

# Final Year Report

Smit Jitesh

Supervisor: Yuhua Li

May 2022

School of Computer Science and Informatics  
Cardiff University

## **Abstract**

Currently there is a need to reduce energy consumption for future sustainability of this planet, one way we can target this is by reducing the energy using within buildings. We will devise some Machine Learning models to forecast the meter-readings within these buildings, where we will take some large dataset, process it, develop our models and then use these to make some conclusions. We will try to optimise the models through pre-processing and hyperparameter tuning stages. At the end we will determine what model is able to most accurately model building energy usage, and the limitations from this study.

## Acknowledgements

I would like to thank my supervisor Yuhua Li, for all the guidance that I've been given throughout this project. I would also like to thank him for all the recommendations and learning material suggested, that has helped develop my knowledge for this study.

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Aims and Goals of the Project</b>	<b>7</b>
<b>3</b>	<b>Background</b>	<b>8</b>
3.1	Problem Classification . . . . .	8
3.2	Machine Learning Algorithms . . . . .	9
3.2.1	Linear Regression . . . . .	9
3.2.2	Decision Tree's . . . . .	12
3.2.3	Gradient Boosting Decision Tree Algorithm . . . . .	12
3.2.4	Neural Networks . . . . .	13
3.3	Software and Libraries . . . . .	14
<b>4</b>	<b>Related Work</b>	<b>15</b>
<b>5</b>	<b>Approach and Design</b>	<b>17</b>
5.1	Machine Learning Pipeline . . . . .	17
5.2	The Dataset . . . . .	18
5.3	Pre-processing of the datasets . . . . .	20
5.4	Selection of Machine Learning Models . . . . .	22
5.4.1	Linear Regression . . . . .	22
5.4.2	LightGBM . . . . .	23
5.4.3	Long Short-term Memory - LSTM . . . . .	23
5.5	Hyperparameter Tuning . . . . .	24
5.6	Evaluation Metrics . . . . .	27
<b>6</b>	<b>Implementation</b>	<b>29</b>
6.1	Environment and Setup . . . . .	29
6.2	Exploratory Data Analysis . . . . .	29
6.3	Training of the Models . . . . .	35
<b>7</b>	<b>Results and Evaluation</b>	<b>40</b>
<b>8</b>	<b>Conclusions</b>	<b>44</b>
<b>9</b>	<b>Future Work</b>	<b>45</b>



# 1 Introduction

In the present time, there is a need to reduce energy consumption for the future sustainability of this planet. With the growing population and economic development, the demand for energy is always increasing. And with the development of countries there is a greater need for energy in multiple aspects such as agriculture, transport, industry and manufacturing. With this growth of energy usage, there is a need to meet future sustainability goals. Current product of energy is heavily related to non-renewable sources like fossil fuels, where these contributed to 84% of energy production in 2019. The burning of these fossil fuels leads to the production of greenhouse gases such as Carbon Dioxide, these have lead to the entrapment of heat within the Earths atmosphere. This exacerbates the issues of global warming and is leading to many issues. We have been observing melting glaciers and polar ice-caps, rising temperatures affecting wildlife and their habitats (also leading to different migration patterns) changing ecosystems, temperature change and further extreme weather. Buildings and construction contribute significantly towards global warming, where these contribute 39% of carbon emissions globally. It is known that ineffective management within buildings may lead to increased costs and severe environmental impacts as above. If we are able to more effectively use existing and advanced controls, we'd be able to reduce energy consumption by 30%, solving some of these complications discussed.

There is large growing amounts of data available, collected from various sensors. Machine learning can make use of large data and make use of predictions to improve efficiencies within buildings. This can also be used to forecast energy consumption, detect and prevent faults (which could've lead to costs in downtime), seasonality and pre-cooling/pre-heating models. We will be investigating on how Machine Learning techniques can be used to effectively forecast and model energy consumption within buildings. Currently there are issues in comparing predictions methods, due to the use of smaller data-sets to optimise specialised buildings. We will be following the ASHRAE Great Energy Predictor III competition, where I will be able to use a large data-set and see what techniques I find important and effective compared to the other participants.

## 2 Aims and Goals of the Project

We aim to develop models using Machine Learning that will be able to accurately predict energy consumption within buildings. Our goal is to be able to use historic usage rates, weather and building data to accurately forecast future energy rates. Following this we will evaluate these models to see what extent and accuracy they are able to perform at. The trends and patterns observed from the obtained models should allow us decide, which of these models and methods are more suited towards this problem. Overall our goal is to see what methods and techniques are effectively towards modelling energy consumption within buildings.

This project aims to contribute towards study of Machine Learning techniques that can be used to effectively study energy consumption in buildings on a larger scale. We have identified two groups that can benefit from our study. With the nature of the models, we will be able to forecast future energy data and this can be used to more efficiently manage buildings. With this more efficient management building owners can benefit from reduced costs. This also relates back to the environmental issues discussed previously. More efficient energy management can allow for lower overall usage, reducing negative impacts such as global warming.

Like stated before towards our project we will be developing a range of models and testing their effectiveness towards this problem. However due to time constraints, it is impossible to test all possible models. With this, it was decided that we'd look at the most promising models and algorithms towards this problem. The way we have selected our models have be discussed further in the "SOMETHING" section.

The overall aims of our study are summarised below:

- Using Supervised Regression Machine Learning algorithms to predict energy consumption within buildings.
- Gauging the level of accuracy of these predictions and decide on the effectiveness of a model (This will be discussed more later).
- Looking at the data-set and how it's quality will affect the ability for the model to learn.

## 3 Background

This section covers some important concepts related to Machine Learning, that will be used heavily throughout the development on my models. I will first cover some Machine Learning algorithms, we will then continue to discuss the metrics that will be used to evaluate the trained models.

### 3.1 Problem Classification

Machine Learning is a sub-field of Artificial Intelligence, focusing on the science and programming of computers so they can learn from data. Arthur Samuel a pioneer in this field has stated Machine Learning as "the field of study that gives computers the ability to learn without explicitly being programmed". With the fast growing of large data-sets and increasingly complex problems, trying to solve these problems by manually defining a set of rules can be impossible i.e image recognition of humans. Machine Learning algorithms allow us to make use of large data to develop models, which are able to make important data-driven predictions. Also the popularity and use of Machine Learning is only growing due to the abundance of data, and now cheap computation. We have been able to see the use of Machine Learning in a multitude of sectors, such as the Energy Industry, our study can be classified within this industry. Machine Learning Systems can be classified into different types, depending on the type of supervision during training. These classifications are shown in the sections below:

#### Supervised Learning

When learning with Supervised learning the algorithms are given an training data-set with the desired solutions, which are also known as the labels (In our case this would be meter readings). The algorithms will analyze the labelled training data to produce an inferred function. This function then can be used to map for new examples and hopefully be able to generalise to new instances. Unseen data can be used to evaluate the accuracy of these models (known as the test data-set, during these evaluations we would take care to reduce results from loss functions, to achieve improved model. Supervised Machine Learning algorithms can be split into further into two main types of problem:



## Regression Problem

With Regression we are typically looking to find some continuous value i.e. salary or building prices. These algorithms will try find correlations between independent and some outcome/dependant variable. With this we can use the models can to forecast outcomes on new and unseen data. This type of problem is an particular interest to us, as our study falls into this category, as energy meter readings are counted as continuous values.

## Classification Problem

Classification problems try to find a class label for any given instance from input data, such as how emails can be classified as *spam* or *not spam*. The models will try to find how to best map input data instances to these class labels. Our problem does not fall into this category, therefore this will not be explored further as this point of time.

## 3.2 Machine Learning Algorithms

Knowing that we're studying an supervised regression problem, we can now look into the relevant algorithms that will be considered during our implementation. In these sections we will only give an overview of the algorithms themselves, the logic behind their usage has been discussed in the Approach chapter.

### 3.2.1 Linear Regression

We can start by looking at more traditional examples of linear regression in the slope-intercept form, the equation for this is shown below.

$$y = \theta_0 + \theta_1 x \tag{1}$$

Here  $x$  would represent the input variables, and  $y$  would be the predictions we've made. The  $\theta_0$  shows the bias, in the case that  $x$  is 0, this would be the value of  $y$ . Finally,  $\theta_1$  represents the weight for the independent variable. The above only deals with finding an linear relationship for one independent variable  $x$ , however we will be dealing with multiple features and with this we will look into multivariate regression.

Generally linear models will make predictions by computing a weight sum of all input features, including the bias term as defined previously. This can be shown with this formula:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n \quad (2)$$

Like before  $\hat{y}$  is the predicted value,  $n$  represents the number of features, where each  $x_n$  would be a feature value.  $\theta_n$  would represent the weight term or regression coefficient - showing how much we expect the predictions to change with respect to  $x$ . To optimise the models above we would need some method to find the most optimal  $\theta_0$  and  $\theta_j$  values, giving us the better predictions possible, to do this we would need some kind of performance measure. An measure that can be used is the RMSE which involves calculating average of the square of the difference between the predictions and actual values. This and other evaluation methods have been discussed in further detail in the "SOMETHING" section. We will now look at some more important concepts with the usage of linear regression methods.

## Gradient Descent

Gradient descents main idea is to adjust parameters iteratively to minimise some cost function (This could be our MSE or RMSE). The gradient is the slope of a curve, where for multivariate functions, it is the vector of derivatives in each main direction along variable axes [REF]. This algorithm will look towards finding the gradient of the error function, and will go in the direction of the descending gradient. In more detail, can use random initialization and fill  $\theta_j$  with random values, it will then work towards decreasing the cost function, until we reach a point of convergence at a minimum. To implement this we will be calculating the partial derivative (finding the change in cost function, if  $\theta_j$  is changed by a small amount). Cost functions such as MSE will work particularly well with gradient descent, since it is a convex function. Below is three important gradient descent applicators:

Batch Gradient Descent - When calculating the partial derivatives, it will calculate these in a single batch and update, rather than doing it individually (In essence doing it over the entire training data-set). This method scales well with the number of data-set features, however can be significantly slower with a larger data.

Stochastic Gradient Descent - Unlike batch gradient descent, this will calculate gradients at every step. Where for every instance of the train data-set, it will compute the gradient based on that one instance. This is a lot faster compared to the above, but due to its random nature, the gradient values can jump around. This also means there is an good chance to escape from a local optima. However, when we reach an minimum, due to the random nature it will struggle to settle, so final parameters might not be optimal (This could be solved by gradually reducing the learning rate).

Mini-Batch Gradient Descent - This combines both of the previous techniques, where the gradients are calculated on mini-batches, which are smaller sets of instances. This algorithm is less erratic and faster (due to hardware optimization for matrix operations) in comparison to SDG, however we may lose the benefit of being able to escape from an local optima.

## **Regularization**

There are further methods that can be used to minimise the MSE on the training data. One way we can do this is by reducing the degree of freedom it has, we can do this on a linear model by constraining the weights of the model. Some of the ways we can apply these constraints are shown below:

Ridge Regression (L2 Regularization) - This is added onto the cost function, where it is modified to minimize the squared absolute sum of the coefficients.

Lasso Regression (L1 Regularization) - This also adds an regularized term to the cost function, where we are modifying to reduce the absolute sum of the coefficients.

### 3.2.2 Decision Tree's

Decision tree's can be both used to solve classification and regression problems. It uses a tree like structure with internal nodes to represent features in the data-set, the image below shows an example of this.

INSERT IMAGE HERE

We can see two types of nodes, Decision and Leaf. Decision nodes are where any decision is made, where leaf nodes contain the outcomes for these decisions. It is also noticeable that a leaf node, will not have further branches. In an generic prediction process we will start at the root node, compare with our data instance and move to a lower node using a branch. It will do this until we reach some leaf node, which will be our output prediction.

Decision tree's are developed with the CART algorithm. The algorithm recursively splits the data-set into two subsets, using a single feature  $k$  and some threshold  $t_k$ , and it will search for the pair of these values that return an lower MSE (cost function) value. It will do this recursively until we reach some maximum set depth, or the MSE cannot be reduced further.

Decision tree's are beneficial in the way it makes very little assumptions towards the training data, however this could lead to the model overfitting. This means the model would not adapt well to new and unseen data, with this we'd need to use some regularization methods. This could be tuning parameters of the algorithm, such as giving a maximum tree depth.

### 3.2.3 Gradient Boosting Decision Tree Algorithm

Boosting algorithms refers to the process of using ensemble methods. The goal is to train predictors sequentially, to solve any residual errors in the previous predictor. We ensemble weaker models, with each trying to reduce the error in the previous tree, overall creating a more accurate and better performing model. This is done by fitting new predictors to the residual errors of the previous stage/model, where the residual error can be identified with some cost function (like MSE). With this higher level of accuracy, we would also need to be conscientious of overfitting, so like before we would need to apply a range of regularization methods, to prevent this.

### 3.2.4 Neural Networks

#### Artificial Neural Networks

Neural networks are algorithms that are modelled after biological human brain and the neurons contained within them. One of the more simplest architectures is known as the Perceptron, based on an artificial neuron called the TLU (Threshold Logic Unit). Inputs to this are numbers, with weights associated to each input, which for the sum is calculated. Some step function is applied to this sum and this result is outputted.

This perceptrons can be combined to create an MLP (Multi-layer Perception), made up of an input layer, a number of hidden layers (layers of TLU's) and an output layer (Final layer of TLU). We can think of each node or neuron as its own linear regression, having its own weight, bias/threshold and output. This method uses backpropagation ( involving gradient decent) training. Firstly data is sent in mini-batches (where each batch is know as an epoch) to the input layer, this is the sent to the hidden layers, where the output is calculated for the neurons before sending the result to the next layer. The algorithm will do this till we reach the output layer (This is the same as making predictions). At each of these predictions, we measure the error and go backwards to see how much each connection contributes towards the error. Finally we adjust the connections weights, to reduce the overall error.

#### Recurrent Neural Network

An RNN is specialised to work with sequential or time-series data. This relates to our own problem, where we will predict energy meter readings, which can be classified as time-series data. Neurons in an RNN are different in the fact, they have an 'memory' that can help store outputs and be able to use them as inputs for the next sequence[]. In an Deep RNN each layer will have their independent weights and biases. Schematically this can be thought as using using a for loop to iterate over time-steps of a sequence, while maintaining some internal state with the information of seen time-steps. This time-step information is kept within the RNN hidden state. All this allows us to long-term dependencies, which other algorithms might be be able to pickup.

### 3.3 Software and Libraries

Here we will look at software and libraries that are related to our study and will be used during our implementation.

- **Pandas** - Python software library dealing with data manipulation and analysis.
- **NumPy** - Python software library dealing with support for large, multi-dimensional array and matrices. It contains a large number of mathematical functions, that can be used.
- **SkLearn** - Python based software library dealing containing multitudes of machine learning techniques such as classification, regression, clustering and dimensionality reduction.
- **Keras** - Keras is a deep learning API written in python. It allows for simple, flexible and powerful implementation of Neural Networks. Keras is built upon Tensorflow.
- **Tensorflow** - Open-source software for machine learning and artificial intelligence, focus on training of deep neural networks.
- **Matplotlib** - Python library allowing for static, animated and interactive visualisations of data[REF].
- **Seaborn** - Another visualisation library in Python, this is based on matplotlib.
- **Google Colab** - Google Colab is an online platform, where it hosts Jupyter notebook service. It allows for the writing of python through the browser, and is suited for machine learning and data analysis work.

## 4 Related Work

We will look at some of the work that other contestants have done towards this study, and I will look at their procedure and solutions.

Authors	Methodology	Returned Score
Isamu Yamashita and Matt Motoki	Procedure includes removing anomalies (long streaks of constant values, large spikes, additional from visual inspection), they will impute missing temperature values, and take the log of the meter-readings. They took two approach's, one brute force (selecting most features) and conservative approach (carefully selected features). They used LightGBM, CatBoost and MLP models, where they used ensemble in post-processing to reduce overfitting.	1.231
Oleg Knaub, Roham Rao, Anton Isakin, Yangguang Zang	Did manual processing, by looking at all the buildings individually and taking out outliers when possible. Did not focus much on feature engineering (less than 30 features). Used models such as XGB, LightGBM and CatBoost. Used post-processing with weighted mean(depending on meter type). Also found the cleaning process important. Focused on using the data from leaks.	1.232

Xavier Capdepon	Used CNN, XGBoost, LightGBM (CPU) and CatBoost (GPU). Recognized building-id as most important feature. Used ensemble techniques for post-processing.	1.234
-----------------	---	-------

I will also look at this paper[REF], and discuss some of the findings from the competition overall. It seems like one of the key methods towards having a high accuracy model depends on how the pre-processing and post-processing was completed. Most of the top solutions have taken more care into their pre-processing, even manually looking at each building to find outliers. Top scores also look at ensembling frameworks, where they've used methods to apply weightings. This paper also talks about how such a large dataset is beneficial towards this study, as these allow us to examine how these models could generalise well to buildings around the world. This meets the goals of our study, since we are trying to create a ML model that will be able to accurately forecast energy usage within buildings.

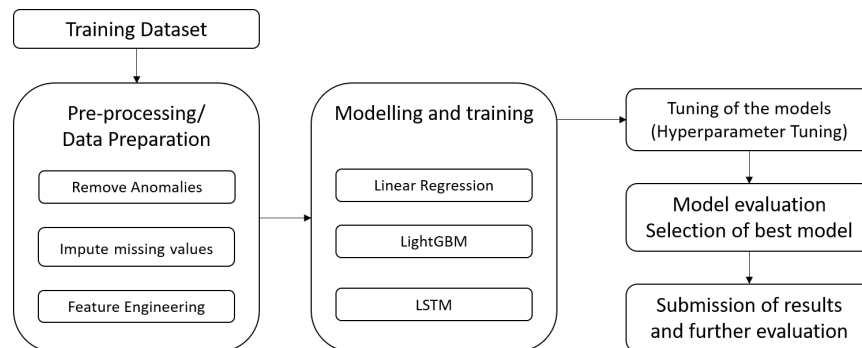


## 5 Approach and Design

In this section we will be going through the how we approach our problem, and the design that will be taken to implement our models. Throughout these I will explain, why I think these methods were the most suitable towards my study. I will start by looking at the overall Machine Learning Pipeline for my problem.

### 5.1 Machine Learning Pipeline

Like stated previously, we are studying the usage of different models to predict energy meter readings for buildings. These models will then be evaluated to see which are the best performing and most accurate. This work will also be compared to work others have completed on the same data, as shown in the **Future Work** section. The pipeline below describes how we'll be developing these models. Below we go through each of the components and give further detail on what those specific stages will entail.



## 5.2 The Dataset

To investigate the energy usage within buildings, we have used the data-set provided by ASHRAE in the Great Energy Predictor III competition. This large data-set is extremely beneficial, as it targets a large number of different types of buildings. This would stop this analysis from becoming too specialised for a singular type of building. With this we can know which type of algorithm will be generally better when it comes to predicting energy usage for buildings accurately. It will be beneficial to be able to compare our results, with those of other contestants, giving us a further idea of how good our developed models are. Below is the break-down of the dataset, where a small description has been given for each feature.

### Building Metadata

This dataset includes data on all buildings the meter readings were taken from, summarized in this table below:

Feature Name	Description
Site ID	Foreign key for weather data file
Building ID	Foreign Key for training dataset
Primary Use	Indicator for the primary category of activities for the building.
Square Feet	Gross floor area for the building
Year Built	The year, the building was opened
Floor Count	Number of floors the building has

### Weather Dataset

This dataset includes data on the weather at the time of the meter reading, summarized in this table below:

Feature Name	Description
Site ID	Foreign key for weather data file
Air Temperature	Measured in Degrees Celsius
Cloud Coverage	Portion of sky covered in clouds, measured in Oktas
Dew Temperature	Measured in Degrees Celsius
Precipitation depth/1hr	Measured in Millimeters
Sea Level Pressure	Measured in millibar/hectopascals
Wind Direction	Compass Direction (0-360)
Wind Speed	Measured in meters per second

### Train Dataset

This is the training dataset that we will be using, which contains the important meter reading labels. This is summarized in this table below:

Building ID	Foreign key for Building Metadata table
Meter	The meter ID code, indicating which type of meter it is (0:electricity, 1:chilled water, 2:steam, 3:hotwater)
Timestamp	The time and date a meter reading measurement was taken
Meter Reading	The energy consumption in kWh, where this is our target variable (our labels)

### 5.3 Pre-processing of the datasets

We can see from the above breakdown that the dataset is spread over multiple tables, and in this form it is currently not as useful for training our models, and will be in various formats. It is also important that any significant outliers, missing values and any other are removed, to remove any incorrect bias in our training process. Throughout this stage, I will use several processing techniques to ensure that data is in a more viable and consistent format. Here we analyse the techniques that we will use, and why these have been the ideal choice towards this processing.

#### Merging of the dataset

We can currently see that the current format of the data is not suitable for when we will train our models. From the break-downs it is easy to see that all the tables are connected through foreign keys, which we will be able to use these to merge the three datasets, into a singular larger one. Currently one record of information only has information on the timestamp, meter type and the reading. When we combine these, a single record will also contain the building and weather information, for that meter reading.

#### Train-Test Split

When training our model we will be splitting into two parts, training and test datasets. The training dataset will be used to train our models, where we will measure the accuracy with some loss function i.e. MSE. However we risk overfitting to this dataset, this is when model will learn well towards a function for the training data, however would not generalise well towards newer information. To prevent this issue we use the test dataset, we can see how the model will adapt to new and unseen data, so we can see if our model is either overfitting or how it will work from an actual production standpoint. Since our dataset is so large, we have chosen to use a 80% train and 20% test split.

## Dealing with Missing Data

When collecting data, it is common for errors to occur, leading to missing entries in the dataset. Dealing with these missing values is important, as it could lead to significant errors when developing our models. To deal with these values, we have a handful of approaches. Firstly we could remove any instance or record where there is a missing value. We will not be using this approach, since it could cause other important information to be lost for our model. Secondly, we can use imputation, where we will fill missing entries with some kind of number. We will be using the median of that feature column to fill missing values, the median would allow us to avoid issues when the data is skewed (with skewed data, mean can perform badly). However if we ever reach the case of large amounts of missing information (above 40%), this entire feature will be removed. Having too much missing data would be hard to impute, and could just bring in more bias towards our model, which is not ideal.

## Dealing with Categorical Data

Since most machine learning algorithms cannot deal with categorical data (such as linear regression), we will need some method to convert it into numerical value. We can see that we have categorical data of Primary use in the **Building Metadata** table, we will need to deal with this. We can firstly look at the Integer encoding method, where we give integer value for each label. This is not sufficient in our case, since the data we have is nominal. and doesn't have any numerical value. With this we have chosen One-Hot encoding, where a new binary variable/feature is added for each categorical label. This method can bring issues with multicollinearity, where these categories could introduce dependency between independent features. Another issue includes if there is a large number of categorical labels present, it can significantly increase the complexity of the problem, we will not incur this issue, as we have a limited number of categorical labels (Discussed in **Implementation** section). We still go through with One-hot encoding since it allows for the encoding of nominal data. Where in the case of issue we can drop dummy variables, to ensure these dependencies are not made.

## Feature Scaling

Scaling is an important consideration to make for the preprocessing of our dataset. Looking at the dataset break-down it we can see that there is a multitude of numerical values, which have different units. In some cases the difference between the large and smaller values may be significant, leading to longer and less accurate training, we could have this due to the difference between large and small meter readings (Show in **Implementation** section). We can use scaling to ensure values are closer together, and hopefully lead to better accuracy. We have decided to use Standard Scaling to do this, where it scales by firstly subtracting the mean (so it is zero) and will get an variance equal to 1. Even-though it will not scale to values between 0-1, this scaling will be less affected by outliers. Since its less affected by outliers, we have chosen to use this approach.

## 5.4 Selection of Machine Learning Models

This problem involves the usage of supervised regression based algorithms to accurately forest energy meter readings for buildings. However due to restricted time for this study, I will be selecting only a few, but promising machine learning algorithms. We will go through how each of the models work on an overview level, then proceed to discuss, why we these could be ideal approach.

### 5.4.1 Linear Regression

We will be using the Linear Regression model provided my SkLearn in Python. This model works by trying to minimize the residual sum of squares between observed targets in the dataset, and the targets predicted by the linear approximation[REF]. This is following procedure of "Ordinary Least Squares", where we are trying to minimise the distance between each data-point and the line of best fit (Minimizing the sum of square residual errors). Linear Regression is a beneficial when we are looking to forecast some value, this will be useful, since we are forecasting future meter readings. This will also allow to see whether our dataset has a linear relationship, indicated by the accuracy this model performs at.

### 5.4.2 LightGBM

LightGBM is based on Gradient Boosting algorithms, where this algorithm will try to grow decision trees vertically (leaf-wise split). This algorithm was chosen due to the wide range of benefits that it would be able to bring for our training process. LightGBM uses histogram-based algorithm commonly used GBDT algorithms. However this histogram based methods have their own limitations, and LightGBM solves these with GOSS and EFB. GOSS (Gradient-based One Side Sampling) involves the exclusion of some low-gradient data instances, where EFB (Exclusive Feature Bundling) bundles mutually exclusive features, reducing the number of overall features. Both of these will have increase the memory usage and speed of the algorithm. Overall this algorithm is faster, because this histogram-based algorithm will place continuous values in discrete bins, reducing memory usage and increasing training speed. This is extremely beneficial in our case, where we have an significantly large dataset. This algorithm can also achieve higher accuracy, due to focus on leaf-wise exploration. During my implementation I will need to be wary of overfitting, due to the leaf-wise nature (could be avoided by setting the max depth hyperparameter).

### 5.4.3 Long Short-term Memory - LSTM

RNN have some issues, which includes the Vanishing/Exploding Gradient problem, this comes from how RNN's uses Backpropagation through time' to adjust weights in the network. With each step of BPTT, it will calculate the partial derivative at each weight in the network, these can act as a chain rule, where in the end we find a sum of these derivatives to find some error. It will use this information to adjust weights within the RNN network, however with longer-term dependencies, the long multiplication of derivatives leads vanishing values, which means the weights of the network are not updated correctly, leading to inaccuracies in the models. Also in the case of initialization of large values, the gradient values can get too large, leading to erroneous values, known as the exploding gradient. LSTM will solve these issues, as its design to deal with dependencies in data. LSTM cells in the hidden layers contain four gates, which will limit information passed through the cell and ensuring only needed information will be passed on. We can chose to overall use LSTM because it allows for the forecasting of time-series data, where it deals with issues in typical RNN, as above.

## 5.5 Hyperparameter Tuning

Hyperparameter tuning is an important part towards the development of the models. Hyperparameter's are values that can help control the learning process of the algorithms, this could help lead to more optimal accuracy. Select ideal values can help avoid the model from over or under-fitting. The below tables show the hyperparameters that have been investigated for this problem, and the reasons behind the targeting of them.

### Linear Regression Hyperparameters

SkLearn's algorithm for Linear Regression has a limited number of hyperparameters, hence not having many to explore in the table below.

Hyperparameter	Functionality + Reasoning
n_jobs	Number of jobs to use for computation. Since we have a large problem , this could lead to a speedup in training

### LightGBM Hyperparameters

LightGBM has an extremely large number of hyperparameters we can work with, however due to time and computational constraints, it is not possible to test all combinations. With this we have selected the hyperparameters that could be the most significantly during training.

Hyperparameter	Functionality + Reasoning
n_estimators	The number of boosted tree's to fit. Controlling the number of tree's can help get an more accurate model. Care should be taken, as too large values can lead to overfitting and longer training times.



learning_rate	Controls the size of the step taking when moving with the gradient of the cost function[]. A higher learning rate, can be faster, but have an lower accuracy. Taking a lower learning rate would allow for slower learning, but can help combat overfitting and have the transitions be smoother.
reg_alpha	This is L1 regularization, where it will make smaller values go towards zero. This can help reduce the number of features, and reduce overfitting.
reg_lambda	This is L2 regularization, where it will restrict larger outlier values. This can also help overfitting, it is important that this value isn't too large, or it could lead to under-fitting instead.
num_leaf	Sets the maximum number of leaves, for the weaker models (DT models). Setting a greater number of leaves, would allow for a higher accuracy, however would need to be careful, as there is an risk of overfitting.
feature_fraction	Does column sampling, where it would select some subset of features on each iteration. This can help speed up training, and help to deal with overfitting.
max_depth	Restricts the maximum depth of each tree. Selecting an optimal value would allow for optimal training times, and could help avoid overfitting. Too large values could lead to long learning times, and overfitting. This parameter has to be carefully considered with the num_leaf.

## LSTM Hyperparameters

The tables below show the hyperparameters that have been chosen to tune the LSTM models. This model had a different approach taken for tuning, where a Bayesian tuning algorithm was used (This will be further shown in the implementation section).

Hyperparameter	Functionality + Reasoning
input_neurons	A greater number of neurons, could lead to an increase in accuracy and complexity of the model, however we would need to be careful in not allowing the model to over-fit.
number of hidden layers	We could test to find a number of hidden layers that would increase the accuracy of the model. However too many layers could lead to overfitting, and this would not be ideal.
dropout number	Tuning this parameter would allow for the managing of regularization of our models. Where certain (probabilistically chosen) LSTM units will be excluded, from activation and weight updates[REF]. This can increase the models performance, while also reducing chances of overfitting.

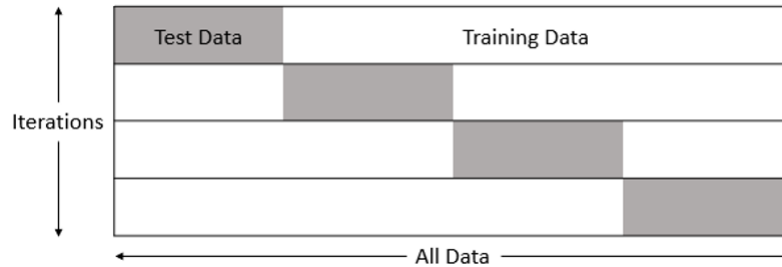
## Approach to Hyperparameter Tuning

There are two main approaches that have been taken when hyperparameter tuning. Both of these methods are discussed below:

### GridSearch

We create a search space or grid of hyperparameters we want to test, then we do an evaluation on every point of that grid. Lists of selected parameters can be fed into the algorithm, exhaustively testing combinations and returning the most ideal combination of these. GridSearch also allows for in-built Cross

validation, this is where the dataset is split into further smaller train-test splits. The image below shows a representation of how this works:



In each iteration it will train the data, and validate it against the k-th fold test data-set. This method will allow for the entire dataset to be considered for validation, and in the end we use the average of the accuracy, as the performance metric for our model. This approach has been chosen as it would allow for evaluation of hyperparameter metrics, where the models with better accuracy (lower MSE) are better. It would also be beneficial, as we would be able to see whether our model over-fits.

### Bayesian Optimisation

Bayesian optimisation works similarly to GridSearch, where we are testing a range of parameters on a model. However with this approach, it will use information on the previous model, to select more promising parameters for the next model iteration. Models such as the LSTM will take significant time to run, it will be infeasible for me to run GridSearch on it, due to time constraints. Bayesian will allow the reaching of optimal hyperparameters for high accuracy, in a lower amount of time.

## 5.6 Evaluation Metrics

To evaluate our models we will need some form of performance measure. Since we are using regression based models and our problem has the goal of being able to forecast future meter readings, we will want to minimise the error on these predictions. The error is measured using the following metrics:

## Root Mean Squared Error - RMSE

The RMSE is the square root of the variance of the residuals, where the residuals are the difference between observed and actual values. This can be calculated with the following formula:

$$\sqrt{\frac{\sum_{t=1}^T (\hat{y}_t - y_t)^2}{T}} \quad (3)$$

The top part of the fraction represents the sum of the difference between the actual and predicted values squared, where  $T$  is the sample size. I have chose this method, since like MSE, the squaring effect will lead to magnified large errors. This punishment make it easy to see, where there is a significant different between our predictions and actual values. Since we also take the square root, values will not be in squared units, allowing for easier reporting and readability. This is why I've chosen this metric over MSE.

## R-Squared Error

This is a measure of how close the data is to the "line of best fit". In essence it will allow us to see the variability been the independent and dependant variables, where its measure between 0-1. Higher the value, can indicate a great fit on the data. We would use the adjusted  $R^2$  value, since we are dealing multiple features, otherwise the algorithm would just keep on adding to this value, even-though some of those variables might not have any significance. I've chosen this metric because of the difference in information it gives compared to RMSE, where this metric cannot tell me whether the model is good or not. A lower RMSE would indicated for a better model, however higher  $R^2$  could indicated overfitting, and even in low-value cases, the model still might be fine. This will overall let us see the correlative nature between our features and dependant variable (our meter readings).

## 6 Implementation

### 6.1 Environment and Setup

This project has been developed in Google Colab, a cloud-based environment allowing access to GPU's and further RAM. Due to the computation restrictions of my own PC, there was a need for a higher performing Jupyter notebook, that wouldn't crash on just loading the large dataset. The extra memory and computing power allowed for easy processing, while the nature of being able to connect to the cloud, allowed for all data to be stored on the google cloud. An zip including the notebook used for this project has been included, the notebook is named "FinalWork", to use this code, upload all files in the zip (as-is) to google drive, so the notebook, will be able to assess all files successfully. The dataset can be found in this link, and will need to be uploaded individually: <https://www.kaggle.com/competitions/ashrae-energy-prediction/data>

### 6.2 Exploratory Data Analysis

It was important to analyse the dataset that I had to find any anomalies and adjust them, to achieve the highest possible accuracy and minimise RMSE as much as possible. I will in this section only be showing some of the more important conclusions, that I have been able to make, and how that has affected my overall preprocessing steps.

```
## Function to reduce the DF size
def reduce_memory_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:.2f} Mb ({:.1f}% reduction)'.format(end_mem, 100 * (start_mem - end_mem) / start_mem))
    return df
```

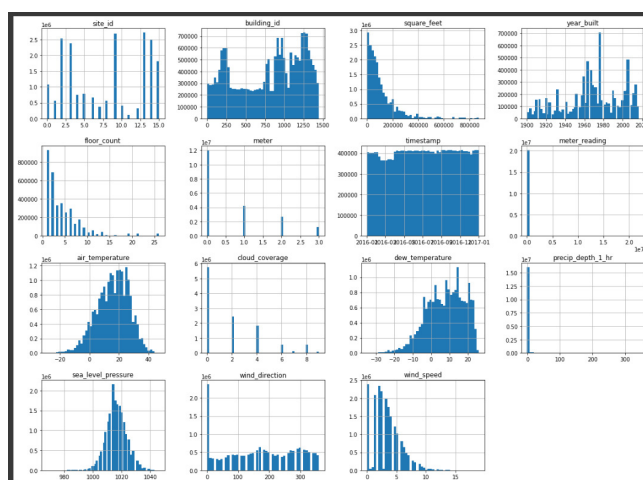
The first step I had to apply was the function above, this is not analysis,

but was needed to prepare for analysis. Due to the large nature of my dataset, it is important that I reduce the memory usage of it, as much as possible. Otherwise it has lead to the notebook's crashing, from using too much memory in a singular session. This is a function that I found that adjusted data types, to reduce the memory usage significantly. It found found that the reduction of the memory usage of these dataset was by at least 60%.

```
traindataset = building_metadata.merge(train, left_on='building_id', right_on='building_id', how='left')
traindataset = traindataset.merge(weather_train, left_on=['site_id', 'timestamp'], right_on=['site_id', 'timestamp'], how='left')
```

	site_id	building_id	primary_use	square_feet	year_built	floor_count	meter	timestamp	meter_reading	air_temperature	cloud_coverage	dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_direction	wind_speed
0	0	0	Education	7432	2008.0	NaN	0	2016-01-01 00:00:00	0.0	25.000000	6.0	20.00000	NaN	1019.5	0.0	0.000000
1	0	0	Education	7432	2008.0	NaN	0	2016-01-01 01:00:00	0.0	24.406250	NaN	21.09375	-1.0	1020.0	70.0	1.500000
2	0	0	Education	7432	2008.0	NaN	0	2016-01-01 02:00:00	0.0	22.796875	2.0	21.09375	0.0	1020.0	0.0	0.000000
3	0	0	Education	7432	2008.0	NaN	0	2016-01-01 03:00:00	0.0	21.093750	2.0	20.59375	0.0	1020.0	0.0	0.000000
4	0	0	Education	7432	2008.0	NaN	0	2016-01-01 04:00:00	0.0	20.000000	2.0	20.00000	-1.0	1020.0	250.0	2.599609

The two above images show the merging of the datasets. This is where we can simply merge them based on the foreign keys that were provided. Firstly the building metadata was merged into the training dataset, following that the weather data was merged into the combination of those. The resulting dataset can be seen in the lower of those two images.



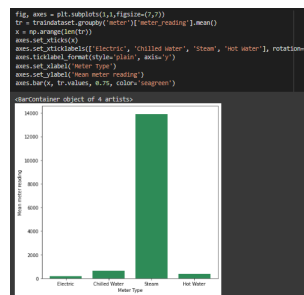
The above image shows an histogram for each of the features. It allows us to make some important conclusions from this. Firstly we can see that the only features that have normal distributions are the air-temperature and sea-level-pressure. We can see that the dew-temperature is slightly skewed to

the right, whereas the floor-count, wind-speed and square-feet are all heavily skewed to the left, this will be dealt with in the coming processing. Here we are able to identify a large issue with our labels, in the fact that they are heavily skewed and will need normalisation. In the histogram, we can only see a singular column, indication there is a large difference between the smallest and largest values, we will need to standardize these readings, so smaller values are also considered in training.

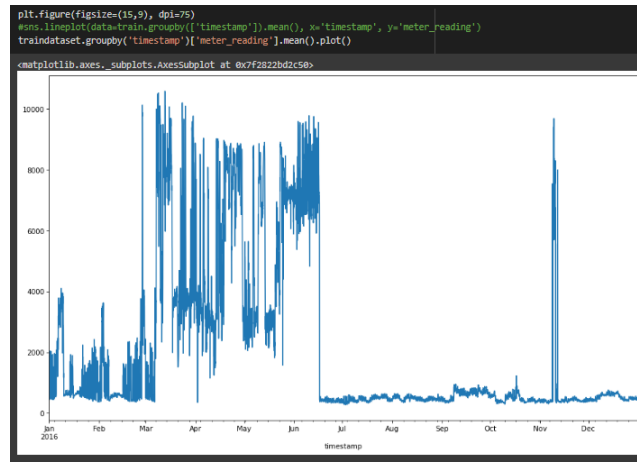
```
print("These are the ratios of missing values: %s")
traindataset.isna().sum()/len(train)*100

These are the ratios of missing values:
site_id          0.000000
building_id      0.000000
primary_use      0.000000
square_feet      0.000000
year_built       59.990033
floor_count      82.652772
meter            0.000000
timestamp        0.000000
meter_reading     0.000000
air_temperature  0.471114
cloud_coverage   41.655111
dew_temperature  0.495348
precip_depth_1_hr 16.548719
sea_level_pressure 6.092515
wind_direction   7.187792
wind_speed       0.710701
dtype: float64
```

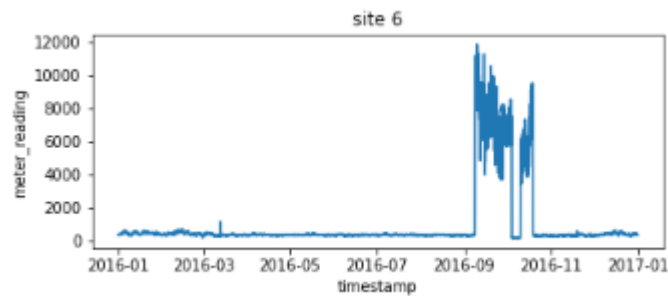
We can now look at the distribution of missing values. It is easy to see that some of the features have an extremely large number of missing values, hence some of these features will be excluded from training. Features years-built, floor-count and cloud-coverage, have above 40% of their values missing, with this I feel like imputing could lead to bias, and therefore I will simply exclude these features.



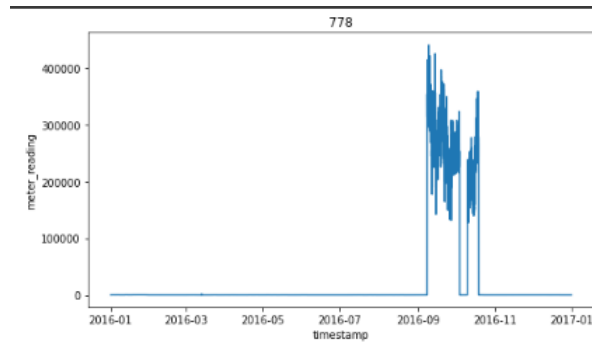
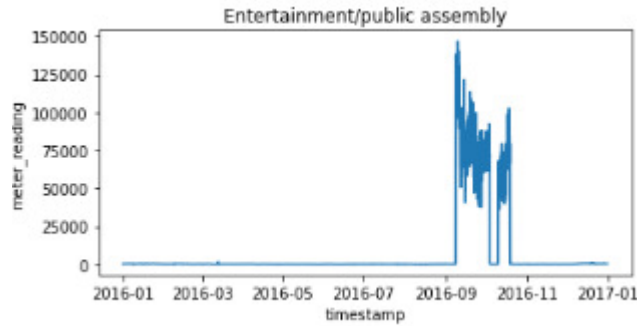
I decided to also investigate the number of meters by their type, and the average readings for them. I was surprised to find that even-though the highest number of meters was electric, the meter-readings for steam-meters was so high compared to the other meter types. It was easy to see that there was some kind of issue in these values, and decided to do further investigation. The first graph on the next page, shows the average meter-readings over a year.



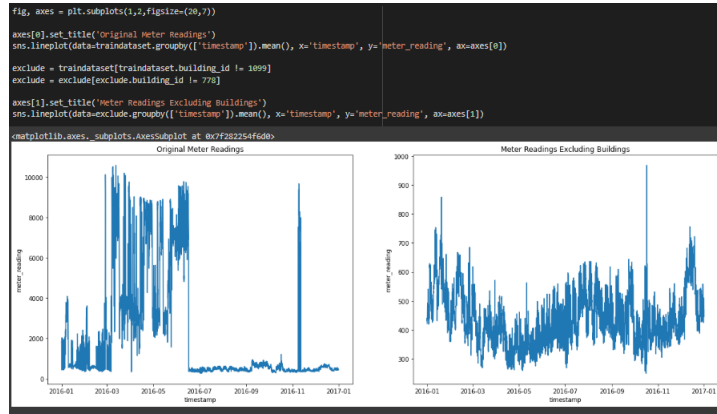
I found this graph very interesting, as it doesn't show an nice distribution like I had expected. I found it odd how the readings suddenly spike around March, then just drop again in July. There also that one peak in November, with this we needed to investigate further the remove errors from the dataset, hopefully improving this distribution, and increasing the accuracy of our predictions.



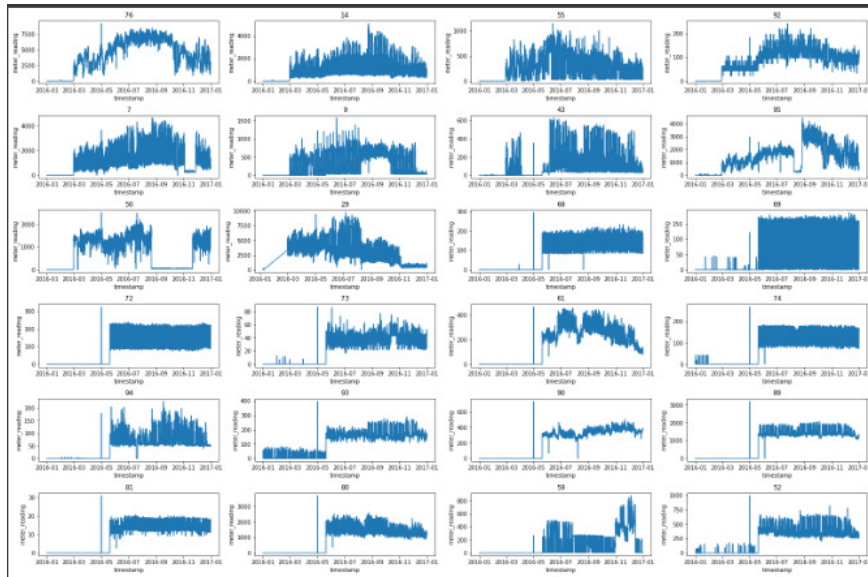




The three above graphs, shows the investigations that had been complete. It was decided that we would organise meter-readings by their sites, it was found that sites 6 and 14 had the most erroneous looking values. In the first image, you can see how there is a sudden spike in readings around 2016-09, where around the rest of the year there are practically 0 readings. Breaking this down further we looked at the type of building this error came from (being entertainment/public assembly) and found that the exact building with an issue was 778. I followed similar procedures for site-13 which shows trends mimicking the average meter-readings over an year, with finding that the building 1099 also has issues. Removing these values gives the following transformed distribution:



We can see how the above investigation allowed for erroneous values to be removed, and overall improving the distribution of my meter-readings. The goals behind doing this, was to try increase the accuracy of my models. Another large error source found was for site 0, where we can find that there were no meter-readings until either March or May for most meters on this site. This could've been some kind of issue with collection, handing or processing of data, and will need to be dealt with. With this we decided to remove all readings until the 20th of May. Some of the graphs showing these 0 readings are below:



## Summary of EDA

Here is a list summary of what I have learnt about the dataset and needs to be changed.

1. Meter-reading labels are heavily skewed and will need to be scaled appropriately.
2. The Years Built, Floor Count and Cloud Coverage features have more than 40% missing values, and hence will be removed.
3. Buildings 778 and 1099 was causing the the overall distribution of meter-readings over an year to be heavily skewed. Removing these values has significantly improved the representation of the data, and should hopefully increase accuracy in training.
4. Site-0 has some kind of error in data collect up till May, these values will be excluded from our training dataset.
5. Site-0 has some readings in the wrong units, these units should be converted for training[REF].

## 6.3 Training of the Models

### Preparation of the training dataset

We first will need to apply all the processing steps that we have discussed throughout the early analysis, We will also need to correctly complete the train-test splits. The below images show this. The above image shows me

```
train_labels = train_processed['meter_reading']
to_train = train_processed.drop('meter_reading',axis=1)

x_train, x_test, y_train, y_test = train_test_split(to_train, train_labels, test_size=0.2, random_state=42)
train_numerical = x_train.drop(['primary_use'], axis=1)
```

creasing a train-test split for the dataset using the `train_test_split` function from SkLearn, We have used an random state of 42 to ensure consistency between runs of my code and to make this study replicable.

The below image shows the pipeline used to pre-process the dataset. We have applied all the processing steps as discussed in my **Summary of EDA** section. We can see the breakdown of the timestamp data into multiple other features, we chose to do this because, it can help detect smaller seasonal trends, in this time data. You may have also noticed how we've taken a sample of the overall dataset, it was found that the dataset is too large for training and google colab's memory restricts will be exceeded. With this we can chose to take an even sample of the data. The below image just

```

train_processed = traindataset.copy()
train_processed.loc[(train_processed['site_id'] == 0) & (train_processed['meter'] == 0), 'meter_reading'] *= 0.2931

train_processed = train_processed[train_processed.building_id != 1899]
train_processed = train_processed[train_processed.building_id != 778]

date = datetime.datetime(2016,5,20)
train_processed = train_processed[(train_processed.site_id != 0) & (train_processed.timestamp < date)]

train_processed['hour'] = train_processed['timestamp'].dt.hour
train_processed['day'] = train_processed['timestamp'].dt.day
train_processed['weekday'] = train_processed['timestamp'].dt.weekday
train_processed['month'] = train_processed['timestamp'].dt.month
del train_processed['timestamp']

train_processed = train_processed.iloc[::4]
train_processed

```

	site_id	building_id	primary_use	square_feet	meter	meter_reading	air_temperature	dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_direction	wind_speed	hour	day	weekday	month
1076662	1	105	Education	50623	0	23.303600	3.800781	2.400391	NaN	1021.0	240.0	3.099609	0	1	4	1
1076666	1	105	Education	50623	0	45.607101	2.300781	1.799605	NaN	1022.5	110.0	1.500000	4	1	4	1
1076670	1	105	Education	50623	0	46.107101	2.190219	1.500000	NaN	1022.5	60.0	1.000000	8	1	4	1
1076674	1	105	Education	50623	0	48.607101	7.000080	4.388438	NaN	1019.0	120.0	5.101562	12	1	4	1
1076678	1	105	Education	50623	0	49.107101	7.601562	6.690219	NaN	1014.0	110.0	5.690219	16	1	4	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20210622	15	1448	Office	92271	0	4.075000	1.700195	1.099609	NaN	1021.0	0.0	0.000000	6	19	3	5
20210626	15	1448	Office	92271	0	4.975000	0.600088	0.600098	NaN	1021.5	70.0	2.099609	10	19	3	5
20210630	15	1448	Office	92271	0	4.600000	13.256875	4.388438	NaN	1022.0	310.0	3.099609	14	19	3	5
20210634	15	1448	Office	92271	0	2.525000	15.601562	3.900391	NaN	1021.5	300.0	6.690219	18	19	3	5
20210638	15	1448	Office	92271	0	1.725000	17.796875	3.900391	NaN	1021.0	280.0	6.190219	22	19	3	5

1782962 rows x 16 columns

shows an basic pipeline used to fill missing values, encode categorical data and use standard scaling on the features. In the below image you can also see we have taken the logarithm of the meter-readings, since the meter-readings were greatly skewed, doing this would give a more symmetrical distribution, which might like the models such as linear regression fit better.

```

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('std_scaler', StandardScaler()),
])

num_attribs = list(train_numerical)
cat_attribs = ["primary_use"]

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])

train_prep = full_pipeline.fit_transform(x_train)

train_numerical = x_test.drop(['primary_use'], axis=1)
x_test = full_pipeline.fit_transform(x_test)

y_train = np.log1p(y_train)
y_test = np.log1p(y_test)

```

## Initial RMSE Scores

We firstly decided to implement both the LightGBM and Linear Regression models. Below shows an simple implementation, where `cross_val_score` function has been used to calculate the RMSE. This function works similar to cross-validation, where it will indicate if the model is overfitting. The Light-

```
lin_reg = LinearRegression()
lin_reg.fit(train_prep, y_train)
scores = cross_val_score(lin_reg, x_test, y_test, scoring="neg_mean_squared_error", cv=10)
lin_rmse = np.sqrt(-scores)

lightgbm = lgb.LGBMRegressor()
lightgbm.fit(train_prep, y_train)
scores = cross_val_score(lightgbm, x_test, y_test, scoring="neg_mean_squared_error", cv=10)
light_rmse = np.sqrt(-scores)

print(f'Initial RMSE scores: \nLinear Regression RMSE: {lin_rmse.mean()} \n\nLightGBM RMSE: {light_rmse.mean()}')

Initial RMSE scores:
Linear Regression RMSE: 1.9111225486635945
LightGBM RMSE: 1.1690078885476836
```

GBM model has been tuned using GridSearch, where exhaustive selections of parameters have been chosen. After each of these searches, the data of these searches have been saved, where we are able reload and see the best performing model. Below shows an example of one of the GridSearch parameter tables used, and the results of the best model. The decision to save these models was again important, these GridSearches take extremely long to complete, and it would insufficient to rerun these every-time.

```
def gridsearch_pipeline(x_train, y_train, model, param_grid, cv=5, scoring_method='neg_mean_squared_error'):
    grid_search = GridSearchCV(estimator = model,
                                param_grid = param_grid,
                                cv = cv,
                                scoring = scoring_method,
                                return_train_score=True,
                                verbose = 3,
                                n_jobs = -1
                                )
    model = grid_search.fit(x_train, y_train)
    return model

param_grid1 = {
    'n_estimators': [500, 1000],
    'max_features': ['auto'],
    'learning_rate': [0.01, 0.05],
    'max_depth': [31],
    'reg_alpha': [0.0, 2.0],
    'reg_lambda': [0.0, 2.0],
    'num_leaf': [31, 1280],
}

print('Loading RMSE from different 60 models\n')
for x in np.arange(1, 7):
    gs = joblib.load(f'content/drive/MyDrive/colab notebooks/gstr{x}.pkl')
    scores = cross_val_score(gs, x_test, y_test, scoring="neg_mean_squared_error", cv=10)
    light_rmse_tuned = np.sqrt(-scores)
    print(f'({gs.get_params()})\nLightGBM ({x}) tuned RMSE: {light_rmse_tuned.mean()}\n\n')

Loading RMSE from different 60 models
Found method (LGBModel.get_params of LGBMRegressor(feature_fraction=0.85, max_depth=31, max_features='auto',
n_estimators=1000, num_leaf=31, reg_lambda=0.0))
LightGBM 1 tuned RMSE: 0.89390258250888

Found method (LGBModel.get_params of LGBMRegressor(feature_fraction=0.85, learning_rate=0.05, n_estimators=1000,
num_leaf=31))
LightGBM 3 tuned RMSE: 0.851410243764794

Found method (LGBModel.get_params of LGBMRegressor(feature_fraction=0.85, learning_rate=0.05, n_estimators=1000,
num_leaf=31))
LightGBM 4 tuned RMSE: 0.824987733293341

Found method (LGBModel.get_params of LGBMRegressor(feature_fraction=0.85, learning_rate=0.05, n_estimators=2280,
num_leaf=31))
LightGBM 5 tuned RMSE: 0.7767288013412186

Found method (LGBModel.get_params of LGBMRegressor(feature_fraction=0.85, learning_rate=0.05, n_estimators=4000))
LightGBM 6 tuned RMSE: 0.730506570601524
```

## Implementation and tuning of Neural Network

To implement LSTM, it was found that the data cannot be entered in the same format. I will first need to transform the data into a 3D format. During development this was a difficulty, as it was hard to understand, what exact format I should use to optimally develop the LSTM model.

```
train_prep2 = np.reshape(train_prep, (train_prep.shape[0], 1, train_prep.shape[1]))
x_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))
```

With this, the outline for the LSTM model was implemented, this is shown below. Keras Tuner has been used to implement Bayesian Optimisations

```
def build_model(hp):
    model = Sequential()
    model.add(LSTM(hp.Int('input_unit', min_value=1, max_value=100, step=10), return_sequences=True, input_shape=[None, 30]))
    for i in range(hp.Int('n_layers', 1, 4)):
        model.add(LSTM(hp.Int(f'layer_{i}_units', min_value=1, max_value=100, step=10), return_sequences=True))
    model.add(Dropout(hp.Float('dropout_no', min_value=0.0, max_value=0.5, step=0.1)))
    model.add(Dense(1, activation = hp.Choice('activation_method', values=['relu', 'sigmoid', 'linear'])))
    model.compile(loss="mean_squared_error", optimizer=keras.optimizers.Adam(learning_rate=0.05), metrics = ['mse'])
    return model

tuner= BayesianOptimization(
    build_model,
    objective='mse',
    max_trials=10,
    executions_per_trial=1,
    project_name = 'lstm6',
    overwrite = False,
    directory = '/content/drive/MyDrive/colab Notebooks/'
)

INFO:tensorflow:Reloading Oracle from existing project /content/drive/MyDrive/colab Notebooks/lstm6/oracle.json
INFO:tensorflow:Reloading Tuner from /content/drive/MyDrive/colab Notebooks/lstm6/tuner0.json

tuner.search(
    x=train_prep2,
    y=y_train,
    epochs=30,
    batch_size=128,
    validation_split = 0.2,
    callbacks = [keras.callbacks.TensorBoard("/content/drive/MyDrive/colab Notebooks/tb_logs")]
)
```

onto the LSTM model. The number of neurons, layers, dropout number and activation methods are being tested upon. The model overall will have an input layer, either 1 to 4 hidden layers, a dropout layer (for regularization) and finally the output layer. The model is also optimized with Adam, where it requires little memory and is computationally efficient, this is also good for problems with large data[REF] (which is what we have). This tuning was chosen since each test iteration takes so long, exhaustive searches like GridSearch will be infeasible in this case, Bayesian will try to find more optimal test parameters, based on the results of the previous, this can save

a lot of time. The Keras tuner has also been beneficial in the fact that, tuning progress is automatically saved, so in the case of an crash or error, we can easily reload the tuning process and carry it on. When the tuning is complete, we can easily obtain the best model from the tuner, this has also been saved for future easy reuse.

## **Issues During Implementation**

During implementation there were several issues that occurred, these are discussed below:

### **Large Nature of the Dataset**

Initially we started by working in local Jupyter Notebook, however it was soon found that this was infeasible. Due to the large nature of the dataset, it wasn't even possible to load the full training dataset, due to my devices limitations. This caused us find other technologies such as google colab, however we found that this still has it's own limitations, where we are limited to 12GB of RAM and an run-time of around 12 hours. Exhaustive searches such as Gridsearch and Bayesian Optimisation take an extremely long time to run, and sometimes it would either reach the maximum RAM capacity or the notebook would timeout, this caused us to have to run code multiple times, adjusting so that it wouldn't crash. This was frustrating, as it made it taker longer to get results, that it normally should've.

### **Getting data into correct format for LSTM**

To use the LSTM architecture, the data would need to be reformatted into a 3D shape. Even-though this was understandable conceptually, it was harder to implement than expected. There are many ways this formatting can happen, and getting into the format of *samples, timestep and features* was confusing. The particular confusion came under how to format the samples and timesteps, due to my lack of knowledge, these steps took longer than expected.

## 7 Results and Evaluation

In this section we will discuss the resulting accuracies from these models, and look at how well these models have been able to adapt. We will first go over the original aims of this study, the goal is to create an optimal machine learning model that is able to accurately forecast meter-readings in buildings. With this goal in mind, three different ML models have been developed, we'll first look and evaluate each of the models individually, then we will compare and see which models perform the best.

### Linear Regression Model

The results from this model are summarised in the table below:

Metric	Unprocessed Result	Pre-processed Result
RMSE	1.9394	1.9111
$R^2$	0.1797	0.1718

After processing we ended up with an RMSE of around 1.911, this is quite large and won't be accurate enough to be able to forecast data in the real world. The RMSE suggests there is an large different between predicted and actual values, which could suggest there isn't a strong linear relationship in this dataset. The small  $R^2$  value also indicates this, where it shows that there isn't a strong relationship between the independent features and our labels. This can indicated that Linear Regression was not the best approach towards this problem. I can also see that my pre-processing hasn't significantly affected the RMSE in any way, this could indicate that my pre-processing has been insufficient towards improving this model.



## LightGBM Model

The results from this model are summarised in the table below:

Result Type	RMSE	$R^2$
Unprocessed	1.2760	0.6448
Pre-processed	1.1669	0.6912
Pre-processed and Tuned	0.7215	0.8819

Firstly it is easy to see that the pre-processing steps was more successful here compared to Linear Regression. You can see how the RMSE has decreased by a moderate amount, the RMSE is also at a reasonable value being at 1.167, before tuning. This indicates that the error between predictions and actual values is smaller in comparison to the previous model. The RMSE significantly decreases after tuning, where its at a value of 0.7215, this is a significantly more accurate model. You can also see that the  $R^2$  value is quite high, this can indicate a strong connection between independent features and our labels. This overall shows that LightGBM has been a superior model over Linear Regression, while it is able to maintain speed and an high level of accuracy.

## LSTM Model

Due to error in my model creation, I have only been able to get RMSE metrics for my LSTM. Also due to the way I design this model, it was tuned automatically using the Keras tuner, when I only had the access to the best model. This means I will not be able to consider the unprocessed/untuned model for this. The results for this model are shown below:

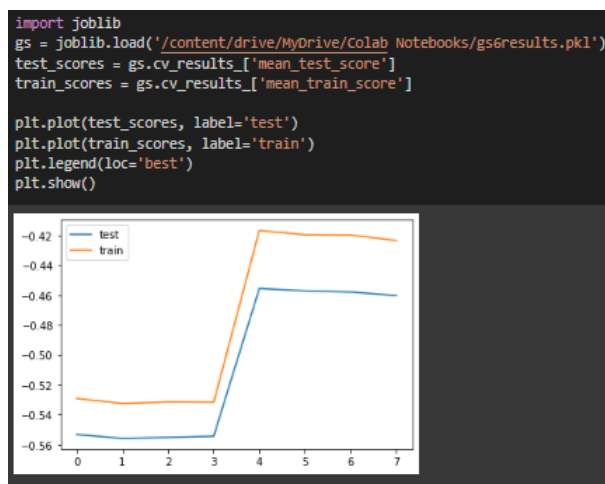
Loss for test model: 1.487947940826416

Loss for train model: 1.4819903373718262

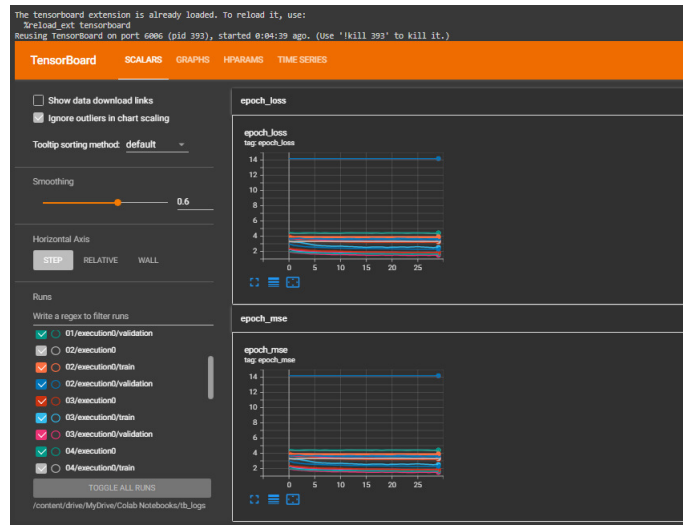
We can see the our LSTM model better than Linear Regression, however not as ideal compared to LightGBM after tuning. This has been a limitation in this study, where due to time constraints, I have not been able to tune the model enough, to be able to give it a fair comparison to the other models. Comparing the train and test losses, I can immediately see that there is very small difference between the two values. This could indicate although it may not be the most ideal model, it has been the model that has not over-fitted significantly.

## Looking at overfitting

We will further look at the LSTM and LightGBM models. Over-fitting is when the model will overly learn onto the training data-set and therefore won't be able to adapt to new data well. I decided to look at how much my models are overfitting, when I was tuning them with cross-validation. For my LightGBM model I simply plotted the validation score vs training score loss, whereas for the LSTM I have used a tool known as Tensorboard (Which will plot this for me), which will allow me to analyse the difference between the training and validation scores. Images for both of these are shown below:



You can see from the image above that my LightGBM model is overfitting, however this is to be expected. This is something that can be targeted in future study, and will be discussed in the coming sections.



You can see how the Tensorboard allows for the analysis of Keras tuner and all of its data. From this interface I can see that other than the first iteration, the overfitting has been minimal. This indicates that the results of this model could be further improved, with more tuning.

## Summary

Overall I can see from all of the analysis above, LightGBM turned out to be the best model for forecasting meter-readings in this study. All the evaluation metrics for LightGBM were lower, indicating a higher level of accuracy. This also follows to the related work I had seen previously, where solutions had also used combinations of Boosting algorithms. It was found that in this study, the pre-processing didn't have as large of an effect as expected (unlike hyperparameter tuning), this was surprising, many of the other participants considered this stage most important. It can also be considered that these models could've been further improved, if I had considered post-processing stages. It seems like the top solutions put an emphasis on post-processing and ensembling, and this could also improved my models even more.

## 8 Conclusions

The original aim of this project was to investigate different Machine Learning technique that will be able to accurately forecast meter-readings for building and weather data. I have been able to learn a multitude of techniques, which I have been able to apply and make some approach to solving this problem. I was able to successfully learn and implement Machine Learning Models and was able to use them to forecast meter-readings. I then evaluated the accuracy of said models, to find LightGBM as the most accurate of those that I studied. I was able to use techniques in hyperparameter tuning to try and get the most ideal model, which was seen with LightGBM. I have now learnt how complex it can become, where trying to get these models to be able to adapt to new instances of data, and the time and work needed to complete this. It was also able to see how my current models are still insufficient, and are suffering from over-fitting, which means my models might not be useful for predictions on unseen data.

## 9 Future Work

The above made it clear, even-though I was able to find the best model in this study, it can be improved to further accuracy. Due to time constraints we were only able to test a small number of models, this could mean there would be a model, which will perform better on this problem. This time constraints also restrict the analysis and pre-processing we could complete, with time, we would instead look at each building individually for outliers, like some of the contestants. With this, we could try and implement further models, using ensembling as suggested in some of those related works. Also due to computational constraints, we were not able to use the whole dataset in training, this means that the model could've been more accurate, had we have that computational power. In future study we could investigate different approaches that can be taken when working with such large data. With further time, we also could've tuned the models further, leading to even higher accuracies. Finally it is important to consider over-fitting at every stage of the study, if continuing or when solving another problem, we will take ore care in researching and implementations methods that will reduce this overfitting.

## 10 Reflection

This project overall has learnt me learn a multitude of skills technical and outside of that. When starting this project, I had no significant experience in Machine Learning. It found it fascinating learning from how I can go from obtaining the dataset, researching appropriate techniques, to implementing these techniques/models. I was also surprised to find how computationally expensive these tasks are, and it made me think significantly about how I could overcome these issues and now how important it is to optimise the tasks that I am completing.

I have also learned how to effectively have meeting with my Supervisor, where every week I would go into meetings with questions (If any issues had occurred that week) and used the advice from those sessions to further improve me work. Through-out the project I was also able to stay organised using the time template on my Initial plan. However I found how much I had underestimated this project, and how long some of those tasks would take. I have taken this information into account, and will not underestimate the depth, that is needed for some of these tasks.

One of the biggest struggled I had during this project was understanding how exactly some of these Machine Algorithms worked, like Recurrent Neural Networks, the math behind these are very complex, as are the concepts. I've learnt how to pick out important information that will aid in my project and implementation, while being able to now have the overview understanding of these concepts.

Overall I have greatly enjoying this introduction to Machine Learning and the applications/techniques that I've gained. I look forward to being able to apply and learn skills in future machine learning projects.

## References

- [1] <https://github.com/Microsoft/LightGBM/issues/695>
- [2] <https://www.kaggle.com/competitions/ashrae-energy-prediction/discussion/307391>
- [3] <https://www.kaggle.com/code/cereniyim/save-the-energy-for-the-future-1-detailed-eda>
- [4] <https://medium.com/geekculture/10-hyperparameters-to-keep-an-eye-on-for-your-lstm-model-and-other-tips-f0ff5b63fcd4>
- [5] <https://medium.com/analytics-vidhya/hypertuning-a-lstm-with-keras-tuner-to-forecast-solar-irradiance-7da7577e96eb>
- [6] <https://stats.stackexchange.com/questions/561357/how-to-identify-overfitting-from-lstm-plot-from-the-prediction-on-trainedunsee>
- [7] <https://stats.stackexchange.com/questions/348959/what-does-l2-regularization-in-lightgbm-do>
- [8] <https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/>
- [9] <https://www.codespeedy.com/diagnose-overfitting-and-underfitting-of-lstm-models-in-python/: :text=Diagnosing>
- [10] <https://blog.minitab.com/en/adventures-in-statistics-2/how-to-interpret-a-regression-model-with-low-r-squared-and-low-p-values>
- [11] <https://stats.stackexchange.com/questions/407328/controlling-overfitting-in-local-cross-validation-lightgbm>
- [12] [https://keras.io/api/keras\\_tuner/tuners/bayesian/bayesianoptimization-classhttps://builtin.com/data-science/gradient-descent](https://keras.io/api/keras_tuner/tuners/bayesian/bayesianoptimization-classhttps://builtin.com/data-science/gradient-descent)
- [13] <https://www.machinelearningplus.com/machine-learning/an-introduction-to-gradient-boosting-decision-trees/>
- [14] <https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/>

- [15] <https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/>
- [16] <https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/>
- [17] <https://medium.com/analytics-vidhya/how-does-linear-regression-work-implementation-with-sklearn-e5a850eddc89>
- [18] <https://www.ibm.com/cloud/learn/neural-networkstoc-neural-net-u3voPJVU>
- [19] <https://machinelearningmastery.com/tune-learning-rate-for-gradient-boosting-with-xgboost-in-python/>
- [20] <https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c>
- [21] [https://en.wikipedia.org/wiki/Cross-validation\\_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics)) : <https://machinelearningmastery.com/regression-metrics-for-machine-learning/>
- [22] <https://www.kaggle.com/competitions/ashrae-energy-prediction/data>
- [23] <https://viejournal.springeropen.com/articles/10.1186/s40327-018-0064-7>
- [24] <https://www.buildingsiot.com/blog/using-machine-learning-to-improve-building-energy-efficiency-bd>
- [25] <https://www.un.org/en/chronicle/article/health-effects-global-warming-developing-countries-are-most-vulnerable>