

Contents:

1. Chapter-1: Abstract, Goal of project and Introduction.
2. Chapter-2: Dataset Description.
3. Chapter-3: Cleaning the Data Set and Removing Outliers.
4. Chapter-4: Creating Light GBM as Baseline Model.
5. Chapter-5: Baseline model using LSTM and Improvements.
6. Chapter-6: Improvements MLP based Neural Network as Best Model.
7. Chapter-7: Conclusion and Kaggle Submission

Chapter-1

1.1 Abstract:

Modern day eco friendly smart buildings practice emission control and substantial development methodology, under which they use optimized infrastructure. ASHRAE - Great Energy Predictor III is a project which aims to provide accurate energy consumption predictions by various metered buildings in order to compare these predicted values with actual energy consumption units of modern day retrofitted buildings. This project helps a building owner to follow pay for usage model under which he pays according to the difference in energy consumption units generated by his retrofitted infrastructure and my predicted values which are trained on a non retrofitted infrastructure.

In this project a series of models are discussed and then the best model is selected which has lowest validation error, various data cleaning, visualization, training and error minimization techniques are discussed throughout this report.

1.2 Goal of the project:

Goal of this project is to predict energy consumption values in kilowatt hour (kWh) based on data collected from various metered buildings for chilled water consumption, electricity consumption, natural gas consumption, hot water consumption, and steam consumption.

1.3 Introduction:

In modern world of globalization it is important follow sustainable practices particularly in case of controlling emissions and saving power. ASHRAE - Great Energy Predictor III is one such machine learning and neural networks based modern day algorithm that predicts energy consumed by various metered buildings according to previous consumption model.

This project report is subsequently divided into 3 sections, in 1st part I have taken the data set given by Jason Zivkovic[1], cleaned and analyzed it with various pre processing, visualization algorithms. This data is typically collected from more than 1,000 buildings for a time span of three years. In the 2nd part I have simulated a baseline model that gives a substantial error. Whereas in 3rd section I have made drastic changes in the base model in order to minimize the error percentage.

Chapter-2: Dataset Description.

2.1 Loading the dataset:

Raw data set can be downloaded from kaggle given by Jason Zivkovic[1], I would like to mention that data processing images are used from [1], it is typically collected from more than 1,000 metered buildings for a time span of three years, data set is given in 5 different csv files.

2.2 Description of the ASHRAE fashion dataset:

Train.csv

It consists of meter readings, timestamp, meter id code, it should be noted that not all rows have valid data, there are few rows which have missing data. The meter id can be further described as '0' for electricity, '1' for chilled water, '2' for steam, '3' for hot water. All energy units are given in kWh (kilo watt hour).

Building_meta.csv:

This file details about information available for various buildings based on energy star property type definitions like total square feet, overall floor area, building completion year, total floor count.

Weather_[train/test].csv:

This file contains weather data from a meteorological station, various parameters like air temperature, cloud coverage, dew temperature, sea level pressure, wind directions, wind speed are given in this file.

Test.csv:

This file has no feature data but it is used to get the predictions in correct order.

2.3 Describing the dataset through Visualizing:

Given data set is a collection of 5 distributed data sets as mentioned above, therefore before visualizing and understanding the data it is important to combine it into one single file while maintaining the same labels throughout this procedure. Additionally it is important to remove skewed values, nan and 0 values from the data set.

Therefore I have combined train, building and weather train datasets together and will perform cleaning operations on this set itself. The joined data frame has 20,216,100 rows and 22 columns. Consider figure 2.3.1 which shows missing variables from the combined data set.

##	building_id	meter	timestamp
##	0	0	0
##	meter_reading	site_id	primary_use
##	0	0	0
##	square_feet	year_built	floor_count
##	0	12127645	16709167
##	air_temperature	cloud_coverage	dew_temperature
##	96658	8825365	100140
##	precip_depth_1_hr	sea_level_pressure	wind_direction
##	3749023	1231669	1449048
##	wind_speed	timestamp_date	timestamp_month
##	143676	0	0
##	timestamp_day	timestamp_day_number	time_ymd_hms
##	0	0	0
##	time_hour		
##	0		

Figure-2.3.1: List of all 22 columns with missing variables.

Part of these missing variables will be interpolated and the remaining will be deleted from the combined data set so that bad and incomplete data should not impact during training. Figure-2.3.2 shows the skewed values present in this data set, which will increase the training and validation error due to their out of scope values.

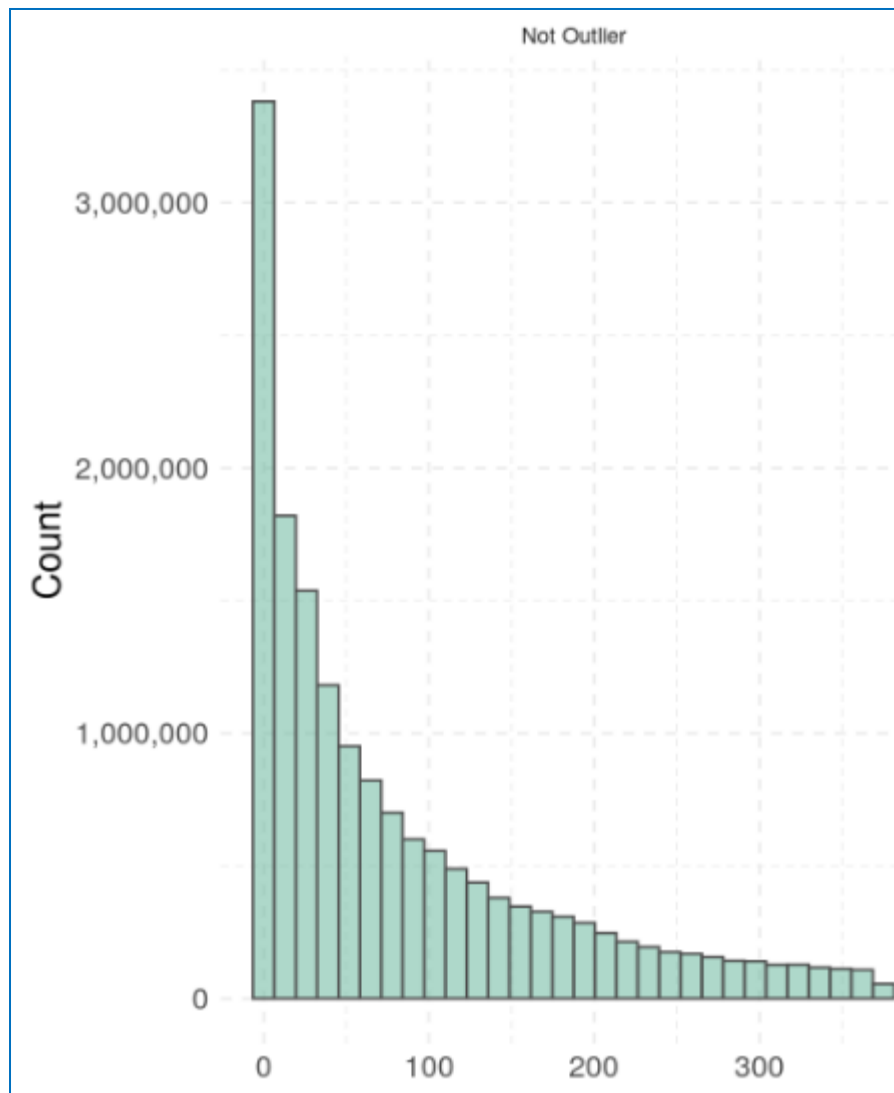


Figure-2.3.2: Skewed values present in data set ranging very high above the average.

The skewed values shown in figure above will be removed, this figure helps us in finding the outliers, I will be taking a log of entire range so that out of average values do not spoil the accuracy of my model. Consider figure-2.3.3 which confirms the range of these skew values.

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0	18	79	2117	268	21904700

Figure-2.3.3: Confirms the range of outlying points is way beyond the average values.

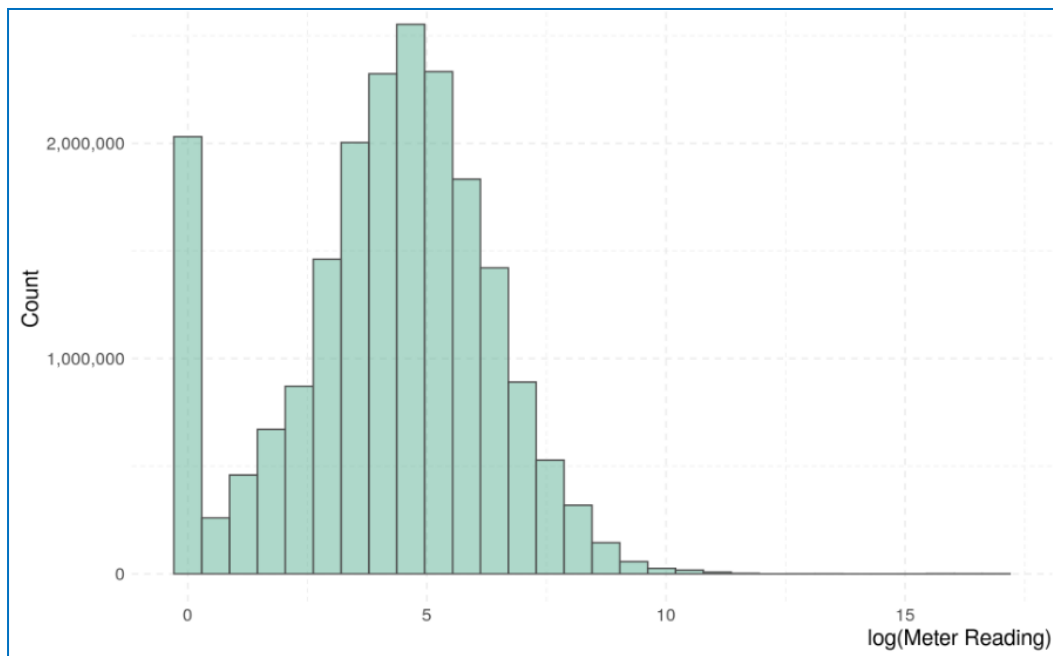


Figure-2.3.4: Scaled down values of meter_reading variable for removing outliers.

In order to scale down the values and make it more readable under a smaller range I have used a log function on this combined data set, refer figure-2.3.4 given below. The out of order values are removed whereas log function on meter_reading variables gives a tight range for better understanding of the values.

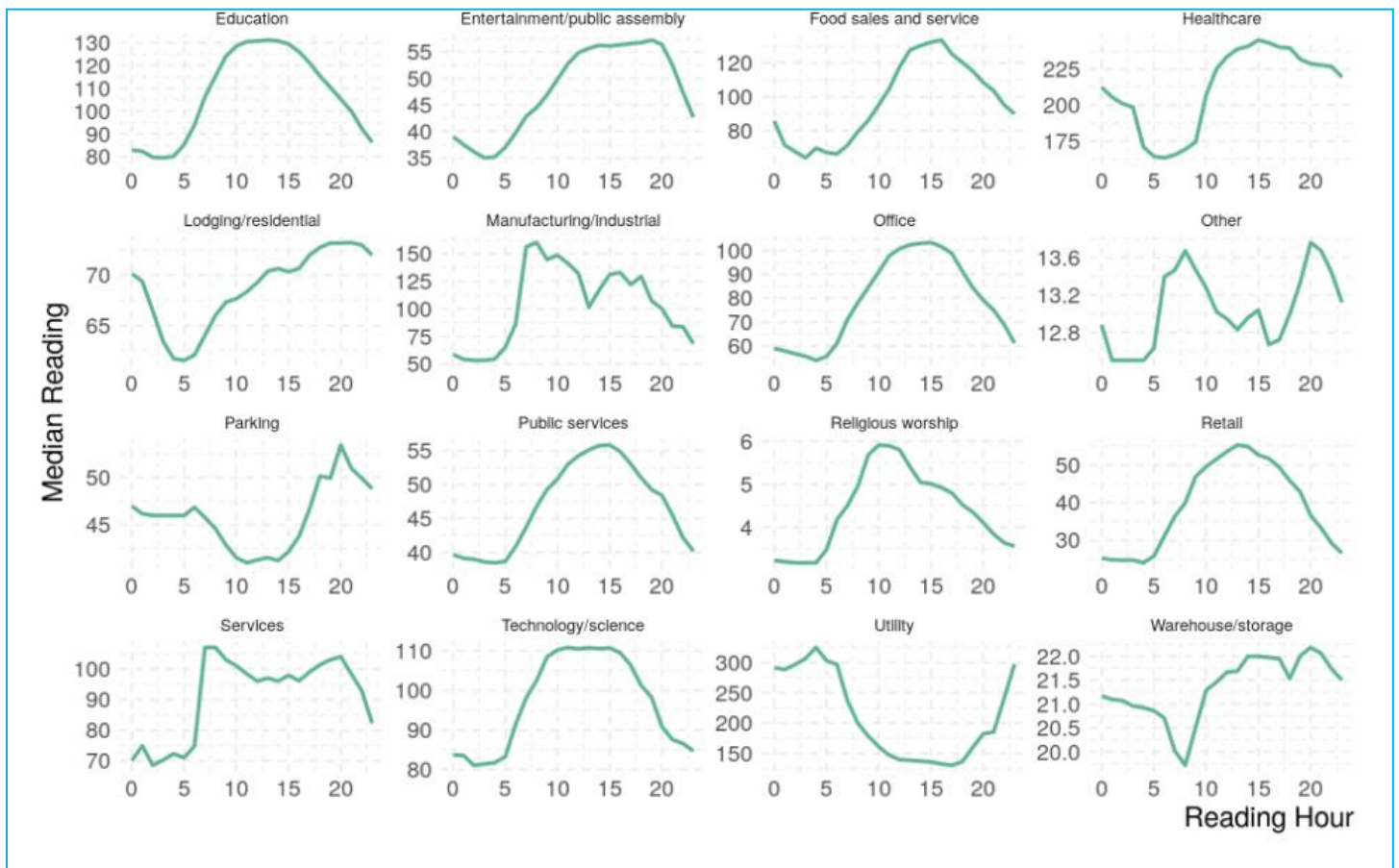


Figure-2.3.5: Energy consumption of different buildings throughout the day.

Consider figure-2.3.5 which shows hurly distribution of meter readings, it is very relevant that residential and warehouse buildings achieve maximum values in later part of the day, whereas manufacturing plants typically achieve maximum energy consumption during early hours. This figure helps us in finding maximum energy consumption hours and lowest energy consumption hours. Additionally it shows that utility and service based building consume maximum energy during early hours of the day.

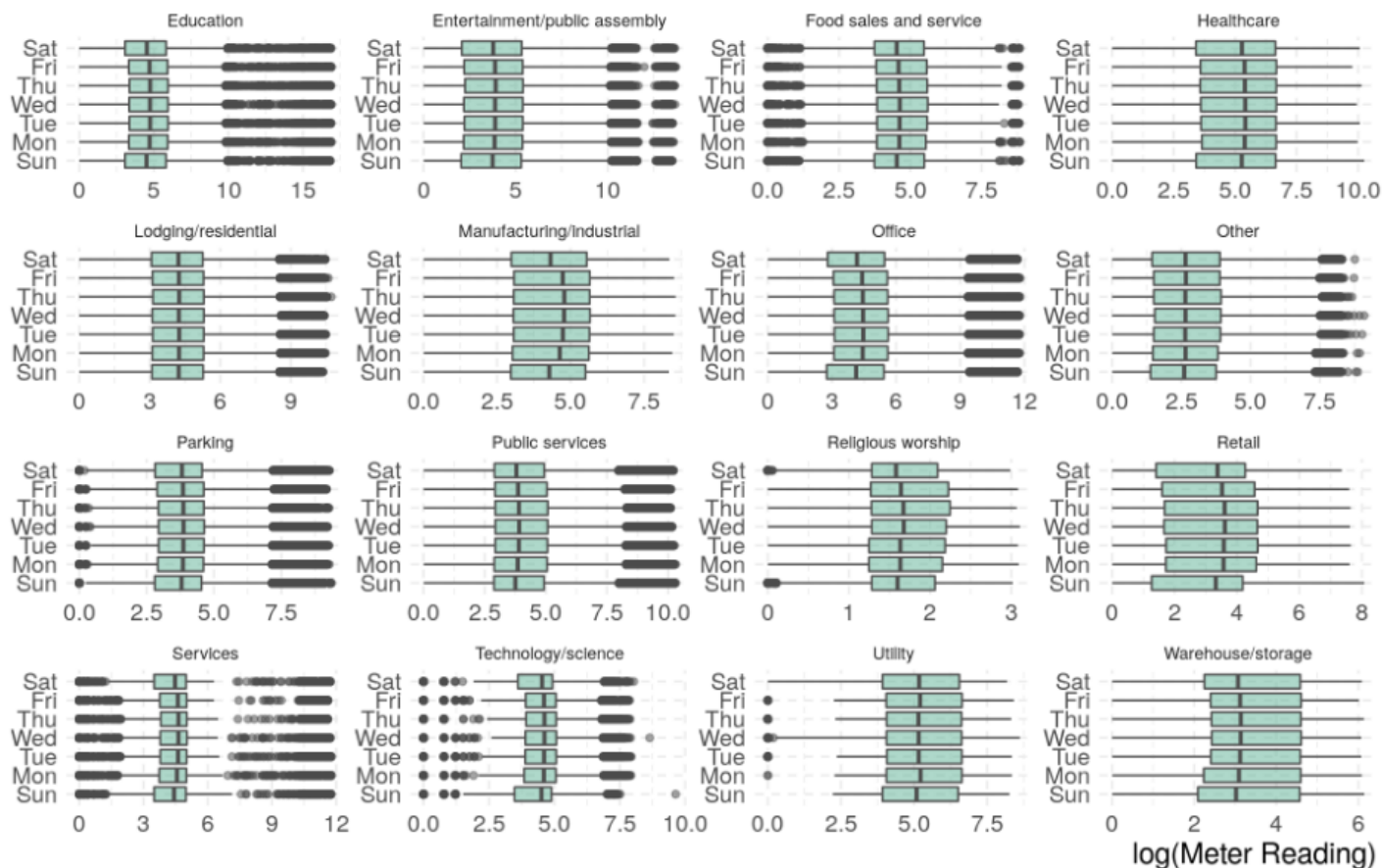


Figure-2.3.6: Energy consumption throughout the week.

Refer to figure-2.3.6 that shows energy consumption units for an entire week, it is clear that places related to religion or places of worship have very energy consumption during weekends, whereas industrial places have higher energy consumption on weekdays.

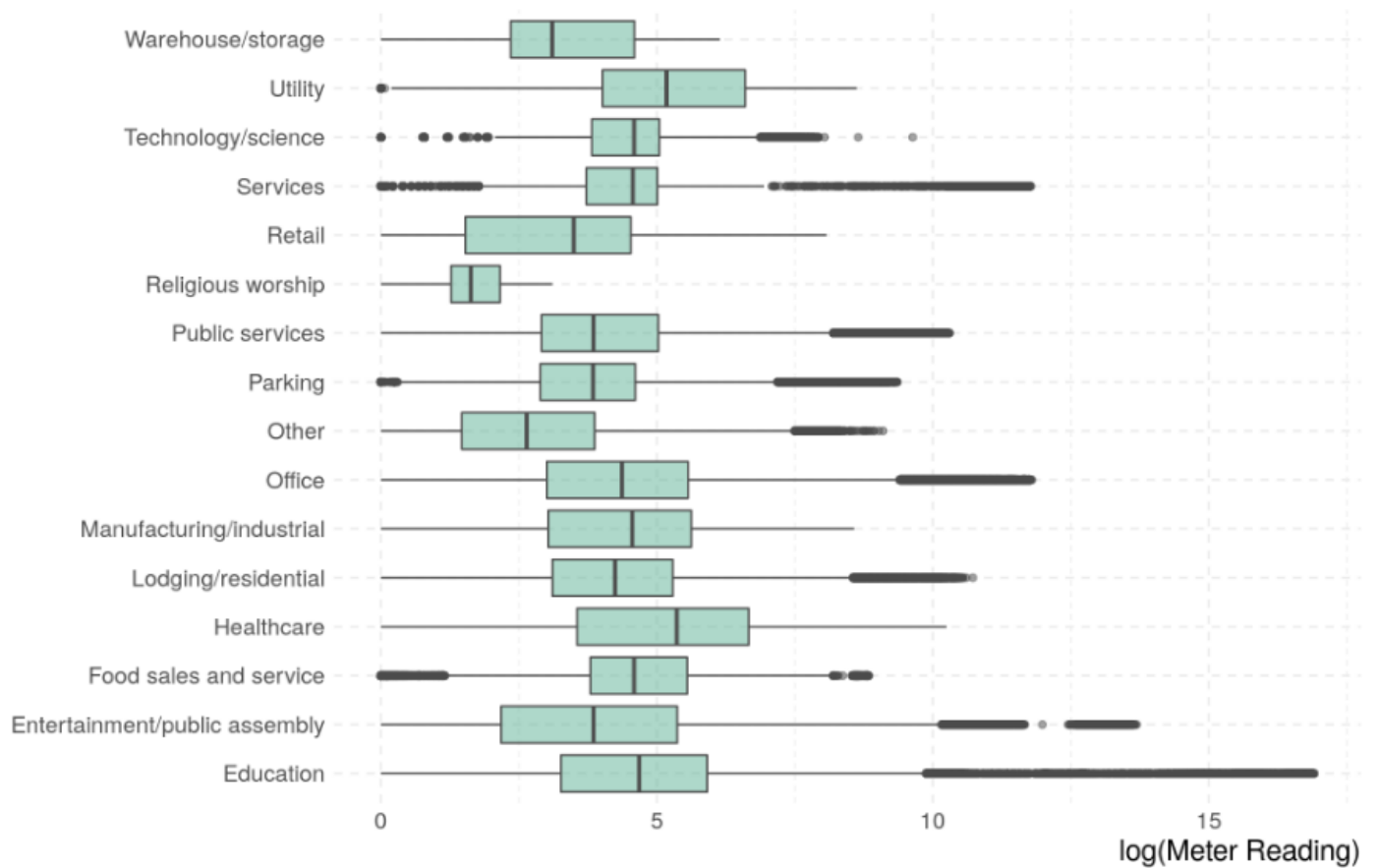


Figure-2.3.7: Energy consumed by various building types on log scale.

Figure-2.3.7 shows energy consumed by various buildings, where healthcare sector and utilities industry have maximum consumption values, this means that maximum optimization and retrofitting is required in these building. Whereas worship places have minimum energy consumption values, similarly educational installations have an average consumption value.

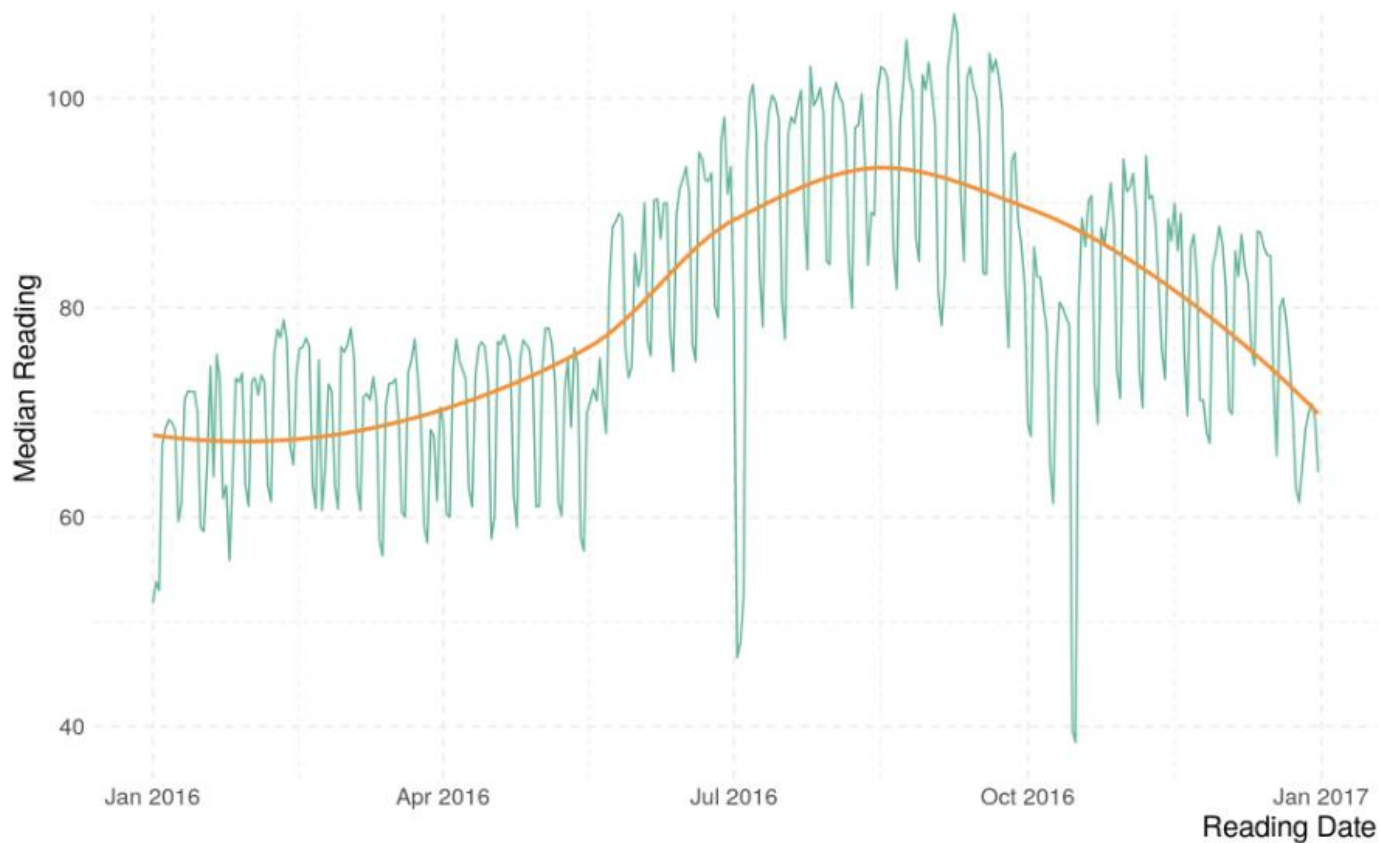


Figure-2.3.8: Median meter reading per reading plotted against number of months.

It is evident from figure-2.3.8 that sudden consumption in median energy values increases between months of July and October, whereas during other part of the year values are mostly around the mean.

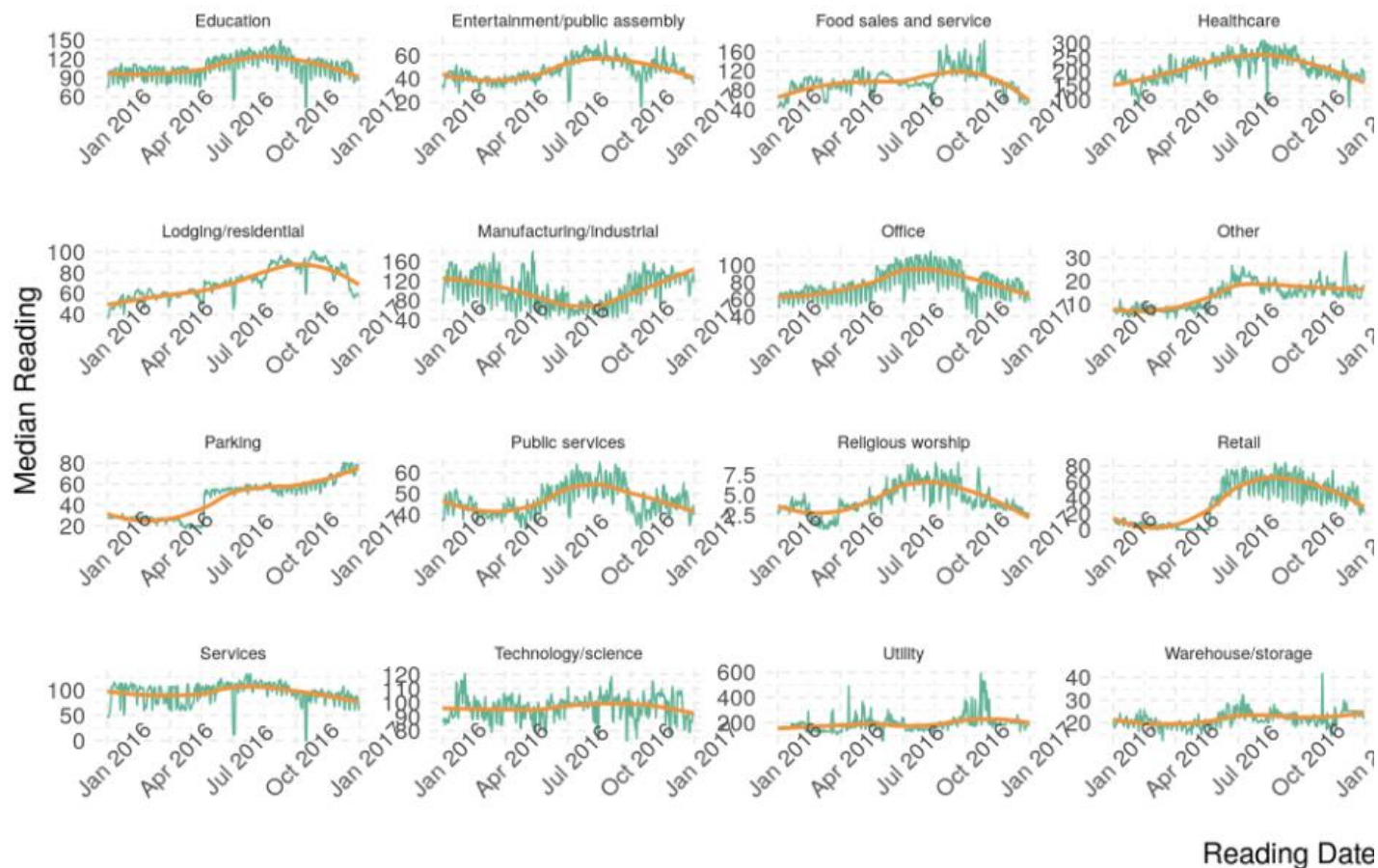


Figure-2.3.9: Meter readings of every type of building over plotted against months.

Consider figure-2.3.9 which shows median reading of all the building types over different months, mostly energy consumption rises between July and October whereas only for manufacturing industries it drops in the same duration.

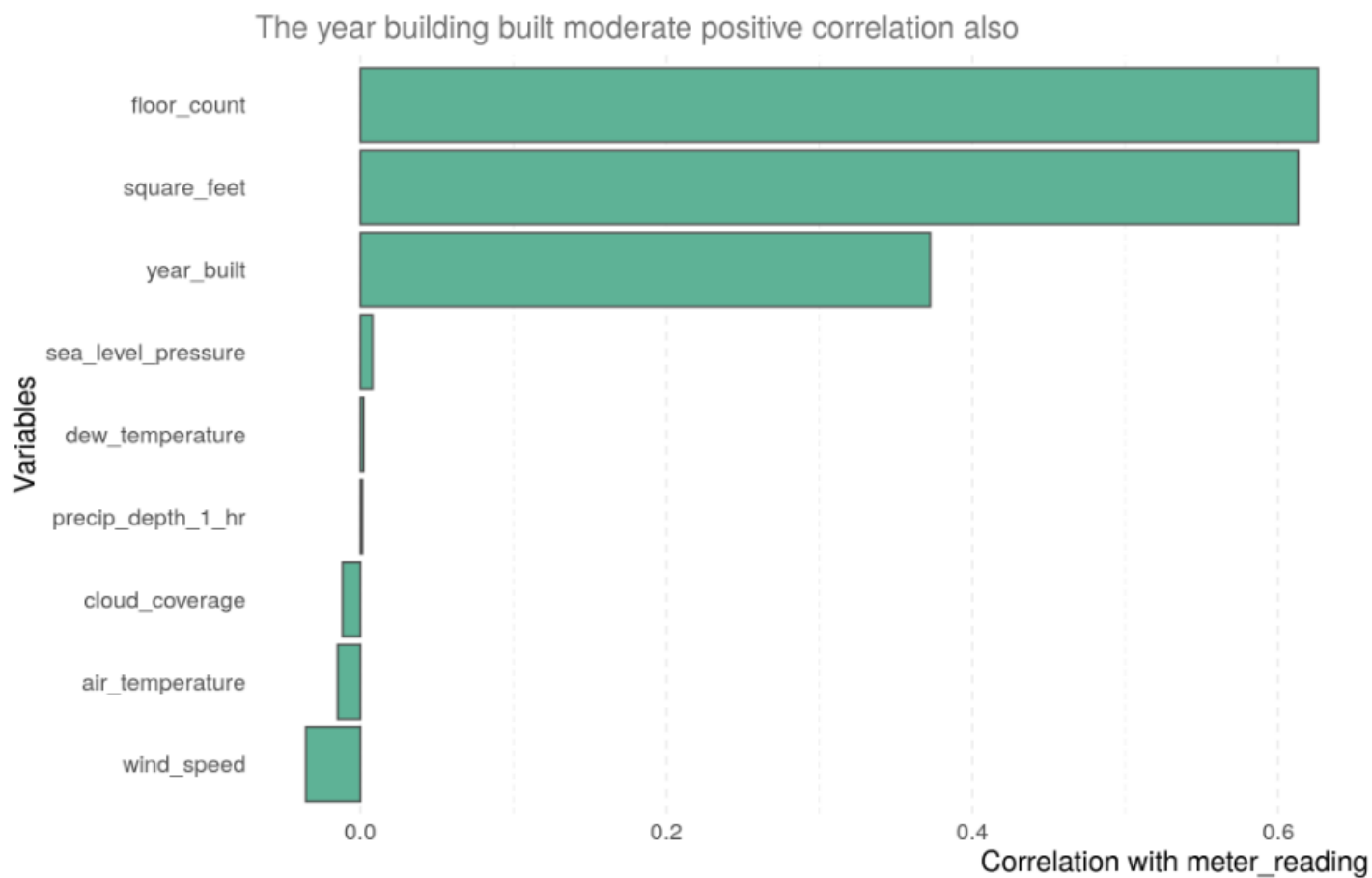


Figure-2.3.10: Correlation between all the parameters given in the data set.

Figure-2.3.10 plots the correlation between all variables given the data set, it is very evident that total floor count with respect to square feet area are highly correlated, where as dew temperature and precipitation depth not correlated.

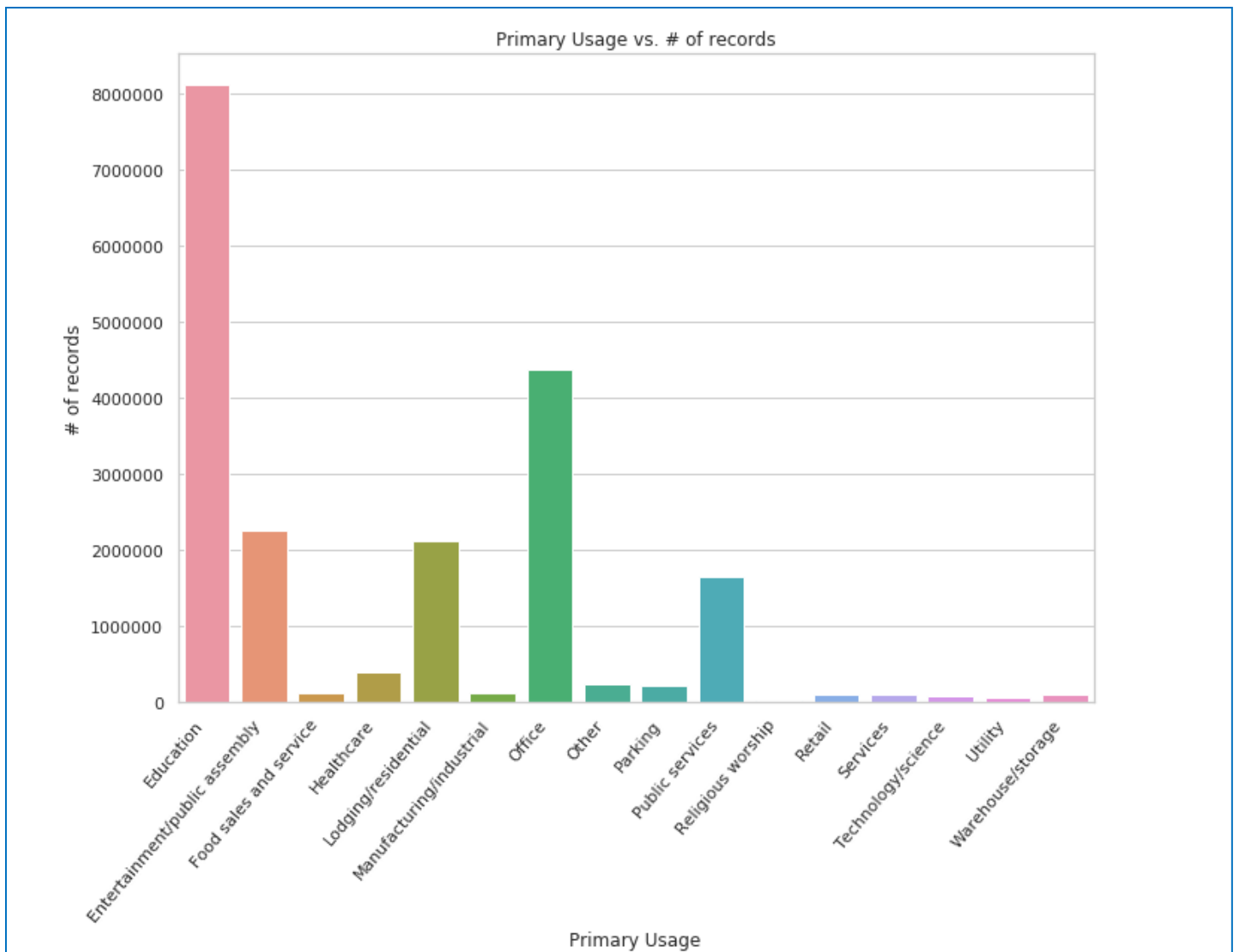


Figure-2.3.11: Total number records of each building vs building type occurring in the data set.

Figure-2.3.11 helps me to understand which building records typically dominates the data set, it is very clear that educational installations have maximum number of records in the data set, followed by office, public assembly, residential and public service buildings.

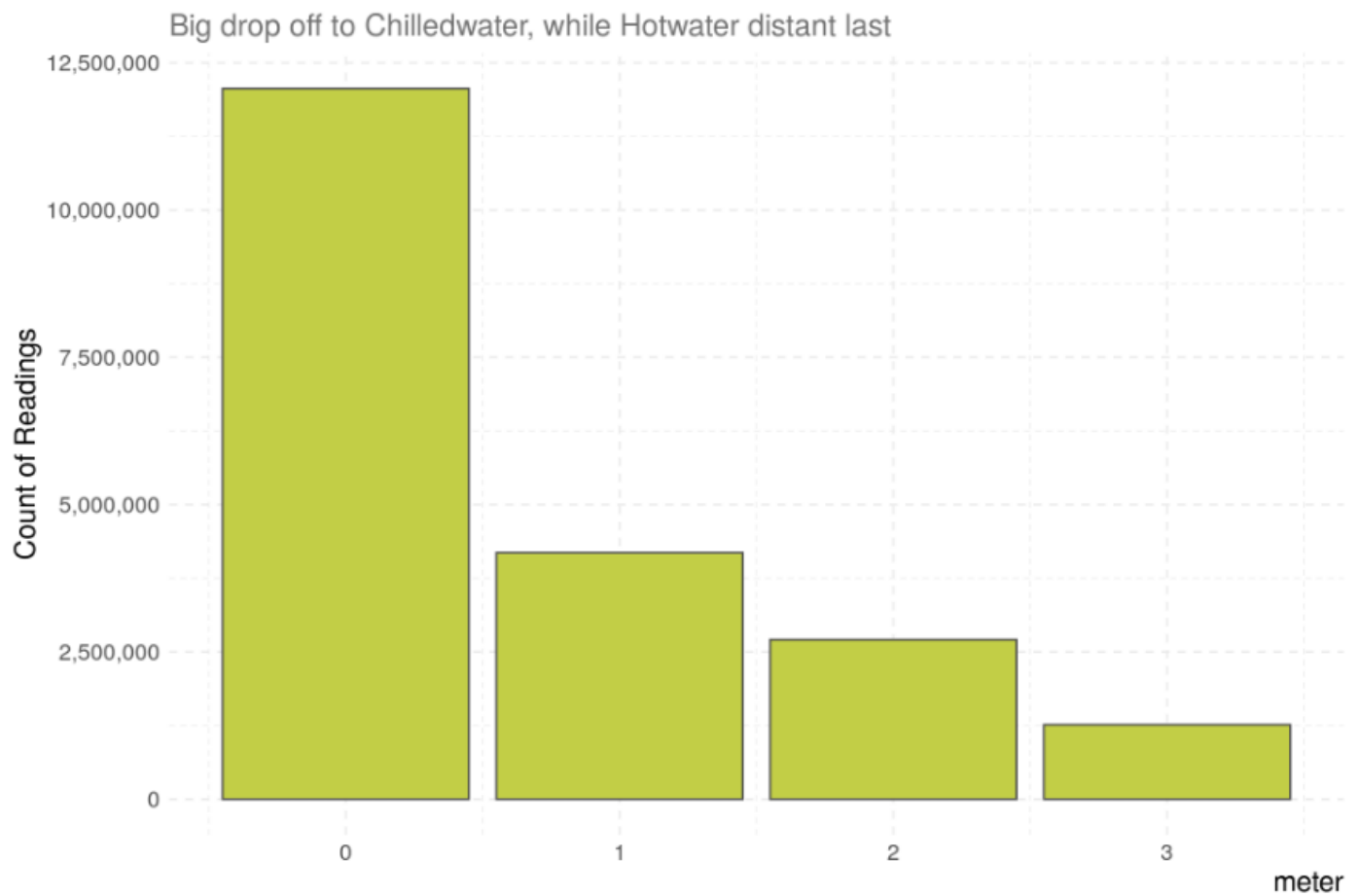


Figure-2.3.12: Frequency of meter type used in the data set.

Figure-2.3.12 shows electricity consumption is very high for all the buildings, whereas hot water consumption is very less. Electricity consumption values range over 11 million, which is 2 times more than steam consumption.

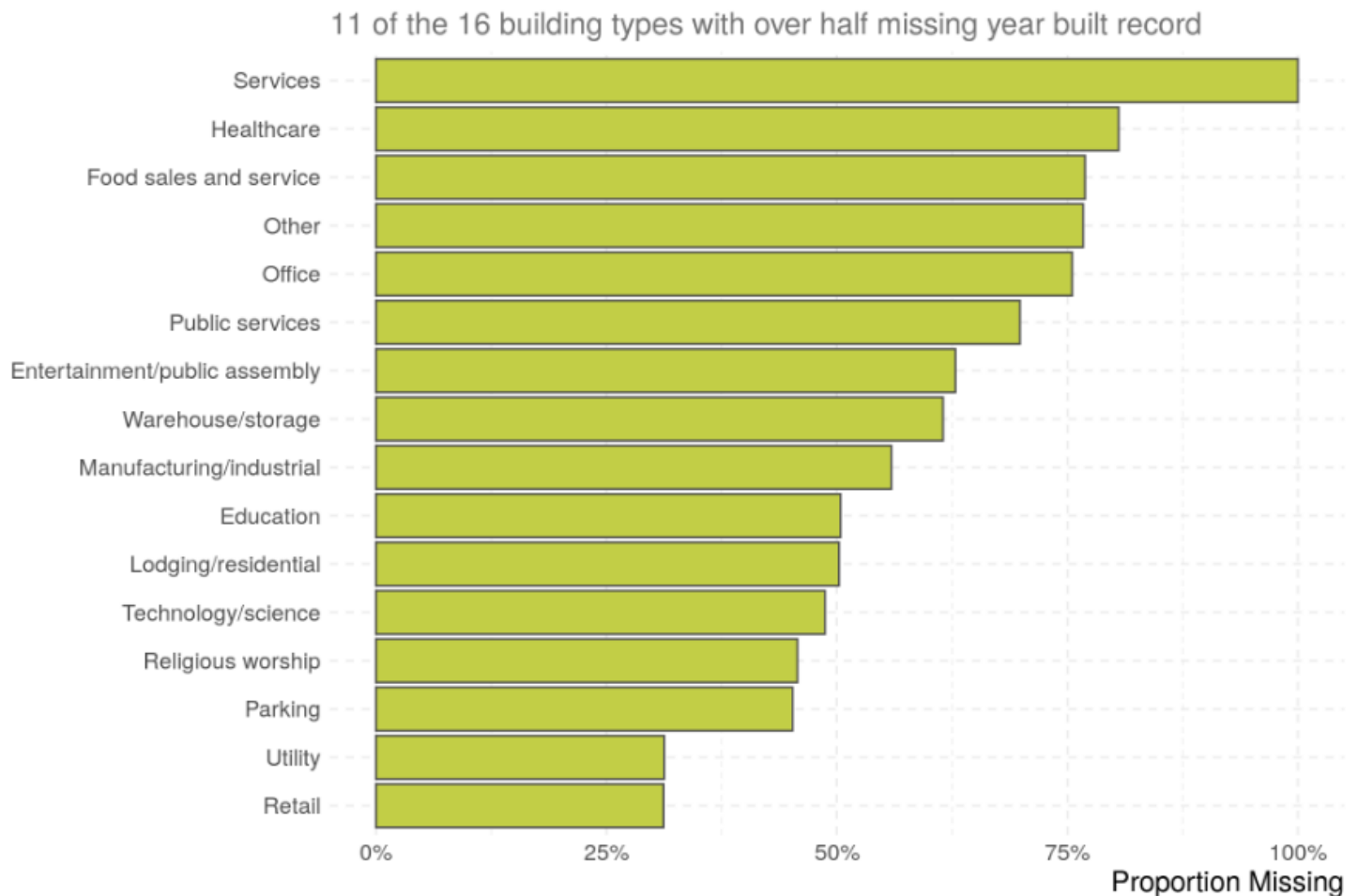


Figure-2.3.13: No available records for building completion year.

Figure-2.3.13 shows that services buildings have no record of completion of their built year, whereas mostly all retail buildings present in the data set have this record. A total of 11 buildings out of total 16 given buildings do not have this record.

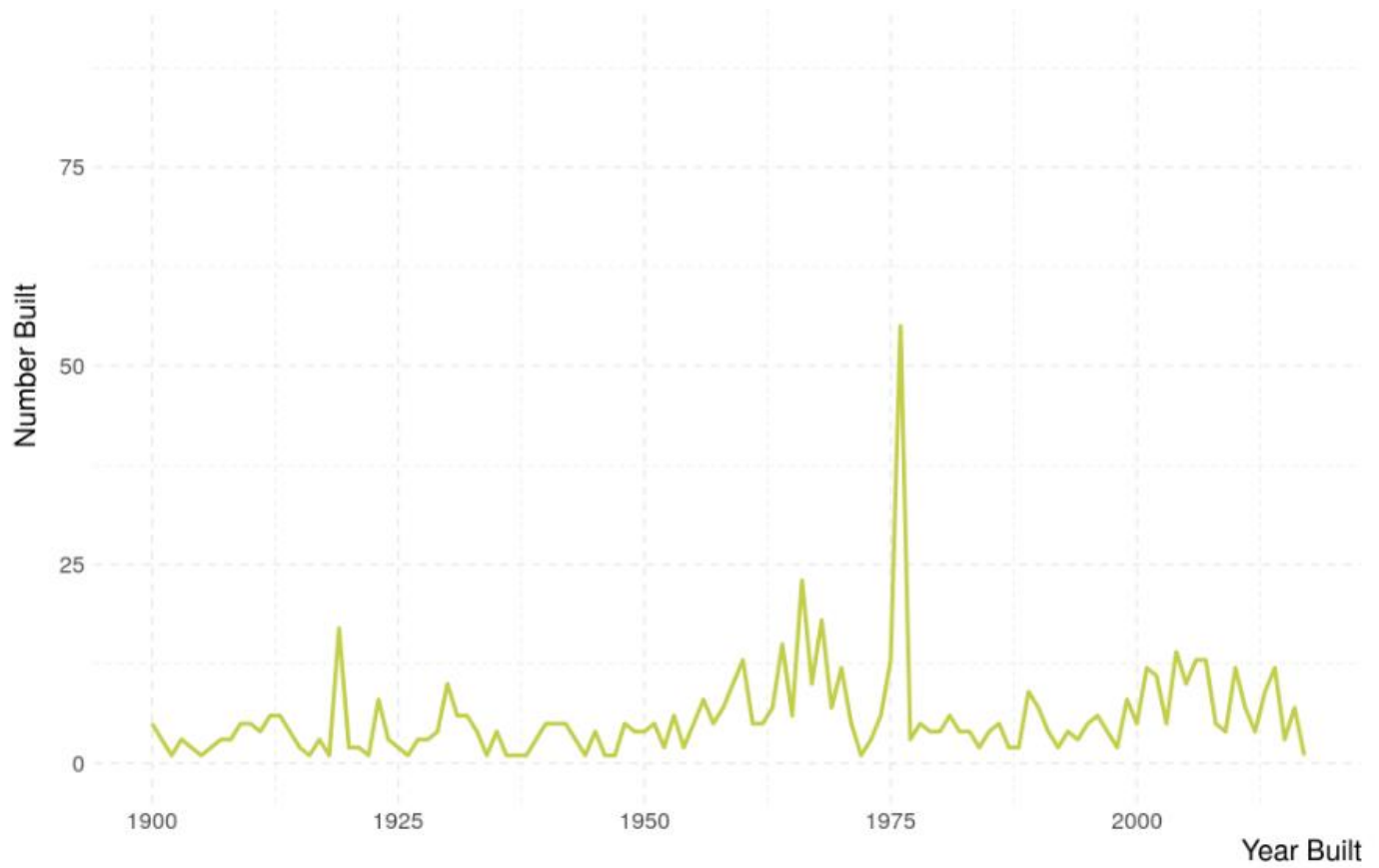


Figure-2.3.14: Total number of buildings made between 1900 and 2016.

It is relevant from figure-2.3.14 that maximum number of buildings were made in 1976, this helps us in finding out the over aged buildings and I can focus on their energy consumption pattern.

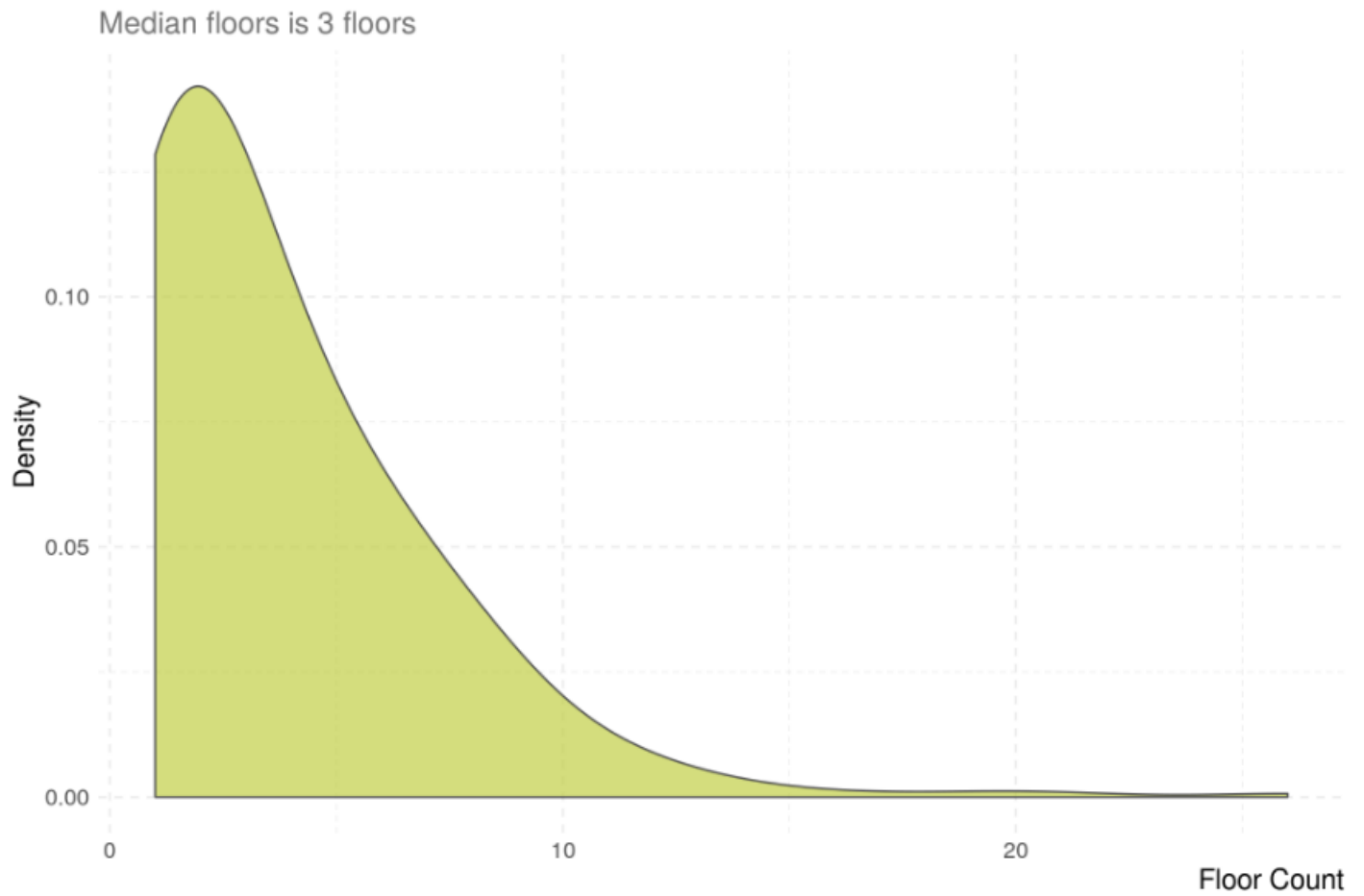


Figure-2.3.15: Total number of buildings vs their total number of floor count.

Figure-2.3.15 shows that mostly all buildings have 3 floors, whereas very few buildings are as high as 30 floors, the average number of floors per building is 3.

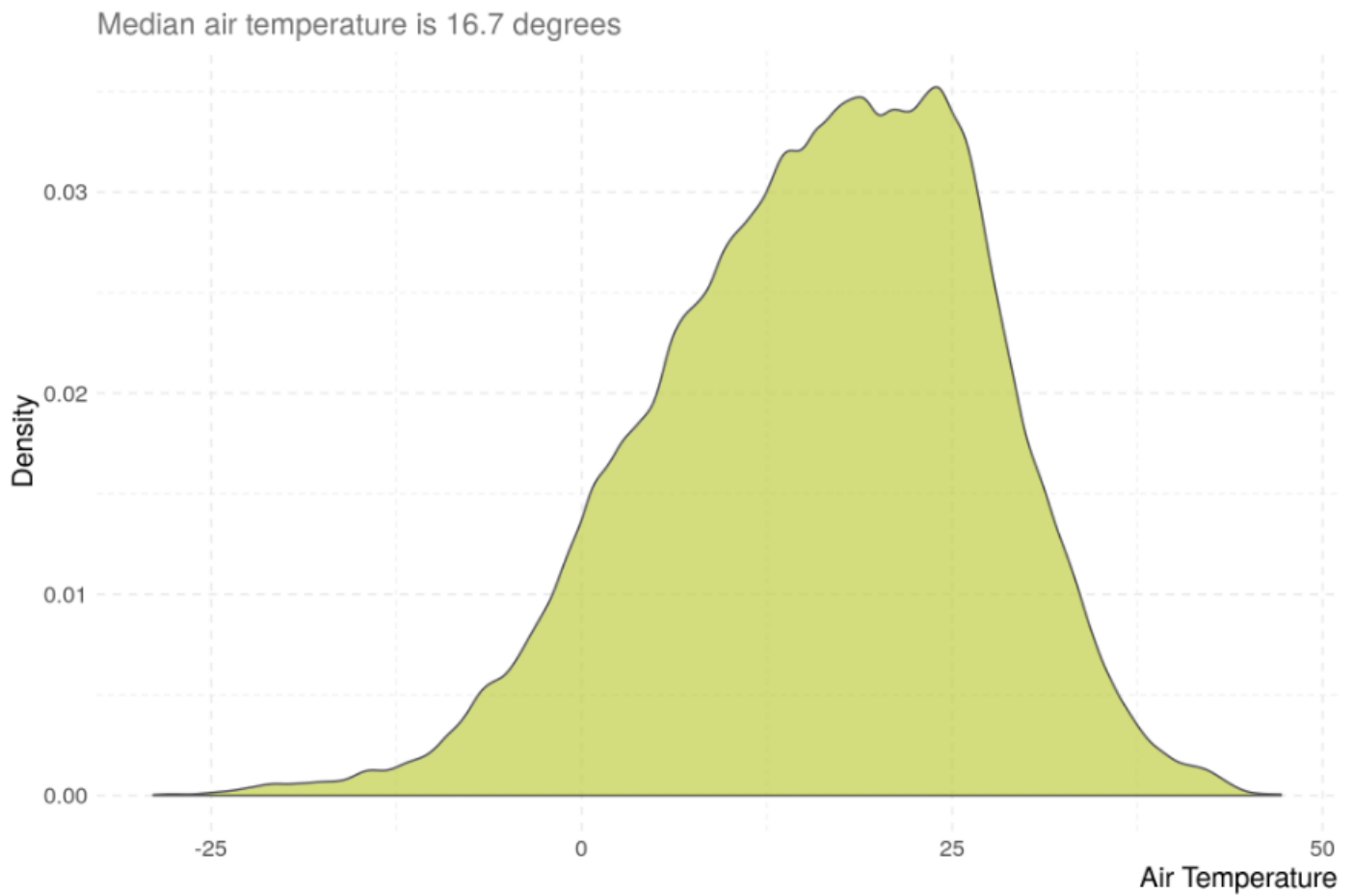


Figure-2.3.16: Distribution of air temperature with average at 16.7 degrees.

Figure-2.3.16 shows average value of air temperature is 16.7 degrees, whereas it maximizes to 50 degrees and minimizes to -30 degrees.

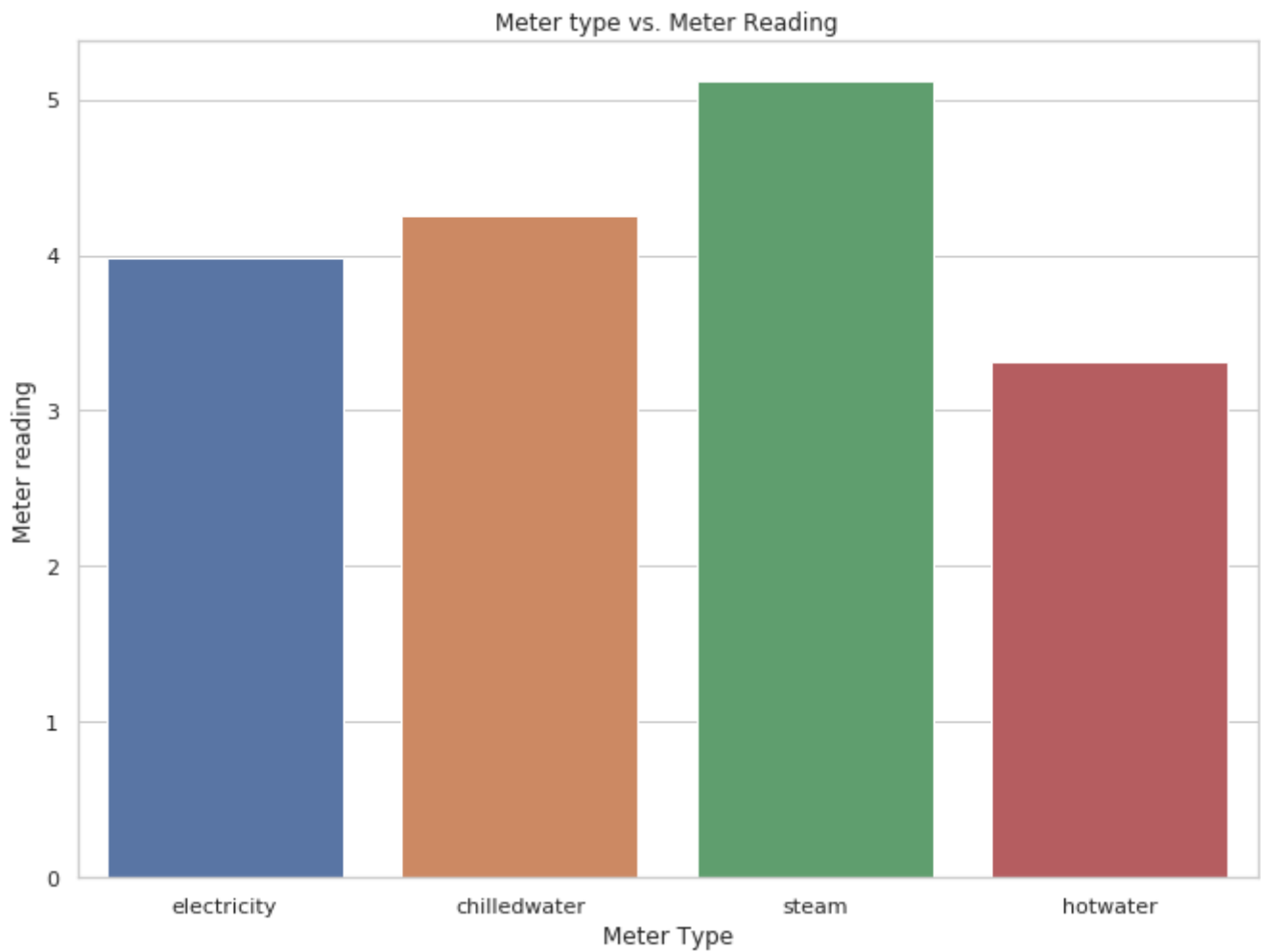


Figure-2.3.16: Meter reading plotted against all 4 meters for estimating range of individual readings.

Figure-2.3.16 shows meter types and their readings, where steam meter record has highest range of readings, followed by child water and electricity.

Chapter-3: Cleaning the Data Set and Removing Outliers.

3.1 Cleaning the data set and removing outliers.

As mentioned earlier this data set requires cleaning and reorganization [4] as shown in figure-3.1.0 below.

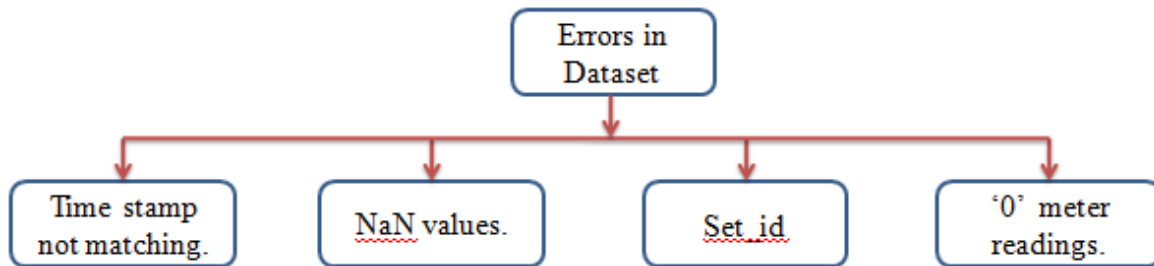


Figure-3.1.0: Most prominent errors present in the dataset.

Whereas figure-3.1.0.1 shows actions performed to mitigate this problem by performing various data cleaning operations, which are discussed in depth under upcoming sections.

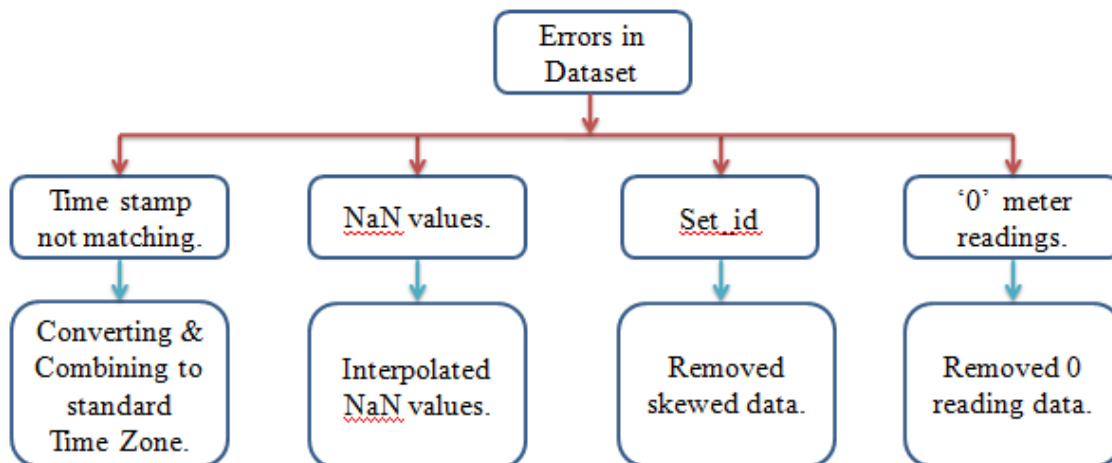


Figure-3.1.0.1: Methods used to mitigate errors present in the dataset.

It is important to understand that errors present here will not allow my model to train properly since skewed time stamp numbers, meter readings with '0' and NaN values will act as an outlier and increase the validation error since weights would not be optimally trained. Hence in sections discussed below, I will be focused on eliminating these errors and skewed values in order to maintain a cleaned dataset.

I have combined train.csv file, weather_train.csv file and building metadata.csv files together since they are related to each other, combined files has 139773 rows and 9 columns.

building_id	meter	timestamp	meter_reading
0	0	1/1/2016 0:00	0
1	0	1/1/2016 0:00	NaN
2	0	1/1/2016 0:00	0
3	0	1/1/2016 0:00	0
4	0	1/1/2016 0:00	NaN
5	0	1/1/2016 0:00	NaN
6	0	1/1/2016 0:00	NaN
7	0	1/1/2016 0:00	NaN
8	0	1/1/2016 0:00	0
9	0	1/1/2016 0:00	0
10	0	1/1/2016 0:00	NaN
11	0	1/1/2016 0:00	0
12	0	1/1/2016 0:00	0
13	0	1/1/2016 0:00	0
14	0	1/1/2016 0:00	NaN
15	0	1/1/2016 0:00	0
16	0	1/1/2016 0:00	0
17	0	1/1/2016 0:00	0

Figure-3.1.1: Input files having 0, NaN and skewed time stamp values.

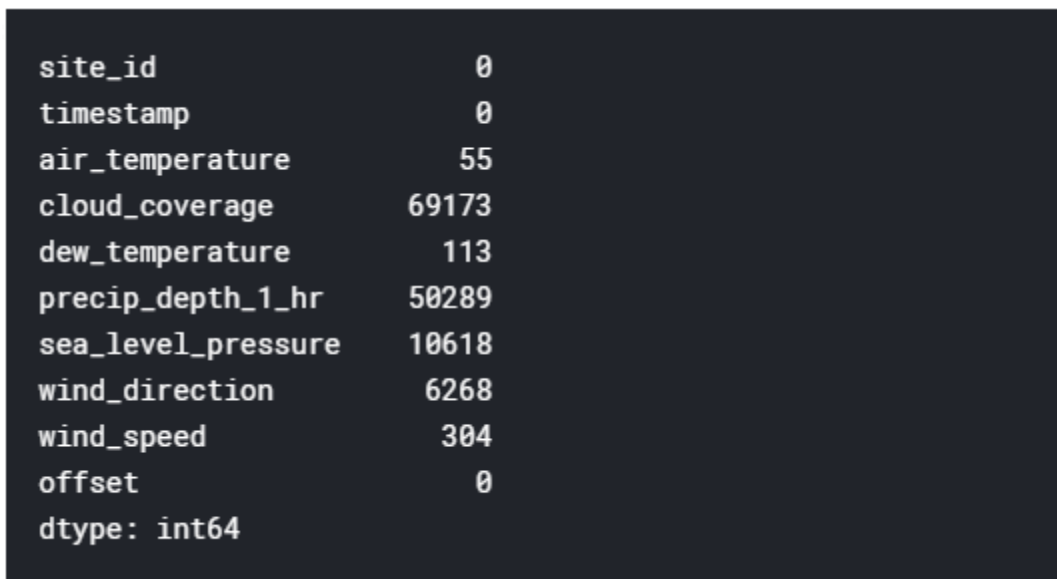
It can be clearly seen in figure-3.1.1 that there are various meter readings which show 0 as their output, similarly there are certain NaN values and all the time stamp values are not arranged according to standard time. Therefore it is very important to remove 0 reading values, interpolate to fill the NaN values, convert all time stamps into standard time and drop all junk data characters under certain fields.

	site_id	timestamp	air_temperature	cloud_coverage	dew_temperature
139768	15	2016-12-31 19:00:00	3.0	NaN	-8.0
139769	15	2016-12-31 20:00:00	2.8	2.0	-8.9
139770	15	2016-12-31 21:00:00	2.8	NaN	-7.2
139771	15	2016-12-31 22:00:00	2.2	NaN	-6.7
139772	15	2016-12-31 23:00:00	1.7	NaN	-5.6

Figure-3.1.2: Converting local time stamp to standard global time stamp.

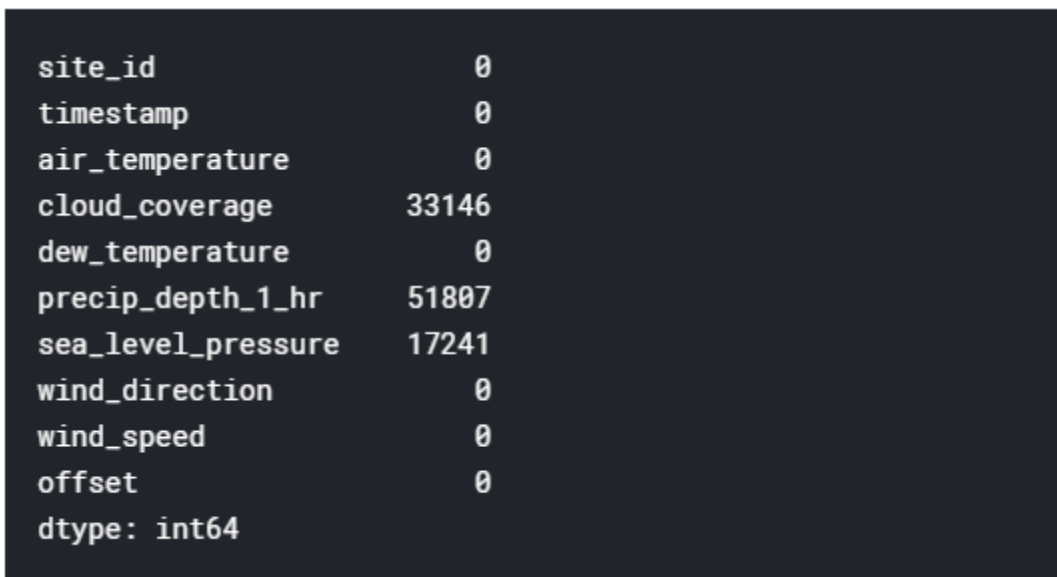
Firstly I have merged all time stamps together and converted them to standard global time from their local time values, so that further operations can become simpler, refer to figure-3.1.2.

Secondly I have interpolated NaN values in both upward and downward directions so that NaN values can be predicted and filled instead of being deleted, this helps in preserving original size of the data set. Refer to figure-3.1.3 which shows number of NaN values for each data variable.



site_id	0
timestamp	0
air_temperature	55
cloud_coverage	69173
dew_temperature	113
precip_depth_1_hr	50289
sea_level_pressure	10618
wind_direction	6268
wind_speed	304
offset	0
dtype:	int64

Figure-3.1.3: Total number of NaN values for every given label.



site_id	0
timestamp	0
air_temperature	0
cloud_coverage	33146
dew_temperature	0
precip_depth_1_hr	51807
sea_level_pressure	17241
wind_direction	0
wind_speed	0
offset	0
dtype:	int64

Figure-3.1.4: Reduced number of NaN values after successfully interpolating them.

Consider figure-3.1.4 which shows reduced number of NaN values after successfully interpolating the NaN values in both upward and downward direction, the left over values could not be interpolated, thus they will be deleted.

Lastly all 0 meter id values are removed from this combined data set since they correspond to no meter data scenario, where meter reading is of no use, refer figure-3.1.5 which shows cleaned data set.

site_id	timestamp	air_temperature	cloud_coverage	dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_direction
0	2015-12-31 19:00:00	25.0	6.0	20.0	-1.0	1019.7	0.0
0	2015-12-31 20:00:00	24.4	4.0	21.1	-1.0	1020.2	70.0
0	2015-12-31 21:00:00	22.8	2.0	21.1	0.0	1020.2	0.0
0	2015-12-31 22:00:00	21.1	2.0	20.6	0.0	1020.1	0.0
0	2015-12-31 23:00:00	20.0	2.0	20.0	-1.0	1020.0	250.0

Figure-3.1.5: Pre-processed and cleaned data set.

Figure-3.1.5 shows pre-processed and cleaned data set which can be now used for training the neural network, the new dimensions of this data set are shown in figure-3.1.6, which shows the shape in form of rows and columns below.

```

Training data shape: (20216100, 4)
Weather training shape: (139773, 10)
Weather training shape: (277243, 10)
Weather testing shape: (1449, 6)
Test data shape: (41697600, 4)

```

Figure-3.1.6: New shape in terms of rows and columns after pre-processing and data cleaning.

From figure-3.1.6 it is evident that after combining all these files together in order to maintain the same test file as I mentioned above, will result in 20112649 rows and 17 columns, this clean and merged data set will be used for training the network.

Additionally, I observed that site_id 0 contained wrong data in place of numerical values, therefore it was necessary to drop this junk data refer to figure-3.1.7 which shows site_id 0 has been cleaned of junk data.

	building_id	meter	meter_reading	site_id	primary_use	square_feet	year_built	floor_count	air_temperature
20215251	1064	0	5.042969	12	0	11.784570	1968.0	4.0	6.5
20215252	1065	0	1.827148	12	6	9.405249	1968.0	4.0	6.5
20215253	1066	0	3.291016	12	0	10.929529	1968.0	4.0	6.5
20215254	1067	0	5.445312	12	0	11.741661	1968.0	4.0	6.5
20215255	1068	0	4.050781	12	0	11.634754	1968.0	4.0	6.5

Figure-3.1.7: New data set with cleaned site_id and other variables.

Cleaned dataset will ensure that my model can achieve optimal weights during training since all the outliers are removed and NaN values have been successfully interpolated, whereas meter readings having '0' values have

been removed. Consider figure-3.1.7 which shows heat map (correlation matrix) for various parameters given in the dataset.

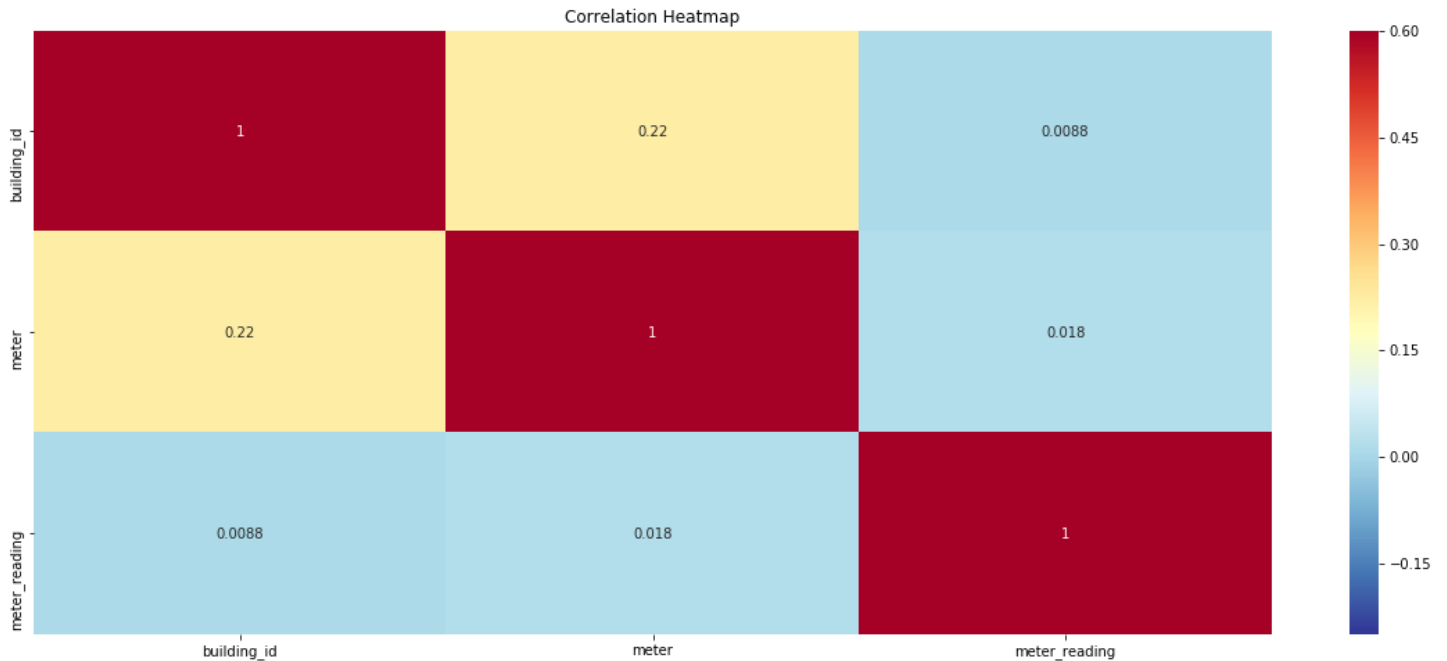


Figure-3.1.7: Heat map based correlation matrix for cleaned dataset.

As expected from the data visualization plots from section 2, categorical data do not correlate with other after removal of all outliers specially meter readings with '0' values. This figure ensures that data has been cleaned successfully, for example data given under meter_reading should not correlate with building_id. More number of '0' values present in the data set earlier would have correlated these variables, which has been successfully avoided after cleaning and combining this data set.

Chapter-4: Creating Light GBM as Base Model.

4.1 Preparing the model and setting parameters.

Following LGBM code has been taken from [5], in this light gradient boosting model I have used leaf based boosting method which helps in conversion of weak learners into strong learners. It uses tree based weight optimizing technique thus after finishing the evaluation of first tree, it increases the weights of those observations that are difficult to classify and lower the weights for those that are easy to classify and second tree is therefore grown on this weighted data, refer figure-4.1.1.

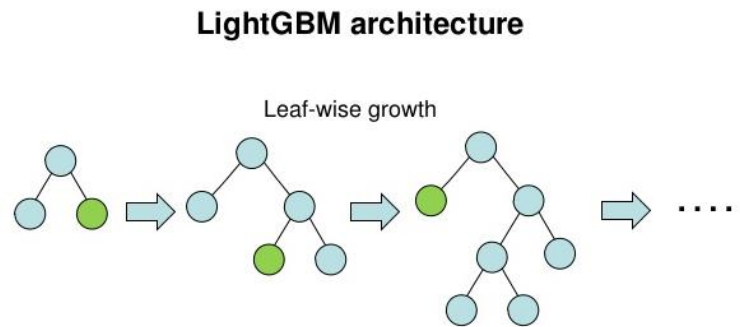


Figure-4.0.1: LGBM leaf wise growth.

Refer to table-4.1.1 which shows all parameters used in this model, in the following section I will detail about these parameters individually.

Table-4.1.1: Detailed parameters used for LGBM tree based model.

Parameter name	Value of the Parameter.
Boosting_type	Gbdt
Objective	Regression
Metric	RMSE
Learning_rate	0.25
Feature_fraction	0.8
Lambda_l1	1
Lambda_l2	1
Subsample	0.4
Num_leaves	31

Gradient boosting GBDT identifies the weaker links or shortcomings by using gradients in the loss function, whereas objective is set as regression, since my problem is a regression based. Error metric is set as rmse [2] which is root mean square error based metric.

Similarly learning rate is set to be a low value of 0.25 so that optimum results can be found, one exciting parameter is feature_fraction which is set to 0.8, that means lgbm randomly selects 80% of parameters for making prediction trees in each iteration [3], lastly lambda specifies regularization which is set to 1.

I have also used subsample as 0.4, this ensures that stochastic gradient boosting and allows reduction of variance with small increment in the bias. Similarly num_leaves means total number of leaves in the entire tree, which is set to default value of 31.

Additionally I have used early stopping method as a common regularization technique so that training stops when test error and validation error do not improve, at this point parameters are stored and returned back in the program.

4.2 Results & Discussions from LGBM base model.

In this section I will share the results produced by proposed model and then I will discuss about the reasons behind these results. Refer to figure-4.2.1, which shows the training rmse and validation rmse values gradually decreasing with increasing number of iterations.

Training Loss: 2.42

Validation Loss: 2.88

Training RMSE: 0.87

Validation RMSE: 1.16

```
Training until validation scores don't improve for 100 rounds.
[100]  training's rmse: 0.860924      valid_1's rmse: 1.1908
[200]  training's rmse: 0.798752      valid_1's rmse: 1.18634
[300]  training's rmse: 0.769253      valid_1's rmse: 1.18272
[400]  training's rmse: 0.746902      valid_1's rmse: 1.1821
Early stopping, best iteration is:
[349]  training's rmse: 0.758543      valid_1's rmse: 1.18162
Training until validation scores don't improve for 100 rounds.
[100]  training's rmse: 0.904698      valid_1's rmse: 1.17205
[200]  training's rmse: 0.829832      valid_1's rmse: 1.16692
Early stopping, best iteration is:
[129]  training's rmse: 0.874455      valid_1's rmse: 1.16487
```

Figure-4.2.1: Results obtained by LGBM 0.87 as training rmse and 1.16 as validation rmse.

It can be clearly seen that best rmse for training data is 0.87, whereas for validation rmse is 1.16, which is a fairly low number. Early stopping regularization method helps in selecting the best rmse value for test and validation since it stops the training when validation and training rmse values do not show any further improvements. This model is preferred for large data sets since gradient boost algorithm optimizes user specific cost function instead of a pre defined loss function, thus it offers a better result and higher control than other models.

It should be noted that LGBM is not a neural network model, since this course is about neural networks therefore I will discuss 2 more models specifically using neural networks. I have not submitted this score and result in kaggle because of the same reason as mentioned above.

Chapter-5: Baseline model using LSTM.

The code for this section is given by Rico Hoffmann [4], which was thoroughly modified by our team. As mentioned in the previous section that LGBM is not a neural network based model since it operates on decision trees which is beyond the scope of this class, therefore I have created a new base model using LSTM. Same dataset is used for this model, which I have cleaned and re-organized in above section of this report using a series of cleaning and visualization techniques as discussed above in depth.

5.0 Reason for selecting LSTM.

Long short term memory (LSTM) is a Recurrent Neural Network (RNN) which means this model has a feedback along with 7 bits information storage capacity in its short term memory [5], thus for given dataset which covers time dependent information for more than 300 days LSTM seems to be an idle choice, refer to figure-5.0.1 for overall LSTM architecture, refer to figure-5.0.2 for LSTM cell architecture.

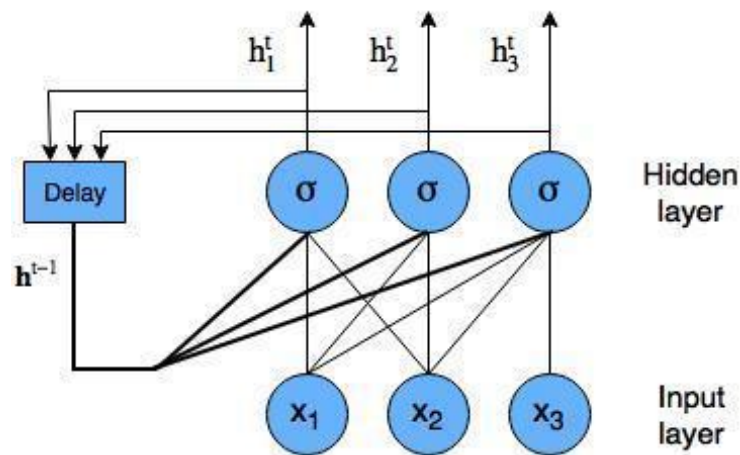


Figure-5.0.1: Overall architecture of LSTM Network.

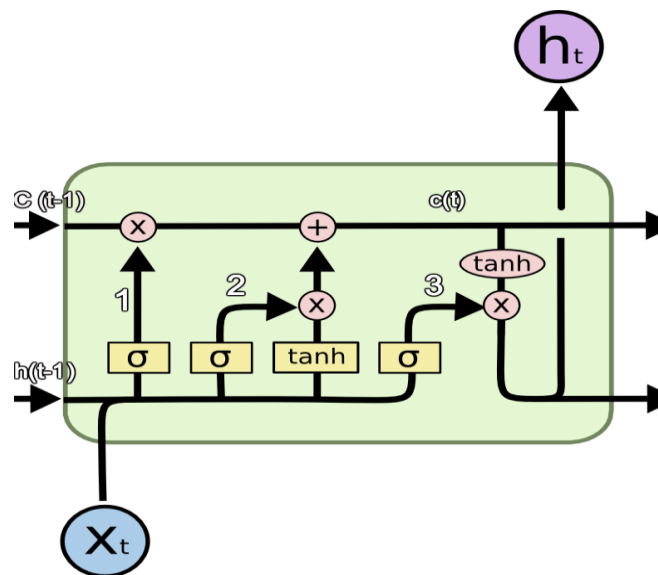


Figure-5.0.2: Internal architecture of LSTM cell.

In the figure shows above, X denotes information scaling, + denotes information addition, σ denotes the sigmoid layer, tanh stands for tanh layer, $h(t-1)$ denotes output from last LSTM unit, $c(t-1)$ denotes memory from last LSTM unit, $X(t)$ is the current input and $c(t)$ is New updated memory.

Therefore I have decided that for a time dependent dataset, LSTM based neural network will be useful since it will learn from previous stage and avoid explosion due to large values of accumulated weights by successfully forgetting some information whereas remembering selected details.

5.1 Building Blocks for LSTM.

1. Use of Sequential API:

I have used Sequential API in this project since it helps in creating model layer by layer, does not allow sharing of layers and it is easy to redefine model over and over again for multiple simulations.

2. Use of Flatten API:

Flatten API is used to convert stacked or layered input into a single flattened out 1 dimensional (1-D) layer, which is easy to process.

3. Use of Dense layer:

A dense layer is a fully connected layer of neurons which receive inputs from previous layers, have their own weights and bias refer [2].

output = activation(dot(input, kernel) + bias).....eq1

For further explanation about dense layer refer eq1 where output of such layers depends upon the output of activation function used in that layer therefore activation refers to activation function, kernel refers to weights associated with that neuron.

5.2 Creating LSTM network based Model-1.

Refer to table-5.2.1, which shows parameters used for creating a LSTM architecture, basic building blocks like dense layer, flatten and sequential API have been explained above.

Table-5.2.1: Parameter names and their values used in setting up LSTM network.

Parameter Name	Parameter Value
Horizontal arrays of cell in LSTM	512
Number of neurons in layer 1	128
Activation function in layer 1	Relu
Number of neurons in layer 2	1
Activation function in layer 2	Relu
Batch Size	1024
Input dimension	24
Error metric	RMSE
Optimizer	Adam

My LSTM network has a total of 512 cells with 2 additional hidden layers having 128 neurons and 1 neuron respectively. I have used ADAM as an optimizer instead of gradient descent algorithm for optimizing cost function in this case, because unlike gradient descent algorithm maintains constant learning rate, Adam typically computes adaptive learning rate by updating from first and second moments of the gradients. Consider figure-5.2.1 which shows summary of the model created above, with output shape of each layer.

Model: "sequential_2"		
Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 512)	1099776
dense_2 (Dense)	(None, 128)	65664
dense_3 (Dense)	(None, 1)	129
Total params: 1,165,569		
Trainable params: 1,165,569		
Non-trainable params: 0		

Figure-5.2.1: Model summary for LSTM network used as a base model.

Figure-5.2.1 shows the summary of model created with above mentioned parameters given in table-5.2.1, as shown above there are 512 LSTM cells, with 2 additional hidden layers having 128 and 1 neuron each respectively, all 1,165,569 parameters of the model described above are trainable.

5.3 Results & Discussions for Model-1based on LSTM.

As mentioned above LSTM based network performs well and produces fairly low loss values for training and validation data, it simultaneously gives low rmse values for both training and validation dataset, refer to the figure-5.3.1 for exact loss values.

Training Loss: 2.42

Validation Loss: 2.88

Training RMSE: 1.52

Validation RMSE: 1.58

```
Epoch 1/1
11845/11845 [=====] - 17547s 1s/step - loss: 2.4228 - rmse: 1.5258 - val_loss: 2.8831 - val_rmse: 1.5829
```

Figure-5.3.1: Loss and rmse values for training and validation.

It is evident that RMSE for training and validation for this model is still higher than 1.0, therefore a few changes can be made, which can hopefully bring error values below 1.0. Therefore in next section I have decided to tweak few hyper parameters of this model in order to make a new model and simulate it to get better results.

5.4 LSTM network based Model-2.

This model is an extension of model-1 discussed before, in this model I am changing few hyper parameters shown in table-5.4.1 below.

Table-5.4.1: Changes in various parameters exclusive for Model-2.

LSTM Base Model – 1 Old parameter values	LSTM Base Model – 2 New Parameter Values
LSTM Cells = 512	LSTM Cells = 8
Number of additional Dense layers = 2	Number of additional Dense layers = 1
Number of neurons in each layer = 128 & 1	Number of neurons in each layer = 1
Training RMSE = 1.52	Training RMSE = 2.01
Validation RMSE = 1.58	Validation RMSE = 1.722
Trainable Parameters = 1,165,569	Trainable Parameters = 1065

Consider figure-5.4.2, which shows results for newly made model-2, results in this new model instead of improving actually deteriorate.

New Training Loss: 4.14

New Validation Loss: 2.73

New Training RMSE: 2.01

New Validation RMSE: 1.72

Epoch 1/1

14806/14806 [=====] - 24118s 2s/step - loss: 4.3577 - rmse: 2.0120 - val_loss: 2.7315 - val_rmse: 1.7224

Figure-5.4.1: Loss and rmse values for training and validation.

In this model RMSE validation increases to 1.722, which is a high rmse value, therefore this is not a good model and it will be rejected hereafter.

Chapter-6: MLP based Neural Network as Best Model.

As mentioned in the previous section that LSTM after certain modification produces high rmse, which resulted in a very low rank for my kaggle competition. Therefore I have decided to use Multi Layer Perceptron model (MLP) for my regression based problem.

6.0 Reason for selecting MLP.

MLP helps in creating a neural network model with a limited depth and known data paths, this ensures that weights will be trained to an optimal solution without exploding or rapidly diminishing due to depth of the neural networks, unlike LSTM based networks. It should be noted that same cleaned and re-organized dataset is used for this MLP model, which is discussed in previous sections of this report.

6.1 Creating MLP based neural network.

Refer to table-6.1.1 which shows all parameters programmed for this model, with total number of hidden layers and number of neurons in each layer.

Table-6.1.1: Detailed description of parameters programmed for MLP model.

Parameter Name	Parameter Value
Hidden layer 1	Number of Neurons = 64, RELU
Dropouts for hidden layer 1	0.2
Hidden layer 2	Number of Neurons = 32, RELU
Dropouts for hidden layer 2	0.1
Hidden layer 3	Number of Neurons = 32, RELU
Dropouts for hidden layer 3	0.1
Hidden layer 4	Number of Neurons = 16, RELU
Dropouts for hidden layer 4	0.1
Learning Rate	0.001
Optimizer	Adam
Error Metrics	RMSE
K-Folds	4 folds have been used
Epochs	10
This model has 4 hidden layers each with RELU activation function.	

My MLP model has 4 hidden layers as mentioned above with 64, 32, 32 and 16 neurons each respectively, each neuron has activation function of RELU. It is worth mentioning that dropout method is used for regularization for this model, unlike other models as discussed above.

Dropout method helps in probabilistically dropping out inputs to given layers, this makes my model more robust and adaptable for a diverse variety of inputs, as shown in the table above dropout values are 2% for first hidden layer and followed by 1% each for other three hidden layers.

Learning rate is knowingly kept small, so that model reaches to an optimal solution, since higher accuracy and lower RMSE is the only reason behind simulating this model. Lower learning rate will allow back propagation algorithm to optimize the cost function to the lowest possible value.

Similar to previous LSTM based architecture I have again used ADAM as an optimizer instead of gradient descent algorithm for optimizing cost function, because unlike gradient descent algorithm maintains constant learning rate, Adam typically computes adaptive learning rate by updating from first and second moments of the gradients.

I have used Relu activation function that returns element-wise $\max(x, 0)$ for all $x > 0$, else it is 0, this provides very simple and fast computation abilities in comparison to sigmoid or tanh functions. Output is always linear but non-negative, therefore it generalizes input data by setting negative inputs to 0 during final output unlike sigmoid function. Additionally in this model I have used K-fold cross validation method with early stopping regularization technique for a total of 10 epochs each, this helps in avoiding over fitting.

6.3 Results & Discussions for New MLP Neural network.

Results of this model are shown in figure-6.3.1 which clearly indicates overall training loss, validations loss, training rmse and validation rmse.

Training Loss: 1.09

Validation Loss: 0.98

Training RMSE: 1.06

Validation RMSE: 0.97

```
Epoch 00004: val_root_mean_squared_error did not improve from 0.99981
Epoch 5/10
14833419/14833419 [=====] - 146s 10us/step - loss: 1.1436 - root_mean_squared_error: 1.0678 - val_loss: 1.0693 - val_root_mean_squared_error: 1.0133

Epoch 00005: val_root_mean_squared_error did not improve from 0.99981
Epoch 6/10
14833419/14833419 [=====] - 144s 10us/step - loss: 1.1288 - root_mean_squared_error: 1.0608 - val_loss: 1.0004 - val_root_mean_squared_error: 0.9793

Epoch 00006: val_root_mean_squared_error improved from 0.99981 to 0.97935, saving model to model_3.hdf5
Epoch 7/10
14833419/14833419 [=====] - 143s 10us/step - loss: 1.1188 - root_mean_squared_error: 1.0561 - val_loss: 0.9887 - val_root_mean_squared_error: 0.9737

Epoch 00007: val_root_mean_squared_error improved from 0.97935 to 0.97373, saving model to model_3.hdf5
Epoch 8/10
14833419/14833419 [=====] - 150s 10us/step - loss: 1.1095 - root_mean_squared_error: 1.0517 - val_loss: 0.9873 - val_root_mean_squared_error: 0.9714

Epoch 00008: val_root_mean_squared_error improved from 0.97373 to 0.97140, saving model to model_3.hdf5
Epoch 9/10
14833419/14833419 [=====] - 149s 10us/step - loss: 1.1044 - root_mean_squared_error: 1.0492 - val_loss: 1.0086 - val_root_mean_squared_error: 0.9841

Epoch 00009: val_root_mean_squared_error did not improve from 0.97140
Epoch 10/10
14833419/14833419 [=====] - 146s 10us/step - loss: 1.0977 - root_mean_squared_error: 1.0461 - val_loss: 1.0058 - val_root_mean_squared_error: 0.9809

Epoch 00010: val_root_mean_squared_error did not improve from 0.97140
*****
```

Figure-6.3.1: Training, validation loss and rmse results for MLP model.

This model trains for 4 different randomly shuffled folds each iterating with 10 epochs, in 1st fold early stopping is used to avoid over fitting, whereas optimum rmse is achieved in 2nd fold. This improvement is saved in “model_1.hdf5” file, which is used as our final submission, consider figure-6.3.2 that shows gradual descent in training and validation RMSE values for all 4 folds.

It is evident that lowest RMSE is achieved in 2nd fold as mentioned above, where as other values hover around the same point. Consider table-6.3.1 which shows training, validation loss and rmse values for all 4 models discussed above.

Table-6.3.1: Training, Validation Loss and RMSE values for all 4 models.

Model Name	Training Loss	Validation Loss	Training RMSE	Validation RMSE
LGBM	0.95	1.12	0.87	1.16
LSTM Model-1	2.42	2.88	1.52	1.58
LSTM Model-2	4.14	2.73	2.01	1.72
MLP (BEST Model)	1.09	0.98	1.06	0.97
Best Neural Network for this project is MLP since it has smallest validation RMSE.				

It is very evident that LGBM has smallest training RMSE but it is ignored since it is not a neural network, similarly MLP based model has lowest validation rmse of 0.97. Therefore our objective for this project is achieved where we intended to achieve rmse values under 1%.

Chapter-7: Conclusion and Kaggle Submission.

This project was majorly divided into 2 broad parts, first part was based on cleaning, combining, reorganizing and thoroughly visualizing the data set.

Second part was based on finding an optimized neural network based model for predicting energy consumption values, for which my MLP based model compiled with rmse of 0.97%. Data set of this project had a too many discrepancies which were essentially required to be removed so that models can be trained using this clean dataset. All the models were given input from same reorganized dataset, out of which MLP outperformed other.

All our objectives for this project were successfully met and it can be concluded that with above mentioned results our kaggle competition rank was highly improved to **2059** whereas our validation rmse was **0.97%**.

It is worth mentioning that correct kwh energy readings can be now predicted by our newly made MLP model as mentioned above for a usage based payment model which is a prime application of this project.

7.0 Conclusion for LGBM

LGBM is not a neural network but it is designed to handle large data sets, this model helps in reducing validation rmse to 1.16 because lgbm typically focuses on high accuracy using decision tree based models as discussed above. In real world applications this is the main reason why lgbm usually outperforms a MLP or LSTM based model, which I have discussed in further sections of this report.

7.1 Conclusion and Explanation for Poor Performance of LSTM.

It can be concluded that model-2 with reduced number of LSTM cells, only 1 additional hidden layer with 1 neuron makes it less complex than 1st LSTM network as discussed previously. An over simplified model-2 will not be able to improve the accuracy since more weights will be required optimize the cost function. Therefore reducing number of LSTM cells, number of hidden layers and number of neurons in hidden layer will conclusively deteriorate overall performance.

7.2 Kaggle submission baseline LSTM Model-1.

As discussed in sections above LSTM Model-1 outperforms LSTM Model-2, therefore we decided to submit model-1 in kaggle. After submission our rank was around 4000 out of 5000 participants, refer to figure-7.2 that shows our poor score of **1.153** based on LSTM model-1.

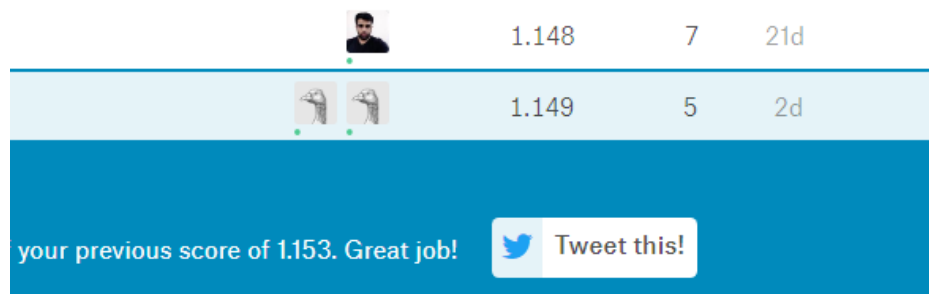


Figure-7.2: Our team's lower score based on my LSTM model.

Our rank is on the lower side after submitting my model, this could be due to various reasons, one of them being that previous teams might have submitted a better model with similar approach. Therefore in next section I will create a new model in order to improve our rank and reduce the error percentage, hence **this is ignored**.

7.3 Discussions: Why MLP defeats LSTM for given data set.

LSTM works on the concept of learning from past and predicting the future values, though the path taken to achieve it is usually very long which either explodes or diminishes the optimal weight values. Whereas this is one major advantage of MLP that depth of the model and path taken to train the weights does not explode or diminish the optimal weights, therefore if a model has a controlled depth and no over fitting problems then MLP produces better results as shown in table above. In this case rmse of LSTM is larger than rmse of MLP model, hence MLP is selected as my best model and scores of this model are submitted in kaggle.

7.4 Conclusion for MLP neural network as “BEST Model.”

It can be concluded that while designing a neural network, depth and complexity of the network should always be kept in mind, a fairly deep neural network like LSTM does not guarantee accurate results. Therefore MLP outperforms LSTM in this case with a validation rmse of 0.97%, which was successfully submitted to kaggle.

7.5 Kaggle submission based on MLP Model “BEST MODEL”

Consider figure-7. 5 which shows the final rank attained by my MLP based submission, this is our second submission on kaggle where we migrated from LSTM based neural network to MLP based neural network with improved validation rmse of 0.97%.

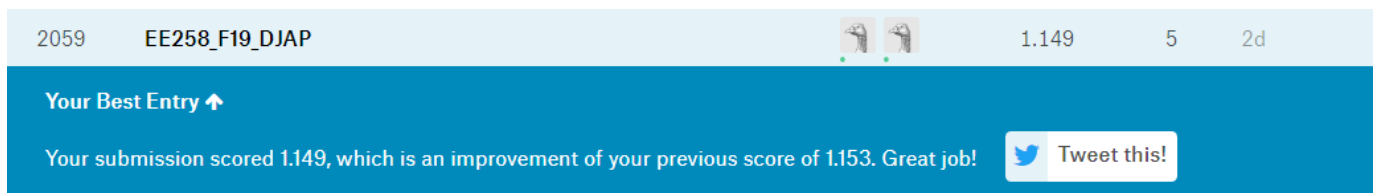


Figure-7.5: Our team’s new and improved rank based on my MLP model.

Our rank has **significantly improved to 2059**, which is a commendable achievements since we jumped up by a significant two thousand places, previously our last submission on kaggle scored a lower rank, which is now highly improved.

Challenges faced:

The biggest issue with this project is time constraint that forces me to limit my work, on top of that dependency on kaggle, training n their platform is a very big overhead.

Training takes 6 hours and every small change made takes a long amount of time, this is very time consuming.

Since we depend on kaggle for simulating the results, instead of simulating them on my laptop, size of the dataset causes very slow updates and thus we had to reduce number of epochs.

Future work:

This project should be executed on laptops that have a gpu, so that more number of epochs can be iterated.

LSTM network should be modified with deeper number of layers and neurons so that it can perform better than MLP.

Model should be deployed on an embedded board like nvidia’s gpu based platforms.

Special Mention:

I will like to mention this project was possible because of the online available MLP code given by Jason Zivkovic [1], LSTM given by Rico Hoffmann [4] and LGBM code given in [5] their codes and data analysis images with appropriate pre processing provided a very strong guidance to our team.

References.

- [1] <https://www.kaggle.com/jaseziv83/a-deep-dive-eda-into-all-variables>
- [2] <https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab>
- [3] <https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc>
- [4] <https://www.kaggle.com/drcapa/ashrae-datagenerator-lstm/notebook>
- [5] <https://www.kaggle.com/aitude/ashrae-kfold-lightgbm-without-leak-1-08>