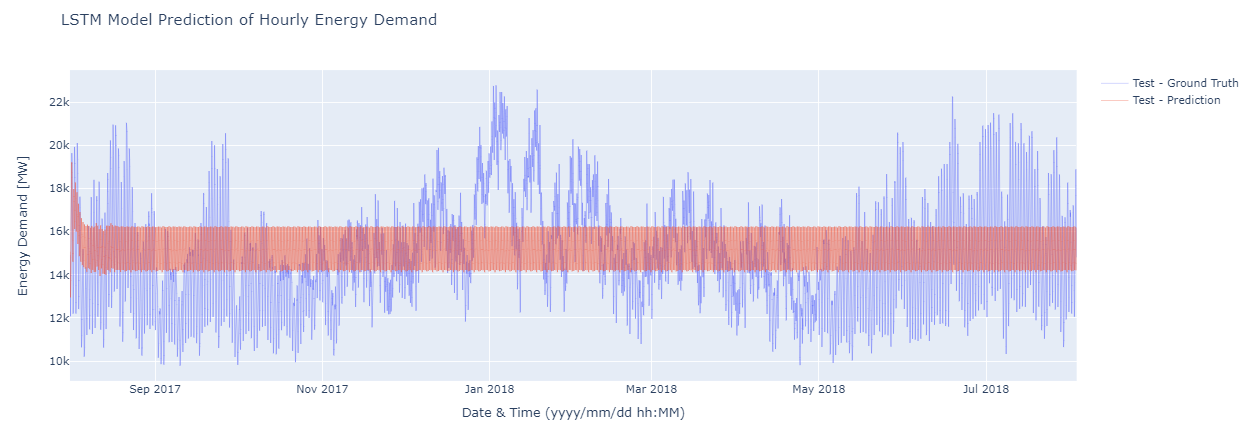
Plotting the Loss graph of 10 epochs



After model fitting we make the prediction and add it to dataframe



Plotting LSTM prediction forecast



Let's look at the model evaluation results





# **References:**

1. For Auto-reg Model, [Autoregression Models for Time Series Forecasting With Python](https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/)
2. Using LSTM, [Time Series Forecasting — ARIMA, LSTM, Prophet with Python](https://medium.com/@cdabakoglu/time-series-forecasting-arima-lstm-prophet-with-python-e73a750a9887)