Time Series Analysis

Forecasting Comparison: Neural Network vs ARIMA

Shourjya Basu(MCS202207)
Sampad Kumar Kar(MCS202215)

20th April 2023

Contents

1 Objective

- We use a power consumption dataset and use it to predict the same using weather conditions.
- We perform EDA and necessary data preprocessing to prepare the data for model fitting.
- We fit 'Time Series models (ARIMA)' as well as 'Neural Network models (LSTM and Transformers)' on the dataset.
- Our goal is to develop a model that takes in weather conditions at a particular time of the day and returns the power consumption at that time.

2 The Data

2.1 Dataset Summary

We are using the Tetuan City power consumption dataset recorder for the year 2017. The dataset records weather conditions (temperature, humidity, wind speed, and drift speed) and power consumption in 3 zones every 10 minutes for an entire year. The city has summer in the middle of the year and winter at the starting and end. Its weather is affected by both the Mediterranean climate and the Sahara climate and which in turn affects the power consumption patterns in the city.

This data contains the power consumption of metrics (in MW: megawatts) for every 10 minutes starting from 2017-01-01 00:00 till 2017-12-30 23:50. It has a total of 52416 entries.

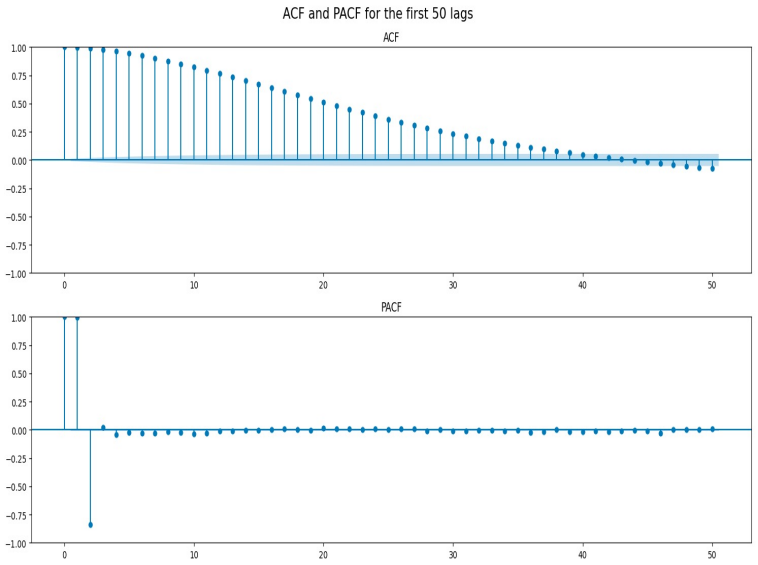
2.2 Data Smoothing

Original data contains the power consumption of metrics (in 'MW': megawatts) for every '10' minutes starting from '2017-01-01' at '00:00:00' till '2017-12-30' at '23:00:00'.

Since, processing the data every '10' minutes is too much for our model, we will aggregate the data to intervals of '1' hour, by taking the mean of the power consumption of every '6' metrics (60 minutes, i.e. six 10 minute intervals. We do the same for other features as well.

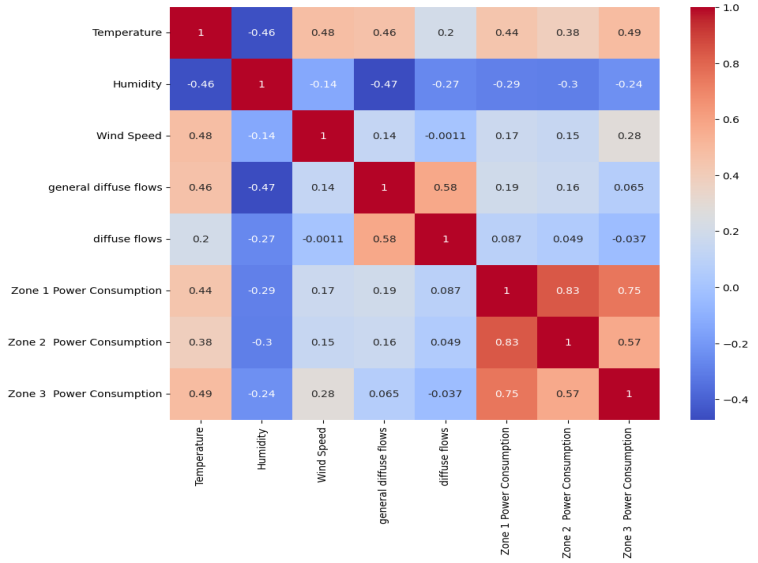
This also results in the reduction of the data from having '52411' examples to '8731'. Another reason why we consider smoothing the data to intervals of '1' hour from intervals of '10' minutes is because of the high correlation caused by long sequences of consecutive data points (in the original data). Since, the average consumption of power typically does not vary by much over short intervals of '10' minutes gap, even after multiple time lags, the power consumption might still be similar to the one at current time.

This can be witnessed by plotting the ACF and PACF plots w.r.t. the target variable for the original dataset (with '10' minute intervals). NOTE: We plot the ACF and PACF after the data was deemed stationarity by the ADF Test. Here are the plots for the same:



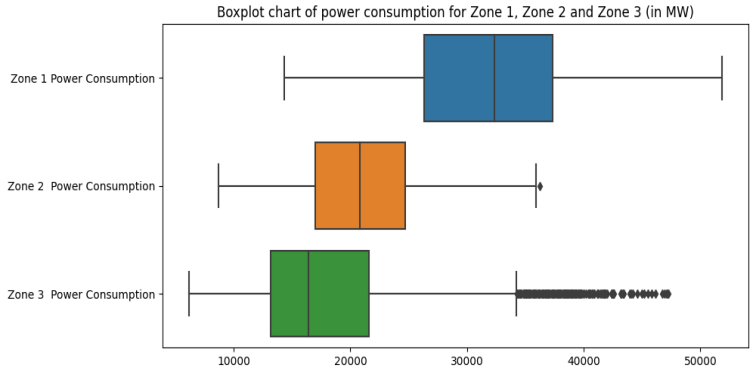
3 Exploratory Data Analysis

3.1 Correlation Matrix



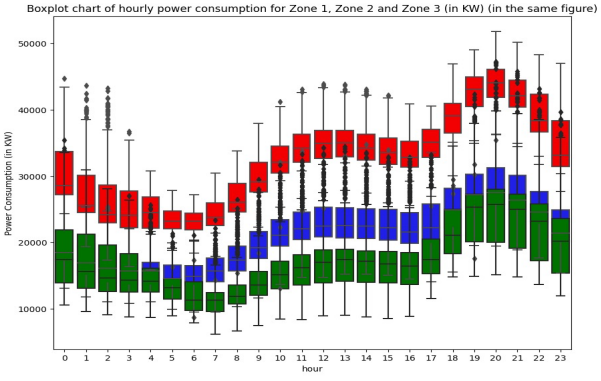
- Temperature, Wind Speed, and general diffuse flows are positively correlated to all the power consumption zones. Higher use of air conditioners could be a possible explanation for this.
- Humidity, on the other hand, shows a slight negative correlation with the power consumption of all three areas.
- Power consumption in each of the three zones, show a high correlation between each other. This suggests that power consumption in these zones, tend to increase or decrease together.

3.2 Power Consumption in Zones



Based on the above boxplot, we can conclude that Zone 1 has the highest power consumption, while Zone 3 has the least.

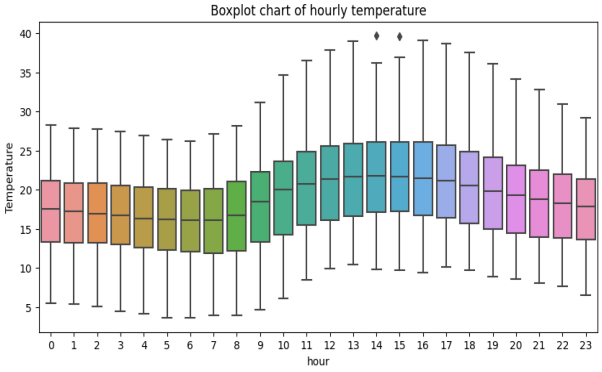
3.3 Hourly Power Consumption



Again, based on the above plot we can see how Zone 1 has higher average consumption compared to Zone 2 and Zone 3. On top of that, the average consumption increases during the evening hours for all three areas.

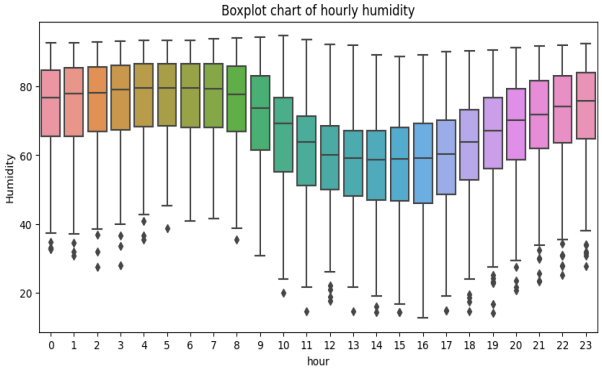
3.4 Hourly temperature chart

The above feature has a direct correlation with the hourly temperature.



3.5 Hourly humidity chart

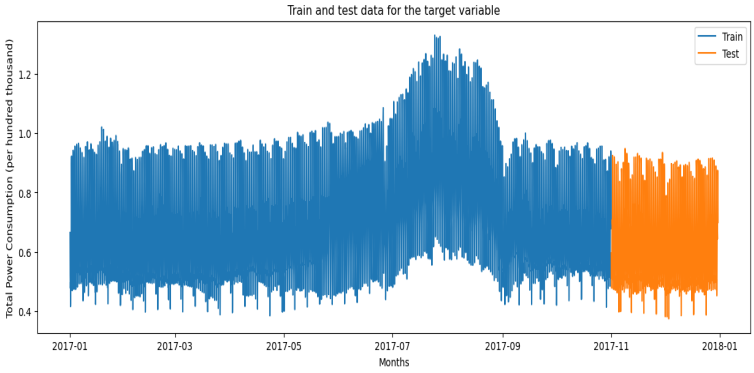
On the other hand, power consumption is inversely correlated with humidity. Hence, humidity follows the exact opposite trend.



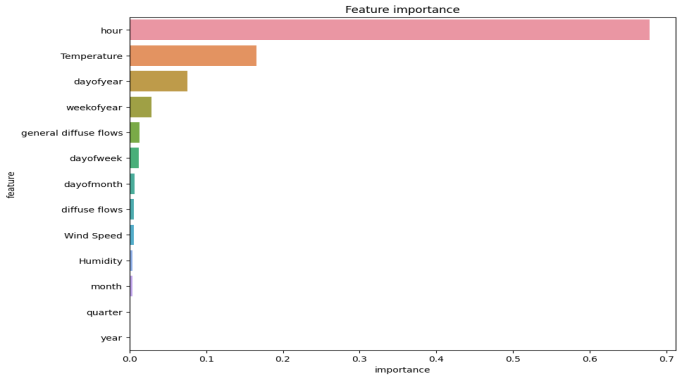
4 Data Preprocessing

4.1 Train-Test Split

Here, 83.33% of the data available is from **January 1st until October 31st**, so that's going to be our **training set**. The remaining 16.67% of the data is from **November 1st until December 30th**, which is our **test set**.



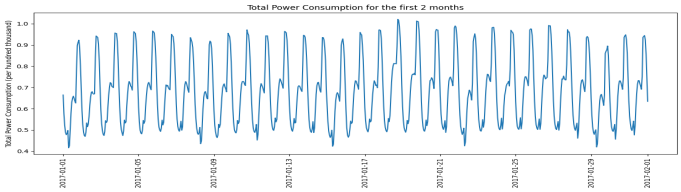
4.2 Feature Importance



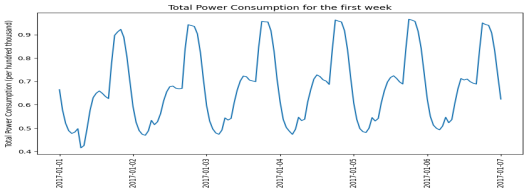
5 Time Series Analysis

5.1 Power Consumption

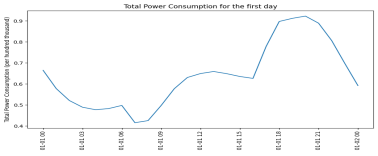
Firstly, we take a look at the data spanning over the first 2 months, to check the variation of the power consumption over a few months of data.



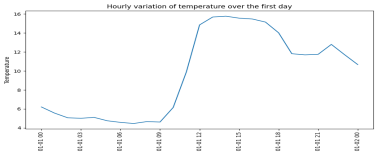
This signifies the presence of a seasonal component in the data. We zoom in further to see the variation in the power consumption over a week.



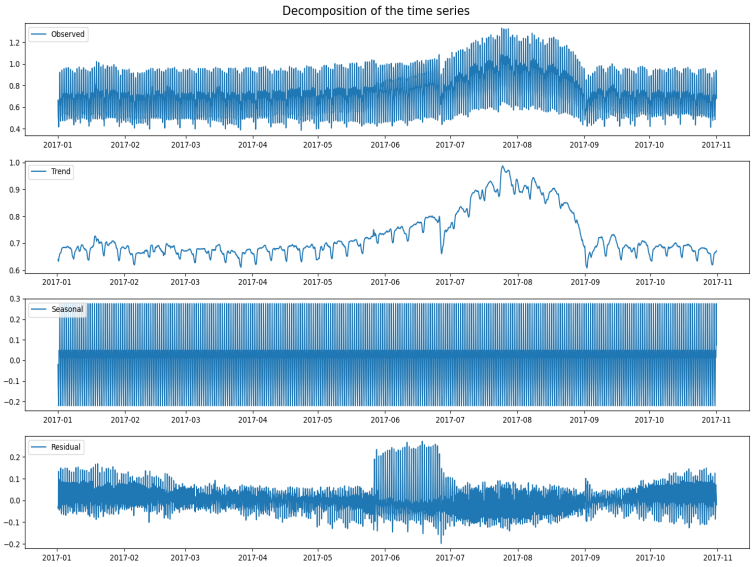
This clearly shows the presence of daily seasonality in the data. Now, we zoom in even further to see the variation in the power consumption over a day, to pin down possible hourly patterns.

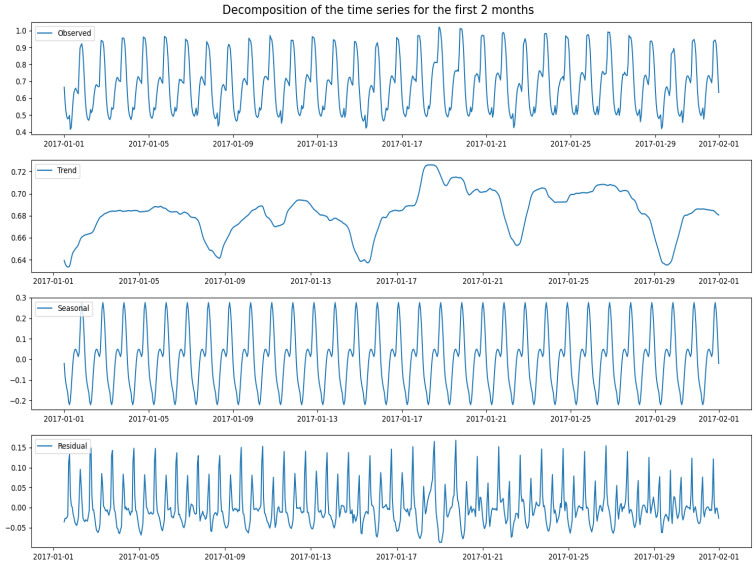


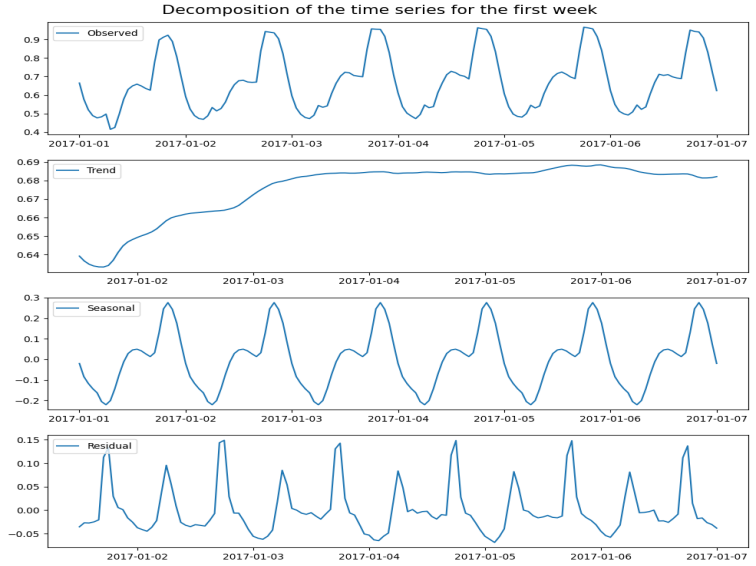
This clearly shows the presence of daily seasonality in the data. Now, we zoom in even further to see the variation in the power consumption over a day, to pin down possible hourly patterns.



5.2 Decomposition




Decomposition of the time series for the first 2 months
Observed
Trend
Seasonal
Residual
2017-01-01
2017-01-05
2017-01-09
2017-01-13
2017-01-17
2017-01-21
2017-01-25
2017-01-29
2017-02-01


Decomposition of the time series for the first week
Observed
Trend
Seasonal
Residual
2017-01-01
2017-01-02
2017-01-03
2017-01-04
2017-01-05
2017-01-06
2017-01-07

5.3 Stationarity Check - ADF test

We use **Augmented Dickey-Fuller** (ADF) test to check for stationarity in the data. The null hypothesis of the ADF test is that the time series is **non-stationary**. The test results comprise of a **test statistic** and some **critical values** for different confidence levels. For **p-values** less than 0.05, we can reject the null hypothesis, i.e. the time series is **stationary**.

5.3.1 ADF-test on observed data

ADF-test Observations (on Original Data):	
Test Statistic	-2.416591
p-value	0.137108
#Lags Used	36.000000
Number of Observations Used	7259.000000
Critical Value (1%)	-3.431251
Critical Value (5%)	-2.861938
Critical Value (10%)	-2.566982

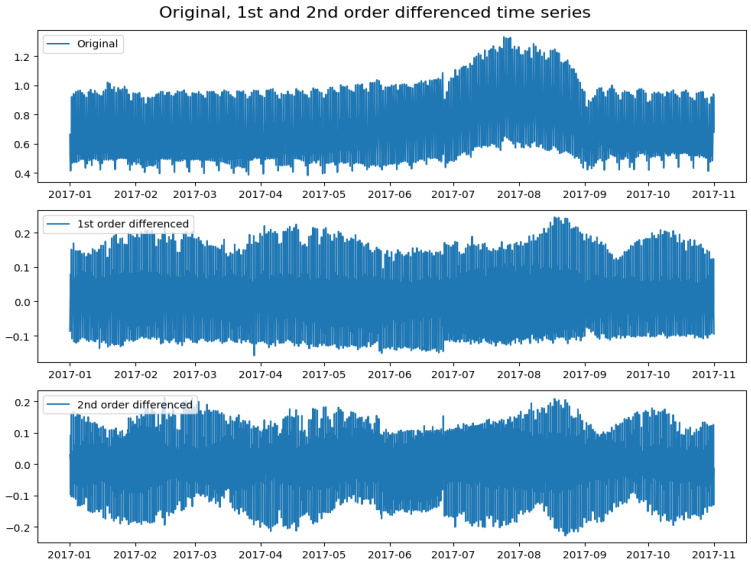
Since the **p-value** (0.13) is greater than 0.05 (significance level of 5%), we fail to reject the null hypothesis and conclude that the time series is **non-stationary**.

5.3.2 ADF-test on first order differenced data

ADF-test Observations: (on First Order Differenced Data)	
Test Statistic	-19.776232
p-value	0.000000
#Lags Used	36.000000
Number of Observations Used	7258.000000
Critical Value (1%)	-3.431251
Critical Value (5%)	-2.861938
Critical Value (10%)	-2.566982

Since the **p-value** (0.0) is smaller than 0.05 (significance level of 5%), we reject the null hypothesis and conclude that the first order differenced time series is **stationary**.

5.4 Differencing

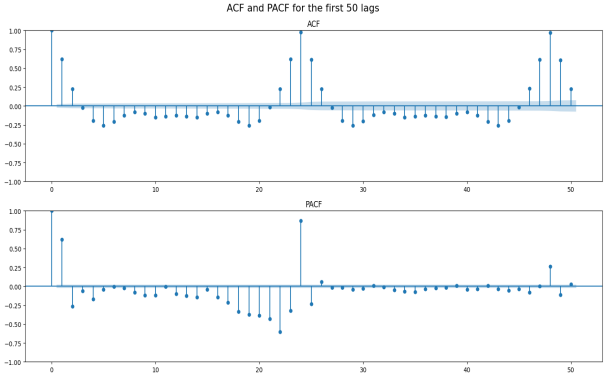


5.5 ACF and PACF

Before trying to fit classical time series models like ARIMA, we need to find the optimal parameters for the model.

We do this by plotting the **Autocorrelation Function (ACF)** and **Partial Autocorrelation Function (PACF)** plots to find the optimal values of p and q for the first and second-order difference data (as they are stationary-based on ADF-Test).

5.5.1 ACF & PACF - 1st order differenced time series



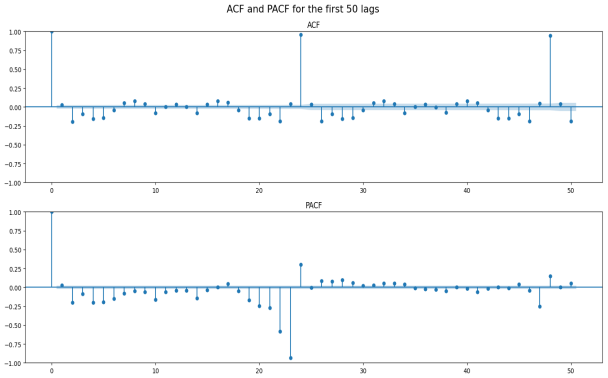
We observe sudden peaks near `lag = 24` and `lag = 24*2 = 48`. These peaks signify the presence of **daily** seasonality in the data.

Based on the `ACF` and `PACF` plots above we also decide upon the following parameters for the `ARIMA` model:

- `d = 1` : This represents the no. of differencing required to make the time series stationary.
- `q = 3` : This represents the no. of moving average terms in the series. After inspecting the `ACF` plot, we can see that the first 3 lags are significant, so we choose `q = 3`.
- `p = 3` : This represents the no. of autoregressive terms in the series. After inspecting the `PACF` plot, we can see that the first 3 lags are significant, so we choose `p = 3`.

So, we try fitting a `ARIMA(3, 1, 3)` model to the data.

5.5.2 ACF & PACF - 2nd order differenced time series



As before, we again observe sudden peaks near `lag = 24` and `lag = 24*2 = 48`. These peaks signify the presence of **daily** seasonality in the data.

Again, based on the `ACF` and `PACF` plots above we decide upon the following parameters for the `ARIMA` model:

- `d = 2` : This represents the no. of differencing required to make the time series stationary.
- `q = 1` : This represents the no. of moving average terms in the series. After inspecting the `ACF` plot, we can see that the first lag is significant, so we choose `q = 1`.
- `p = 1` : This represents the no. of autoregressive terms in the series. After inspecting the `PACF` plot, we can see that the first lag is significant, so we choose `p = 1`.

So, we try fitting a `ARIMA(1, 2, 1)` model to the data.

5.6 ARIMA

ARIMA stands for Autoregressive Integrated Moving Average, which is a popular time series analysis technique used to model and forecast time series data. ARIMA models capture both autocorrelation (correlation between observations in the same series at different time points) and stationarity (constant statistical properties over time) in time series data.

ARIMA models are made up of three components:

- Autoregression (AR) - A regression model that uses past values of the time series to predict future values.
- Integrated (I) - The differencing step used to make the time series stationary.
- Moving Average (MA) - A model that uses the errors (residuals) from past forecasts to predict future values.

ARIMA models can be used for both short-term and long-term forecasting and are useful in a variety of applications such as finance, economics, and weather forecasting.

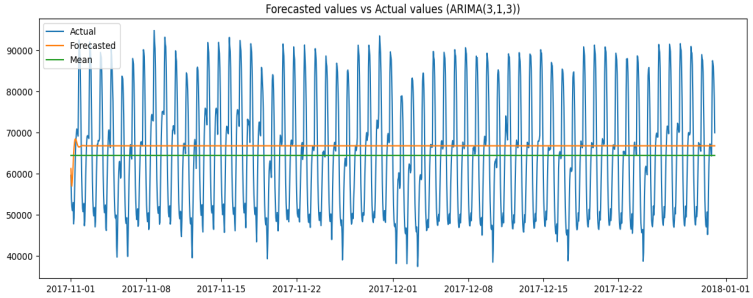
5.6.1 ARIMA(3,1,3)

SARIMAX Results

Dep. Variable:	Total Power Consumption	No. Observations:	7296
Model:	ARIMA(3, 1, 3)	Log Likelihood	12253.014
Date:	Thu, 20 Apr 2023	AIC	-24492.027
Time:	09:15:42	BIC	-24443.763
Sample:	01-01-2017 - 10-31-2017	HQIC	-24475.431
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.7186	0.096	7.467	0.000	0.530	0.907
ar.L2	0.5080	0.121	4.196	0.000	0.271	0.745
ar.L3	-0.5042	0.047	-10.718	0.000	-0.596	-0.412
ma.L1	-0.0889	0.095	-0.932	0.351	-0.276	0.098
ma.L2	-0.8388	0.060	-14.052	0.000	-0.956	-0.722
ma.L3	-0.0570	0.041	-1.370	0.168	-0.138	0.024
sigma2	0.0020	2.04e-05	99.638	0.000	0.002	0.002

Ljung-Box (L1) (Q):	0.46	Jarque-Bera (JB):	8580.73
Prob(Q):	0.50	Prob(JB):	0.00
Heteroskedasticity (H):	1.02	Skew:	1.17
Prob(H) (two-sided):	0.70	Kurtosis:	7.77



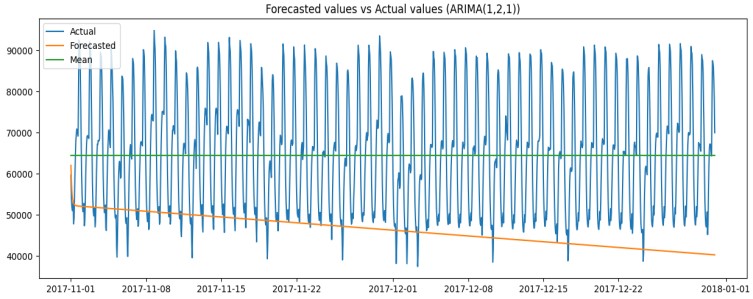
5.6.2 ARIMA(1,2,1)

SARIMAX Results

Dep. Variable:	Total Power Consumption	No. Observations:	7296
Model:	ARIMA(1, 2, 1)	Log Likelihood	11402.593
Date:	Thu, 20 Apr 2023	AIC	-22799.186
Time:	09:34:05	BIC	-22778.501
Sample:	01-01-2017 - 10-31-2017	HQIC	-22792.073
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.6230	0.011	58.207	0.000	0.602	0.644
ma.L1	-1.0000	0.221	-4.534	0.000	-1.432	-0.568
sigma2	0.0026	0.001	4.676	0.000	0.001	0.004

Ljung-Box (L1) (Q):	198.00	Jarque-Bera (JB):	2331.66
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	1.04	Skew:	0.70
Prob(H) (two-sided):	0.32	Kurtosis:	5.39



We see that the ARIMA(3,1,3) model performs better than the ARIMA(1,2,1) model. Now we fit our data on Neural Network models to compare and find out if they perform better than the classical Time Series models.

6 Neural Networks

6.1 About Neural Networks

Neural networks are a powerful tool for processing sequential data for time series forecasting. Time series data is sequential in nature, meaning that the order of the data points matters and the previous data points can influence the future values. Neural networks, specifically recurrent neural networks (RNNs), are well-suited for modeling such sequential data.

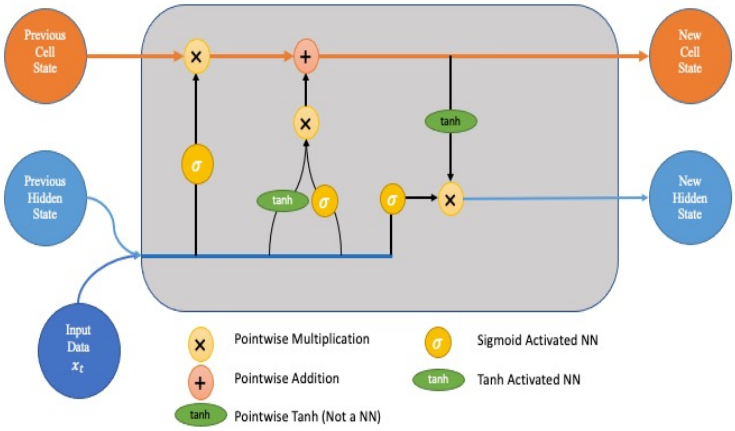
There are several popular neural network-based models for processing sequential data, including RNNs, gated recurrent units (GRUs), long short-term memory (LSTM) networks, and transformers. RNNs are a class of neural networks that can process sequences of varying lengths and remember past information through recurrent connections. GRUs are similar to LSTMs but with fewer parameters, making them faster and easier to train. LSTMs are a type of RNN that use memory cells to selectively retain information from previous time steps. Transformers are a relatively new neural network architecture that have shown excellent results in processing sequential data by attending to relevant parts of the input which can effectively capture long-term dependencies and patterns in the data. They achieve this by using self-attention mechanisms, which allow them to focus on relevant time steps while ignoring irrelevant ones. This makes transformers particularly effective for tasks such as predicting future values in a time series or generating text, where capturing long-term dependencies is critical.

6.2 Generating Sequences

Since, we are moving away from classical time series models, i.e. ARIMA in our case to neural networks based models, we convert the dataset into sequential data, to enable the neural networks to incorporate historical information to forecast into the future. Since we are looking at data over intervals of ‘1‘ hour each, the period of the data is ‘24‘ (i.e. the power consumption patterns tend to repeat at one-day intervals, which is ‘24‘ hours).

We decide to use hourly power consumption data spanning 2 weeks into the past, to forecast the next hourly power consumption in the future. This means we use a sequence of length ‘2*7*24 = 336‘ as our sequential data.

6.3 LSTM



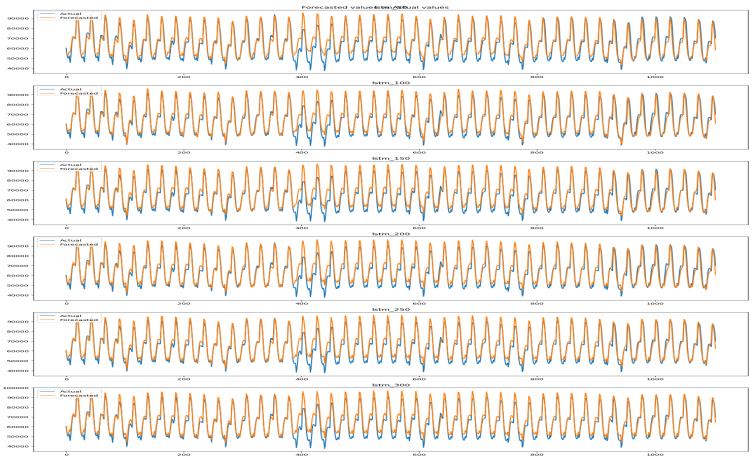
LSTMs (Long Short-Term Memory networks) are a type of recurrent neural network (RNN) that are particularly useful for processing sequential data for time series forecasting. They are designed to overcome the vanishing gradient problem that occurs in traditional RNNs, which limits their ability to learn long-term dependencies in sequential data. LSTMs achieve this by introducing a memory cell and three gates: an input gate, an output gate, and a forget gate, which allow the network to selectively remember or forget information from previous time steps. This makes them particularly useful for modeling complex relationships in time series data.

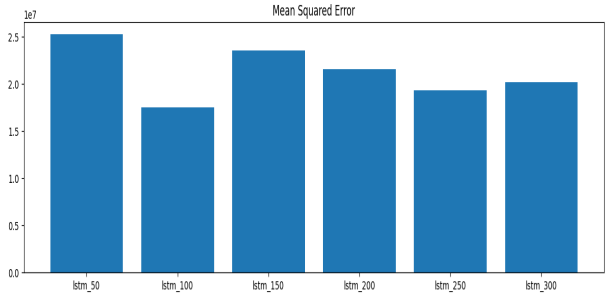
The architecture of the LSTM model implemented in the code consists of an LSTM layer followed by a fully connected (linear) layer. The LSTM layer has two main parameters: the number of hidden units and the number of layers. In this implementation, the default values for these parameters are 128 hidden units and 2 layers, respectively. The input to the LSTM layer is a sequence of length 336 (i.e., the two-week period of hourly power consumption data) and the output of the LSTM layer is fed to the fully connected layer, which produces a single output value for the next hour's power consumption prediction.

The model is trained using mean squared error (MSE) loss and optimized using the Adam optimizer, which is a popular optimization algorithm for neural networks. The number of epochs is set to 300, and the learning rate is set to 0.001. Additionally, the model is saved every 50 epochs.

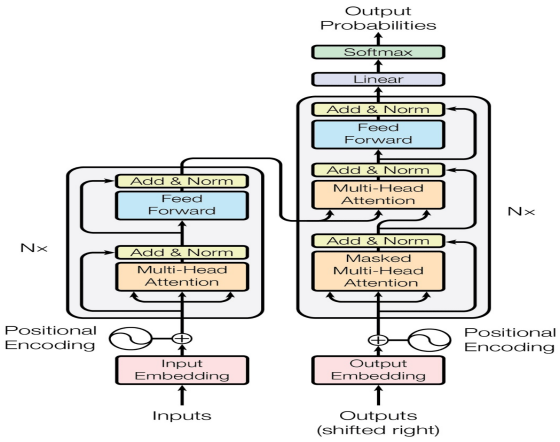
The running time for training the LSTM model on Kaggle P100 GPUs for 300 epochs was 12 min 52 sec and the saved model size was 814 KB on the disk.

6.3.1 LSTM-results





6.4 Transformers



Transformers are a type of neural network architecture that has gained popularity in recent years for natural language processing tasks, but they can also be used for time series forecasting. The main advantage of Transformers over LSTMs and other sequential models is that they are able to capture long-term dependencies in the time series data without the need for recurrent connections. This is achieved through the use of an attention mechanism that allows the model to focus on the most important parts of the input sequence at each time step. Additionally, Transformers can be trained much more quickly than LSTMs due to their ability to parallelize computations across the input sequence.

The Transformer architecture consists of an encoder and a decoder. The encoder is responsible for taking in the input time series data and generating a set of hidden representations for each time step. The decoder then takes in these hidden representations and generates a prediction for the next value in the time series. The key feature of the Transformer is its attention mechanism, which allows the model to selectively focus on different parts of the input sequence at each time step. The attention mechanism involves computing a set of attention scores for each time step based on its similarity to the other time steps in the sequence, and then using these scores to compute a weighted sum of the hidden representations for each time step.

The architecture of the Transformer implemented in the code consists of an encoder and a decoder, each comprising multiple blocks of Transformer layers. The encoder takes in the input data of shape (batch_size, sequence_length, input_size) and maps it to an embedding of size (batch_size, sequence_length, encoder_embedding_size) using a linear layer. The encoder embedding is then fed through multiple TransformerEncoderLayers to generate a contextualized representation of the input. Each TransformerEncoderLayer consists of a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The decoder takes the contextualized representation generated by the encoder and maps it to an embedding of size (batch_size, sequence_length,

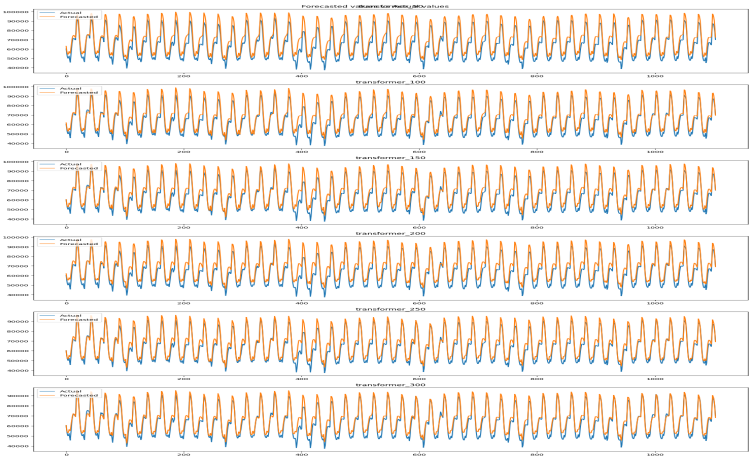
decoder_embedding_size) using another linear layer. This embedding is then reshaped to (batch_size, sequence_length*decoder_embedding_size) and projected to (batch_size, output_seq_size, decoder_embedding_size) using a linear layer and a batch normalization layer. The projected embedding is then fed through multiple TransformerDecoderLayers. Each TransformerDecoderLayer comprises a multi-head self-attention mechanism, a multi-head attention mechanism that attends over the encoder outputs (i.e., memory), and a position-wise fully connected feed-forward network.

The model is trained using mean squared error (MSE) loss and optimized using the Adam optimizer again.. The number of epochs is set to 300, and the learning rate is set to 0.001. Additionally, the model is saved every 50 epochs.

The running time for training the Transformer model on Kaggle P100 GPUs for 300 epochs was 19 min 01 sec and the saved model size was approximately 33.1 MB on the disk.

NOTE: Even though the model size for transformers is almost 42 times larger than LSTM on the disk, it only took 1.47 times as much time to train the model till the same no. of epochs, which goes on to demonstrate the gain in training speed caused due to the parallelization capabilities of Transformer.

6.4.1 Transformers-Results



7 Results

Models	MSE on Test Data (MW^2)
ARIMA(3,1,3)	203,237,826.00
ARIMA(1,2,1)	539,569,276.00
LSTM (Best)	17,565,033.00
Transformer (Best)	**15,999,286.00**

MSE Ratio	MSE Ratio
ARIMA(3,1,3)/Transformer	12.70293099
ARIMA(1,2,1)/Transformer	33.72458471
LSTM/Transformer	1.097863555

8 Conslusion

Transformers give the best results out of the few models that we implemented. Even though we ended up using a relatively simple Transformer model against a reasonably more complex LSTM, it ended up outperforming all (and the training loss was still decreasing, which goes on to demonstrate the potential of this architecture for forecasting). We also see that Neural Network outperform classical Time Series Models by quite a large margin.