CAPSTONE PROJECT BY TEAM D

A DATA SCIENCE APPROACH TO FORECASTING ELECTRICITY DEMAND IN NSW, AUSTRALIA

Baheerathan Gnanasundram, Matthew Seery, Mohammad Ahsan Ullah, Rahul Lobo.

School of Mathematics and Statistics
UNSW Sydney

June 2021

Submitted in partial fulfilment of the requirements of the capstone course ZZSC9020

Plagiarism statement

I declare that this thesis is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere.

I acknowledge that the assessor of this thesis may, for the purpose of assessing it:

- Reproduce it and provide a copy to another member of the University; and/or,
- Communicate a copy of it to a plagiarism checking service (which may then retain a copy of it on its database for the purpose of future plagiarism checking).

I certify that I have read and understood the University Rules in respect of Student Academic Misconduct, and am aware of any potential plagiarism penalties which may apply.

By signing this declaration I am agreeing to the statements and conditions above.

Signed: _____ Date: _____

Signed: _____ Date: _____

Signed: _____ Date: _____

Signed: _____ Date: _____

Acknowledgements

All thanks must go to our families and university professors who have guided us through this Capstone Project. Without them this would not be possible.

15/06/2021.

Abstract

Forecasting electricity demand is an important requirement as there are now more interested stakeholders associated with the generation and distribution of this service. Traditionally electricity demand was just the concern of governments. However, this has now been extended to market bodies and owners and operators of the underlying infrastructure required for this service. Forecasting accurate energy demand not only supports network infrastructure, but also aides investment decisions about power generation. It is thus of interest to a variety of stakeholders including power utilities, energy policymakers, and private investors just to name a few. This study aims to demonstrate how neural network (specifically LSTM neural networks) can be used to accurately forecast short-term energy demand. The focus will be on attributes such as temperature, month of the year, day of the week and time of the day as well as how past time lags can be used to predict future demand values. It is envisaged that the results of this study will provide useful insights for governments and market bodies to assist in the rules, policies and pricing that are invoked on the energy sector. Additionally, the businesses operating in this sector can use this information to guide their decision-making regarding electricity generation and distribution, as well as for the purpose of price benchmarking.

Contents

Chapter 1

Introduction

Towards the end of the 20th century, countries throughout the world began moving away from regulated government-controlled energy supply to a deregulated sector influenced by market forces [1]. This means that forecasting for electricity demands is essential information required beyond previously regulated governmental structures. It is now required by the market bodies working in the sector and private industries who invest in the generation and distribution of electricity. In turn, this aids in better rules, policies, and pricing in the energy sector. However, for this study, the focus will primarily be on short-term forecasting as this can greatly aid vested parties concerned in maximising profits and cutting costs.

Temperature plays a significant role in electricity demand. This is because heating is utilised more by customers in the cooler months while cooling is required for the hottest months of the year. For that reason, temperature is an essential attribute that will be investigated in this study. However, temperature does not account for other factors such as humidity or varying electricity usage patterns associated with specific days of the weeks. For example, a temperature of 25 degrees in Spring may not lead to as much usage of air conditioning as a humid day in Summer with the same temperature. Additionally, the usage rate of electricity will inevitably vary between weekdays, weekends, and public holidays due to the varying ratios of usage between business and residential customers. Therefore, the month of the year, day of the week and time of day will also be examined in this study.

Chapter 2

Literature Review

There have been various studies undertaken where more traditional statistical methods such as multiple linear regression (MLR) have been utilised to predict demand [2]. However, in more recent times there has been greater focus on the use of neural networks to help solve the problem of forecasting demand [3]. The advantage of neural networks is they are very suitable for determining non-linear relationships [3] and have been widely used for short-term demand forecasting [4]. They are also very flexible and easy to configure when dealing with time-series data [5]. For neural networks, monthly values have been a popular unit of measure and this is particularly true when short-term forecasting of demand has been employed [5]. However, this proposed solution differs because the focus will be on intervals shorter than even a day as this is of greater interest to the client. Nevertheless, the month of the year will still be considered as an input factor for the model for its potential to distinguish factors such as humidity that are not fully explained by temperature alone. Another distinguishing factor for this study will be the emphasis on the day of the week where each record will be categorised as either a weekday, weekend, or public holiday.

Up until the time of writing, the majority of deep learning models being appliied to energy forecasting fall under the subset of three major types [6]. These are: A feed forward neural network (FFNN)/Multi Layer Perceptron (MLP) through the process of increasing the number of hidden layers, some form of recurrence through a recurrent neural network (RNN), long-short term memory (LSTM) or gated recurrent unit (GRU), or through sequentially combining different types of algorithms into an overall structure. In 2020, Xue et al. contrasted these different approaches in order to forecast the heating demand of a district system [7]. Their experiments showed that the LSTM models were amongst the highest-performing of the models tested.

When applied to natural gas forecasting, a close relative to electricity demand, RNN models have proven to be useful in prediction. Wei et al. (2019) explored the application of LSTM for forecasting natural gas consumption of four cities [7]. In addition, LSTM models used for forecasting were compared with other model techniques, including multiple linear regression, feed-forward neural networks, and a support vector regression in this study. The authors' rigorous testing demonstrated that LSTM models achieved a greater accuracy than the other data-driven models. It appears that RNN and LSTM models have formed the majority of accurate forecasting implementations to date, with other machine learning techniques such as SVMs performing as well as, but not better than RNN and LSTM based models.

An example of this is demonstrated by Amarasignhe et al. (2017), who contrasted the deep learning techniques of a Convolutional Neural Network (CNN) and LSTM, against a traditional machine learning technique (SVM) in order to forecast the energy demand of a building. This forecast was for a time period of sixty hours ahead and used 4 years of training data [8]. Their experiments proved that the deep learning based forecasting models obtained less forecasting error when compared with standard machine learning-based techniques; in particular, the LSTM obtained the smallest error.

Due to the significance of the LSTM accuracy across a variety of these studies, it will form the basis of this analysis.

Chapter 3

Material and Methods

3.1 Software

A variety of software was used in the analysis as well as for collaboration and organisational purposes. GitHub was used as the main tool for collaboration and version control. It is not only extensively used in the industry but also enables the ability to collaborate on code seamlessly from local machines. This was the best choice as it is also used in the course instruction.

For data analysis purposes, Python, R/Rstudio and Tableau were all used. R was utilised in the initial exploratory data analysis phase in conjunction with Tableau, to visualise the dataset and to observe any obvious trends in the energy data. Tableau was especially useful in visualisation as it allowed the segmentation of days, times and months to be easily discerned in reference to energy demand. The main software used for the electricity demand modelling was Python. It was also used to clean, transform, and replace missing values in the supplied data and for merging separate datasets including the final LSTM model which was constructed.

Furthermore, Excel was used to fill in missing values for the temperature dataset and the report for this analysis has been constructed in RMarkdown.

3.2 Description of the Data

Three csv files were supplied for this project. The file totaldemand_nsw.csv contains three columns which record a date and timestamp (DATETIME) for the electricity consumed (TOTALDEMAND) for the associated region (REGIONID). The DATETIME columns contains dates between the 1st of January 2010 and midnight on the 18th of March 2021. Also included with each date is a timestamp with values for hours and minutes. There is a timestamp for each hour and thirty-minute interval within each hour for the aforementioned range of dates. All observations have the same value for the REGIONID column which is 'NSW1'.

The file temperature_nsw.csv also contains three columns. The columns record the location for the temperature observation (LOCATION), the date and time it was recorded (DATETIME) and the actual recorded temperature (TEMPERATURE). The values in the LOCATION column are all the same containing the value 'Bankstown' that describes the Bankstown weather station. The values in the DATETIME column are in the same format and range of dates as the temperature_nsw.csv file except some observations are outside the on hour or thirty-minute interval within each hour time periods. The file also contains duplicates and has

missing values between the dates of the 1st of January 2010 and 18th of March 2021.

The third file, forecastdemand_nsw.csv, contains periodic forecasts (FORECASTDEMAND) for a set date and time (DATETIME) and the date and time the forecast was made (LASTCHANGED). The values in the FORECASTDEMAND column align with the same range of dates in the totaldemand_nsw.csv file. The format of the date and time in this column and the DATETIME column is the same as the DATETIME column in the totaldemand_nsw.csv file. The forecastdemand_nsw.csv file also includes a REGIONID column and all with the same values as the totaldemand_nsw.csv file. There is also a column called PREDISPATCHSEQNO which contains a unique id number for each forecast and a PERIODID column which contains a unique ID for forecasts of each unique date and time in the DATETIME column. The data in this file was not used in the neural network models that were tested.

3.3 Pre-processing Steps and Data Cleaning

The data in the file totaldemand_nsw.csv required no further cleaning other than the removal of the REGIONID column as it contains redundant data. For the temperature_nsw.csv file, further cleaning was required. There were 14 duplicates found which were removed from the dataset. There were also 579 records that were either missing or had date and timestamps that did not align with those in the totaldemand_nsw.csv file. An attempt was made to reassign the inconsistent timestamps to the nearest 30-minute interval if one for that period did not already exist. However, this only reduced the number of incomplete records down to 564. Instead, it was decided to fill in the missing values by sourcing these temperatures elsewhere.

For dates between March 2016 and April 2018, hourly temperature recordings were collected for the Bankstown weather station (Station Number 66137) from datasets found at data.gov.au [9]. Between May 2010 and May 2011, there were sections of the dataset where more than 10 consecutive timestamps were missing. Additionally, they passed through time periods where the maximum or minimum temperature of the day normally occurred. Therefore, either a maximum or minimum temperature for these days was sourced from the Bureau of Meteorology [10] for the Bankstown weather station. These temperatures were then assigned to an estimated hour of the day where the maximum or minimum temperature was likely to have occurred based on when maximum or minimum temperatures occurred on the 3 proceeding and following 3 days. There was also missing timestamps for the 21st of May 2018 between 10:30 and 17:00. However, not even a maximum temperature could be sourced for this day. Therefore, an estimate for the maximum temperature and the time of day it occurred was made using the proceeding and following 3 days as a guide. These sourced temperatures and their associated timestamps were placed in a separate csv file and then merged with the original temperature dataset.

All remaining missing temperatures were filled using the resample and interpolate functions associated with panda dataframes in python. This assigned incremental and evenly spaced values ranging between the values of the previous and next existing temperatures that surrounded the consecutive missing values. Prior to the merging of the two datasets, the LOCATION column was dropped from the temperature dataset as all the values were the same and thus redundant.

Using python, the temperature dataset was also extended with new features using the DATETIME column as a base. First, separate columns were made for day, day of the week, month, year, hour and minute. Most columns maintained the same values as represented in the original date and timestamp. The only exception was the day of the week which used a numbering scale from 0 (Monday) to 6 (Sunday) to represent the days of the week. The values in the day of week column were then used to generate a Boolean column where 0 signifies a weekday and 1 represents a weekend.

Additional columns were also generated using the values in the HOUR and MONTH columns. The 0 to 23 hours in the HOUR column were broken up into 4 sectors where 0 (Midnight) to 5 represents early morning, 6 to 11 represents late morning, 12 (Midday) to 17 represents afternoon and 18 to 23 represents evening. These grouping were given values from 1 to 4 respectively. Using the MONTH column, a column called SEASON was also created with 1 representing summer, 2 representing autumn, 3 representing winter and 4 representing spring. The temperature dataset was then merged with the total demand dataset. After the transformed dataset was saved as a csv file, one additional column was added in Microsoft Excel called PUBLIC_HOLIDAY with a value of 1 given if the date was a public holiday and 0 otherwise. See Fig. 3.1 for the final processed dataset used in the modelling.

DATETIME	TEMPERATURE	DAY	DAY_OF_WEEK	MONTH	YEAR	HOUR	MINUTE	WEEKEND	PUBLIC_HOLIDAY	TIME_OF_DAY	SEASON	TOTALDEMAND
1/01/2010 0:00	23.1	1	4	1	2010	0	0	0	1	1	1	8038
1/01/2010 0:30	22.9	1	4	1	2010	0	30	0	1	1	1	7809.31
1/01/2010 1:00	22.6	1	4	1	2010	1	0	0	1	1	1	7483.69
1/01/2010 1:30	22.5	1	4	1	2010	1	30	0	1	1	1	7117.23
1/01/2010 2:00	22.5	1	4	1	2010	2	0	0	1	1	1	6812.03
1/01/2010 2:30	22.4	1	4	1	2010	2	30	0	1	1	1	6544.33
1/01/2010 3:00	22.3	1	4	1	2010	3	0	0	1	1	1	6377.32
1/01/2010 3:30	22.3	1	4	1	2010	3	30	0	1	1	1	6282.85
1/01/2010 4:00	22.1	1	4	1	2010	4	0	0	1	1	1	6211.49
1/01/2010 4:30	22.2	1	4	1	2010	4	30	0	1	1	1	6248.31

Figure 3.1: The first 10 records in the transformed dataset used for modelling of the neural networks.

3.4 Assumptions

It is assumed that the provided data was accurate and that the actual demand figures are correct. There may be errors in temperature recordings as evident by missing records for temperature. In addition to that, recorded temperatures at one location may not be a true representation for the entire region.

It is recognised that the Solar PV data influences the actual electricity demand. Furthermore, it is assumed that currently there is no mechanism to get this data to the granule of the region for each timestamp.

With these limitations, the assumption is that it the given data is sufficient to predict the forecast demand within a range 5-10% of the actual demand.

3.5 Modelling Methods

Before the dataset described in Chapter 3.3 could be used for modelling, transformations were required for some attributes. Columns with ordinal number values, including the columns DAY, DAY_OF_WEEK, MONTH, HOUR, TIME_OF_DAY and SEASON, were transformed using one-hot encoding. The dataset was then split into a training and test set. A minimum maximum scalar was then fitted for each of continuous values in the training set. This was performed on the TEMPERATURE and TOTALDEMAND columns where values were proportionally reassigned to values between 0 and 1. These fittings were then used to transform the TEMPERATURE and TOTALDEMAND columns in both the training and test set. From experimentation we limited the inputs to month and hour as categorical variables, identifying whether it is a weekday or weekend and identifying if the day was a public holiday.

The LSTM model has been designed to use the attributes of a designated number of immediately previous timestamps to the record that is either being trained or predicted. This number is set by a variable called 'time_steps'. A function was then created to transform both the X value for the training and test datasets into a three-dimensional matrix. The three dimensions represent the number of records in the dataset, the number of immediately prior records (time_steps) to the current record and the number of features used for these prior records. This therefore means that for each dataset the first number of records, equal to the value of the time_steps variable, cannot be trained or predicted by the model. This is because the number of previous records in the dataset for these records is less than the value of the time_steps variable. It should also be noted that the total demand values from the previous records were also included in the matrix of X values. The y value fed to the neural network is the total demand for the current record being either trained or predicted.

The neural network consists of a single layer bidirectional LSTM with 50 neurons which uses 'relu' for the activation function. This feeds into a single dense neuron. As part of the python tensorflow library, keras was used to compile the model. Mean square error was used to measure the performance of the model and Adam was used to optimise the model. When fitting the model, 30 epochs were used with a batch size of 32 with 0.1 (10%) of the training dataset set aside for validation. The values of the test dataset were then used to make predictions for the y values of the test dataset with an inverse transform then applied to restore these predictions back to the pre-scaled values. Root Mean Square Error (RMSE) was also used to determine how well the model was performing.

The test result was recorded along with the 'DATETIME' timestamp, temperature, actual and predicted demand in a .csv file for further analysis and visualisation by any appropriate tools such as Tableau.

The built model and the scalers of temperature and demand were saved so that the prediction program can work independently by loading this model and scalers. Various combinations of the attributes were used until the best combination was determined. The final columns used in the model were TEMPERATURE, WEEKEND, PUBLIC_HOLIDAY, MONTH (one-hot encoded), HOUR(one-hot encoded) and TOTALDEMAND.

3.6 Prediction Algorithm

Based on the model, the forecast temperature is an important factor for forecasting demand. The data for 'time_steps' (e.g., 24) number of records of actual demand prior to the forecast period is also required for the time series LSTM model to work. All other inputs can be prepared based on the 'DATETIME' timestamp.

Suppose the actual demand information available weekly is on Sunday midday, and the requirement is to provide a forecast of demand for the following week. With the forecast of temperature for that week, the input data will be built with the appropriate timeslot related input, temperature, with the actual demand field set to zero.

The last 24 timeslots of complete data records including actual demand will be joined in front of the forecast records. The prediction program loads the saved neural network model, the scalers for temperature and demand and then loads the entire dataset and converts it to the appropriate input format with the necessary scaling. As the month and hour data (if the forecast is only for a few hours and not for 24 hours or a week) will be limited, it can not use standard one hot encoding. So, the program builds this encoding programmatically.

Once the input is transformed, the algorithm picks the first 24 records, and predicts the demand for the 25th slot. While recording this information for output, the actual demand for the 25th record will be updated with this predicted demand value and the oldest record (1st record) will be dropped.

Now having complete records for the first 24 records again, the above process is repeated until all the necessary forecast demands having been performed. Practically the future forecast demand is based on the forecast, so it is possible to have the error build cumulatively. However, if the actual demand data is available say every hour, then the program can predict the forecast hourly with much accurate input. The forecast period also will be decided by the stakeholders.

The test result was recorded along with the 'DATETIME' time stamp, temperature, predicted demand in a .csv file for distribution to relevant stakeholders. The model can be re-trained every month or even daily to cover latest environmental and policy impacts as training will take less than 10 minutes on an average computer without the GPU support for 3 year data.

Chapter 4

Exploratory Data Analysis

4.1 Using Tableau

Year to year monthly temperature-demand trend

- Given temperature and demand data is from Jan'10 to Mar'21
- Monthly avg. temperature varies 2/3 degrees
- Demand is comparatively higher during winter season (Jun-Aug)
- Overall demand shows reduction in the recent years

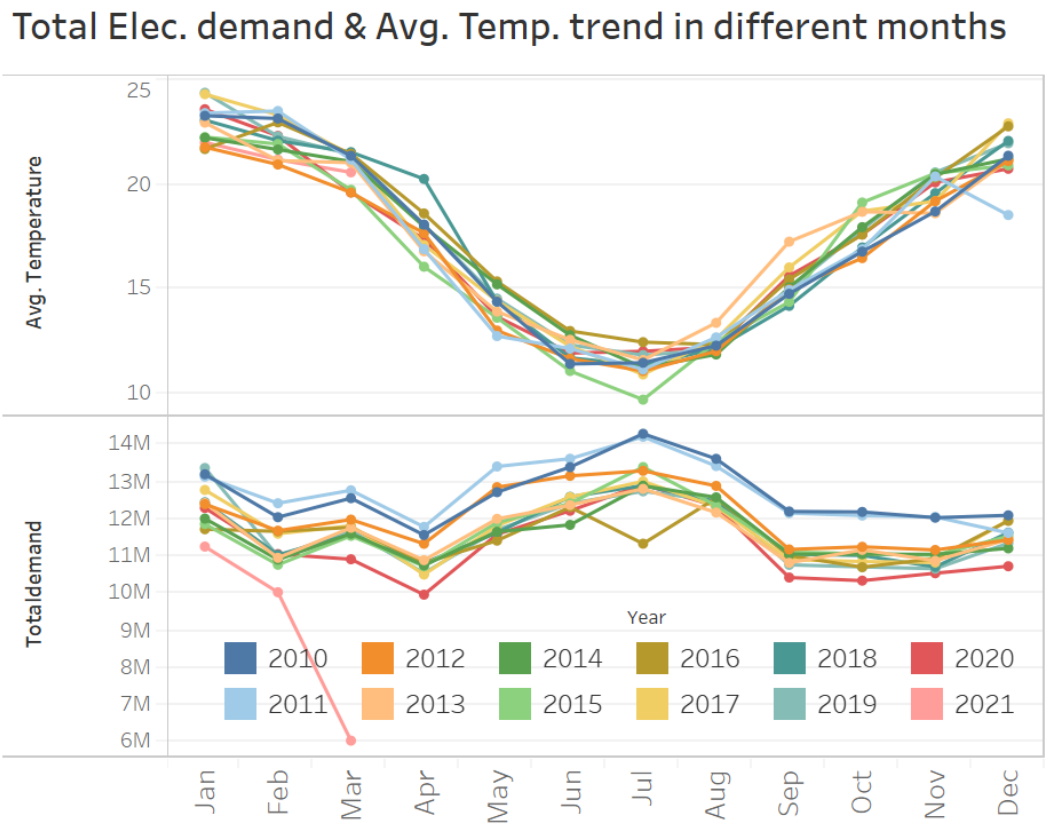


Figure 4.1: Total Electricity Demand and Avg. Temperature trend in different months

Monthly temperature-demand in diff. weekdays trend

- Different months have different demand pattern in different weeks days
- Year to year the pattern is not same

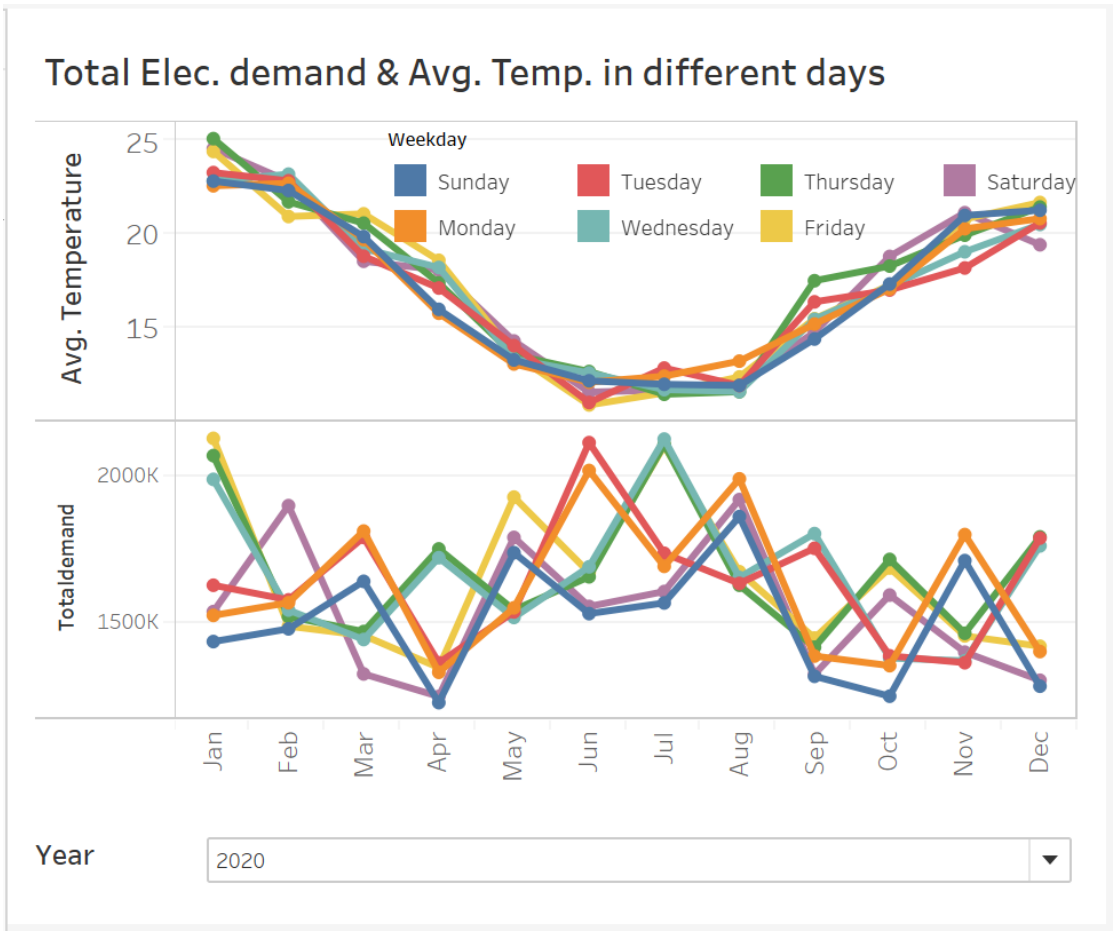


Figure 4.2: Total Electricity Demand and Avg. Temperature trend in different days

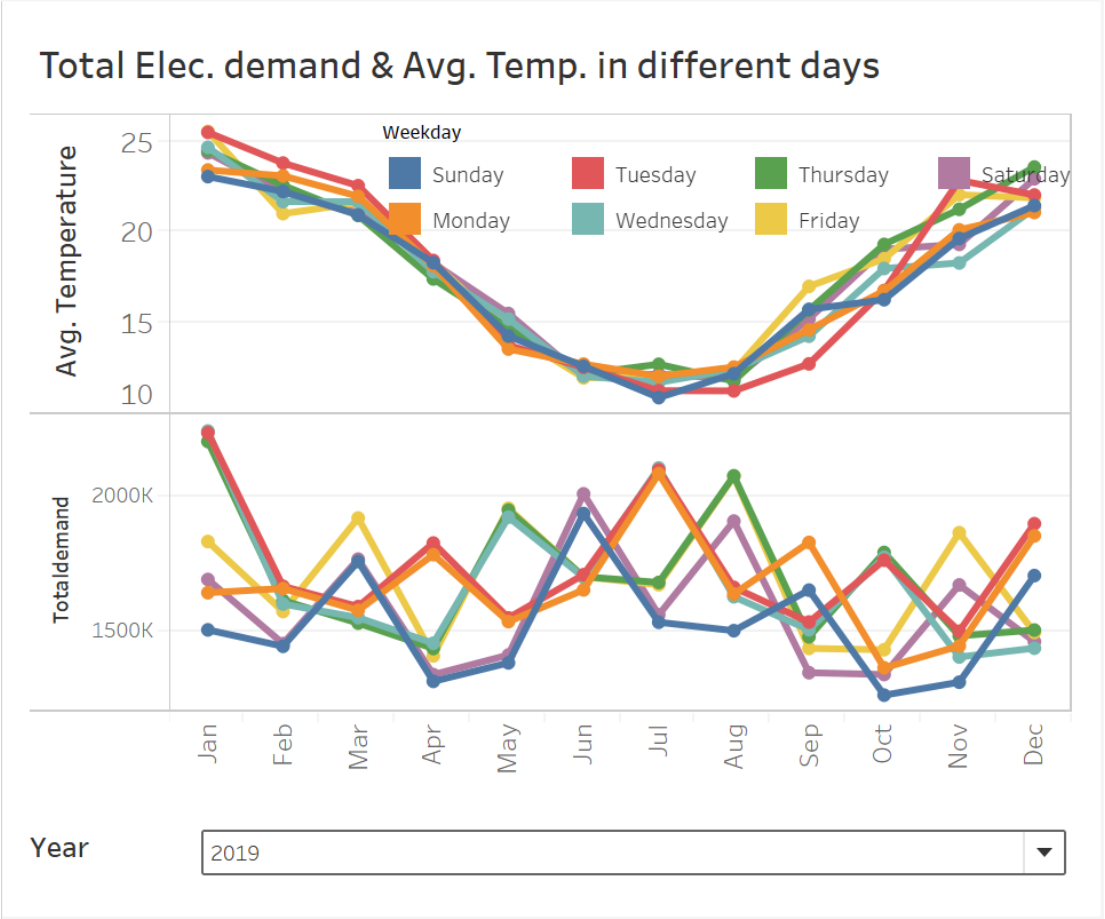


Figure 4.3: Total Electricity Demand and Avg. Temperature trend in different days

Hourly temperature-demand in diff. weekdays trend

- Demand somewhat follows temperature during off peak (night-time) time
- Demand variation during daytime varies in different days and not always proportional to temperature

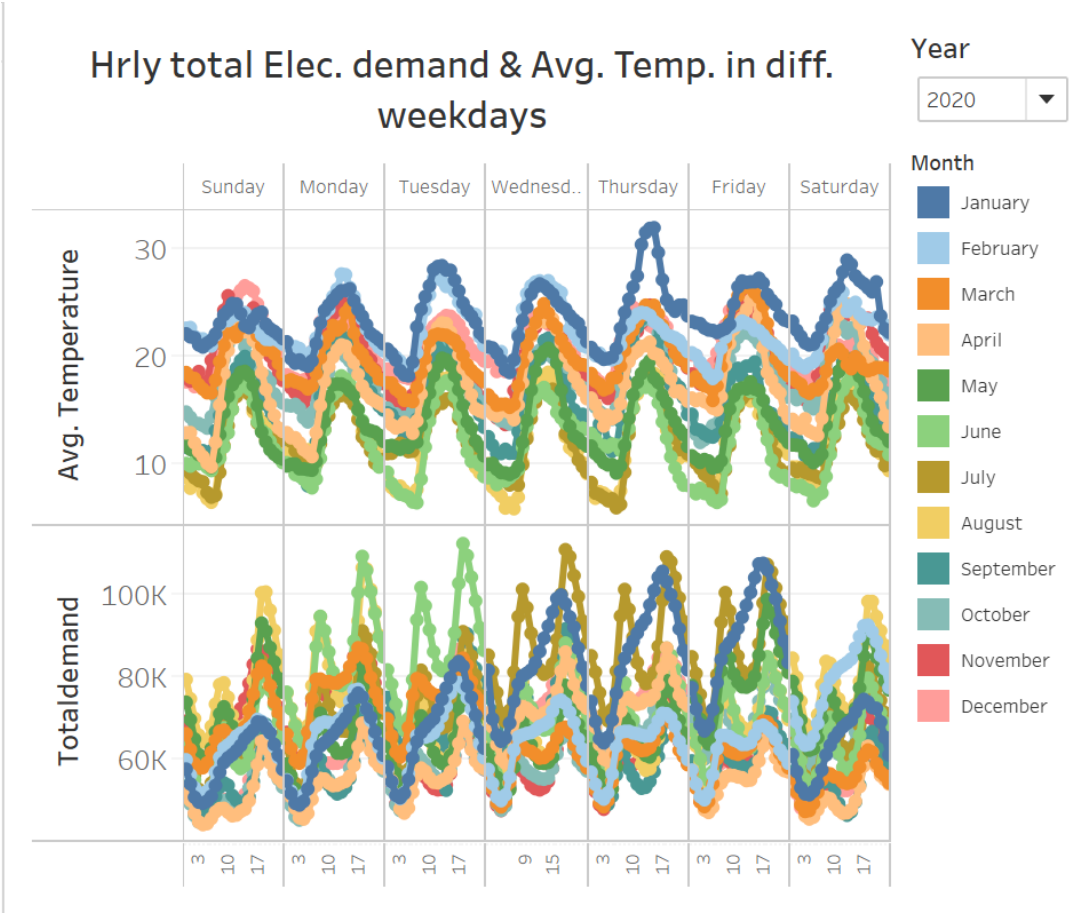


Figure 4.4: Hourly total Electricity Demand and Avg. Temperature in different weekdays

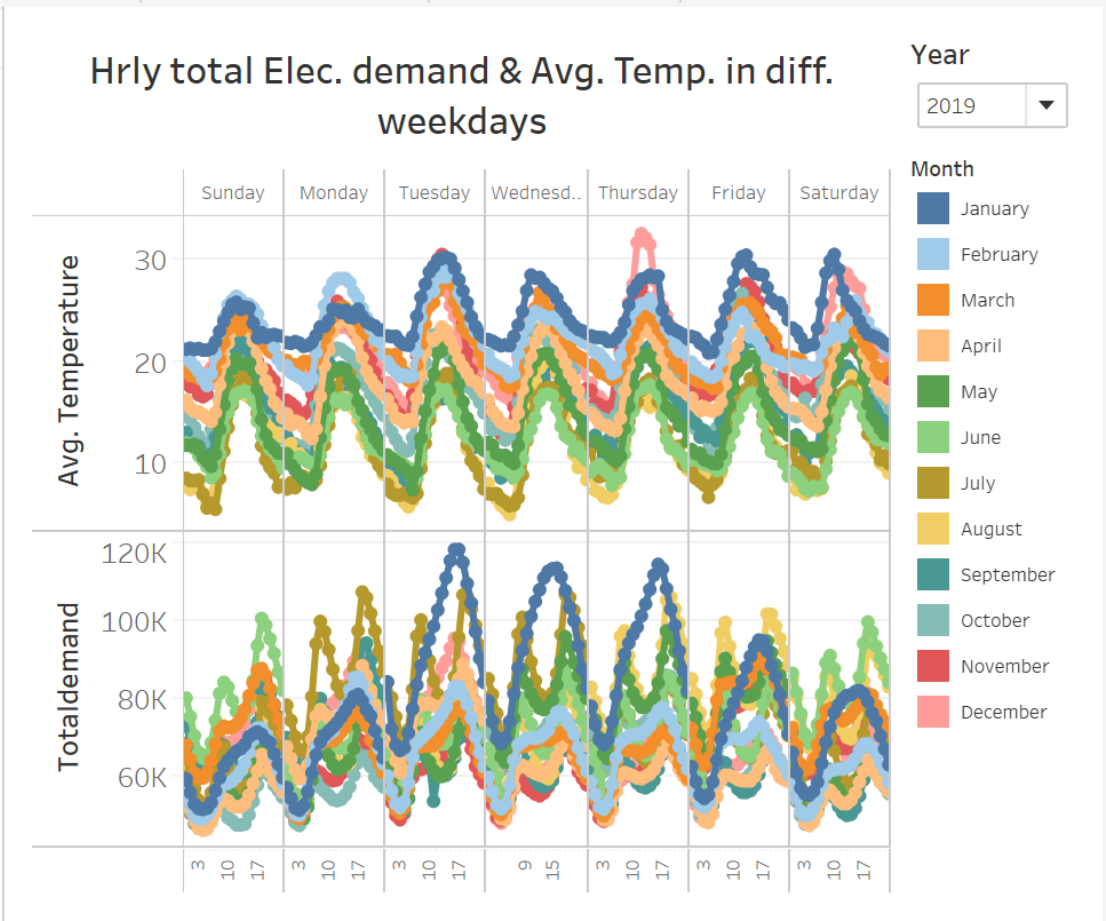


Figure 4.5: Hourly total Electricity Demand and Avg. Temperature in different weekdays

Hourly Forecast vs. Demand (Jan and Aug 2020 first 10 days)

- Current forecasts are pretty good in some hours and in some months
- Forecast suffers accuracy during daytime

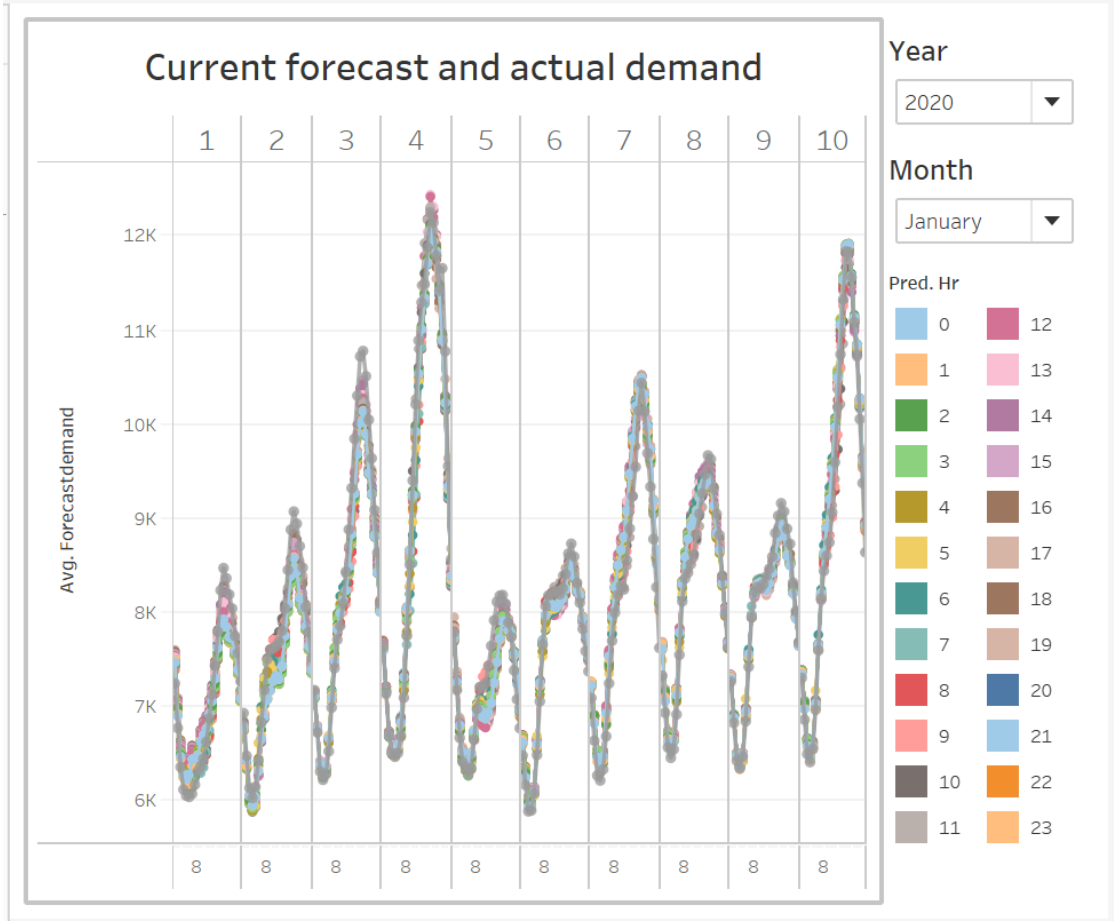


Figure 4.6: Current forecast and actual demand

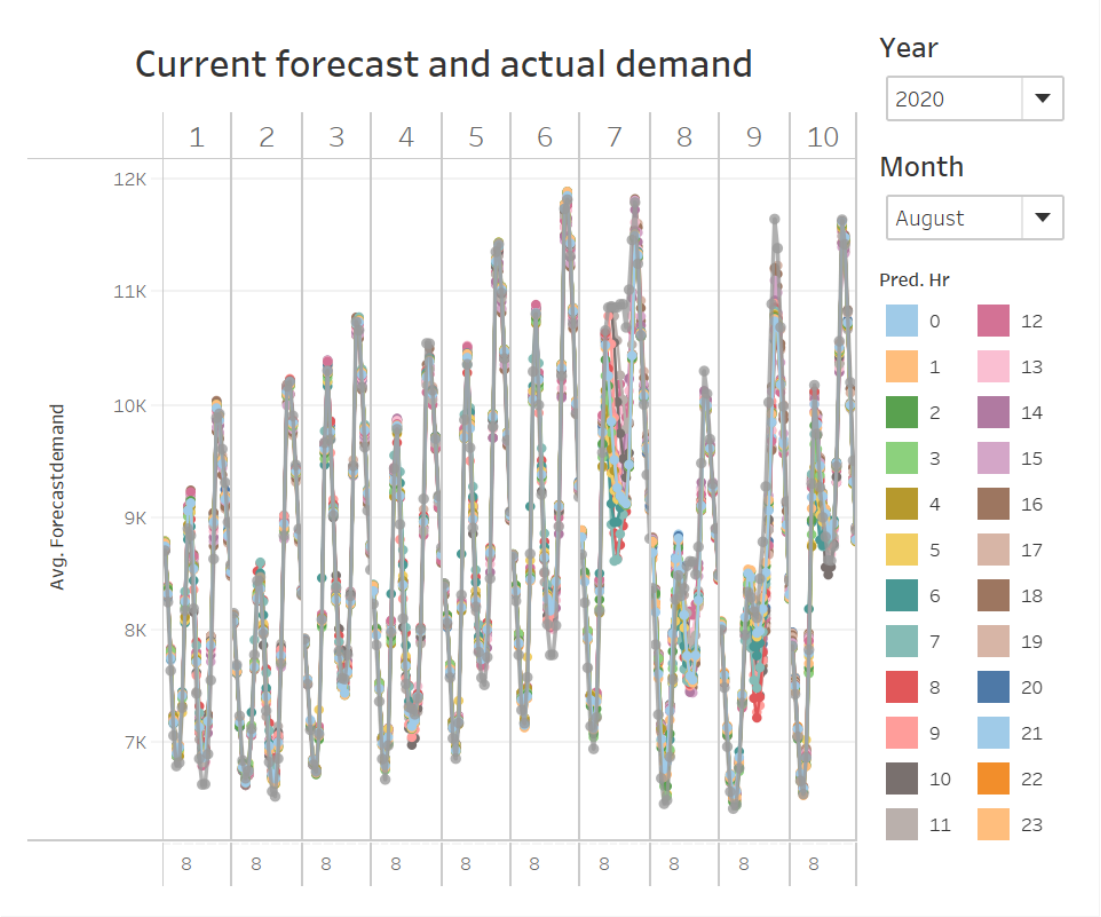


Figure 4.7: Current forecast and actual demand

2021 prediction results vs actual demand

- LSTM model output shows good performance during off peak time
- In some month (March) prediction seems very good

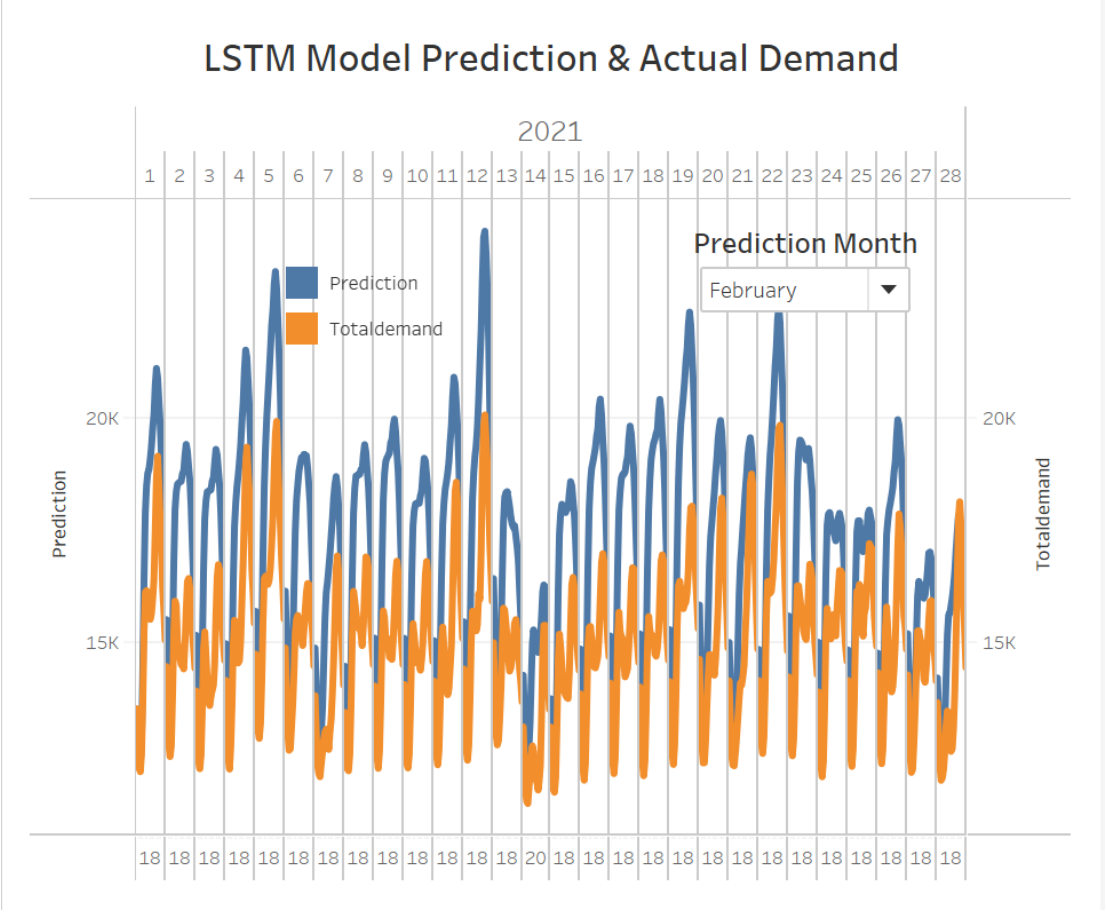


Figure 4.8: LSTM Model Prediction and Actual Demand

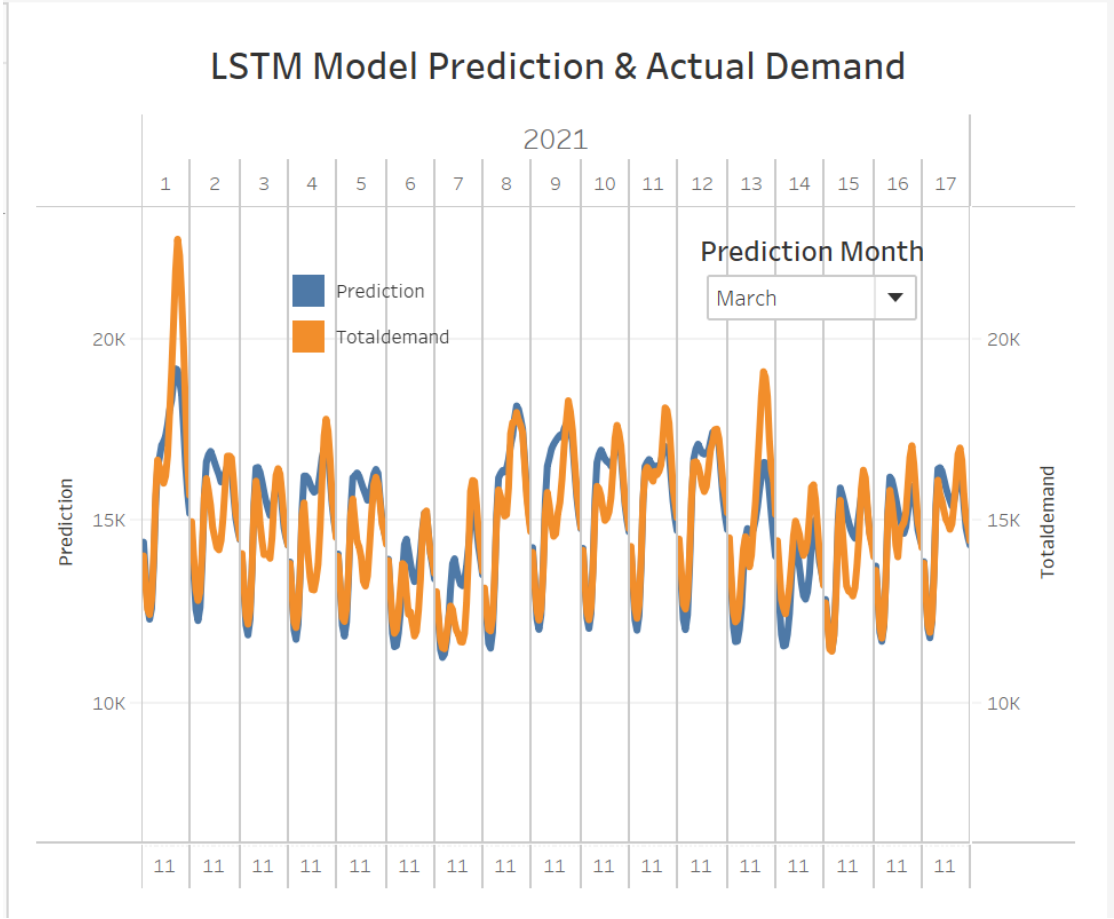


Figure 4.9: LSTM Model Prediction and Actual Demand

Chapter 5

Analysis and Results

5.1 A First Model

The categorical variables: month, day of the week and the 48 half-hour timestamps along with the temperature and actual demand were used to train the Multilayer Perceptron (MLP) model. With various tunings of hyper parameters with regards to dropout, learning rate, hidden layers and neurons, the best achieved Root Mean Square Error (RMSE) was about 450-500MW on both the train and test dataset. Based on this result, it was clearly indicating that the bias is high in the morning session timestamps, which could suggest that some features were missing and that the model didn't incorporate all of the variation in energy demand. One of these key missing features here is Solar PV, which is unable to be accurately provided for each region and timeslot at this stage.

The MLP model used 3 hidden layers of 200, 100 and 200 neurons with the final output layer, all using 'Relu' as the activation function. The ultimate model should be significantly better than the error associated with this MLP model, and the benchmark RMSE should be around 85MW as per the currently deployed model. The provided data is a time series which should hence produce a better result. Based on the literature review, a LSTM structure is the ideal model to work with.

5.2 LSTM Model – Only Using Previous Total Demand Values

The early LSTM structure, which happens to be a mono-directional model just uses the time series of actual demand with a lookback of 24 records, produced a significantly better result than the first model. The model trained with an RMSE of 100MW, which was superior to the 450-500MW RMSE achieved previously. Naturally, this model was picked for fine-tuning with appropriate feature inputs and hyper parameters.

5.3 Ulimate LSTM Model

The temperature is a key feature in the dataset - extreme temperatures require heavy cooling or heating (depending on where they are on the spectrum) which puts severe demand on electricity. Based on the time of day, as well as if it is a weekday or weekend, the electricity usage will differ. This implies various categorical inputs based on the timestamp play a bigger role. As described in the modelling methods chapter of the analysis, categorical variables span MONTH and HOUR along with weekday/weekend status and whether it is a public holiday or not. It was noted that

the temperature and demand scaled between 0 and 1 which improved the result. A BIDRIECTIONAL LSTM structure takes a bit more time to train but further improved the result.

The final model was produced with a 50 node bidirectional single LSTM with 'Relu' as activation function. This was trained with data from 2018-2020 with 30 epochs, a batch size of 32 and a lookback of 24 timestamps. Furthermore, a validation split of 10% and test set of 2021 produced an excellent result, with an RMSE less than 70MW (66.55MW) on test data as well as an average percentage difference of 0.66%.

Chapter 6

Discussion

The results presented in the previous chapter demonstrate that a bidirectional LSTM presents an effective approach to developing accurate forecasting models for energy demand. The bidirectional LSTM outperforms all other tested models, including a mono-directional LSTM and basic MLP, which do not appear to have the same forecasting abilities in this application with the time-series data with a limited amount of features.

The basic MLP model produced a mean square error of around 7 times higher than the final model, which perhaps speaks of its inability to work with this type of demand timeseries data. Part of this poor performance is the difference in how inputs were processed for the models - with the MLP treating each timestamp as a categorical variable. In the final LSTM model, the timestamps are treated as a sequence which may perhaps explain this reduction in error. It was also of note that the morning timestamps often had the biggest errors associated with them in the MLP model. Although it is difficult to theorise the variation here, a possible explanation for poor performance in the MLP model in this specific area is that energy demand dramatically picks up once people start to wake up and that the categories associated with the early morning timestamps have miscalculated weights in the network.

The LSTM model that was originally tested overcame this issue of treating the timestamps as categories - rather it took the timestamps in sequence which appeared to dramatically decrease the RMSE, approaching those benchmark error figures of 85MW. Most of the variation in energy demand was captured here and no consistent trends were found in the errors unlike in the MLP model. As was discovered in the literature review prior to the analysis, this was expected as LSTM networks are highly suited for classification and regression prediction problems, based on time series data - which this energy demand dataset is.

The final bidirectional model was far superior to all models tested, beats benchmark RMSE figures and gives sufficient evidence to the client that this can be used profitably to price energy. Due to the model being market-leading, it can both be used for a variety of reasons including for private investors, market efficiency or to aide governments in appropriate electricity distribution. The model works on the basis of fairly straightforward idea when compared to the mono-directional LSTM. The main difference here is that the first recurrent layer in the network is duplicated so that there are two layers alongside each other. Furthermore, the first layer

receives the input sequence as an input as-is, whilst the second layer is provided a reversed copy of the input sequence.

As mentioned previously, perhaps the addition of extra datasets (either Solar PV data or otherwise) with timestamps associated may improve the quality of predictions and reduce the model error even further. Although the bidirectional LSTM worked extremely well for this application, other network architectures may also be explored in future.

Chapter 7

Conclusion and Further Issues

The bi-directional Long short-term memory (LSTM) model performs well when predicting electricity demand for NSW, Australia. Using the last 24 timestamps, the pre-processed data is given as input which is then fed through the model and gives a Root Mean Square Error (RMSE) of 66.55MW and an average percentage error difference of 0.66% in the sample tested. This can be used profitably by the client through the appropriate pricing based on model outputs, with an appropriate profit margin to account for variance in the model. Although the bi-directional LSTM has worked well, there is no doubt that more data (i.e. Solar PV data or even an increased granularity in timestamps) could decrease the model error and make pricing more efficient for the client. Furthermore, other deep-learning architectures may be explored as well in the future, even though this model architecture worked successfully on this occasion. However, it is important to note, as was evident in this analysis, that inducing a more complex network structure to model the data does not always improve model accuracy.

References

[1] J. Catalão, S. Mariano, V. Mendes, L. Ferreira, Short-term electricity prices forecasting in a competitive market: A neural network approach, Electric Power Systems Research 77 (10) (2007) 1297–1304. doi:https://doi.org/10.1016/j.epsr.2006.09.022.
URL https://www.sciencedirect.com/science/article/pii/S0378779606002422

[2] Z. Mohamed, P. Bodger, Forecasting electricity consumption in new zealand using economic and demographic variables, Energy 30 (10) (2005) 1833–1843. doi:https://doi.org/10.1016/j.energy.2004.08.012.
URL https://www.sciencedirect.com/science/article/pii/S0360544204003639

[3] E. González-Romera, M. Jaramillo-Morán, D. Carmona-Fernández, Monthly electric energy demand forecasting with neural networks and fourier series, Energy Conversion and Management 49 (11) (2008) 3135–3142, special Issue 3rd International Conference on Thermal Engineering: Theory and Applications. doi:https://doi.org/10.1016/j.enconman.2008.06.004.
URL https://www.sciencedirect.com/science/article/pii/S0196890408002288

[4] G. Ciulla, A. D'Amico, Building energy performance forecasting: A multiple linear regression approach, Applied Energy 253 (2019) 113500. doi:https://doi.org/10.1016/j.apenergy.2019.113500.
URL https://www.sciencedirect.com/science/article/pii/S0306261919311742

[5] E. G. D. Carmona, M.A. Jaramillo, J. Alvarez, Electric energy demand forecasting with neural networks.
URL https://www.sciencedirect.com/science/article/abs/pii/0301421595001166

[6] R. Kumar, R. Aggarwal, J. Sharma, Energy analysis of a building using artificial neural network: A review, Energy and Buildings 65 (2013) 352–358. doi:10.1016/j.enbuild.2013.06.007.

[7] G. Xue, C. Qi, H. Li, X. Kong, J. Song, Heating load prediction based on attention long short term memory: A case study of xingtai, Energy 203 (2020) 117846. doi:10.1016/j.energy.2020.117846.

[8] D. L. Marino, K. Amarasinghe, M. Manic, Building energy load forecasting using deep neural networks (Oct 2016).
URL https://arxiv.org/abs/1610.09460

[9] Historical rainfall and temperature forecast and observations hourly data.
URL https://data.gov.au/data/dataset/

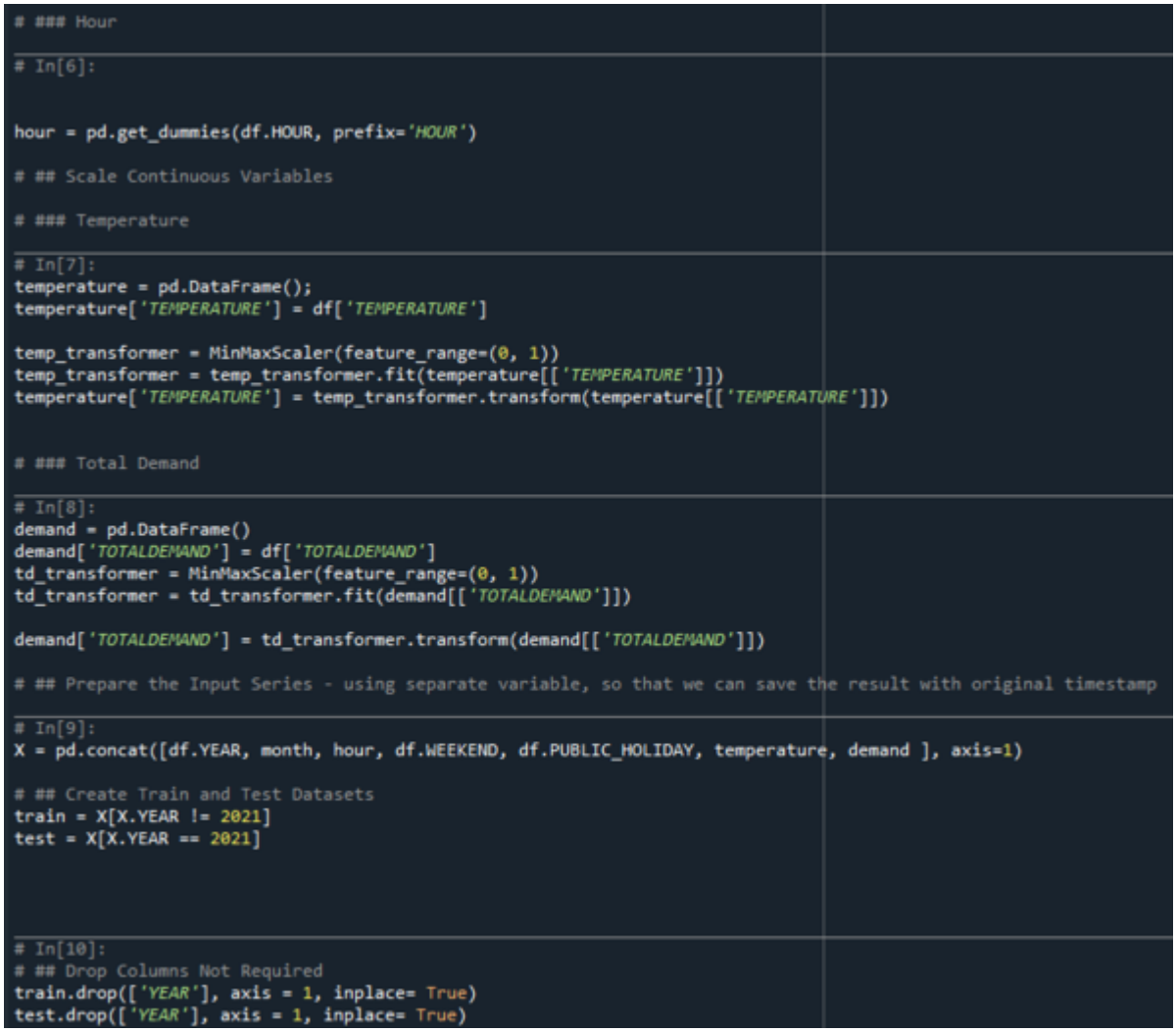
[10] Weather station directory.
URL http://www.bom.gov.au/climate/data/stations/

Appendix

Codes

```
# ## Load Packages
# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.layers import Dense, Bidirectional
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.optimizers import Adam

from pickle import dump


# ## Import Data
# In[2]:


df = pd.read_csv('../../data/merged_all_values_v2.csv')

# In[3]:


df.head()


# ## Take Subset Of Data To Only Include Years 2017, 2018, 2019 and 2020
# In[4]:


df = df[(df.YEAR == 2018) | (df.YEAR == 2019) | (df.YEAR == 2020) | (df.YEAR == 2021)]

# prepare the categoorical inputs with one hot encoding

# ## One Hot Encoding

# ### Month
# In[5]:


month = pd.get_dummies(df.MONTH, prefix='MONTH')
```

Figure 7.1: LSTM Model Build 1



```
# ### Hour

# In[6]:


hour = pd.get_dummies(df.HOUR, prefix='HOUR')

# ## Scale Continuous Variables

# ### Temperature

# In[7]:
temperature = pd.DataFrame();
temperature['TEMPERATURE'] = df['TEMPERATURE']

temp_transformer = MinMaxScaler(feature_range=(0, 1))
temp_transformer = temp_transformer.fit(temperature[['TEMPERATURE']])
temperature['TEMPERATURE'] = temp_transformer.transform(temperature[['TEMPERATURE']])


# ### Total Demand

# In[8]:
demand = pd.DataFrame()
demand['TOTALDEMAND'] = df['TOTALDEMAND']
td_transformer = MinMaxScaler(feature_range=(0, 1))
td_transformer = td_transformer.fit(demand[['TOTALDEMAND']])

demand['TOTALDEMAND'] = td_transformer.transform(demand[['TOTALDEMAND']])

# ## Prepare the Input Series - using separate variable, so that we can save the result with original timestamp

# In[9]:
X = pd.concat([df.YEAR, month, hour, df.WEEKEND, df.PUBLIC_HOLIDAY, temperature, demand ], axis=1)

# ## Create Train and Test Datasets
train = X[X.YEAR != 2021]
test = X[X.YEAR == 2021]



# In[10]:
# ## Drop Columns Not Required
train.drop(['YEAR'], axis = 1, inplace= True)
test.drop(['YEAR'], axis = 1, inplace= True)
```

Figure 7.2: LSTM Model Build 2

```
# ## Configure Lookback Period And Apply To Train And Test Datasets

# In[11]:


def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        v = X.iloc[i:(i + time_steps)].values
        Xs.append(v)
        ys.append(y.iloc[i + time_steps])
    return np.array(Xs), np.array(ys)


# In[12]:
print(test.head())

time_steps = 24

# Prepare The Train And Test Sets

X_train, y_train = create_dataset(train, train.TOTALDEMAND, time_steps)
X_test, y_test = create_dataset(test, test.TOTALDEMAND, time_steps)

print(X_train.shape, y_train.shape)



# ## Configure LSTM Neural Network

# In[13]:


model = Sequential()

model.add(
    Bidirectional(LSTM(units=50, input_shape=(
        X_train.shape[1],
        X_train.shape[2]),
         activation='relu')
         ))


model.add(Dense(1))
opt = Adam()
model.compile(loss='mean_squared_error', optimizer=opt)


# ## Fit Model To Train Dataset

# In[14]:

history = model.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.1, shuffle=True)
```

Figure 7.3: LSTM Model Build 3

```
# ## Plot Losses Per Epoch
```

```
# In[15]:


plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss Per Epoch')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()


# ## Make Predictions For Test And Train Dataset
```

```
# In[16]:


y_pred = model.predict(X_test)
y_pred_train = model.predict(X_train)


# ## Convert Values Back To Regular Values
```

```
# In[17]:


# Typically you would not scale y values. However, as the scaled total demand values were used to create both the x and y
# values, the scaling for the prediction and y test now needs to be reversed.
y_pred = td_transformer.inverse_transform(y_pred)
y_test = td_transformer.inverse_transform([y_test])

y_pred_train = td_transformer.inverse_transform(y_pred_train)
y_train = td_transformer.inverse_transform([y_train])


# ## Calculate RMSE

# ### Test Dataset
```

```
# In[18]:


y_test = y_test.reshape(-1)
y_test = pd.Series(y_test)

y_pred = y_pred.reshape(-1)
y_pred = pd.Series(y_pred)

rmse = mean_squared_error(y_test, y_pred, squared = False)

print('Test RMSE: %f' % rmse)
```

Figure 7.4: LSTM Model Build 4

```
# ### Train Dataset (for comparison with Test dataset)
# In[19]:


y_train = y_train.reshape(-1)
y_train = pd.Series(y_train)

y_pred_train = y_pred_train.reshape(-1)
y_pred_train = pd.Series(y_pred_train)

rmse = mean_squared_error(y_train, y_pred_train, squared = False)

print('Train RMSE: %f' % rmse)


# ## Average Percentage Difference Between Prediction And Actual Values
# In[20]:


predict_actual = y_test.to_frame()
predict_actual['PREDICTION'] = y_pred
predict_actual.columns = ['ACTUAL', 'PREDICTION']
predict_actual['PERCENT_DIFF'] = abs(predict_actual.ACTUAL - predict_actual.PREDICTION) / predict_actual.ACTUAL * 100


# In[21]:


avg_perc_diff = sum(predict_actual.PERCENT_DIFF) / len(predict_actual)
print('Average Percentage Difference', avg_perc_diff)

# ## Save The Result in a .csv with Timestamp, Temperature, Actual & Predicted Demand
# In[22]:


df = df[(df.YEAR == 2021)]

# reset the index as it is a filtered set
df.reset_index(drop=True, inplace=True)

# drop the records used for the first prediction
df.drop(range(0,time_steps), axis = 0, inplace=True)

# reset the index as we dropped records.
df.reset_index(drop=True, inplace=True)

print(predict_actual.head())
print(df.head())
KeyData = ['DATETIME', 'TEMPERATURE', 'TOTALDEMAND']
```

Figure 7.5: LSTM Model Build 5

```
df_out = pd.concat([df[KeyData], predict_actual['PREDICTION']], axis=1)
df_out['Percentage diff'] = 100 * (df_out['PREDICTION'] - df_out['TOTALDEMAND']) / df_out['TOTALDEMAND']
print(df_out.head())
df_out.to_csv('./output/LSTM_result.csv', index = False)

# ## Save The Model AND Scalers
# In[23]:

# ### Save The Modle
model.save('./output/LSTM_model.h5')  # creates a HDF5 file

# ### Save The Scalers For Demand AND Temporarture
dump(td_transformer, open('./output/scaler_td.pkl', 'wb'))
dump(temp_transformer, open('./output/scaler_temperature.pkl', 'wb'))
```

Figure 7.6: LSTM Model Build 6

```
# ## Import Libraries
import pandas as pd
import numpy as np

from tensorflow.keras.models import load_model
from pickle import load

# ## Load Model
model = load_model('./output/LSTM_model.h5')

# ## Load Scalers
# ### Save The Scalers For Demand AND Temporarture
td_transformer = load(open('./output/scaler_td.pkl', 'rb'))
temp_transformer = load(open('./output/scaler_temperature.pkl', 'rb'))


# ## Import Data
df = pd.read_csv('../../data/data_for_forecast.csv')

# prepare the categoorical inputs with one hot encoding
# ## One Hot Encoding
# We have to do this manually, as the required forecast will not cover all the months,
# and possibly not all the time slots.


# ### Month & Hour

records = len(df)

month_col = ['MONTH_1', 'MONTH_2', 'MONTH_3', 'MONTH_4', 'MONTH_5', 'MONTH_6', 'MONTH_7',
                'MONTH_8', 'MONTH_9', 'MONTH_10', 'MONTH_11', 'MONTH_12']

month = pd.DataFrame(np.zeros((records, len(month_col)), dtype=int), columns = month_col)


hour_col =['HOUR_0', 'HOUR_1', 'HOUR_2', 'HOUR_3', 'HOUR_4', 'HOUR_5', 'HOUR_6', 'HOUR_7',
           'HOUR_8', 'HOUR_9', 'HOUR_10', 'HOUR_11','HOUR_12','HOUR_13','HOUR_14','HOUR_15',
           'HOUR_16','HOUR_17','HOUR_18','HOUR_19', 'HOUR_20', 'HOUR_21','HOUR_22','HOUR_23']

hour = pd.DataFrame(np.zeros((records, len(hour_col)), dtype=int), columns = hour_col)

# fill in appropriately
for i in range(records):
    m = df.at[i,'MONTH']
    month.iat[i, m-1] = 1
    h = df.at[i, 'HOUR']
    hour.iat[i, h] = 1
```

Figure 7.7: LSTM Forecast 1

```
# ## Scale Continuous Variables

# ### Temperature
temperature = pd.DataFrame();
temperature['TEMPERATURE'] = df['TEMPERATURE']
temperature['TEMPERATURE'] = temp_transformer.transform(temperature[['TEMPERATURE']])

# ### Total Demand
demand = pd.DataFrame()
demand['TOTALDEMAND'] = df['TOTALDEMAND']
demand['TOTALDEMAND'] = td_transformer.transform(demand[['TOTALDEMAND']])

# ## Build Required Data
X = pd.concat([month, hour, df.WEEKEND, df.PUBLIC_HOLIDAY, temperature, demand ], axis=1)

# ## Configure Lookback Period And Apply To Dataset

def create_dataset(X, time_steps=1):
    Xs = []
    v = X.iloc[0:time_steps].values
    Xs.append(v)

    return np.array(Xs)

# ##Algorithm - Prediction one at a time

#
# Pass the time_steps number of records to predict the next forecast.
# Update the forecasted demand as actual demand for the relevant record.
# Drop the first record, and send the next time_steps number of records.
# Continue this until all prediction done.

#
# Precondition, that the original data file feeded should have time_steps number of records
# with actual demand
#

# ### Predict the demand one at a time

time_steps = 24

# collect all the prediction
df_pred = pd.DataFrame()
df_pred = pd.DataFrame(np.zeros((records-time_steps, 1)), columns = ['PREDICTION'])
```

Figure 7.8: LSTM Forecast 2

```
for i in range(records - time_steps):
    X_batch = X.head(time_steps)
    X_test  = create_dataset(X_batch, time_steps)

    # predict
    y_pred = model.predict(X_test)
    df_pred.at[i,'PREDICTION']= y_pred

    #
    # Update total demand with the predicted value
    # And drop the oldest record for next prediction
    #
    X.at[time_steps,'TOTALDEMAND'] = y_pred
    X = X.tail(-1)
    X.reset_index(drop=True, inplace=True)

print(df_pred.head())

# ## Save The Prediction.

# In[10]:



# reset the index as it is a filtered set
df.reset_index(drop=True, inplace=True)

# drop the records used for the first prediction
df.drop(range(0,time_steps), axis = 0, inplace=True)

# reset the index as we dropped records.
df.reset_index(drop=True, inplace=True)
KeyData = ['DATETIME', 'TEMPERATURE']

# rescale the forecasted demand
y_pred = td_transformer.inverse_transform(df_pred)
y_pred = y_pred.reshape(-1)
y_pred = pd.Series(y_pred)
y_pred = y_pred.to_frame()
y_pred.columns =['PREDICTION']

df_out = pd.concat([df[KeyData], y_pred], axis=1)

print(df_out.tail())
df_out.to_csv('./output/LSTM_predictions.csv', index = False)
```

Figure 7.9: LSTM Forecast 3

Model Results

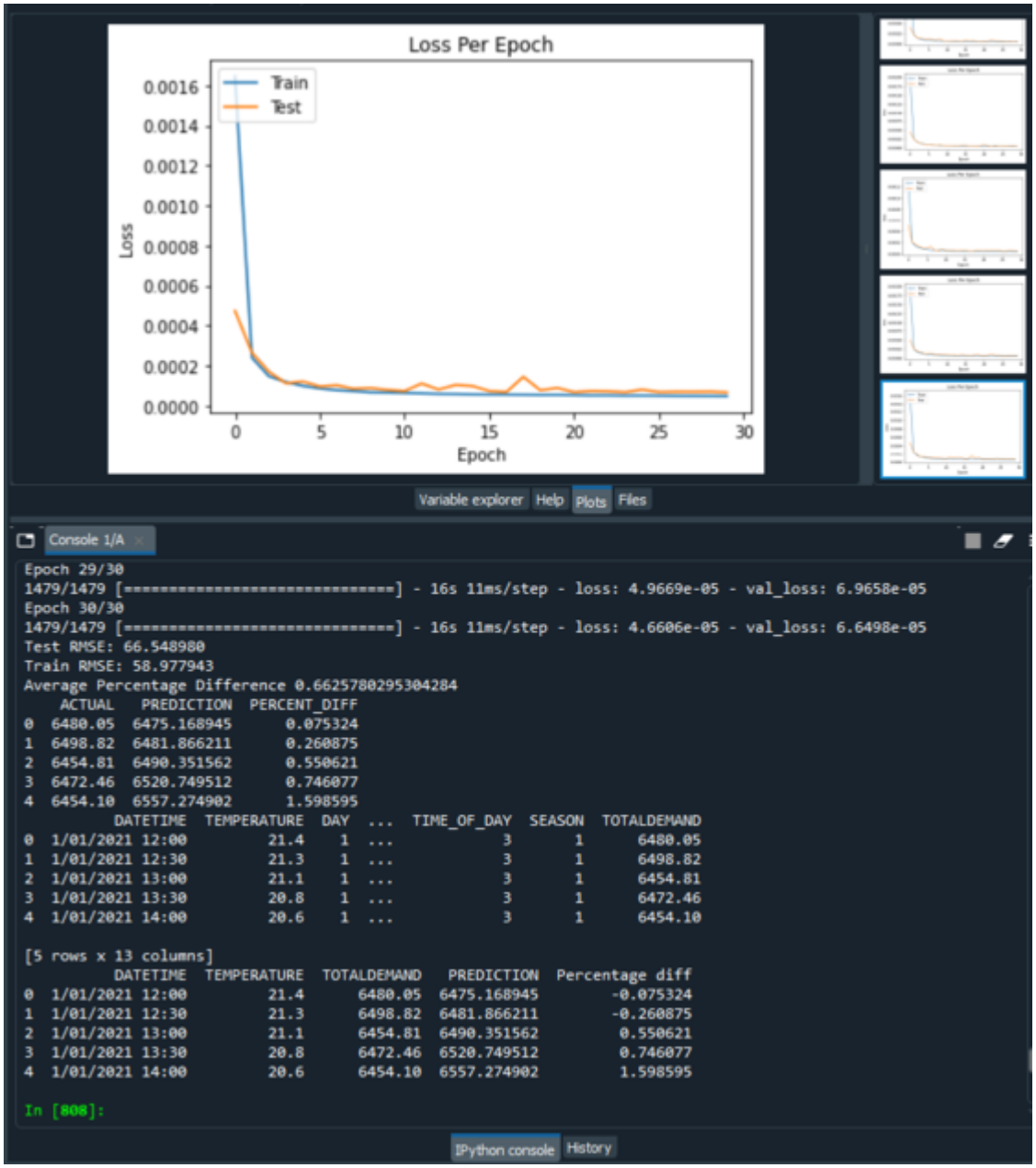


Figure 7.10: LSTM Model Test Result