CMT 307 Coursework 2 Group Project

Group number	20
Project title	Energy usage prediction
Supervisor	Hanzhi Wang
	C21114121 C1992365 C21052217 C21108612 C21104212 C21107597 C21119371 C1826277

Table of Contents

1. Introduction **4**

2. Literature review / Related work **4**

3. Descriptive analysis of the dataset **5**

3.1 Missing value and duplicates 9

3.2 Outliers 9

3.2 Features' analysis 10

3.2.1 Meter reading 10

3.2.2 Primary use 13

3.2.3 Meter 15

3.2.4 Other features 17

3.3 Correlation analysis 21

4. Data pre-processing **24**

4.1 General pre-processing (for both models) 24

4.1.1 Correct the unit 24

4.1.2 Outliers 24

4.1.3 Missing values 25

4.2 Model Specific Pre-Processing 25

4.2.1 Neural Network Pre-Processing 25

4.2.2 Supervised Learning Models Pre-Processing 26

5. Methodology & Implementation **27**

5.1 Regression Models 27

5.1.1 Feature Selection - Univariate Selection 27

5.1.2 Parameter Tuning 27

5.2 Recurrent Neural Networks 27

5.2.1 Gated Recurrent Unit (GRU) 29

5.2.2 Long Short Term Memory (LSTM) 29

5.2.3 General Setting of GRU/ LSTM 31

5.2.4 GRU/LSTM Implementation 31

6. Results **33**

6.1 Neural Network Results 33

6.1.1 GRU 33

6.1.2 LSTM 34

6.1.3 GRU vs LSTM 35

6.2 Regression Model Results 36

7. Analysis **36**

8. Conclusion and future work **37**

9. Reference List **38**

1. Introduction

The energy efficiency of buildings is a topic of great concern to many investors. The efficiency of energy use affects the return on investment and the environment. At present, there are some estimation methods to predict the energy consumption of different buildings. The aim is to accurately forecast and predict consumption rates as Building owners will pay the difference between these forecasts and actual consumption.

However, there is currently no clear standard for energy prediction. Scholars also have different opinions on the evaluation indicators in this issue. Some models are designed with a bias towards a certain indicator, so the positions of opinion are scattered like sand. The existing estimation methods still do not have a unified standard, so if one specific model is implemented as a policy, it will inevitably lead to various disputes.

In this project, we will develop models of metered building energy usage in the following areas: chilled water, electric, hot water, and steam meters. The models aim to assist building owners' ability to measure their costs more accurately. In addition, these models can also be used to support some individuals or companies who are interested in investing in energy conservation.

2. Literature review / Related work

Given the interest in energy usage prediction there are numerous publications and articles available that discuss the approaches taken and models used.

Brown, Barrington-Leigh and Brown (2011) used Kernel Regression to model Real-Time Building Energy Analysis. The model is claimed to exceed performance of neural networks when the training dataset is small. The dataset used was normalised with a total of 14 parameters, and separate models were trained for weekdays, weekday/holidays. Results: the KR using KNN approximation model performed the best in ¾ cases and outperformed neural networks.

Cao et al (2021) developed two models for two types of application using the ASHRAE dataset. The first model, a gradient boosting (GB) based model for medium to long-term predictions, and the second a long short-term memory (LSTM) based model for short term predictions. Multiple approaches for preprocessing were explored, with the best suggested for building energy predictions. Results: The GB based model achieves RMSE of 0.49 for electricity, 1.10 for chilled water, 1.25 for steam, and 1.32 for hot water. The LSTM model has about 38% lower prediction errors than the baselines, which are averages of energy consumptions from similar historical days.

Fu and Miller (2021) discuss usage of the ASHRAE dataset with ‘Google Trends’ to predict building energy consumption based on occupant behaviours. The approach utilised the search volume of topics (e.g., education or Microsoft Excel) on the Google Trends platform as a proxy of occupant behaviour and building use (e.g. holidays). Results show that highly correlated Google Trends data can effectively reduce the overall RMSLE error and the potential of using Google Trends to improve energy prediction.

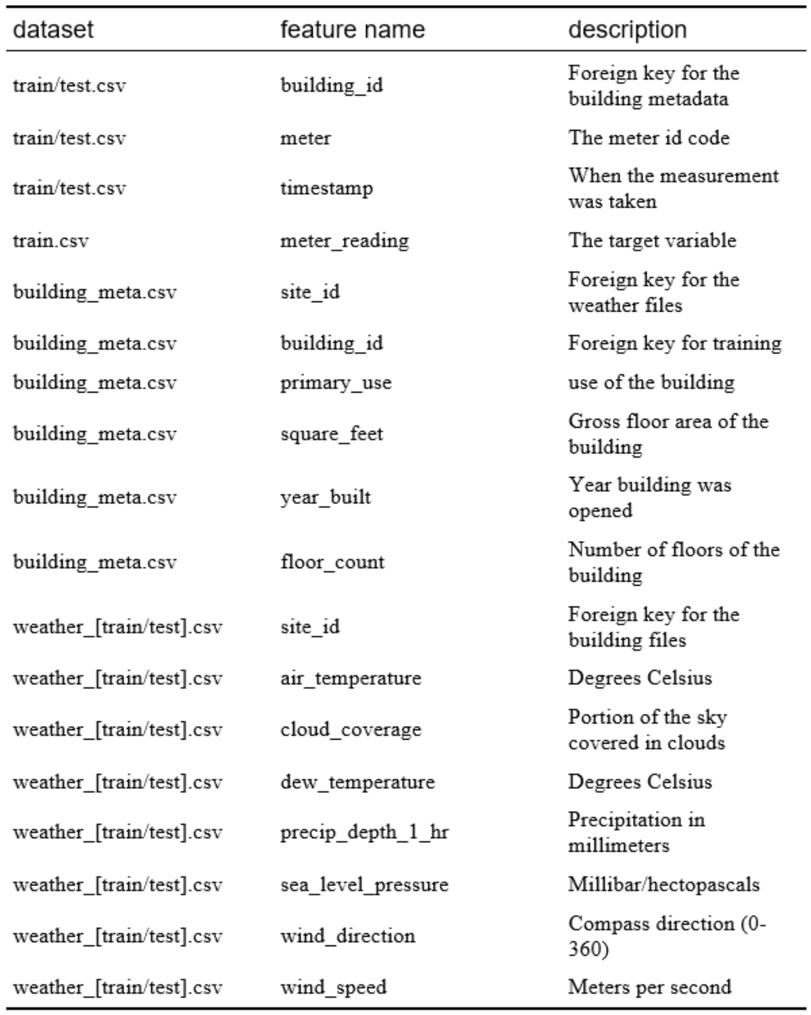
Ahmad et al (2018) conducted an analysis of electricity requirement forecasting using supervised based machine learning models with limited data information. Energy consumption data was categorized into monthly, seasonally and yearly basis to foresee performance for the short, medium and long term. Four-supervised based machine learning models were used: i) Binary Decision Tree; ii) Compact Regression Gaussian Process; iii) Stepwise Gaussian Processes Regression; iv) Generalized Linear Regression Model. Utilisation of limited energy usage and environmental data was found to significantly enhance the efficiency of energy requirement forecasting.

Paterakis (2017) compared the use of Multi Layer Perceptrons (MLP) with the most commonly used machine learning methods to predict aggregated energy consumption. Results showed that MLP outperformed other methods used in the study.

Eseye et al (2019) investigated a machine learning (ML)-based integrated feature selection approach to identify the most relevant and nonredundant predictors for accurate short-term electricity demand forecasting in distributed energy systems.

3. Descriptive analysis of the dataset

The following figure demonstrates the five data files Along with the feature names of each data file and its descriptions.



dataset	feature name	description
train/test.csv	building_id	Foreign key for the building metadata
train/test.csv	meter	The meter id code
train/test.csv	timestamp	When the measurement was taken
train.csv	meter_reading	The target variable
building_meta.csv	site_id	Foreign key for the weather files
building_meta.csv	building_id	Foreign key for training
building_meta.csv	primary_use	use of the building
building_meta.csv	square_feet	Gross floor area of the building
building_meta.csv	year_built	Year building was opened
building_meta.csv	floor_count	Number of floors of the building
weather_[train/test].csv	site_id	Foreign key for the building files
weather_[train/test].csv	air_temperature	Degrees Celsius
weather_[train/test].csv	cloud_coverage	Portion of the sky covered in clouds
weather_[train/test].csv	dew_temperature	Degrees Celsius
weather_[train/test].csv	precip_depth_1_hr	Precipitation in millimeters
weather_[train/test].csv	sea_level_pressure	Millibar/hectopascals
weather_[train/test].csv	wind_direction	Compass direction (0-360)
weather_[train/test].csv	wind_speed	Meters per second

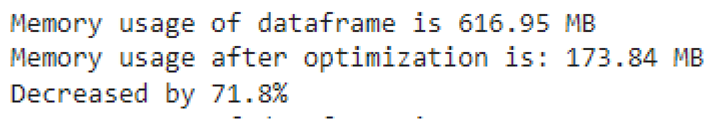
Fig 1: Data overview

The project's goal is to make predictions of meter_reading on test.csv, so the test.csv does not have meter_reading initially.

Table 1: Shape of the data files

Data name	Rows	Columns
train.csv	20,216,100	4
test.csv	41,697,600	4
building_meta.csv	1,449	6
weather_train.csv	139,773	9
weather_test.csv	277,243	9

The table1 shows that this is a big dataset with millions of rows. In order to manipulate these data in a limited memory workspace, we created a reduce memory usage function to deal with it which reduced over 70% of memory.



```
Memory usage of dataframe is 616.95 MB
Memory usage after optimization is: 173.84 MB
Decreased by 71.8%
```

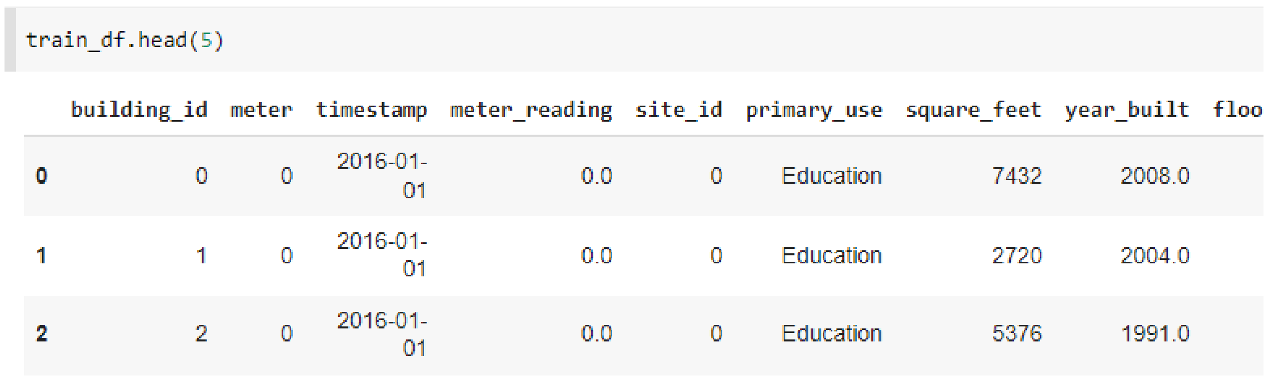
Fig 2: A example of memory optimization

According to the foreign keys (building_id, site_id, timestamp), we combined the train, building_meta and weather_train into train_df, and the test, building_meta, weather_test into test_df.

The train_df contains the data recorded in 2016 whilst the test_df in 2017 and 2018.
The test_df is for the model evaluation in the end and its structure is similar to train_df. So we mainly base on the train_df to do exploratory data analysis and build our models.

Moreover, the electric meter readings for site 0 were not properly converted to units of kWh and are in kBTU which influence over 900k rows in train_df. To fix this issue we multiply relevant instances by 0.2931 to convert the unit into kWh like the other sites, and multiply 3.4118 to get back to kBTU for scoring in the end.

Here is the glance of the train_df. Over 20 million instances were collected from four energy meters {0: electricity, 1: chilled water, 2: steam, 3: hotwater} at 1448 buildings from 16 sources, and the meter_reading was been record hourly, from 2016-01-01 00:00:00 to 2016-12-31 23:00:00.

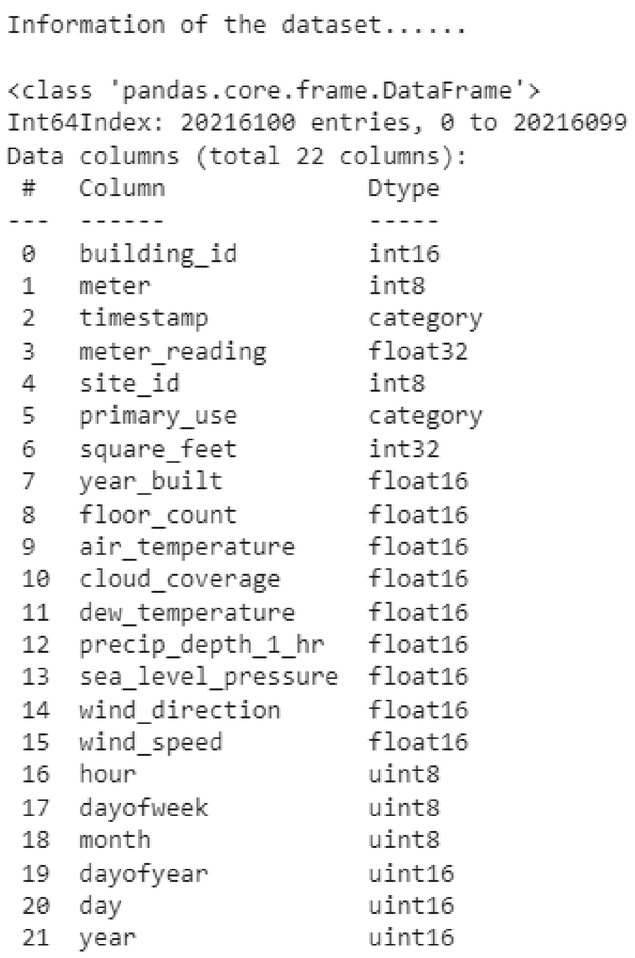


```
train_df.head(5)
```

	building_id	meter	timestamp	meter_reading	site_id	primary_use	square_feet	year_built	floo
0	0	0	2016-01-01	0.0	0	Education	7432	2008.0	
1	1	0	2016-01-01	0.0	0	Education	2720	2004.0	
2	2	0	2016-01-01	0.0	0	Education	5376	1991.0	

Fig 3 : Part of the train_df

The timestamp feature contains year, month and day in a string like format. In order to do exploratory data analysis in different time frames, we split it into five other features. Named: hour, day, month, dayofyear and dayofmonth.



```
Information of the dataset......

<class 'pandas.core.frame.DataFrame'>
Int64Index: 20216100 entries, 0 to 20216099
Data columns (total 22 columns):
 #   Column              Dtype   
---  ------              -----   
 0   building_id         int16   
 1   meter               int8    
 2   timestamp           category
 3   meter_reading       float32 
 4   site_id             int8    
 5   primary_use         category
 6   square_feet         int32   
 7   year_built          float16 
 8   floor_count         float16 
 9   air_temperature     float16 
 10  cloud_coverage      float16 
 11  dew_temperature     float16 
 12  precip_depth_1_hr   float16 
 13  sea_level_pressure  float16 
 14  wind_direction      float16 
 15  wind_speed          float16 
 16  hour                uint8   
 17  dayofweek           uint8   
 18  month               uint8   
 19  dayofyear           uint16  
 20  day                 uint16  
 21  year                uint16  
```

Fig 4: Information of the dataset

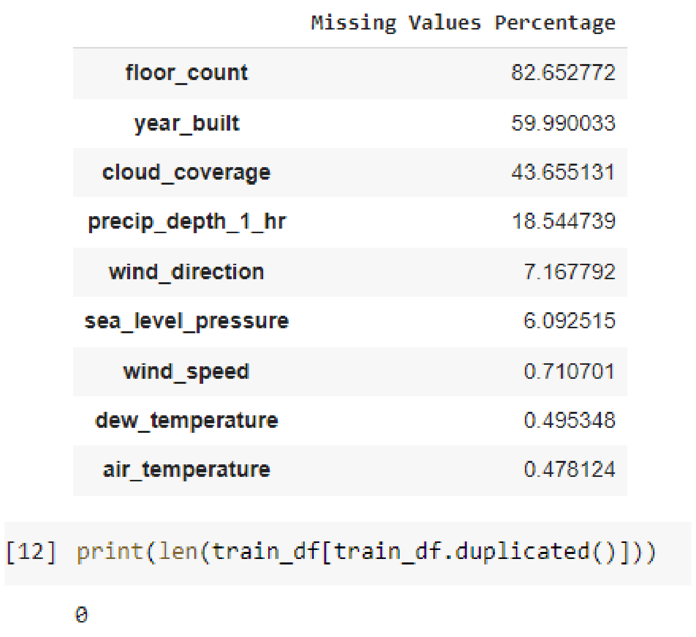
Next, we classified features in the table2. The bold meter_reading is the target variable. Classifying the variables' type is essential for subsequent encoding and scaling in the data pre-processing stage.

Table 2 : Features and its type

Categorical variables		Numeric variables	
Nominal	Ordinal	Interval-scaled	Ratio-scaled
timestamp, building_id, site_id, primary_use, meter, wind_direction,	cloud_coverage,	air_temperature , dew_temperature, year_built, floor_count, hour month, dayofweek, dayofyear, day, year	**meter_reading**, square_feet, sea_level_pressure, wind_speed, precip_depth_1_hr

3.1 Missing value and duplicates

Through the missing value analysis, it can be concluded that floor_count, year_buit has the most missing values followed by cloud_coverage. And there is no duplicate value in the train_df.



	Missing Values Percentage
floor_count	82.652772
year_built	59.990033
cloud_coverage	43.655131
precip_depth_1_hr	18.544739
wind_direction	7.167792
sea_level_pressure	6.092515
wind_speed	0.710701
dew_temperature	0.495348
air_temperature	0.478124

```
[12] print(len(train_df[train_df.duplicated()]))

     0
```

Fig 5: Missing value and duplicates

3.2 Outliers

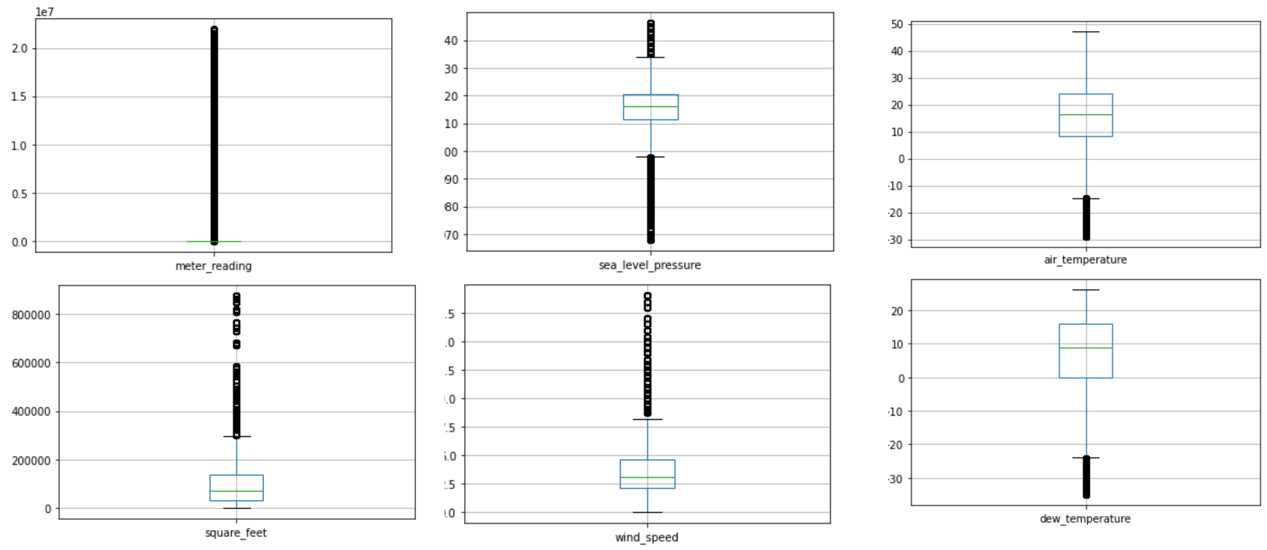


Fig 6: Box Plots

We selected some features to draw boxplots to observe the outliers. From the graphs above we can see there are some outliers in the selected features. Especially the meter_reading.

3.2 Features' analysis

Here are the 14 main features' distribution plots of the train_df. They are explored in this section.

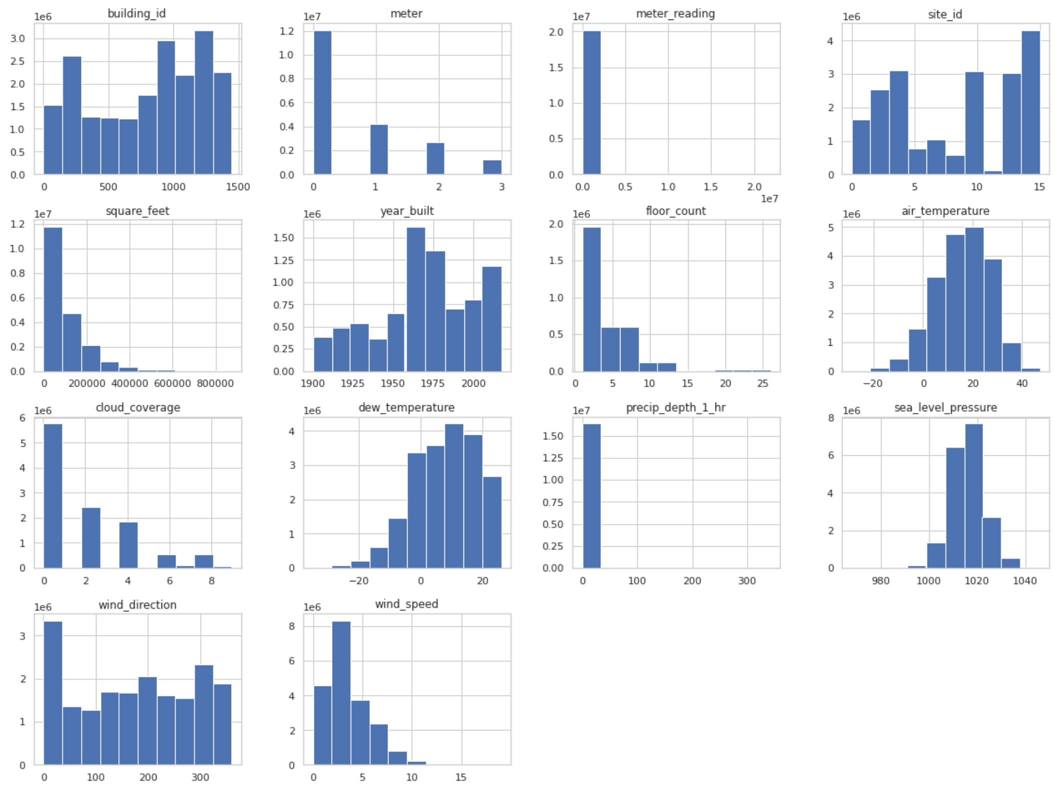
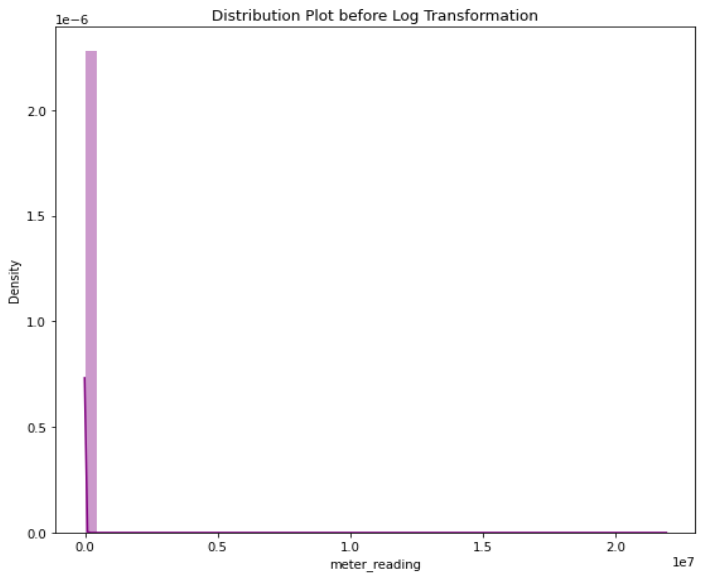


Fig 7: Distribution plots of main features in train_df

From the chart, we can see the 'building_id' and 'wind_directioin' roughly have a uniform distribution. And the 'air_temperature', 'sea_level_pressure' and 'year_built' basically follow the normal distribution. Besides, the 'meter_reading', 'square_feet', 'wind_speed' and 'floor_count' have right skewed distribution.

3.2.1 Meter reading

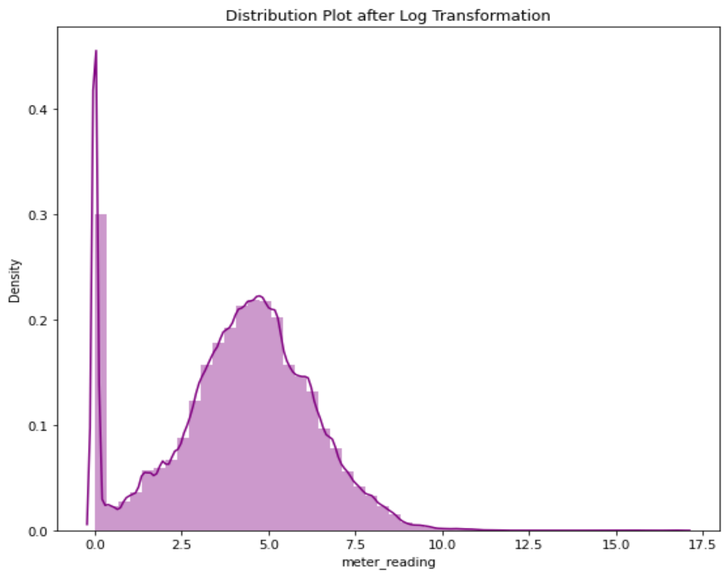
The target variable: meter_reading is essential for model building. By drawing the distribution plot, we can see the meter_reading is heavily skewed, and the value range is between 0 to over 2 millions which can not get much insight from it.



```
count    2.021610e+07
mean     1.987455e+03
std      1.532159e+05
min      0.000000e+00
25%      1.797590e+01
50%      7.509000e+01
75%      2.526000e+02
max      2.190470e+07
Name: meter_reading, dtype: float64
```

Fig 8: Distribution plot and statistical information about the meter reading

In order to boost the validity of our statistical analyses, we fixed the skewness by using log transformation, which makes the dataset align with the normality assumption in statistics. After the transformation we can see the bell shape graph. The meter reading lies between 0 and 17, and the most reading value is zero, followed by around 5.



```
count    2.021610e+07
mean     4.164197e+00
std      2.071129e+00
min      0.000000e+00
25%      2.943170e+00
50%      4.331917e+00
75%      5.535758e+00
max      1.690221e+01
Name: meter_reading_log, dtype: float64
```

Fig 9: Meter reading after log transformation

Then we named a new column ‘meter_reading_log’ for transformed ‘meter_reading’. And we use it to explore the trend of meter reading in hour, week and month.

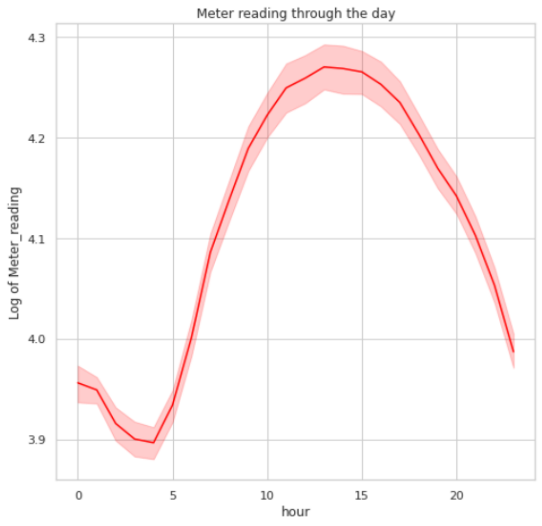


Fig 10: Log of meter reading by hour

It can be observed from the above plot that the meter readings drop in the early hours of the morning around 4am and rise again after 6am. The hourly usage peaks in the afternoon around 3pm. And after 3pm it starts to decrease.

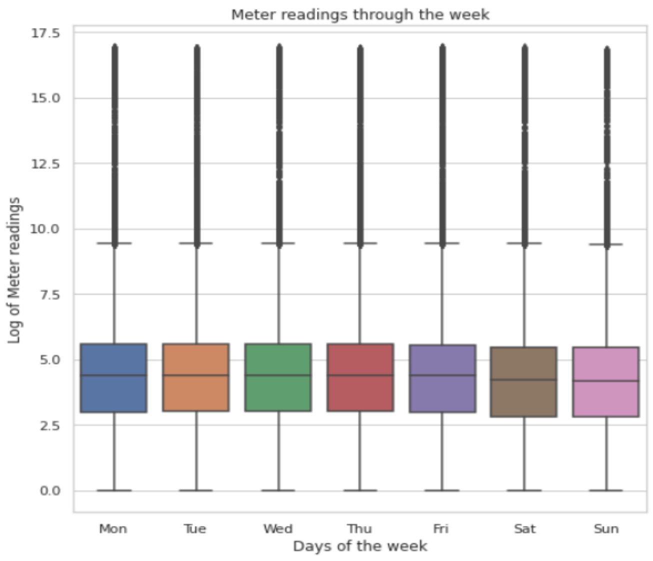


Fig 11: Log of meter reading by day

There is not much of a difference in day to day basis of usage. Still, weekends' usage is relatively lower than weekdays. There are many outliers which use more energy than others.

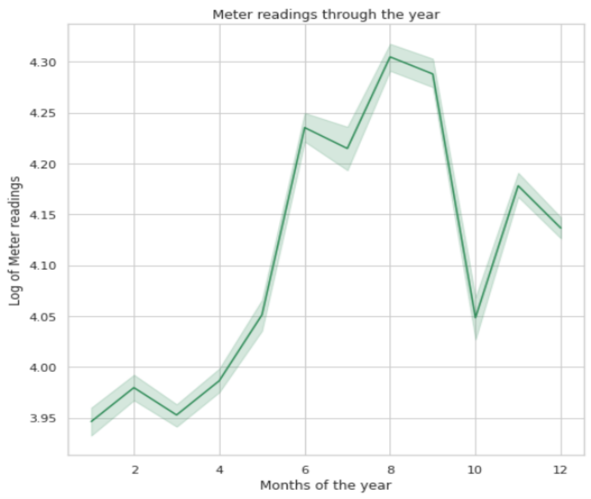


Fig 12: Log of meter reading by month

The average monthly consumption is minimal in the initial months and then rises after April. Then the energy usage rises sharply between April and June, and peaks in August.

3.2.2 Primary use

By drawing the count plot for the primary_use variable, we can see most readings are present for educational institutes followed by office and entertainment.

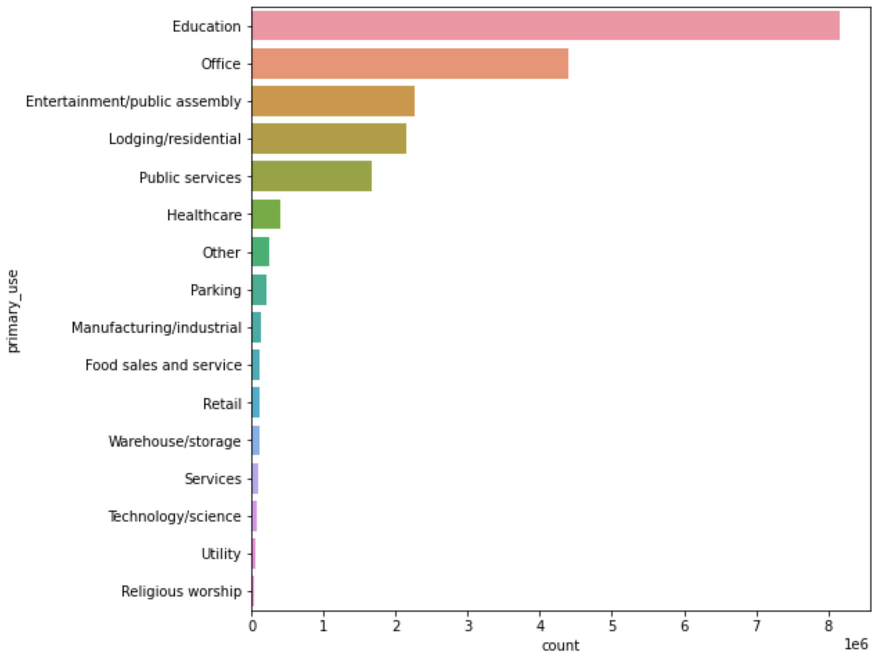


Fig 13: Count plot for each primary usage category

Next, we segmented the data by hours, days, and months to observe the energy usage trend respectively.

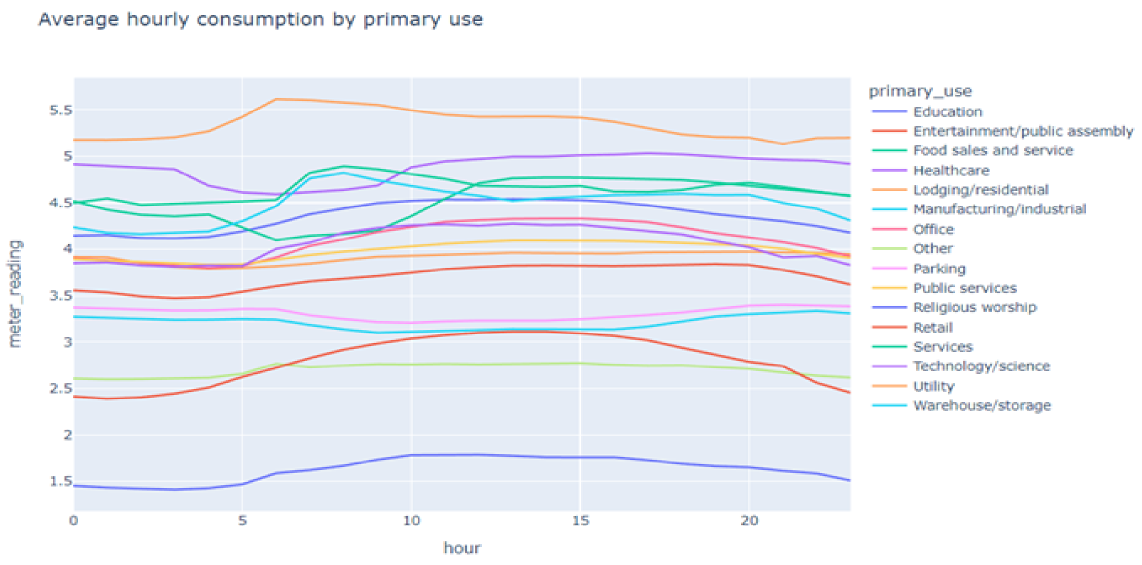


Fig 14: Average hourly meter_reading_log by primary use

Some features such as Public Service, lodging/residential, parking, and warehouse/storage have relatively stable changes throughout the day, whilst others have more fluctuation.

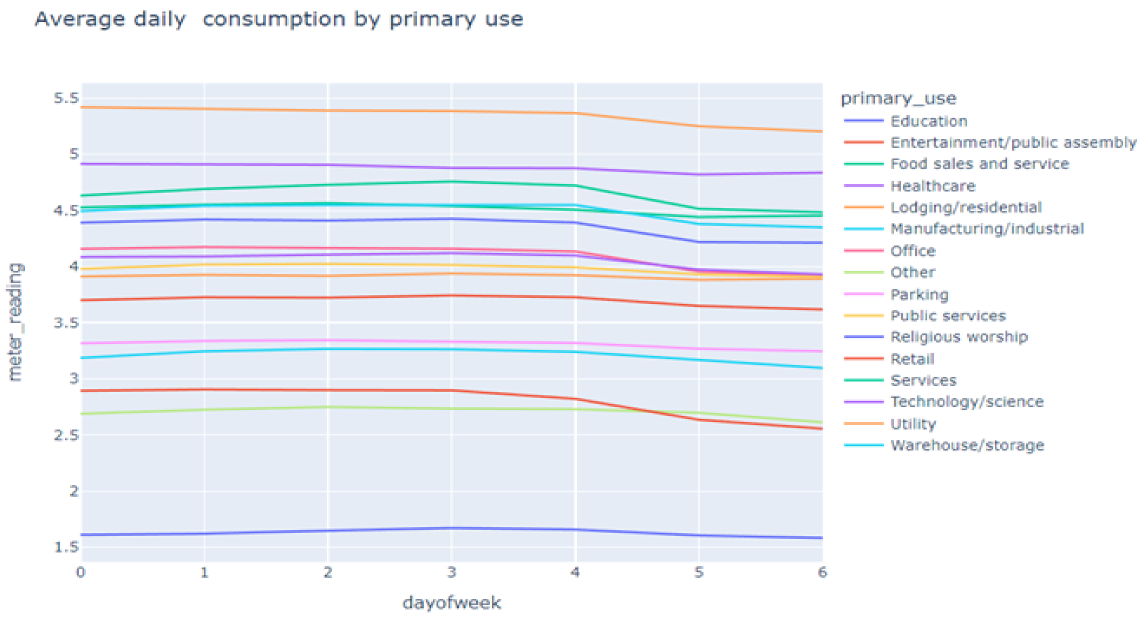


Fig 15: Average daily meter_reading_log by primary use

The figure above starts from 0 (Monday). Overall, the energy usage is stable on weekdays and it would go down on weekends.

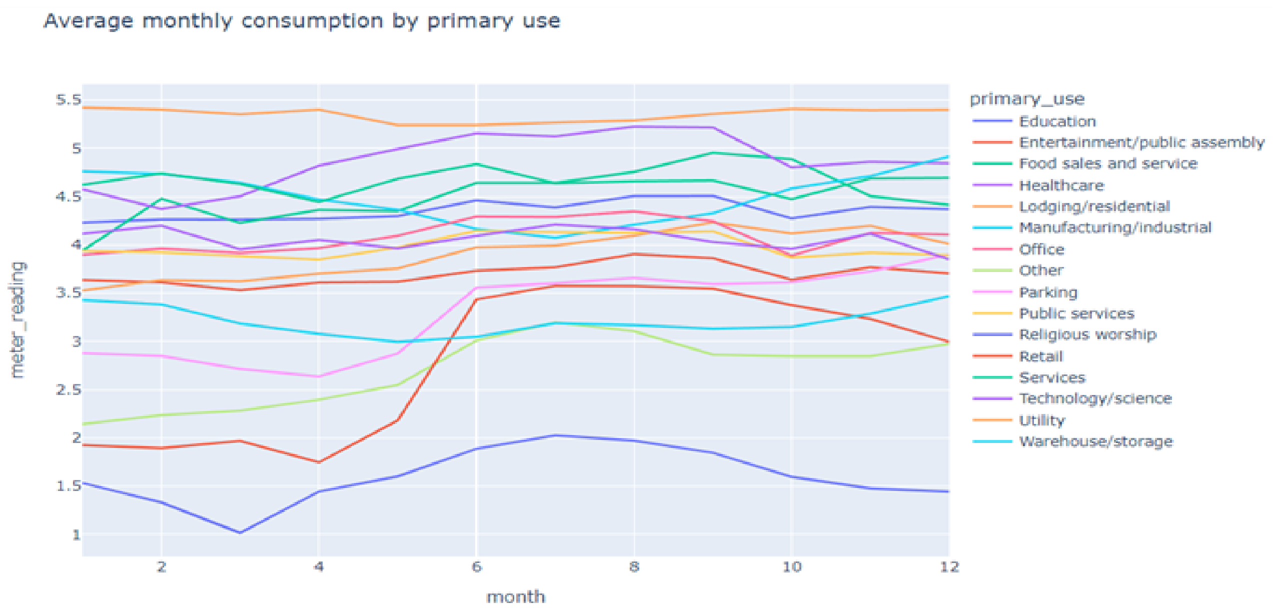


Fig 16: Average monthly meter_reading_log by primary use

In the month scale, we can find that Education, Entertainment, Healthcare, Office and Public service have the same trend in usage throughout the year. They will have a longer peak energy usage in June to October. Other usages are affected by the dual effects of periodicity and seasonality, as well as special holidays.

3.2.3 Meter

By creating the count plot of the meter type, we found that the electricity meter generated the most of the reading data whilst the hot water meter generated the least.

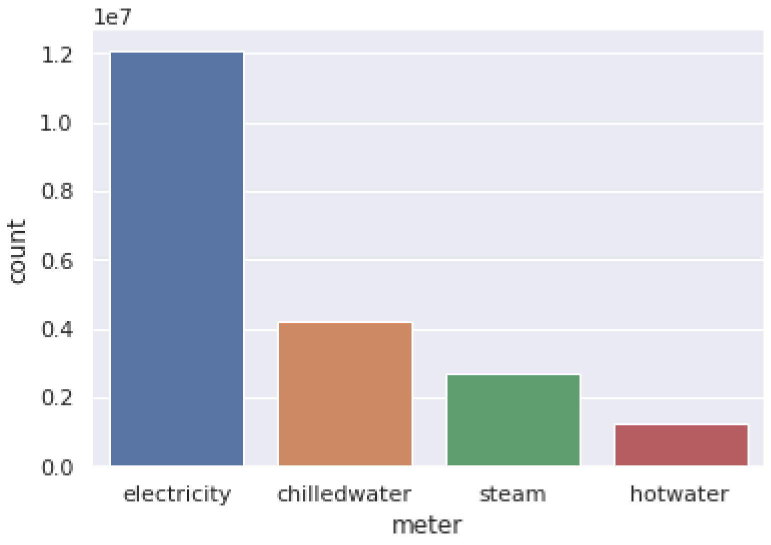


Fig 17: Count plot of meter types

However, the meter type counting is not always the same. There are 1448 buildings total, but not every building has every meter record at any time. For example, there are only 798 buildings that have meter 0 (electricity) records on March 13 2016, as the graph indicates below.

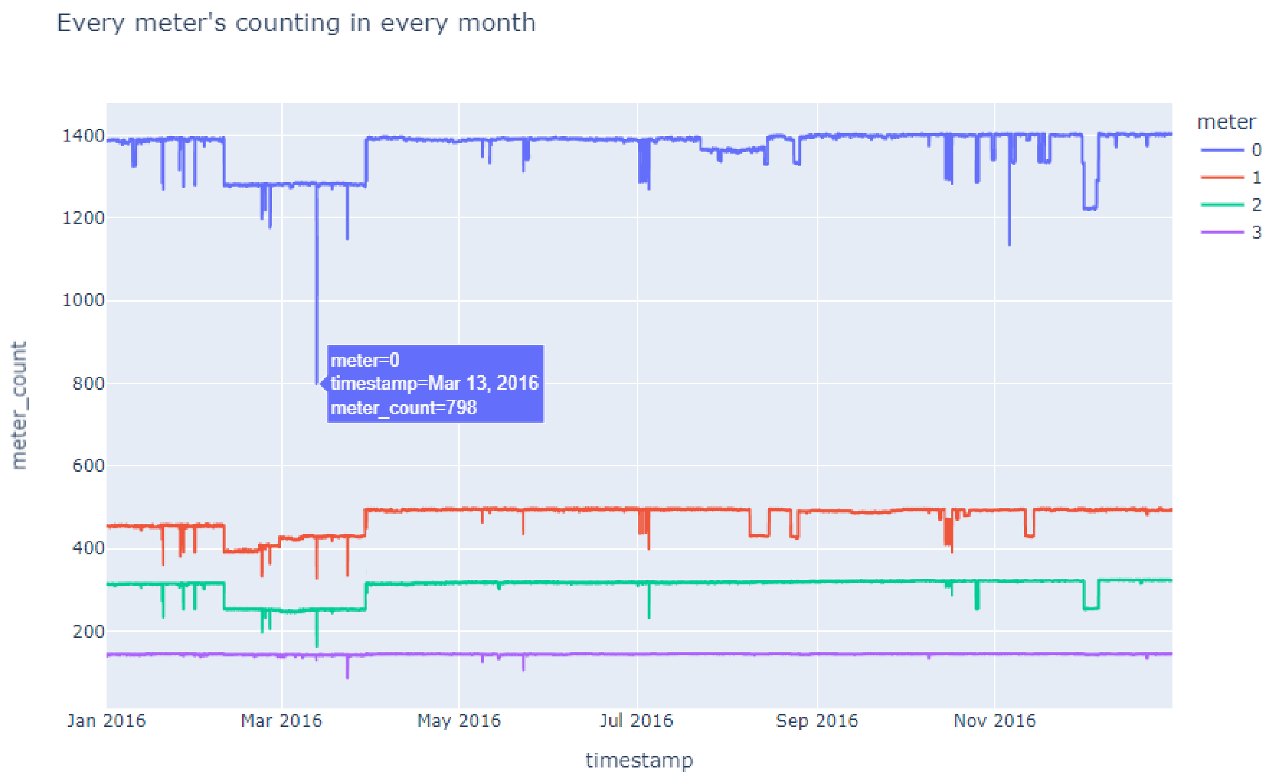


Fig 18: Meter count changing by timestamp

In terms of the consumption, the steam meter consumes the most energy whilst the hot water meter consumes the least.

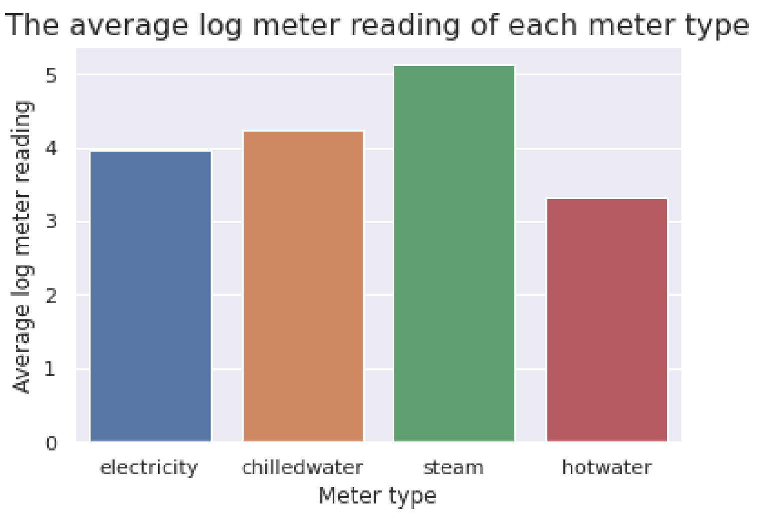


Fig 19: The average log meter reading of each meter type

3.2.4 Other features

The square feet is heavily skewed to the right according to the left distribution plot below. Most floor area of the building is between 0 to 400 thousands feets. After implementing the log transformation the skewness was fixed as the right part of the chart. And we add a new column ‘square_feet_log’ for the transferred values.

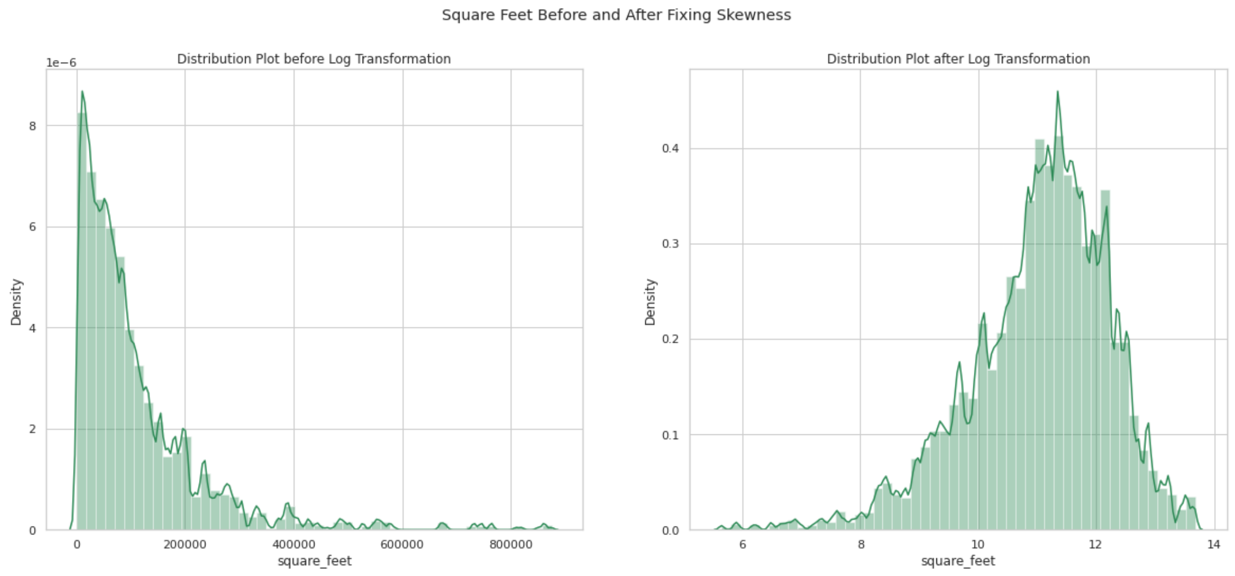


Fig 20: Square Feet Before and After Fixing Skewness

About the buildings’ height, most buildings are concentrated in floors 1-5. Not much of the buildings have floors greater than 10.

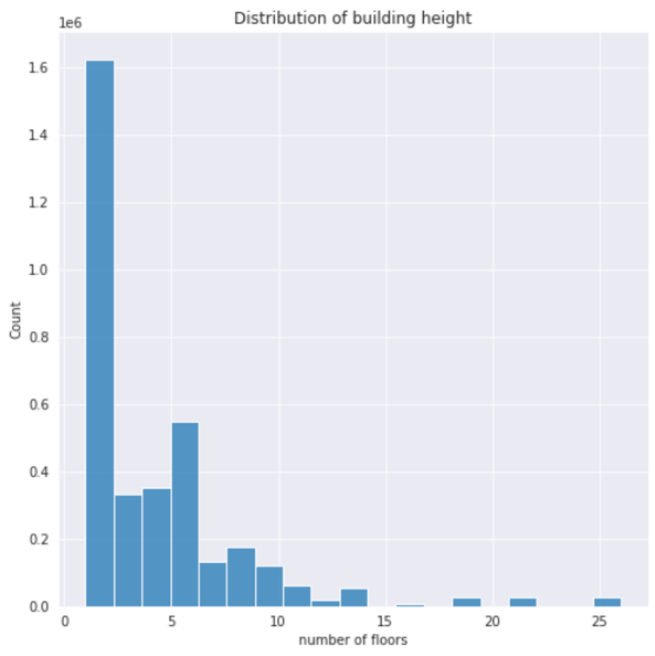


Fig 21: Distribution of building height

The ‘year built’ spread over 117 years, from 1900 to 2017. In general, most of the buildings were built between 1960 to 1980, peaking at around 1975.

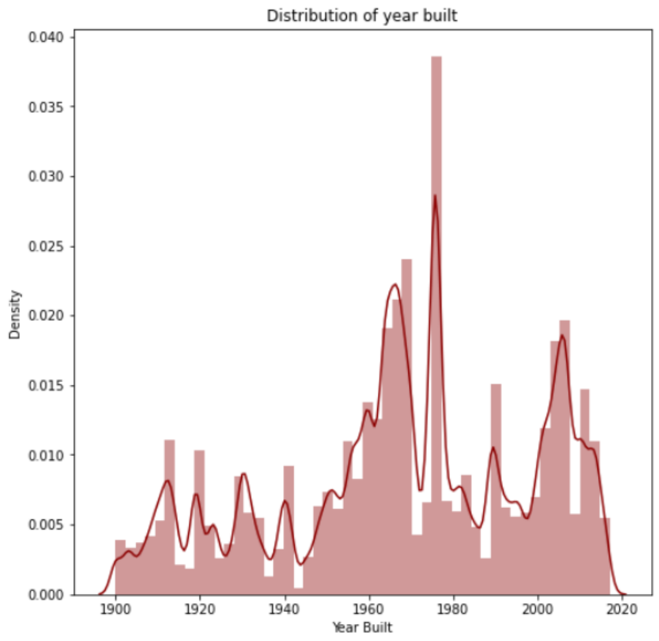


Fig 22: Distribution of construction being built time

The air temperature roughly follows a normal distribution, which peaks at around 14 degrees celsius and the most observations lie between -20-40 degrees celsius.

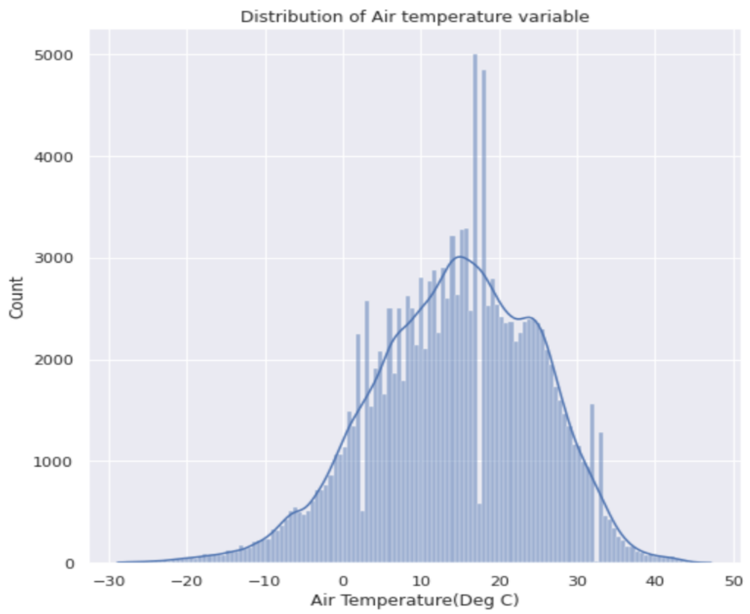


Fig 23: distribution of air temperature

The dew temperature is a bit skewed towards the left. Most observations lie between -30-25 degrees celsius.

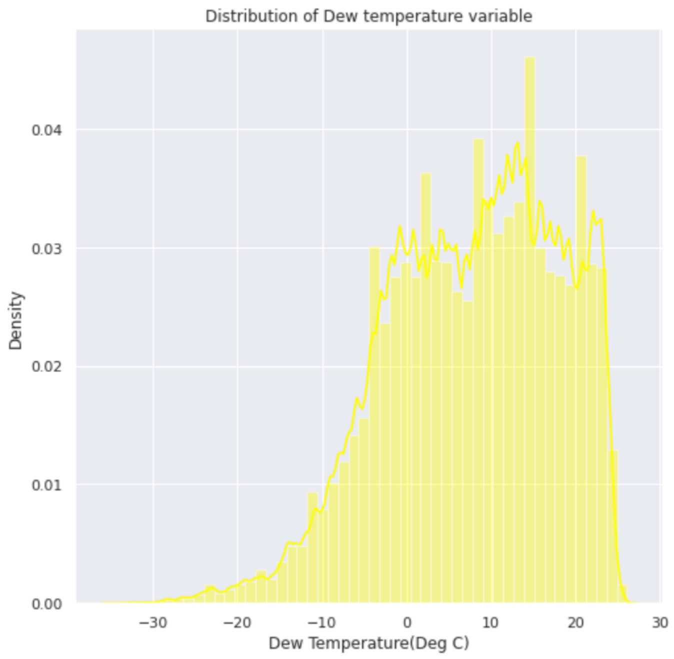


Fig 24: Distribution of dew temperature

The sea level pressure follows a normal distribution with maximum of the observations lying between 1000 to 1030 millibar.

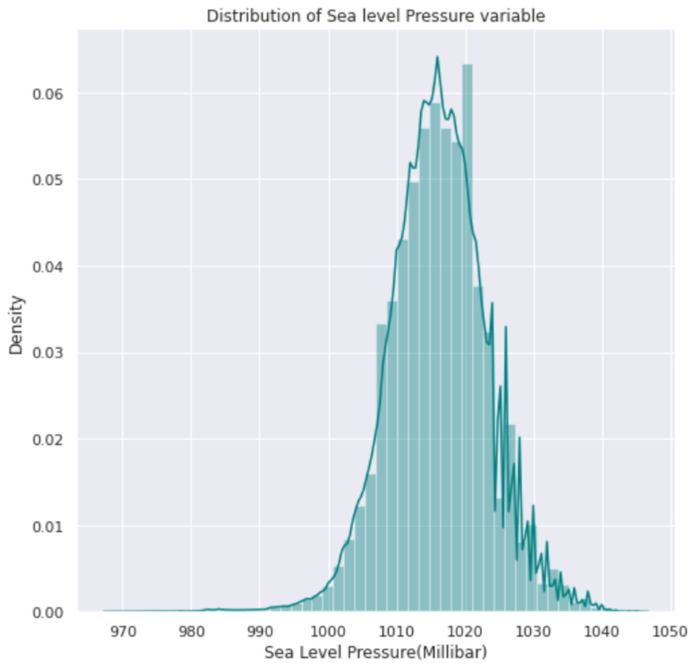


Fig 25: Distribution of sea level pressure

Majority of the observations have cloud coverage as 0 (Sky clear).

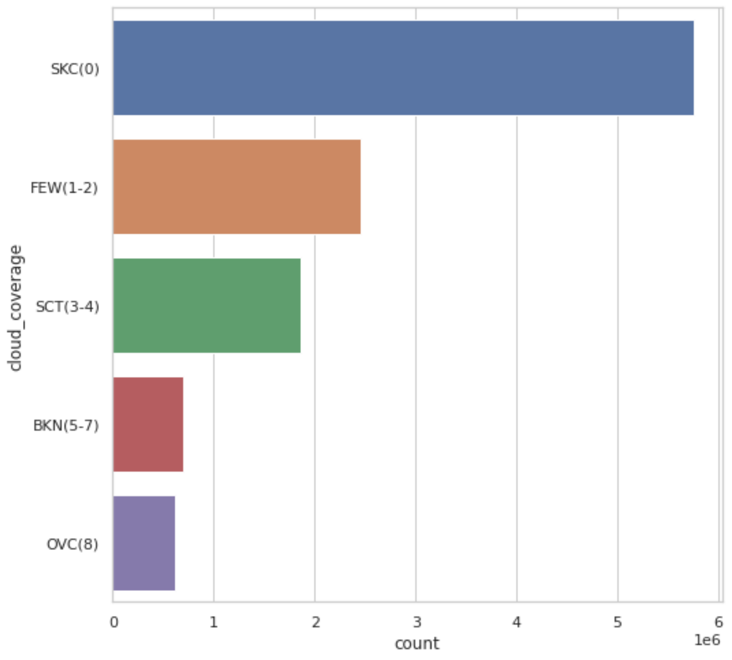


Fig 26 : Count plot of cloud coverage

The windrose diagram shows that most of the time the wind blows from the east direction (90 degrees), followed by the west direction(270 degrees). And most of the wind speed is between 0 to 11.4 m/s from any direction.

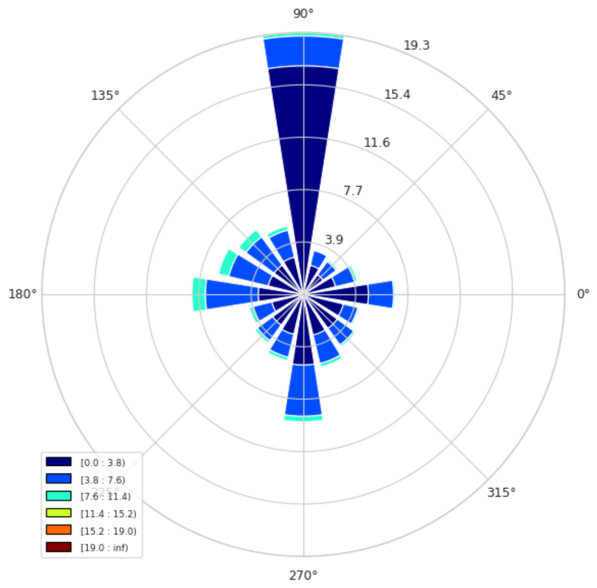


Fig 27 : Windrose diagram

3.3 Correlation analysis

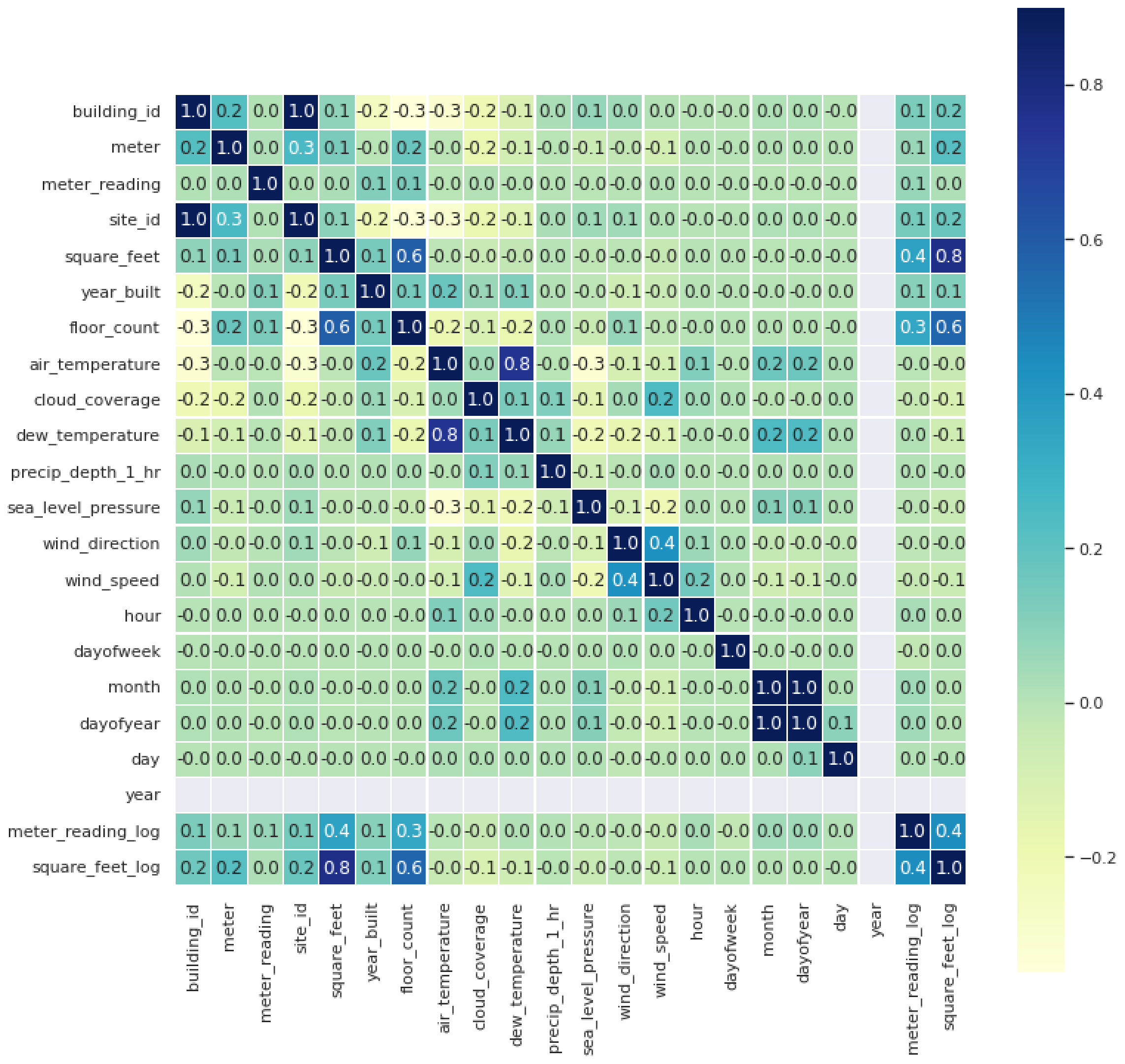


Fig 28: Correlation heatmap

The shade of colour indicates the strength of the correlation between variables. Based on the heatmap value and interests, we picked the following paired features to check correlationship.

(1). “floor_count” and “square_feet”

Although they have a 0.6 (correlation value) in the heatmap, which is the second highest correlation value, our review found that even at higher correlation values, its curve still did not strictly fit a statistically linear relationship .

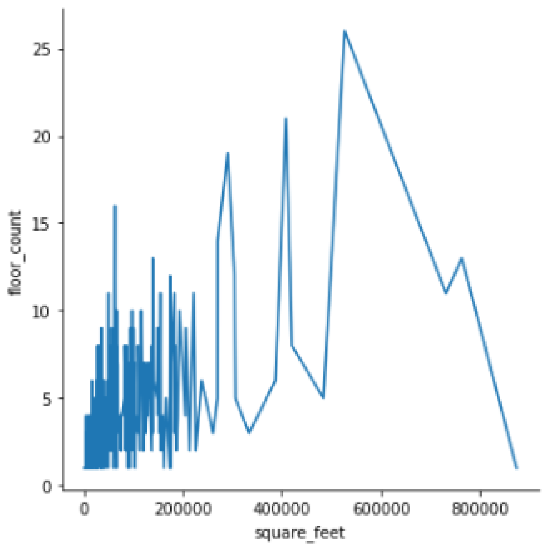


Fig 29: Correlation between floor_count and square_feet

(2). “dew_temperature” and “air_temperature”

This pair of variables obviously has a linear correlation. After checking the relevant instructions, it is also verified that the two are converted and must have correlation.

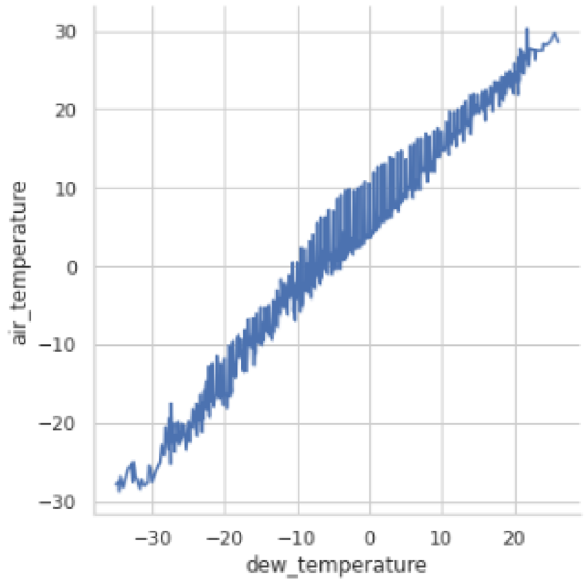


Fig 30: Correlation between dew_temperature and air_temperature

(3). “air_temperature” and “meter_reading_log”

The average meter readings are high when the temperature is negative, once it starts increasing, the meter consumption reduces. It again increases when the temperature rises above 15 degrees celsius.

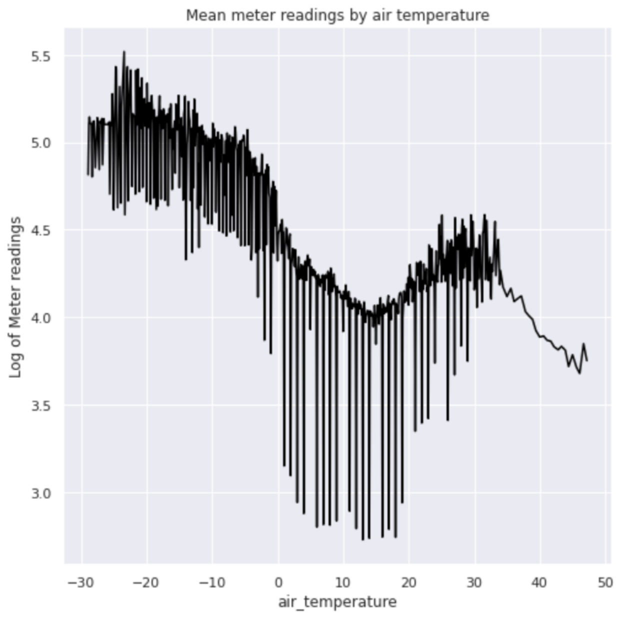


Fig 31: The mean of meter readings by air temperature

(4) 'meter_reading_log' and 'square_feet_log'

After implementing log transformation for both meter_reading and square_feet, we can see a clear positive relation between them. As the size of the building increases, the meter consumption also increases.

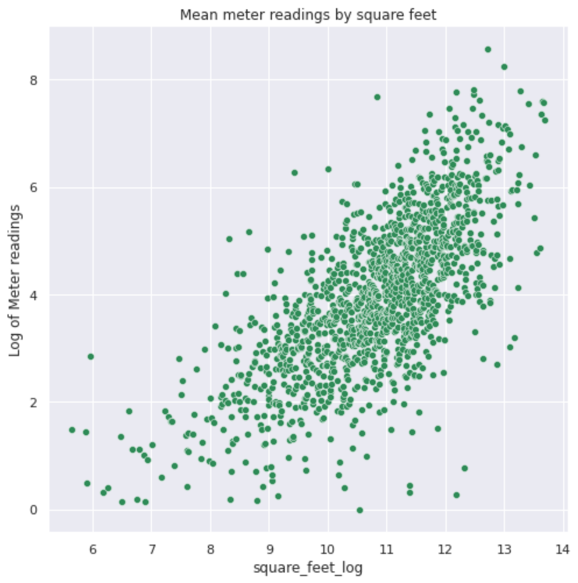


Fig 32 : Mean meter_reading_log by square_feet_log

4. Data pre-processing

Prior to running our chosen models the data was processed to support the demands of particular models and to achieve the best possible results. For each model: Neural Networks and Supervised Learning Models.
Due to the nature of the dataset, we carried out some pre-processing activities in T1 to form the dataset for modelling.

4.1 General pre-processing (for both models)

4.1.1 Correct the unit

The Kaggle challenge organisers pointed out an error in the issued datasets where the electric meter readings for site 0 were not properly converted to units of kWh and are in kBTU.

The suggested correction, to multiply the site 0 electric meter readings by 0.2931 to convert to kWh, was made in T1.

4.1.2 Outliers

Group removal of outliers was carried out, all numerical features with a standard deviation larger than 3 were removed.
Following that, there were still couple of buildings were identified with extremely high meter reading values.

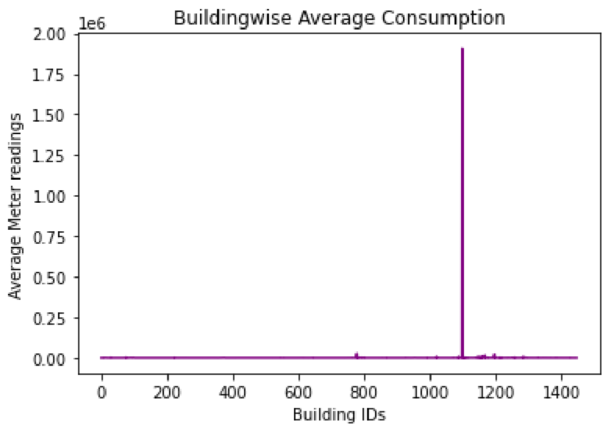


Fig 33 : Before dropping building 1099

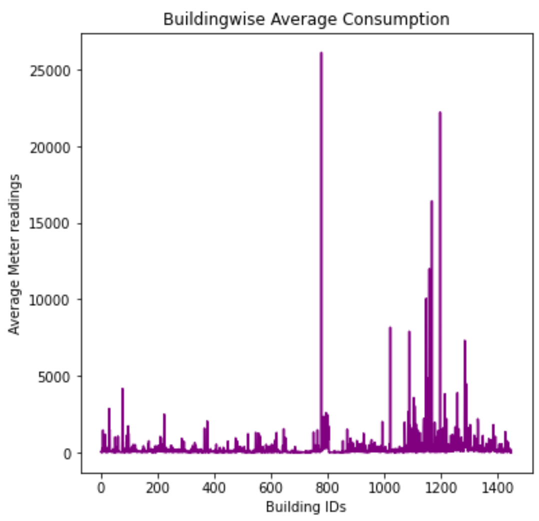


Fig 34 : After dropping building 1099

The building; building_id 1099 was dropped from the dataset. In addition meter readings equal to 0 were dropped.

4.1.3 Missing values

Referring to the missing value check in the T1 (3.1). Analysis of the dataset found a number of missing values with floor_count at 82.6% and year_bulit at 59.0% being particularly high. Replacing so many missing values would risk distorting the data so both features were dropped.

4.2 Model Specific Pre-Processing

4.2.1 Neural Network Pre-Processing

- Data Refinement: To deal with data more efficiently and make model implementation smoother, a group by statement is used which groups the training dataset of five features and aggregates it on six more features. Tests were carried out to find the most influential features, others were removed. A specific order was used to groupby the dataset which allowed for further categorization of our dataset.

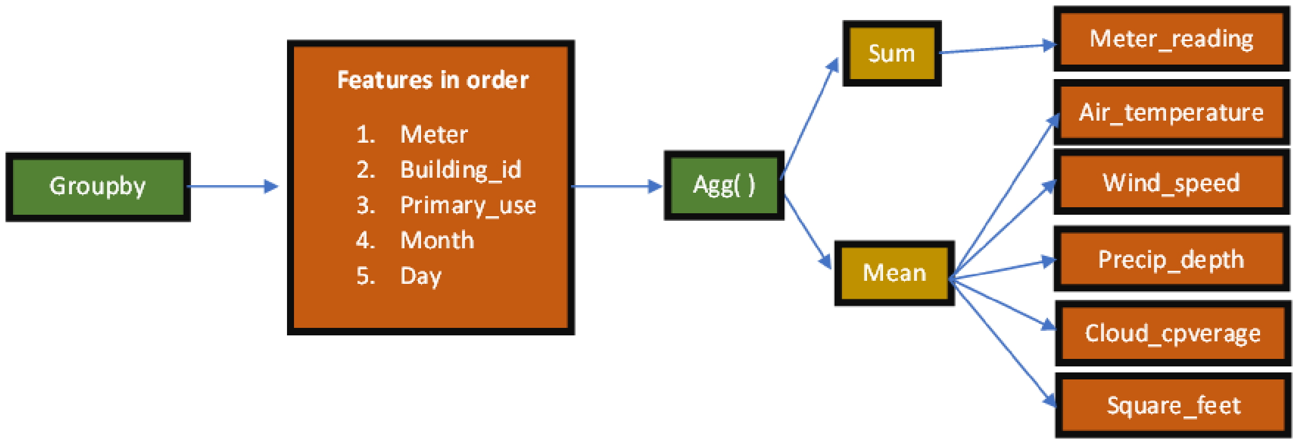


Fig 35 : Flow Chart explaining the groupby process

- The diagram above also shows the imputation of the dataset.
- The next step is to convert features like “wind_speed”, “air_temperature” and “precip_depth_1_hr” to “float32” to allow model implementation.
- After applying groupby null values were found in the following features: “wind_speed”, “air_temperature”, “precip_depth_1_hr”, “cloud_coverage” and “square_feet” subsequently these were treated during the preprocessing stage.
- Feature named “primary_use” was label encoded as it was previously categorical, so that it can be used during model implementation.
- Scaling of numerical features wasn’t carried out because during the groupby stage aggregate of features was taken with respect to the 5 groupby features upon observing them closely it was more sensible not to scale them since they had very little standard deviation so scaling them would not positively effect the model performance.

4.2.2 Supervised Learning Models Pre-Processing

- Features with significant levels of missing values (Fig 5: Missing value and duplicates): air_temperature, dew_temperature, wind_direction, wind_speed, cloud_coverage, precip_depth_1_hr and sea_level_pressure. These missing values were replaced with the mean or median value for each feature.

- Two new features were added:
  - Season to specify spring, summer, autumn or winter.
  - isDayTime to flag if it is daytime or not.

- A pipeline was used to:
  - Transform the categorical features using LabelEncoder (“primary_use”,”season”)
  - Drop the feature timestamp as this has already been split into five other time features
  - Deal with NaNs, using means and medians to fill the data.

5. Methodology & Implementation

5.1 Regression Models

As a way to test different types of learner methods we decided to use 2 regression models, KNN which is a lazy learner and Decision tree which is an eager learner. We decided this as a means to not only allow a fair comparison between the regression models and the neural network but also allows us to see if one form of learning performs better than the other. The kaggle scores later in the report gives a clear picture of which regression was a better fit for this dataset.

5.1.1 Feature Selection - Univariate Selection

Univariate Selection was the method of feature selection as it was very easy to implement and gave a very strong idea of the features and their relationships to the data. This is further shown when the accuracy of all the classifiers improved when dropping “precip_depth_1_hr” and “month” by a significant degree.

5.1.2 Parameter Tuning

We decided to use accuracy_score from the sklearn.metrics library as it allows for fluidity in the parameter testing and allowed us to go through each parameter separately to see which was the best for accuracy. The scores are as follows, KNN Accuracy: 10.71%, Decision Tree Accuracy: 29.60%. By using this we can decide which classifications will have the highest chance of success and make assumptions about the kaggle score.

5.2 Recurrent Neural Networks

A recurrent neural network (RNN) is a type of artificial neural network which either uses sequential data or time-series data. Like convolutional neural networks (CNNs), recurrent neural networks utilise training data to learn. The main thing that distinguishes them is their “memory” factor as they take information from previous input and influence current input and outputs. The classical difference between a feed-forward network(Right) and a recurrent neural network(Left) can be seen from the figure below.

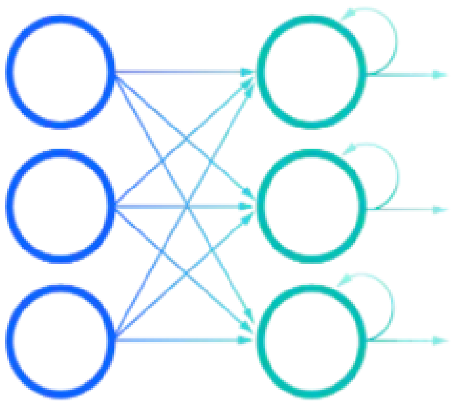
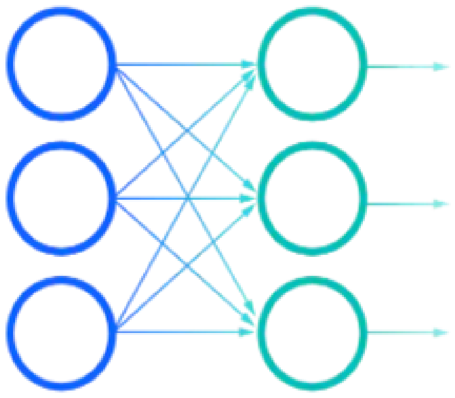



Fig 36 : RNN (Left) and Feed Forward Neural Network (Right)

Architecture of a single unit of RNN is as follows here, it takes input from the previous step and current state Xt and uses tanh as an activation function; in some cases only the previous input is needed for the model to work but this is very rare.

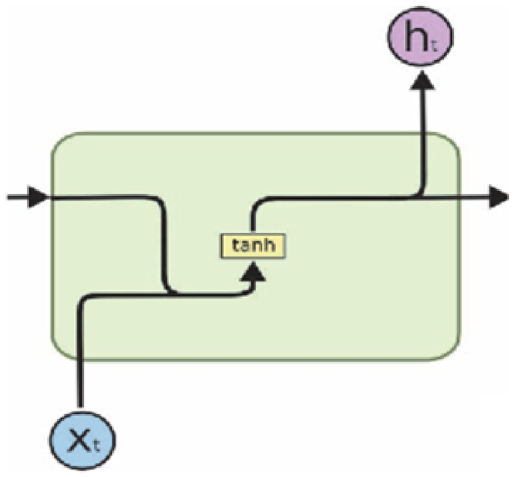


Fig 37: RNN Architecture

When a normal RNN is applied to long sequences it tends to lose the information because of its inability to store long sequences since it only stores the latest information this problem is known as the vanishing gradients.

To overcome this Long short-term memory (LSTM) and Gated Recurrent Unit (GRU) architectures are used which are based on RNN.

5.2.1 Gated Recurrent Unit (GRU)

To cater for the vanishing gradient problem present in RNN. GRU uses two gate operating mechanisms called update gate and reset gate.

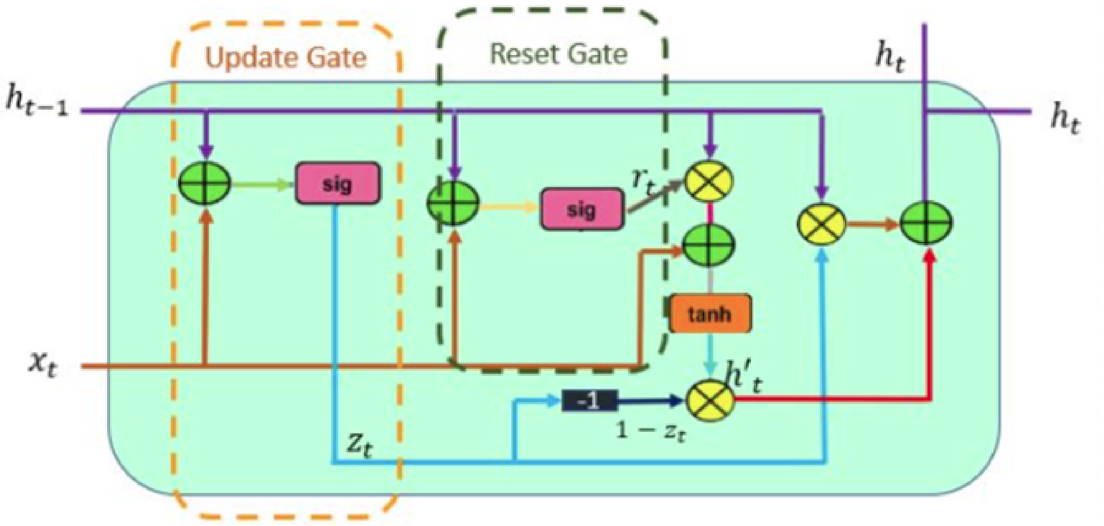


Fig 38: GRU Architecture

The basic working of GRU can be explained by explaining its two gates

Table 3 : GRU Gate Configuration

Gate Name	Action
Update Gate	● Responsible for the amount of previous information that passes to the next state. ● Can send all the information from the past and eliminate the risk of vanishing gradient.
Reset Gate	● Decides whether the previous cell state is important or not.

5.2.2 Long Short Term Memory (LSTM)

LSTM is a kind of RNN that was designed to avoid long-term dependency problems by remembering long sequences for a longer period of time.

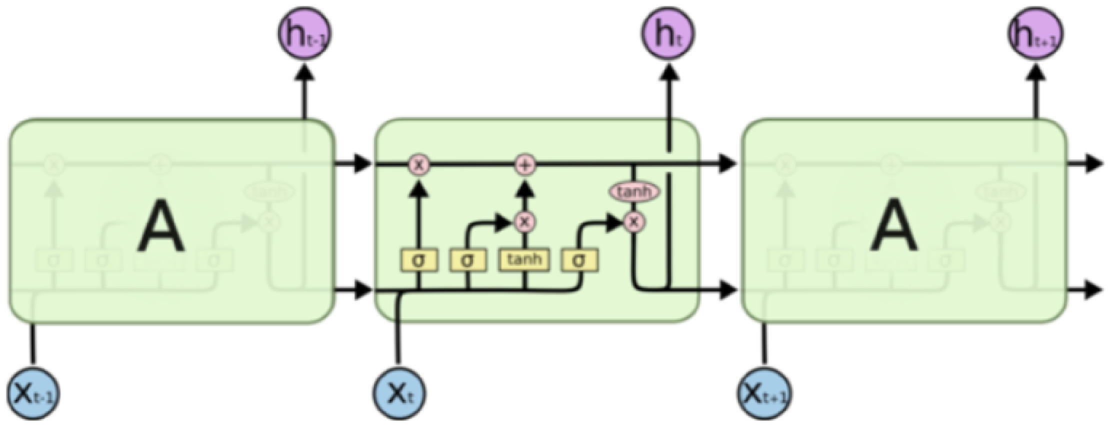


Fig 39: LSTM Architecture

An LSTM architecture takes input from three different states

- Current input state
- Short term memory from the previous cell
- Long term memory

This architecture uses gates in the form of a filter to select and pass information among these cells the actions taken by these can be shown in the form of a table.

Table 4: LSTM Gate Configuration

Gate Name	Action
Input Gate	● Decides which information from current input and short-term memory needs to be stored in the long-term memory.
Forget Gate	● Decides which information from long term memory be kept or discarded ● Multiplies incoming long-term memory by a forget vector generated by the current input and incoming short memory.
Output Gate	● Takes current input, the previous short term memory, and newly computed long term memory to produce new short term memory

5.2.3 General Setting of GRU/ LSTM

During preprocessing we divided our data into two halves to bypass system limitations. All the code cells for neural networks were run on colab pro because from preprocessing to implementation they needed RAM upto 25 GB.

First, we split our data using train_test_split function which splits data into two sets of X and y both for the training set and validation set here the test size used is 0.2.
We have defined a function “transform” which takes the model setup, training data, validation data, batch size, and callbacks. This function does three things

- This converts the training and validation data’s shape to 3D shape using the reshape function since this is the shape (samples, time steps, and features) required by the model.

- During the exploratory analysis, it was found that our target feature “meter_reading” is heavily skewed so to deal with that this function applies log transformation to this.

- In the end, the function fits the model on 3D transformed train and validation sets and uses batch size and all the necessary callbacks being used during the function call.

Early stopping criteria have been used to put a stop to our model if there’s no improvement in the output here; the patience set for the epochs with no change is 5 and the entity being monitored is validation RMSE.

5.2.4 GRU/LSTM Implementation

The function model_gru/model_lstm function is used to set up our GRU/LSTM models the key features of this setup are

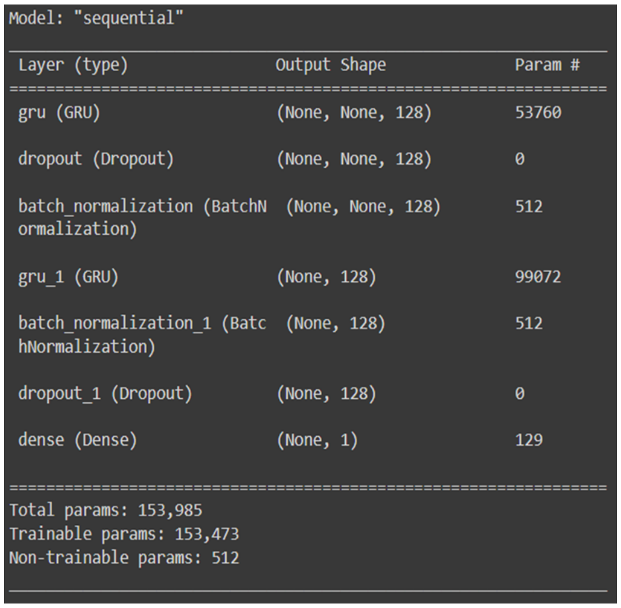
- Stacked GRU/LSTM architecture is used here to reinforce learning at a deeper level here there are two GRU/LSTM layers with 128 neurons that are used back-to-back the argument return_sequence in the first layers is set to TRUE so that it outputs a 3D array as an input for the subsequent GRU/LSTM layer.

- We have used two batch normalisation layers used to stabilise the learning process and to reduce the number of training epochs required to train the network.

- Two drop out layers with a drop rate of 0.2 are being used in the model set up to limit neurons in the same layer to extract the same or similar hidden features from input data; also known as Co-adaptation.

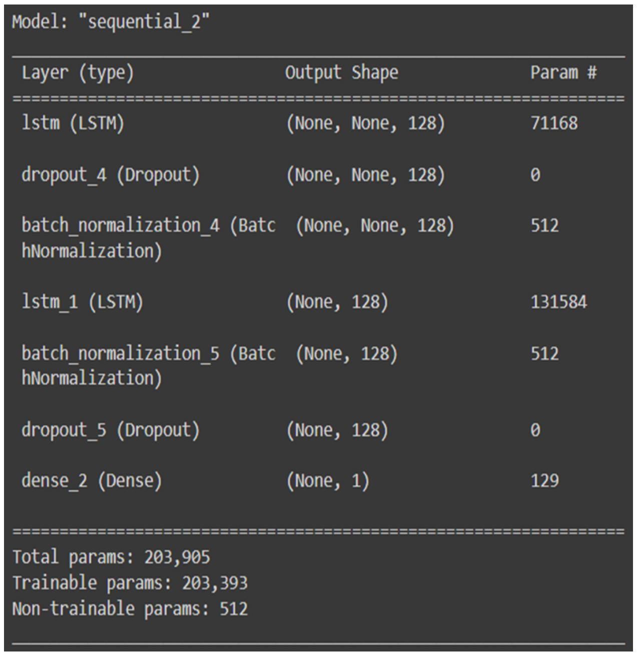
- Metrics used here to evaluate our model are root mean squared value and accuracy but our analysis is based on RMSE.

- The default optimizer used is Root Mean Squared Propagation ("rmsprop") because as compared to "Adam" optimizer "rmsprop" restricts the gradient oscillations in the vertical direction. So by using RMSprop, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, None, 128)         53760

 dropout (Dropout)           (None, None, 128)         0

 batch_normalization (BatchN  (None, None, 128)        512
 ormalization)

 gru_1 (GRU)                 (None, 128)               99072

 batch_normalization_1 (Batc  (None, 128)              512
 hNormalization)

 dropout_1 (Dropout)         (None, 128)               0

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 153,985
Trainable params: 153,473
Non-trainable params: 512
_________________________________________________________________
```

Fig 40: GRU Model Summary



```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, None, 128)         71168

 dropout_4 (Dropout)         (None, None, 128)         0

 batch_normalization_4 (Batc  (None, None, 128)        512
 hNormalization)

 lstm_1 (LSTM)               (None, 128)               131584

 batch_normalization_5 (Batc  (None, 128)              512
 hNormalization)

 dropout_5 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 203,905
Trainable params: 203,393
Non-trainable params: 512
_________________________________________________________________
```

Fig 41: LSTM Model Summary

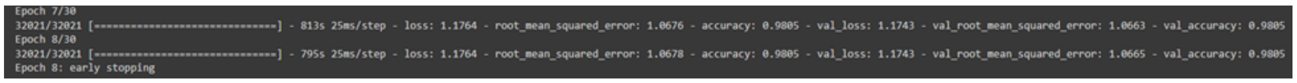
6. Results

6.1 Neural Network Results

The results for neural networks reported here are for two halves of the data and we'll be looking at the RMSE values for training and validation sets and will also compare RMSE of train vs val loss for number of epochs in a line graph

6.1.1 GRU

- For the first half of data the model stopped after 8 epochs



```
Epoch 7/30
32021/32021 [==============================] - 813s 25ms/step - loss: 1.1764 - root_mean_squared_error: 1.0676 - accuracy: 0.9805 - val_loss: 1.1743 - val_root_mean_squared_error: 1.0663 - val_accuracy: 0.9805
Epoch 8/30
32021/32021 [==============================] - 795s 25ms/step - loss: 1.1764 - root_mean_squared_error: 1.0678 - accuracy: 0.9805 - val_loss: 1.1743 - val_root_mean_squared_error: 1.0665 - val_accuracy: 0.9805
Epoch 8: early stopping
```

Fig 42: GRU Model Output (1st Half)

- The line graph for train and val loss RMSE is as follows

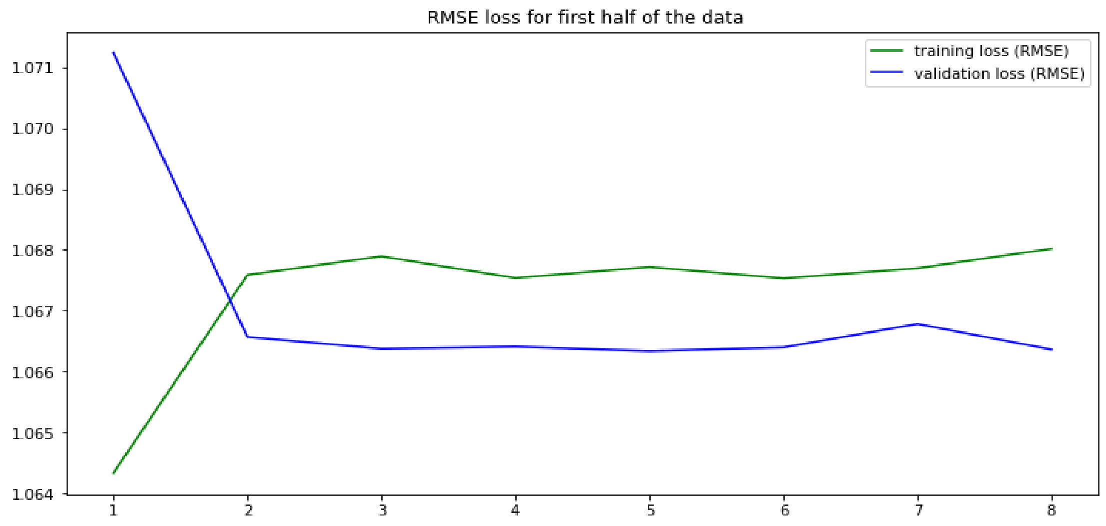
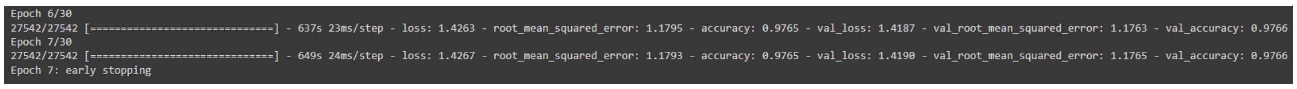


Fig 43: GRU Model RMSE Line Graph(1st Half)

- For the second half of the data model stopped after 7 epochs



```
Epoch 6/30
27542/27542 [==============================] - 637s 23ms/step - loss: 1.4263 - root_mean_squared_error: 1.1795 - accuracy: 0.9765 - val_loss: 1.4187 - val_root_mean_squared_error: 1.1763 - val_accuracy: 0.9766
Epoch 7/30
27542/27542 [==============================] - 649s 24ms/step - loss: 1.4267 - root_mean_squared_error: 1.1793 - accuracy: 0.9765 - val_loss: 1.4190 - val_root_mean_squared_error: 1.1765 - val_accuracy: 0.9766
Epoch 7: early stopping
```

Fig 44: GRU Model Output (2nd Half)

- The line graph for train and val loss RMSE for second half of data is as follows

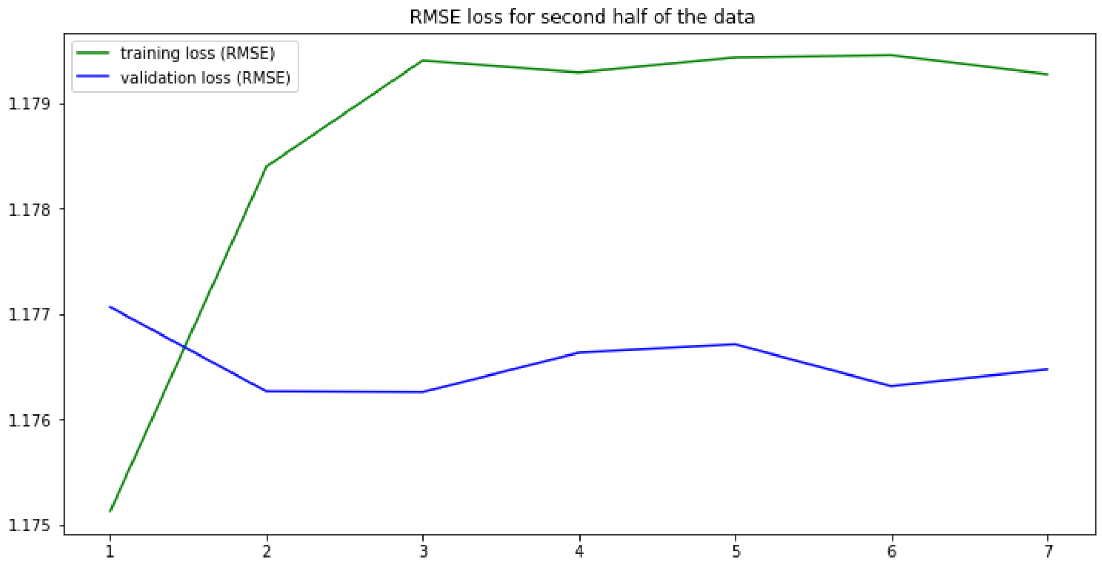
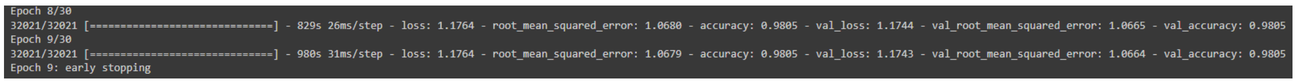


Fig 45: GRU Model RMSE Line Graph(2nd Half)

6.1.2 LSTM

- For the first half of data the model stopped after 9 epochs



```
Epoch 8/30
32021/32021 [==============================] - 829s 26ms/step - loss: 1.1764 - root_mean_squared_error: 1.0680 - accuracy: 0.9805 - val_loss: 1.1744 - val_root_mean_squared_error: 1.0665 - val_accuracy: 0.9805
Epoch 9/30
32021/32021 [==============================] - 980s 31ms/step - loss: 1.1764 - root_mean_squared_error: 1.0679 - accuracy: 0.9805 - val_loss: 1.1743 - val_root_mean_squared_error: 1.0664 - val_accuracy: 0.9805
Epoch 9: early stopping
```

Fig 46: LSTM Model Output (1st Half)

- The line graph for train and val loss RMSE is as follows

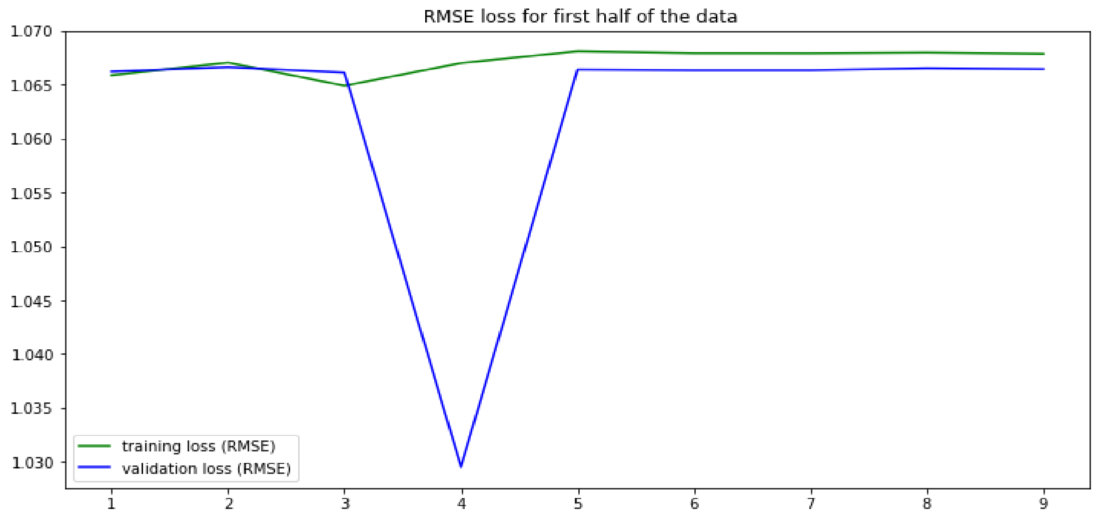
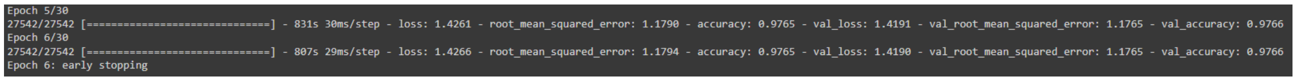


Fig 47: LSTM RMSE Line Graph (1st Half)

- For the second half of the data model stopped after 6 epochs



```
Epoch 5/30
27542/27542 [==============================] - 831s 30ms/step - loss: 1.4261 - root_mean_squared_error: 1.1790 - accuracy: 0.9765 - val_loss: 1.4191 - val_root_mean_squared_error: 1.1765 - val_accuracy: 0.9766
Epoch 6/30
27542/27542 [==============================] - 807s 29ms/step - loss: 1.4266 - root_mean_squared_error: 1.1794 - accuracy: 0.9765 - val_loss: 1.4190 - val_root_mean_squared_error: 1.1765 - val_accuracy: 0.9766
Epoch 6: early stopping
```

Fig 48: LSTM Model Output (2nd Half)

- The line graph for train and val loss RMSE for second half of data is as follows

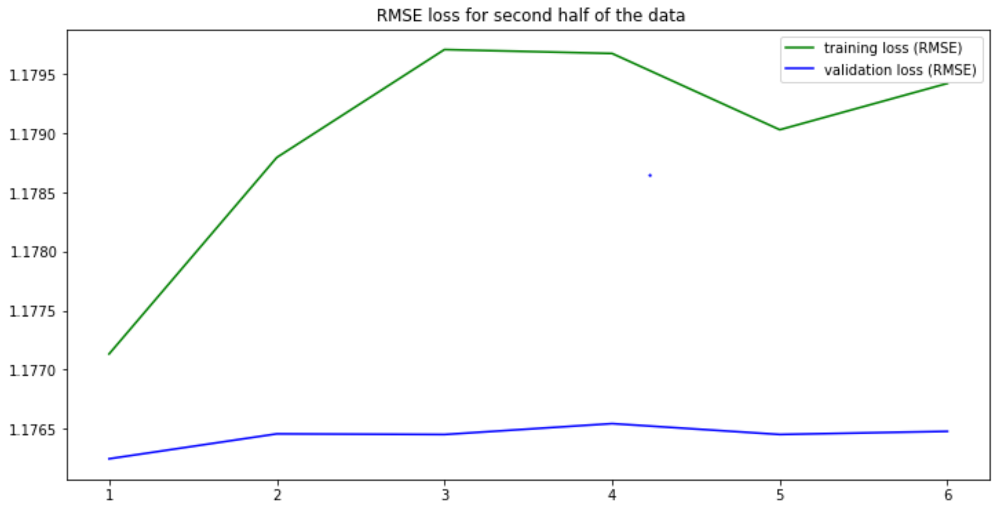
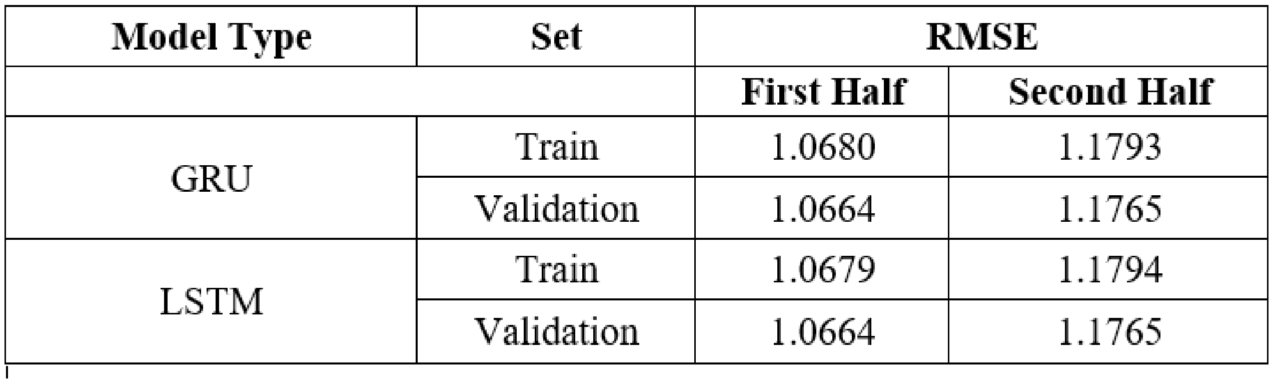


Fig 49: LSTM RMSE Line Graph (2nd Half)

6.1.3 GRU vs LSTM

The root mean squared error comparison between GRU and LSTM is as follows

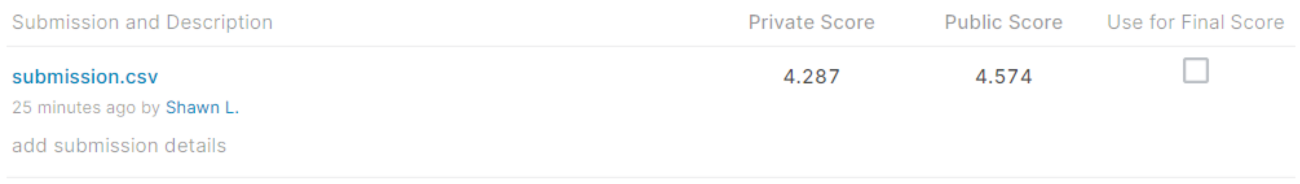


Model Type	**Set**	**RMSE**	
		First Half	**Second Half**
GRU	Train	1.0680	1.1793
	Validation	1.0664	1.1765
LSTM	Train	1.0679	1.1794
	Validation	1.0664	1.1765

Fig 50: The comparison between GRU and LSTM

Both models performed almost the same but GRU model proved itself to be more stable and was able to converge quickly in lesser epochs as compared to LSTM.

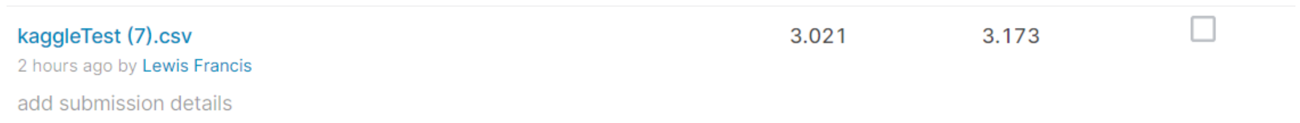
The kaggle score for GRU based mode which was trained on only half of training data and was used to predict the whole test set data is as follows.



Submission and Description	Private Score	Public Score	Use for Final Score
submission.csv 25 minutes ago by Shawn L. add submission details	4.287	4.574	☐

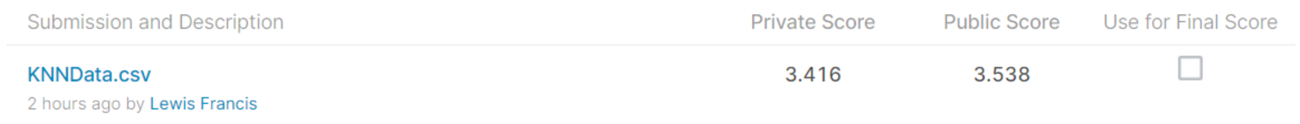
Fig 51: GRU Kaggle Score

6.2 Regression Model Results



kaggleTest (7).csv 2 hours ago by Lewis Francis add submission details	3.021	3.173	☐

Fig 52: Decision Tree Regression Kaggle Score



Submission and Description	Private Score	Public Score	Use for Final Score
KNNData.csv 2 hours ago by Lewis Francis	3.416	3.538	☐

Fig 53: KNN Regression Score Kaggle Score

We can see that both regression models had a reasonably low kaggle score which indicates that the accuracy is reasonably high. As well as this we can see the confirmation from earlier in the report where we see that the decision tree classifier has a higher accuracy score, we also see the same thing with the kaggle score with a reasonably high difference between the 2 regression models.

7. Analysis

During the implementation of the regression models we decided that in order to keep a high enough accuracy while also keeping resources low that we would only use half of the training set, as you can see from the kaggle scores above I believe this was successful as we have maintained a good score without compromising on using excess data.

During the implementation of neural networks it was observed that training loss is higher is than validation loss this can be explained as during training the disabling neurons (dropout), some of the information about each sample is lost, and the subsequent layers attempt to construct predictions based on incomplete representations but during the validation the network has its full computational power available that's why it's performing better than the training data.

Although the RMSE values for both training and validation set was quite good and showed that model worked well. However the Kaggle score wasn't upto the mark this can be explained by one of two reasons

- The GRU model trained was only on one half of the data and the prediction was made on the full test test this can account for low score
- Use of log operation on meter reading because of skewness and then use of exponential after prediction to convert it back could've caused the score to fall

8. Conclusion and future work

As a conclusion for the regression models we can see that the Decision tree regression model has the highest kaggle score when compared to the neural network that we have implemented as well as the KNN regression model, realistically this should not occur as we'd assume a neural network would have a higher accuracy but there were some problems when developing the neural network.

In terms of the neural networks, GRU is computationally more efficient. From the performance we can clearly state that Recurrent Neural Networks are a good fit for this energy prediction problem.

However, we were much aware throughout this project we were lacking behind with time and computational power. Due to this, for future work, we recommend implementing a more robust hyperparameter tuning process.We also believe that if we obtain historical data, it will improve our model and help to identify patterns.We would further apply transformations both on our model features ,training and validation set in order to check which variation gives us the best score on kaggle. Furthermore, a model trained on a full training set would result in a good prediction score which we couldn't train due to computational limitations.

9. Reference List

Ahmad, T., Chen, H., Huang, R., Yabin, G., Wang, J., Shair, J., Akram, H.M.A., Mohsan, S.A.H., Kazim, M. 2018. Supervised based machine learning models for short, medium and long-term energy prediction in distinct building environment, Energy, Volume 158, 2018, doi:10.1016/j.energy.2018.05.169.

Brown, M., Barrington-Leigh, C., Brown, Z. 2011. Kernel regression for real-time building energy analysis, Journal of Building Performance Simulation, 5(4):1-14, doi:10.1080/19401493.2011.577539.

Brownlee, J. 2022. Stacked Long Short-Term Memory Networks. Available at: https://machinelearningmastery.com/stacked-long-short-term-memory-networks/ [Accessed: 19 April 2022].

Cao, T., Gao, L., Aute, V.C., Hwang, Y. 2021. A Data-driv A Data-driven Model for Generalized Building Ener Predictions. International High Performance Buildings Conference. Paper 361. https://docs.lib.purdue.edu/ihpbc/361.

Dhar, V. 2021. ASHRAE- Great Energy Predictor III- A Machine Learning Case Study. Available at: https://medium.com/analytics-vidhya [Accessed: 13 April 2022].

Eseye, A.T., Lehtonen, M., Tukia, T., Uimonen, S., Millar, R.J. 2019. Machine Learning Based Integrated Feature Selection Approach for Improved Electricity Demand Forecasting in Decentralized Energy Systems, IEEE Access, volume 7, pp 91463-91475, doi: 10.1109/ACCESS.2019.2924685.

Fu, C., Miller, C. 2022. Using Google Trends as a proxy for occupant behavior to predict building energy consumption, Applied Energy 310, 118343. doi:10.1016/j.apenergy.2021.118343

Htoon, K., 2020. Log Transformation: Purpose and Interpretation. Available at: https://medium.com/@kyawsawhtoon/log-transformation-purpose-and-interpretation-9444b4b049c9 [Accessed 16 April 2022].

IBM Cloud Education. 2022. What are Recurrent Neural Networks?. Available at: https://www.ibm.com/cloud/learn/recurrent-neural-networks [Accessed: 17 April 2022].

Lendave, V. 2022. LSTM Vs GRU in Recurrent Neural Network: A Comparative Study. Available at: https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/ [Accessed: 19 April 2022].

Miller, C, et al. 2020. The ASHRAE great energy predictor III competition: Overview and results. Science and Technology for the Built Environment 26.10, pp. 1427-1447.

Miller, C., Kathirgamanathan, A., Picchetti, B., Arjunan, P., Young Park, J., Nagy, Z., Raftery, P., Hobson, B.W., Shi, Z., Meggers, F. 2020. The Building Data Genome Project 2, energy meter data from the ASHRAE Great Energy Predictor III competition, Sci Data, 7: 368. doi: 10.1038/s41597-020-00712-x

Nitin1901. 2022. Dropout in Neural Networks. Available at: https://www.geeksforgeeks.org/dropout-in-neural-networks/ [Accessed: 19 April 2022].

Paterakis, N. G., Mocanu, E., Gibescu, M., Stappers, B., van Alst, W. 2017. Deep learning versus traditional machine learning methods for aggregated energy demand prediction, IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), pp. 1-6, doi: 10.1109/ISGTEurope.2017.8260289.

Pham, A-D., Ngo, N-T., Truong, T.T.H., Huynh, N-T., Truong, N-S. 2020. Predicting energy consumption in multiple buildings using machine learning for improving energy efficiency and sustainability, Journal of Cleaner Production, Volume 260, 121082, doi:10.1016/j.jclepro.2020.121082.

Smiley, S. 2020. Estimating Counterfactual Energy Usage of Buildings with Machine Learning. Available at: https://towardsdatascience.com/ [Accessed: 13 April 2022].