Abstract

Efficient production and consumption of electricity have become the focal point of modern civilization and industrialization. Due to economic and political reasons, we have yet to find any better yet scalable technology to find the next best alternative to the massively inefficient way of producing electricity with fossil fuel. In this task, I propose alleviating this problem by focusing on electricity consumption and exploring ways to find what modern design can offer. By exploring Site Energy Usage Intensity (EUI) on various stages and scales, I have identified sectors that contribute most to the inefficiencies. I also researched the various extraneous relevant factors that have some effects on it. Parameterizing these variables, I have modeled the prediction of site EUI using basic machine learning models such as Linear Regression and Gradient Boosting. Tuning these models' hyperparameters using standardized methods yields a highly reliable prediction of EUI. Finally, I have experimented with feature importance analysis to test the validity of our hypothesis. This experiment elicits further improvement scope of the proposed system, which is crucial to identify the most important factors to focus our research on the efficiency of electricity consumption.

Acknowledgements

Table of Contents

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 1: Introduction

Electricity is the primary driver of the modern advancement of civilizations. It is an interchangeable form of energy that can provide heating, lighting, or other potential energy. Thus, it has started a modern renaissance that will accelerate even faster in the coming future. However, its ubiquity posed a unique challenge from its inception. As it is produced en-masse at an ever-increasing rate, how it is produced is always likely to have a significant effect on its environment over time. Coal-powered plants were the earliest fully commercial producer of electricity.
The modern era of mass industrialization and globalization has gifted humanity with an unprecedented level of accessibility and a wonderful amount of production capability. Even in just the last century, our capability to scale up far outpaced the progress of the 500 years before. While it has gifted us with certain perquisites that our ancestors could never even dream of, the way this progress was achieved was not all benevolent for the environment we live in. From the outset of the 21st century, the majority of scientific research and our general observations pointed to one major realization that this progress that we are experiencing is at the cost of the very environment that we are living in. Studies have shown that there is a general surge in the average temperature all around the world (Tollefson, 2021) which has resulted in droughts, floods, stronger storms, and many other environmental calamities. A plethora of animal species have phased out of existence because of how inhospitable we have made the environment. Overall, the rapid rise in industrial prosperity has exacerbated the ecological imbalance far more than was anticipated before. Many of the crucial natural processes, like plant pollination and thriving coral reefs, are gradually being phased out of existence. A glaring outcome of the uptick in temperature can be observed in the vanishing polar icecaps - melting at a rate faster than ever. Studies have shown that the sea level is rising by 3.2 mms globally each year (MIMURA, 2013), and by 2050, most of the major coastal regions will lose a large chunk of their land to the oceans if it keeps continuing this way(Kulp and Strauss, 2019).

It has been widely acknowledged that inefficient energy production and an unrestricted rise in power consumption are two of the major reasons for accelerating the declining situation of the environment. We still rely on fossil fuels (i.e. Coal, Natural Gas etc) to fulfill 80% of our energy requirements globally (Maurya et al., 2021). These fuels emit a large amount of CO_2 gas, which accounts for 65% (United States Environmental Protection Agency, 2022) of all the Green House Gases (GHG) produced collectively. Thus, it is imperative to ensure efficiency during power production as long as we are forced to rely on fossil fuels. Additionally, we need to retrofit our current grid systems, electricity transportation, and the systems of our users to make sure we are being as efficient as possible even when we are reaching to the grass root populations. With that

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

view in mind, for this project, In this project, I have explored variables that may affect Site EUI. While exploring these variables revealed some interesting patterns, I have also explored the possibility of accurately predicting Site EUI. The main reasoning behind this is to have an advanced estimate of how much the establishment can incur in efficient electricity use. This can help policymakers target retrofitting efforts which can decrease GHG emissions significantly. In addition, I have experimented with dimensionality reduction and grid search to find the most significant factors in predicting the given variable. The contributions of my research is as follows:

- I identify the relationship between different variables and site EUI and analyze these variables over time. Moreover, I evaluate and find out the biases that may be underlying in the dataset.
- I predict the site EUI using basic machine learning models such as Linear regression, Gradient boosting, Decision tree regressor, and Random forest regressor.
- I do hyperparameter tuning to find the best parameters of my trained models, and I also analyze the importance of different features which have been used to train the models.

-

Related Studies

A recent study published by the IEA (International Energy Agency) has shown that 37% of all global energy-related and process-related CO2 emissions are directly or indirectly attributed to the lifecycle of the infrastructure we build. According to reports, the buildings in industrialized or developed nations with 50–65% of power usage used roughly 35–40% of the total energy (Eia.gov, 2018). For example, more than 60% of the power used in the United Kingdom is used by different industries, services, and domestic sectors, which make up the majority of energy consumers(Department for Business, Energy and Industrial Strategy, 2021). In the United States, 41% of the primary energy consumers were comprised of residential and commercial buildings in 2010 (Yang, C., Choi, J.H., Noble, D. and Schiler, M., 2015).
There are several ways to lower carbon dioxide emissions and the energy consumption of these buildings - while keeping buildings' energy needs met - by using a combination of enforcing implementation of some meticulous strategies and utilizing available technologies during the planning and the entire duration of the life cycle of the building. Energy efficient technology, smart design, low carbon appliances, and high efficiency HVAC systems that are already well established and widely used are just a few ways to meet that goal (IPCC, 2019) (Pacala, 2004).

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

In recent years, the concept of ‘retrofitting’ or re-engineering existing buildings and infrastructures has gained much traction among the concerned authorities. Indeed, it has been demonstrated that making the right long-term choices for building renovation and efficiency may greatly improve thermal performance and thus lower energy use (Aboelata, 2021) (Liao and Cao, 2013). Several studies suggest (Sharma et al., 2022)(Kharseh and Al-Khawaja, 2016),that we can reduce the air conditioning energy requirement by up to 50% by simply retrofitting the pre-existing buildings with apt equipment and changes in some small building planning. Consequently, this would imply a reduction in the amount of CO2 these buildings tend to emit by up to 30%(Zafirah, M.F. and Mardiana, A., 2014).

However, prior to taking a step towards building sustainable structures and, consequently, a sustainable future, it is important to first quantify the current energy consumption of existing buildings. Buildings are intricate, unique systems with many different physical, functional, and environmental characteristics. Numerous building-related factors are included in energy consumption modelling. Cross-sectional and temporal patterns of energy use are influenced by climatic factors including temperature, precipitation, and snowfall, among others. Building height, orientation, volume, floor area, façade area, site area, window-to-wall ratio, volume-to-façade area ratio, etc. are only a few of the multitudes of aspects that make up a building's features. Therefore, projecting how much energy will be used by such facilities presents a number of challenging obstacles. Popularly, a measure called "Site Energy Usage Intensity" or EUI (Andrews and Krogmann, 2009) is used to estimate the level of energy consumed by a building and contains aggregated energy performance information, as reflected in utility bills.

Hypothesis

Power production and, consequently, the use of it remain one of the most contentious issues regarding the new era of climate friendly policy implementation. For this project, I would like to focus on the energy consumption and related datapoints to explore the usage of power in different facets of life (commercially or residentially). If we can explore the pattern of behavior and identify the major inefficient consumers of this system, we can shift our research and development focus to sectors that will need it the most.

My hypothesis here is that there are some significantly irresponsible consumer infrastructures that can be identified through this project. The analysis will also be able to discern the underlying patterns giving rise to such behaviors. Consequently, arming those particular entities with the most efficient technologies will help us move forward to a better, more sustainable future rapidly.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 2: Dataset Analysis

Power consumption datasets that allow for the advance evaluation of created algorithms are required by scientists and experts in smart energy monitoring systems in order to test energy efficiency solutions and suit actual scenarios of daily power usage. It is challenging to choose a database for addressing energy efficiency issues because there are numerous datasets that can be obtained in the literature, and each one has its own unique qualities. For this project, I have primarily used the publicly available Building Energy Efficiency Dataset collaboratively built by the US Environmental Protection Agency (EPA), MIT Critical Data, Climate Change AI (CCAI), and Lawrence Berkeley National Laboratory (Berkeley Lab). The collected dataset* contains over 75k observations of building energy usage records within several states in the United States. It was collected for over seven years. It contains buildings characteristics data such as floor area, facility types, floor area, foundation year, and energy star rating. It also contains the surrounding weather data for the given buildings and energy usage for the building, measured as Site Energy Usage Intensity or SIte EUI. There are in total 63 columns of information in this dataset.

Chapter 2.1: Data Introduction

Main Feature Columns

1. id: building id
2. Year_Factor: anonymized year in which the weather and energy usage factors were observed
3. State_Factor: anonymized state in which the building is located
4. building_class: building classification
5. facility_type: building usage type
6. floor_area: floor area (in square feet) of the building
7. year_built: year in which the building was constructed
8. energy_star_rating: the energy star rating of the building
9. ELEVATION: elevation of the building location
10. january_min_temp: minimum temperature in January (in Fahrenheit) at the location of the building
11. january_avg_temp: average temperature in January (in Fahrenheit) at the location of the building

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

12. january_max_temp: maximum temperature in January (in Fahrenheit) at the location of the building
13. february_min_temp: minimum temperature in February (in Fahrenheit) at the location of the building
14. february_avg_temp: average temperature in February (in Fahrenheit) at the location of the building
15. february_max_temp: maximum temperature in February (in Fahrenheit) at the location of the building
16. march_min_temp: minimum temperature in March (in Fahrenheit) at the location of the building
17. march_avg_temp: average temperature in March (in Fahrenheit) at the location of the building
18. march_max_temp: maximum temperature in March (in Fahrenheit) at the location of the building
19. april_min_temp: minimum temperature in April (in Fahrenheit) at the location of the building
20. april_avg_temp: average temperature in April (in Fahrenheit) at the location of the building
21. april_max_temp: maximum temperature in April (in Fahrenheit) at the location of the building
22. may_min_temp: minimum temperature in May (in Fahrenheit) at the location of the building
23. may_avg_temp: average temperature in May (in Fahrenheit) at the location of the building
24. may_max_temp: maximum temperature in May (in Fahrenheit) at the location of the building
25. june_min_temp: minimum temperature in June (in Fahrenheit) at the location of the building
26. june_avg_temp: average temperature in June (in Fahrenheit) at the location of the building
27. june_max_temp: maximum temperature in June (in Fahrenheit) at the location of the building
28. july_min_temp: minimum temperature in July (in Fahrenheit) at the location of the building
29. july_avg_temp: average temperature in July (in Fahrenheit) at the location of the building
30. july_max_temp: maximum temperature in July (in Fahrenheit) at the location of the building
31. august_min_temp: minimum temperature in August (in Fahrenheit) at the location of the building

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

32. august_avg_temp: average temperature in August (in Fahrenheit) at the location of the building
33. august_max_temp: maximum temperature in August (in Fahrenheit) at the location of the building
34. september_min_temp: minimum temperature in September (in Fahrenheit) at the location of the building
35. september_avg_temp: average temperature in September (in Fahrenheit) at the location of the building
36. september_max_temp: maximum temperature in September (in Fahrenheit) at the location of the building
37. october_min_temp: minimum temperature in October (in Fahrenheit) at the location of the building
38. october_avg_temp: average temperature in October (in Fahrenheit) at the location of the building
39. october_max_temp: maximum temperature in October (in Fahrenheit) at the location of the building
40. november_min_temp: minimum temperature in November (in Fahrenheit) at the location of the building
41. november_avg_temp: average temperature in November (in Fahrenheit) at the location of the building
42. november_max_temp: maximum temperature in November (in Fahrenheit) at the location of the building
43. december_min_temp: minimum temperature in December (in Fahrenheit) at the location of the building
44. december_avg_temp: average temperature in December (in Fahrenheit) at the location of the building
45. december_max_temp: maximum temperature in December (in Fahrenheit) at the location of the building
46. cooling_degree_days: cooling degree day for a given day is the number of degrees where the daily average temperature exceeds 65 degrees Fahrenheit. Each month is summed to produce an annual total at the location of the building.
47. heating_degree_days: heating degree day for a given day is the number of degrees where the daily average temperature falls under 65 degrees Fahrenheit. Each month is summed to produce an annual total at the location of the building.
48. precipitation_inches: annual precipitation in inches at the location of the building
49. snowfall_inches: annual snowfall in inches at the location of the building
50. snowdepth_inches: annual snow depth in inches at the location of the building
51. avg_temp: average temperature over a year at the location of the building
52. days_below_30F: total number of days below 30 degrees Fahrenheit at the location of the building

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

53. days_below_20F: total number of days below 20 degrees Fahrenheit at the location of the building
54. days_below_10F: total number of days below 10 degrees Fahrenheit at the location of the building
55. days_below_0F: total number of days below 0 degrees Fahrenheit at the location of the building
56. days_above_80F: total number of days above 80 degrees Fahrenheit at the location of the building
57. days_above_90F: total number of days above 90 degrees Fahrenheit at the location of the building
58. days_above_100F: total number of days above 100 degrees Fahrenheit at the location of the building
59. days_above_110F: total number of days above 110 degrees Fahrenheit at the location of the building
60. direction_max_wind_speed: wind direction for maximum wind speed at the location of the building. Given in 360-degree compass point directions (e.g. 360 = north, 180 = south, etc.).
61. direction_peak_wind_speed: wind direction for peak wind gust speed at the location of the building. Given in 360-degree compass point directions (e.g. 360 = north, 180 = south, etc.).
62. max_wind_speed: maximum wind speed at the location of the building
63. days_with_fog: number of days with fog at the location of the building

Categorical Feature Columns

Categorical Features

index	Unique Values
Year_Factor	6
State_Factor	7
building_class	2
facility_type	60

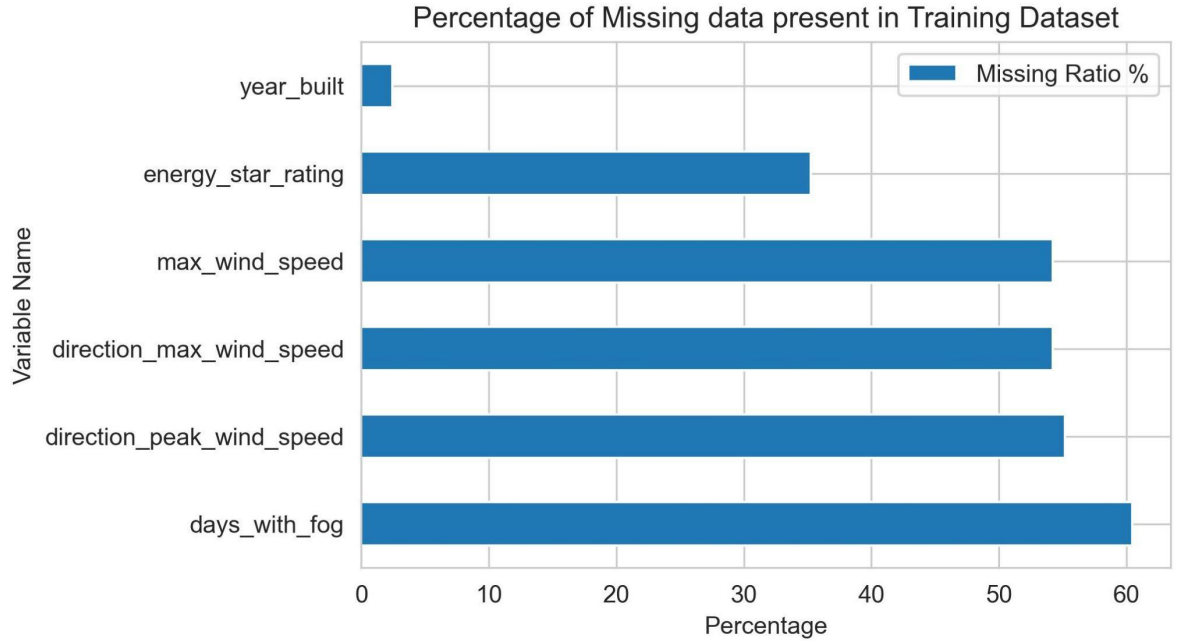
*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Target

site_eui: Site Energy Usage Intensity is the amount of heat and electricity consumed by a building as reflected in utility bills

Chapter 2.2: Missing Data

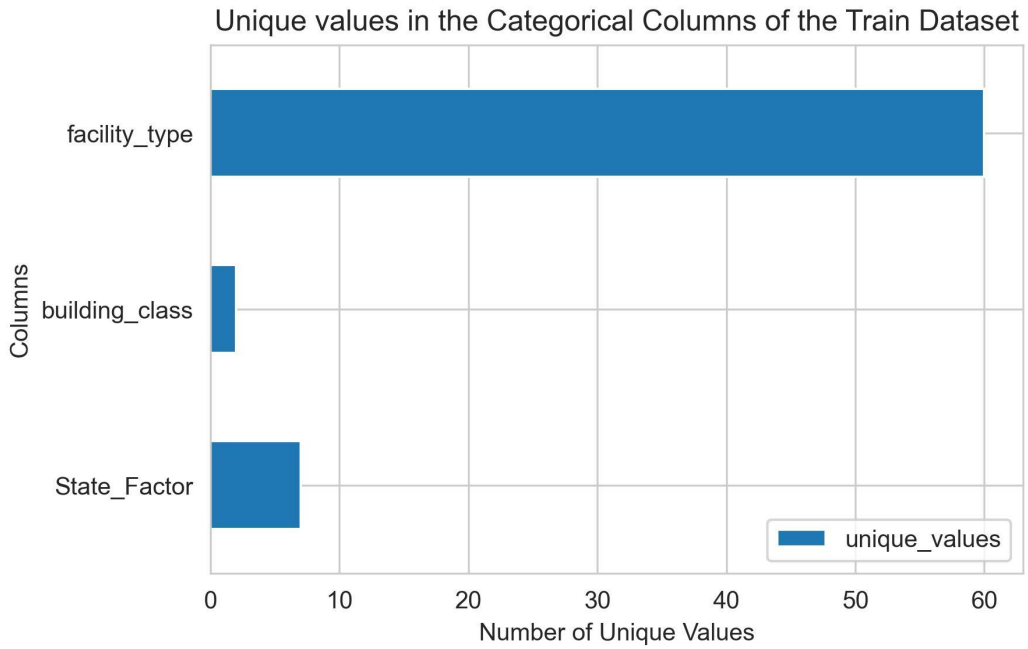
First, I look into the amount of missing or invalid data in our dataset. While most columns have all the necessary information, Days with Fog, Wind Speed related data,



and Energy Star Rating data have a significant number of missing data points. Days_with_fog contains 60.5% missing data points. One possible reason behind it could be that we do not see many foggy days in a year. Moreover, We do not have ‘year_built’ data for 2.4% of the building. Wind speed was also not collected for a significant number of areas for reasons unknown.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 2.3: Categorical Values



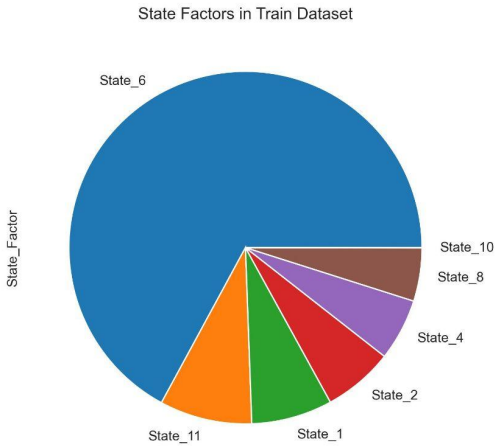
There are three categorical columns in the dataset: facility_type, State_Factor and building_class. From the plot below we find that :

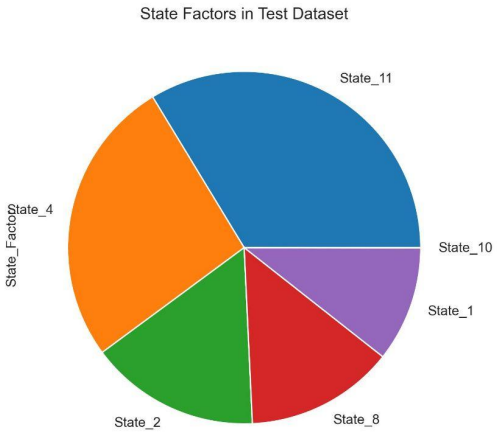
- facility_type has 60 unique values
- building_class has 2 unique values
- state_factor has 7 unique values

We will need to convert these categorical values into numerical values for the model to be able to understand them.

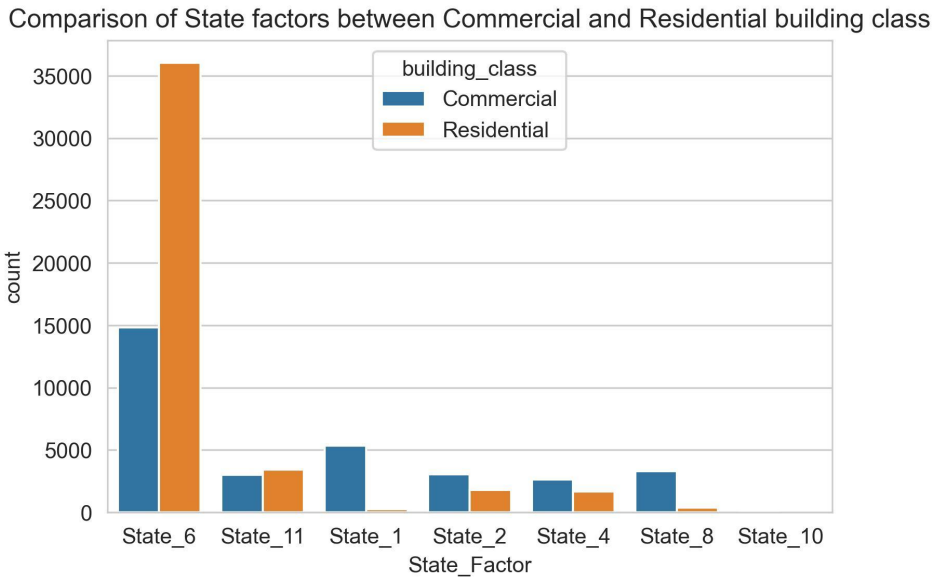
*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 2.3: State Factor





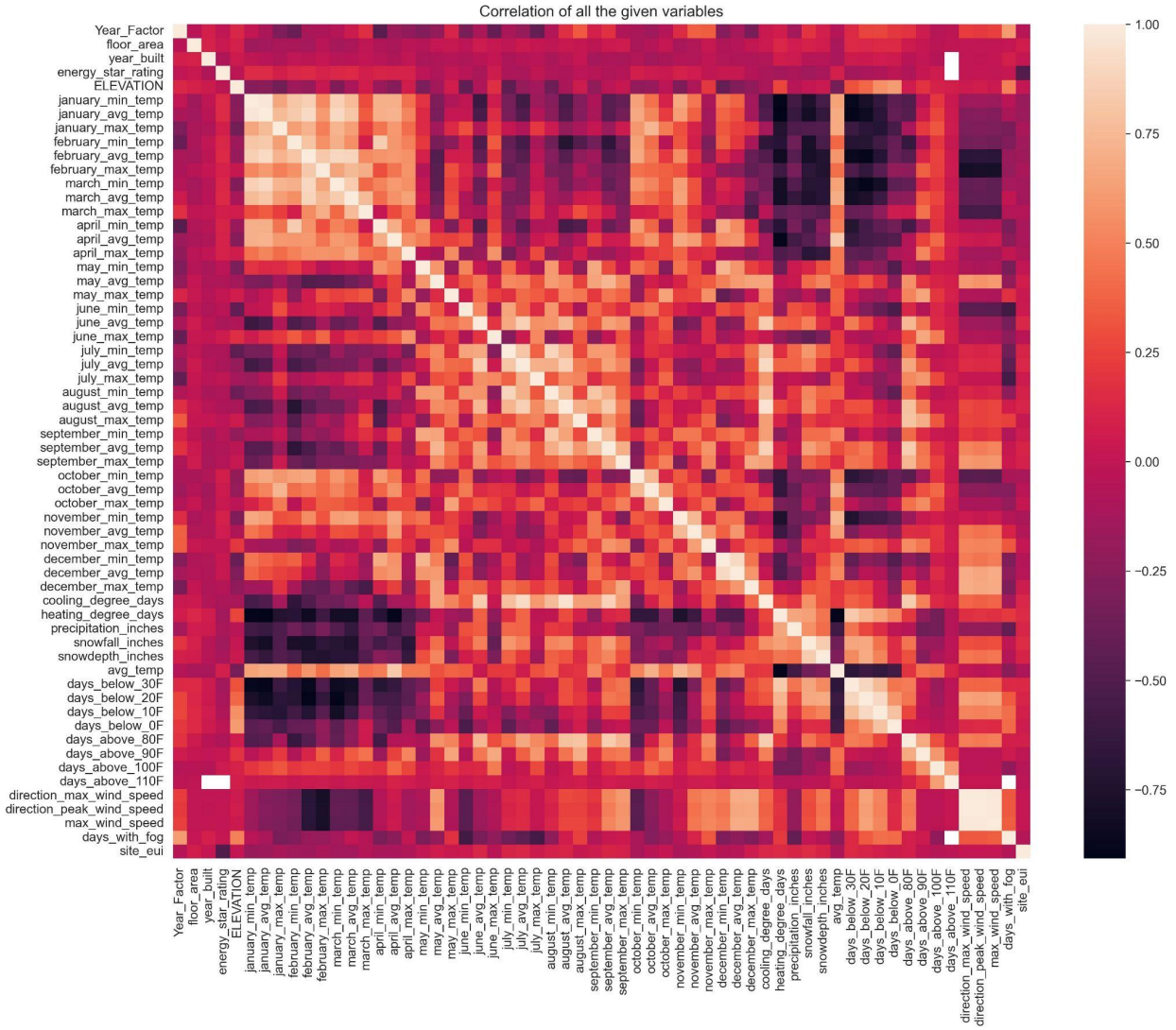
Our test dataset and train dataset contains the ratio of the data that are vastly different from each other concerning state_factor. State_factors represent the anonymized state information about the building where it is located.



The bar chart shows that state_6 is the most data-rich state, while state_10 has negligible amount of data compared with other states mentioned in the training dataset. This imbalance may introduce bias into our prediction where our prediction for state_10 may not be very accurate.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 2.5: Correlation of all the variables among each other

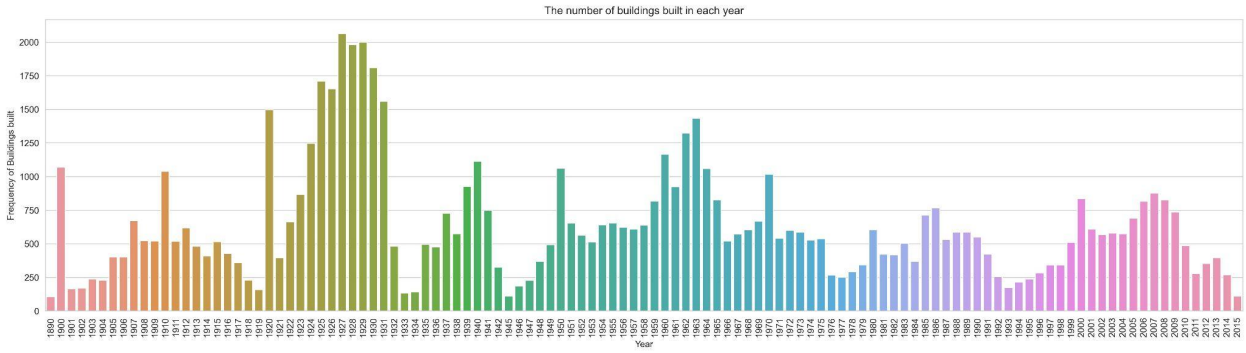


This plot shows the correlation of different factors in the dataset. The lightest point shows the highly correlated data points, and the darkest point shows the least correlated data points. There are 4 critical findings from this chart:

- The *elevation* is a minor factor behind the weather data collected for each building.
- *year_built* has a surprisingly high correlation with the *days_above_110F*. But this correlation seems purely coincidental.
- *energy_star_rating* has a high correlation with the hotter areas.
- *Precipitation* negatively correlates with *avg_temp*.

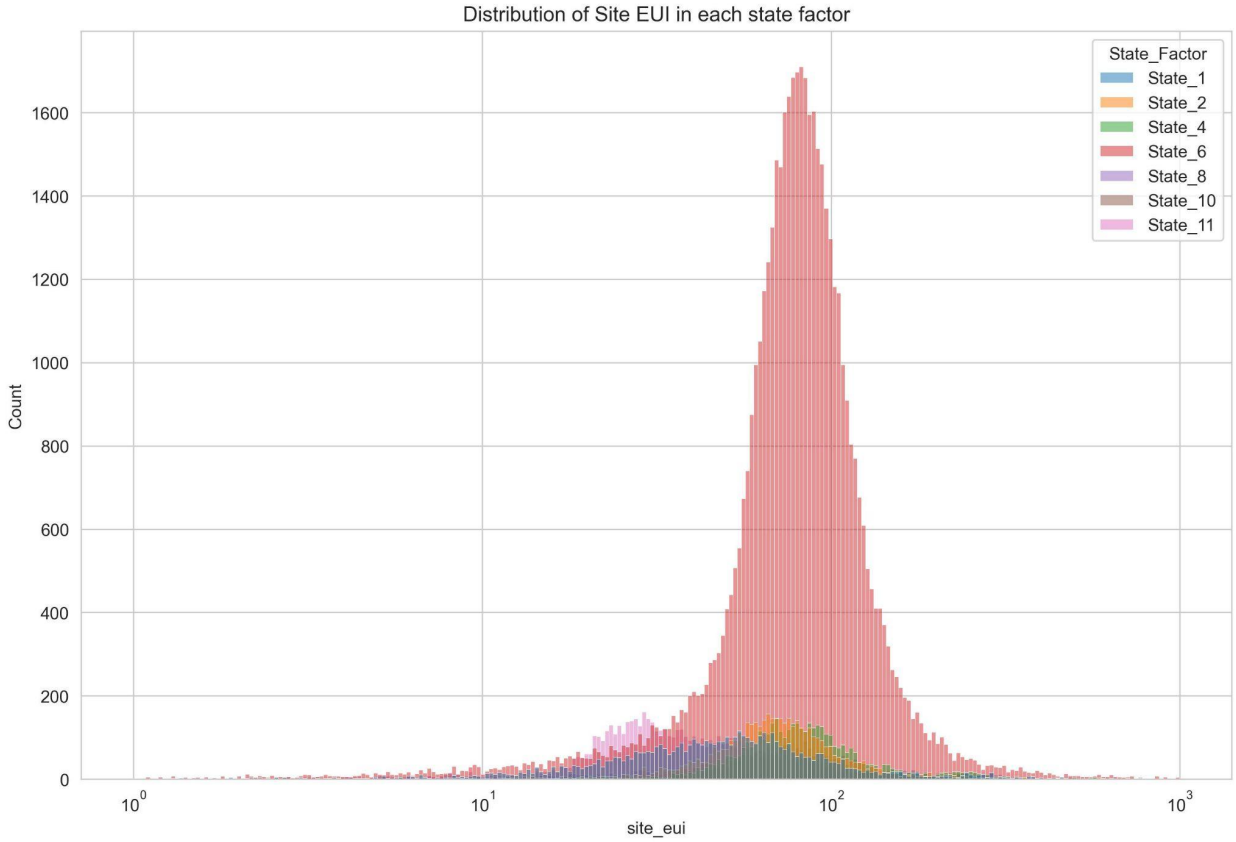
*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 2.6: The Number of buildings built in each year



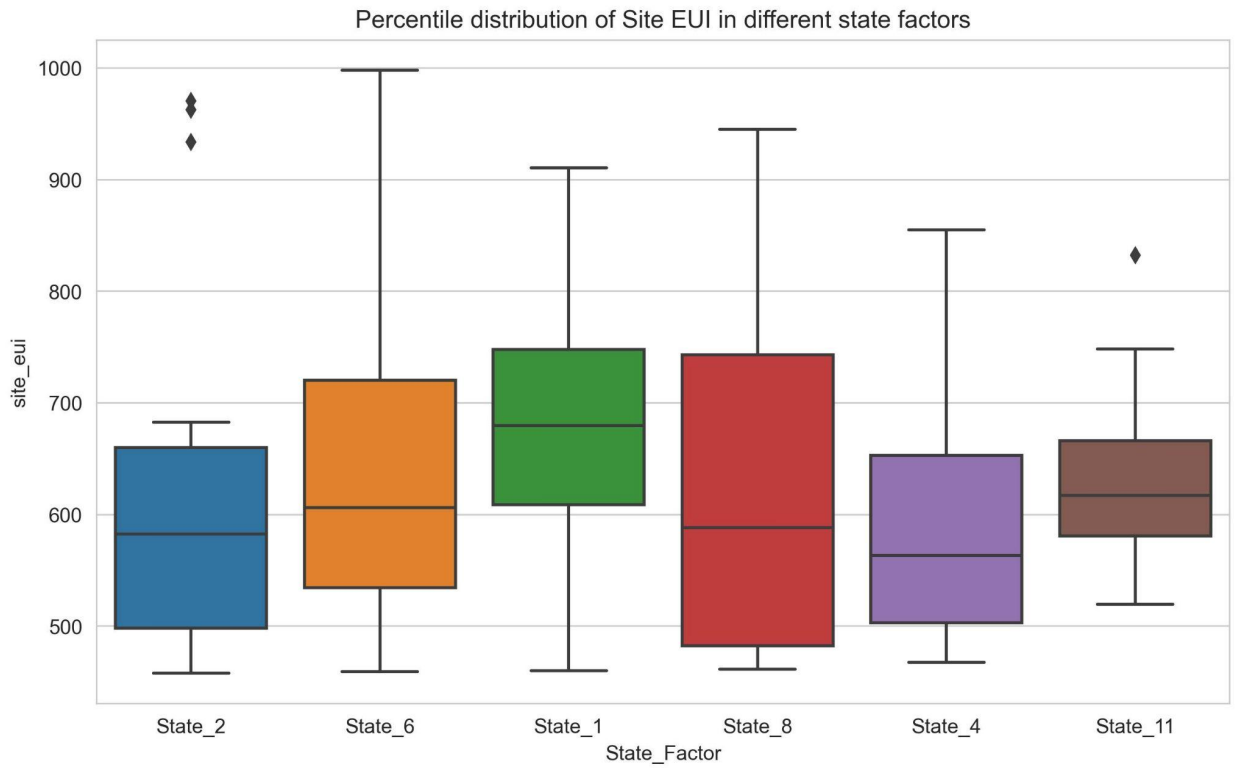
The dataset maintains an even distribution of samples of house data from every decade. However, buildings built during the 1920s represent significantly more data points than in any other decade.

Chapter 2.7: Contrasting Site EUI with State Factor



This plot shows the distribution of site_eui for each state factor. This was plotted on log scale as the range for the site_eui data is large enough to contain extreme outliers. Ignoring the extreme outliers in the given chart shows that the site_eui data follows a normal distribution pattern.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022



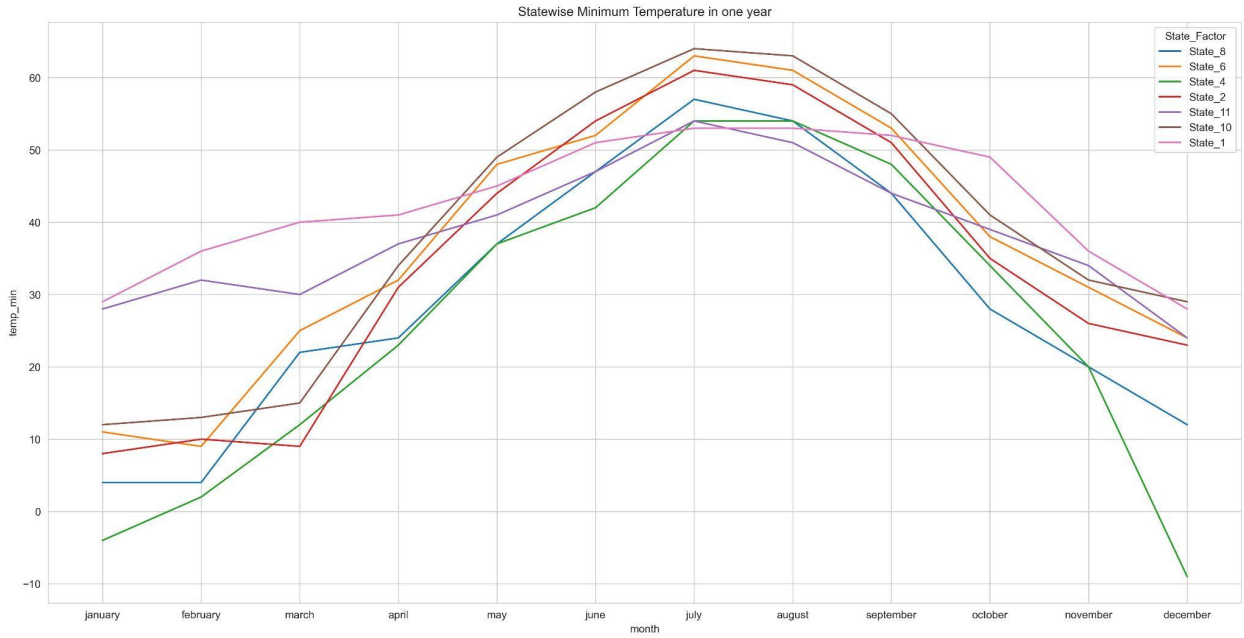
Percentiles measure the skews and variants in a dataset. *State_2* have relatively small range with a median value of 580. While *State_6* has a greater range, median values are quite similar to *state_2.* However, the final quartile range spans from 720 to 1000 and it is left skewed. *State_1* and *State_11* has an even distribution with median value of 680 and 620 respectively. *State_8* and *State_4's site_eui* values are left skewed with median values less than 600.

Chapter 2.8: Contrasting minimum temperature per month with State Factor

Electricity demand for cooling is significantly affected by climate and weather:

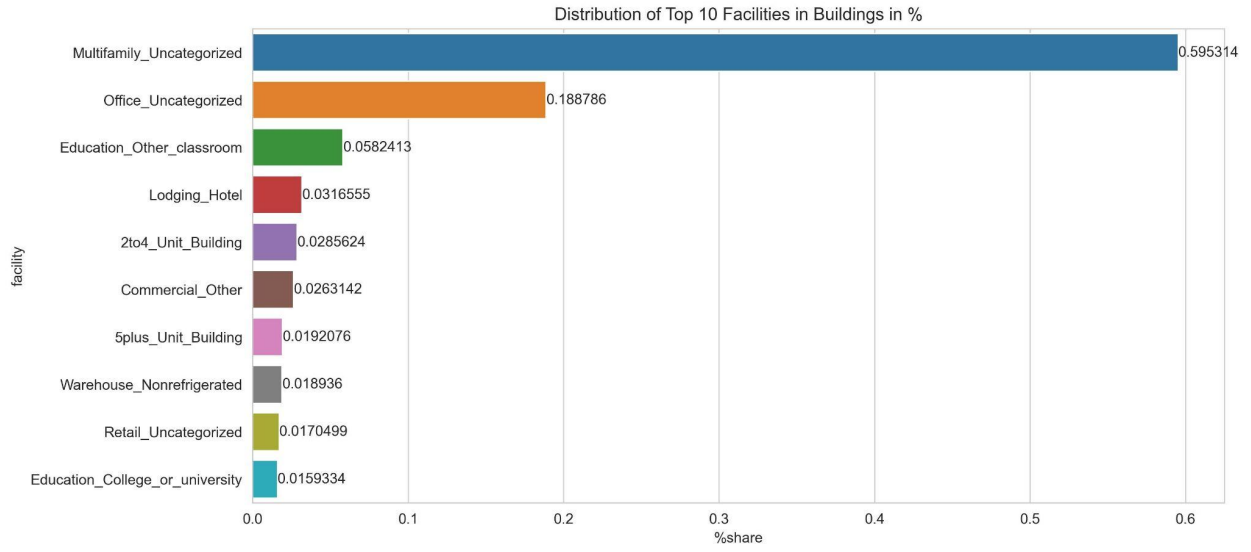
- *State_4* seems to be having the lowest min temperature among other states
- It appears that, in comparison to other states, *state_11* has a moderate temperature. The temperature trend line reveals that *State_11*'s lowest and maximum temperatures range from the lower 30s to the upper 50s and from the upper 60s to the upper 90s, respectively.
- Media temperature for state 10 is higher than the rest of the states

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022



Chapter 2.9: Distribution of Top 10 facilities in buildings

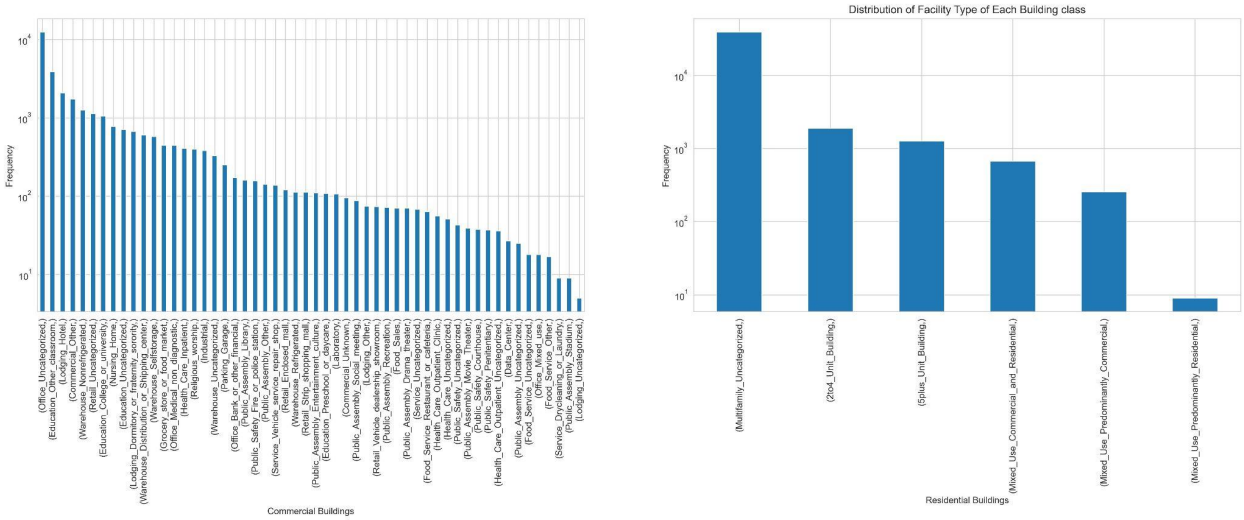
50% of our training dataset are for multifamily residents while only 1% data are for Educational institutions, Retail shops and warehouses. It's safe to assume that we will have a better capability of understanding the behavior of energy usage of residential buildings from the given dataset.



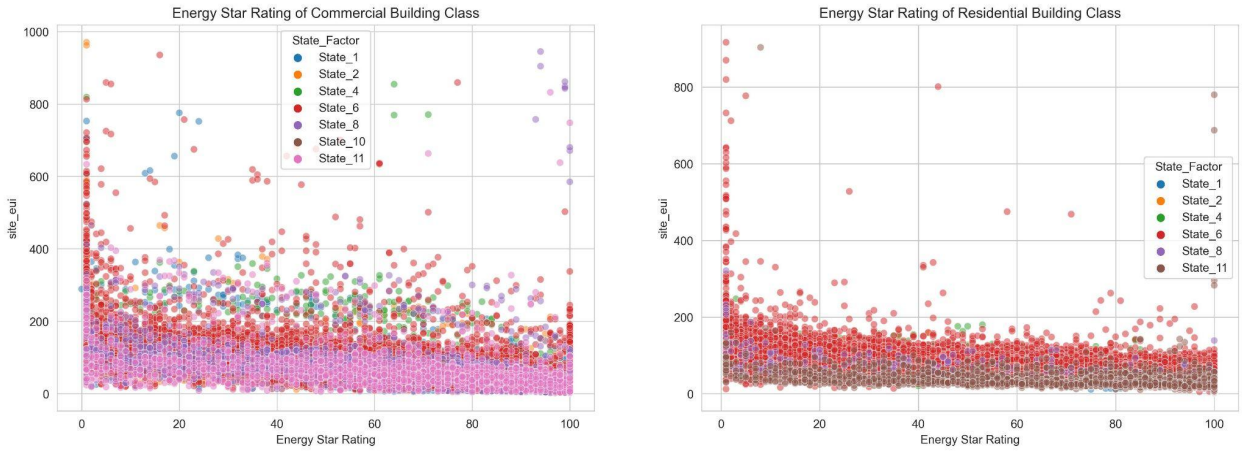
*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 2.10: Contrasting the distribution of facility type of each building class

There are only 6 types of residential facility buildings available in the dataset. The rest are all commercial buildings. Therefore, The dataset is more likely to have a better understanding of the energy usage of commercial buildings.



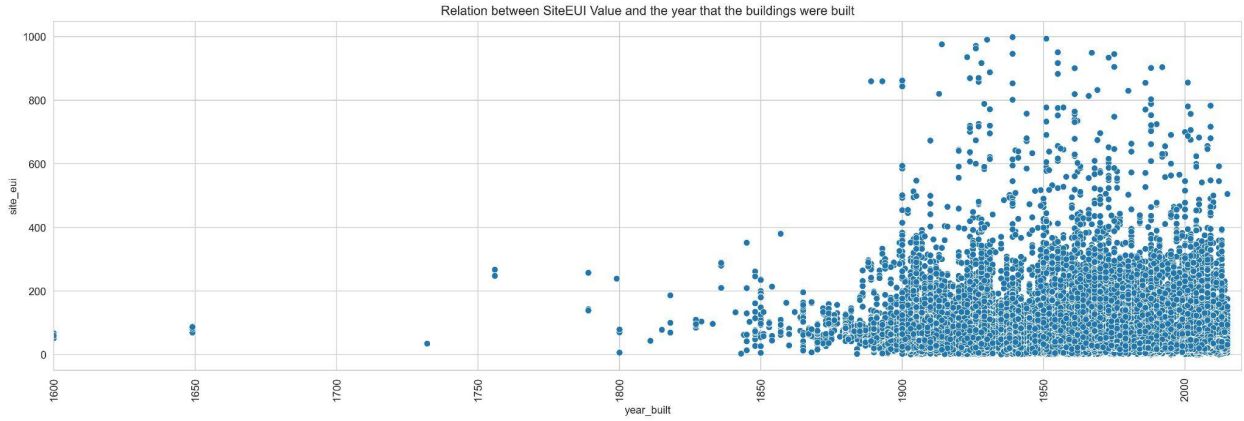
Chapter 2.11: Contrasting the distribution of Energy Star Rating of each building class



The Building Energy Star rating is usually high for buildings that have low *Site_EUI*. Another observation here is that *state_10* is missing for Residential buildings.

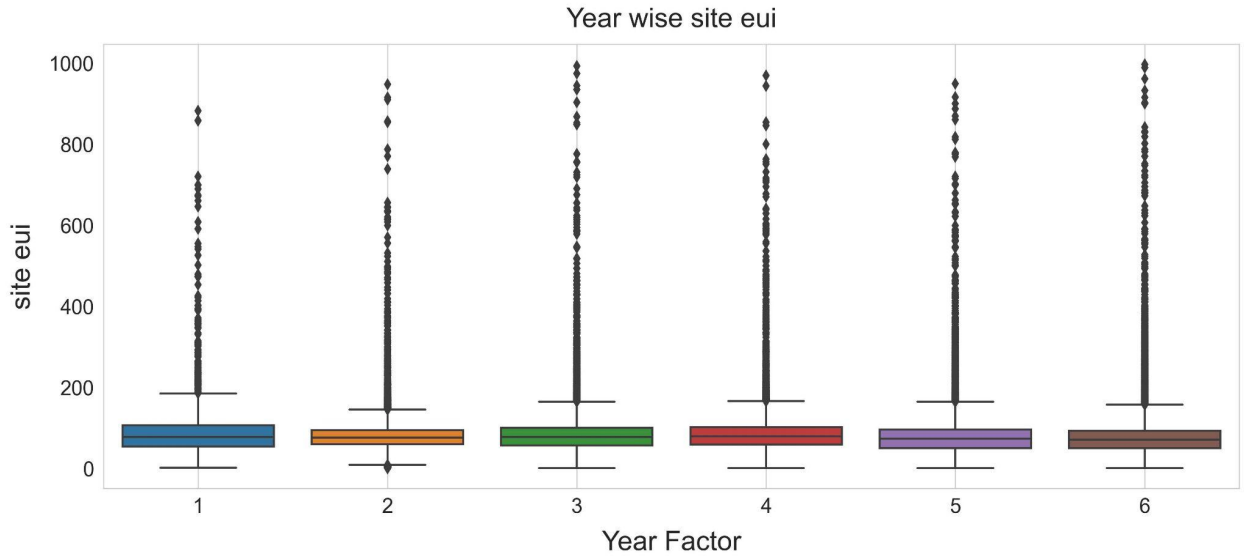
*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 2.12: Scatterplot of SITE EUI Values in accordance of their year built in the training data



From the range and percentile value of site_eui over the years, we can see that the range of site_eui has significantly increased. This trend can be attributed to the electrical appliances and machinery we have used over the years. Mainly since the 1980s, the increase of site_eui has been observed significantly.

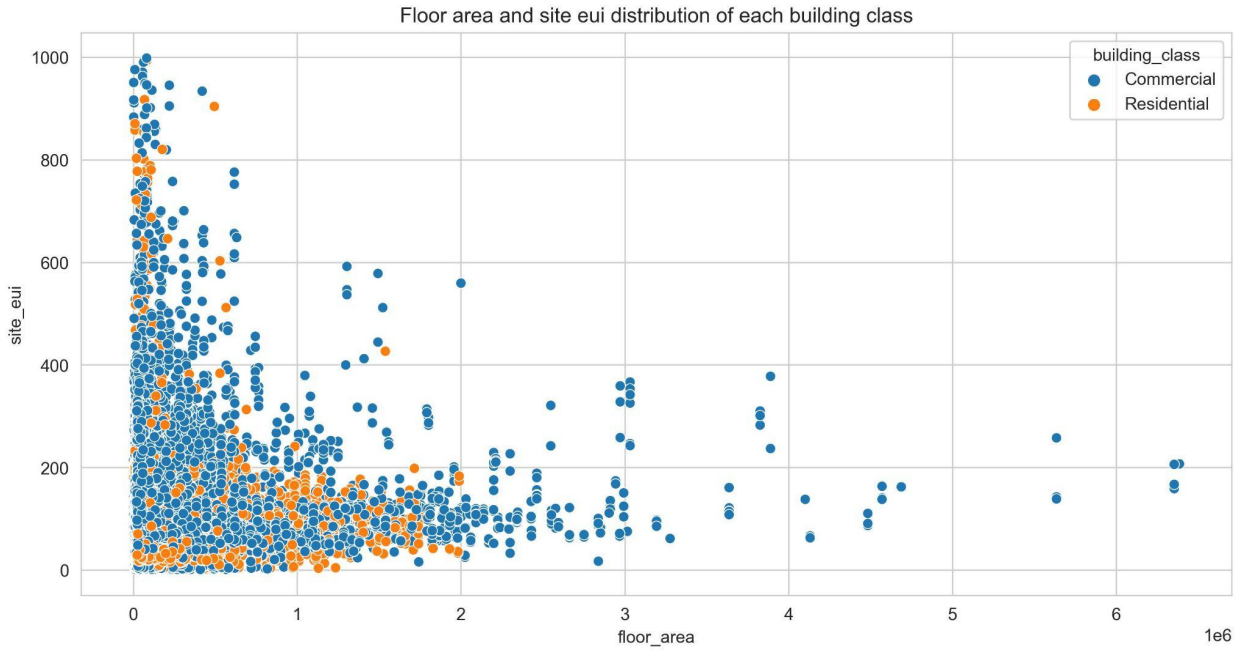
Chapter 2.13: Contrasting Site_EUI values to year factor



From the plot it is evident that - year factor contributes very little to the variation of SITE EUI values and remains the same across all values of year factor.

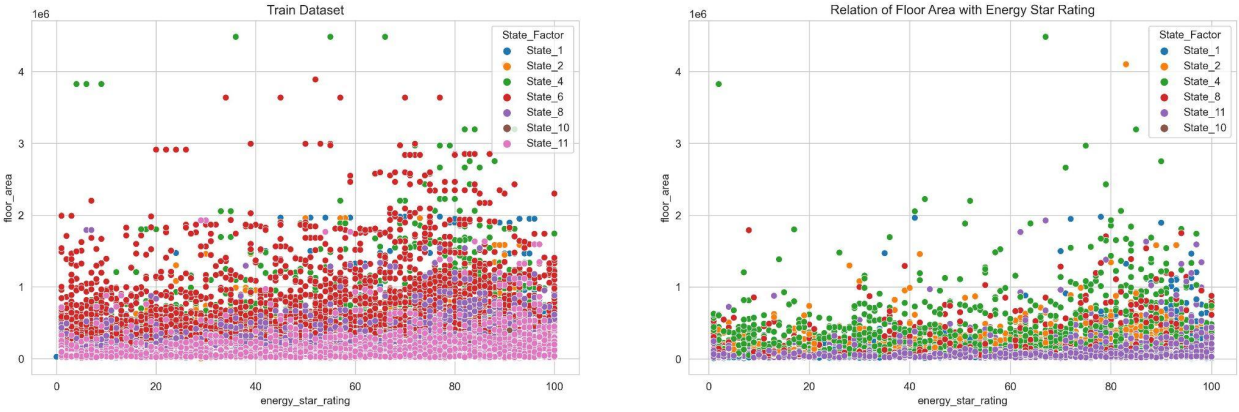
*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 2.14: Floor area distribution of Each Building class



The plot shows that the site_eui generally ranges from extremely high to low for both commercial and residential areas, but larger commercial areas always have a meager amount of site_eui. This means that while residential neighborhoods are smaller than commercial areas due to the usage of appliances, utility bills are always significantly higher for residential areas, and big commercial warehouses don't need that much weather control inside. Additionally, the floor area outliers are mainly related to commercial buildings.

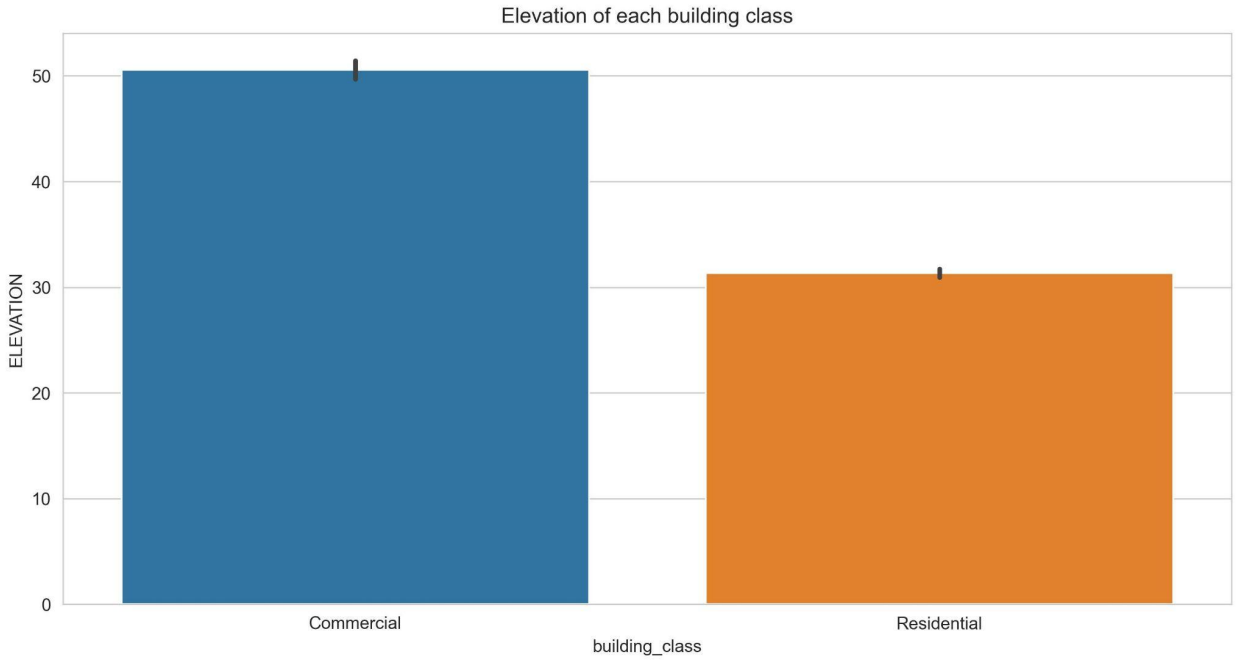
Chapter 2.15: Contrasting floor area to energy star rating



*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

State_6 is missing in the test data in this plot. However, the distribution of datapoints across *floor_area* and *energy star rating* seems to follow the same trends and range, with some outliers.

Chapter 2.16: Contrasting Elevation of different building classes



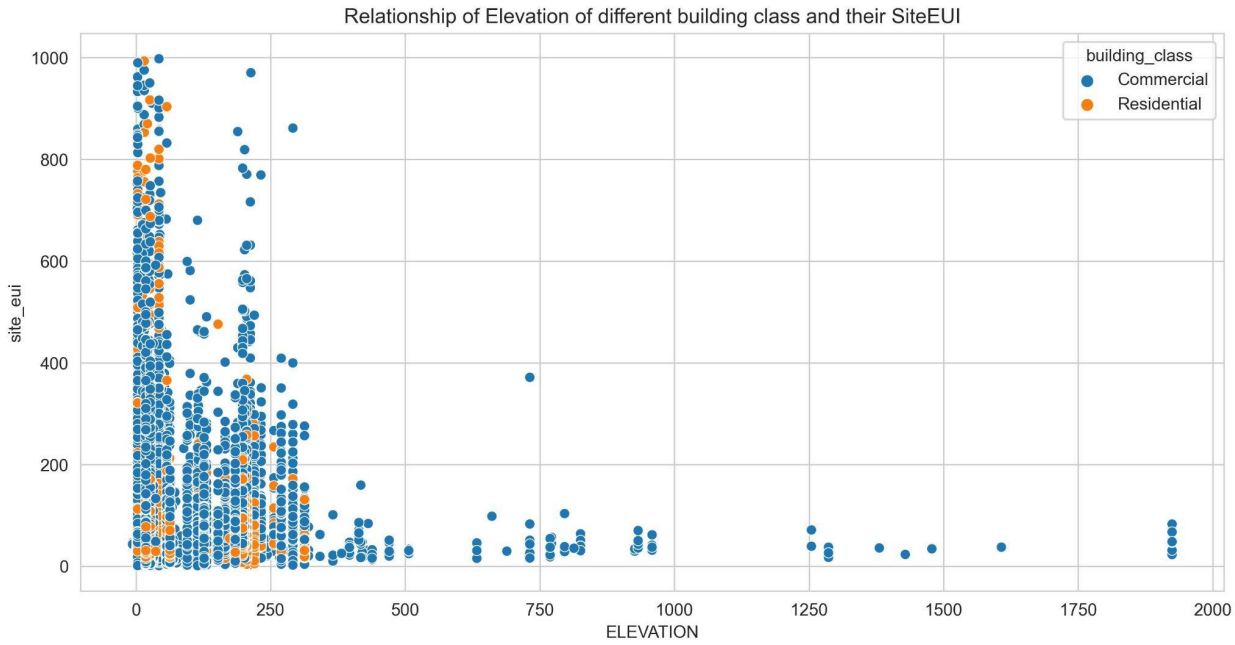
Commercial buildings make up the majority of the structures located over 300 feet (elevation). However, as there are no residential structures over 500 feet (Elevation), commercial structures are what the Elevation variable's outliers are tied to.

Chapter 2.17: Contrasting Elevation of different building classes with their EUI

Elevation is also a noticeably significant factor for site_eui. Areas most closely located to the ground have a higher amount of air circulation control than those high above the ground, which incurs a very low amount of utility bills.
The plot shows that buildings below 300 (elevation) are using more energy. So, more significant energy users (like Data warehouses) can be suggested to be placed high up

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

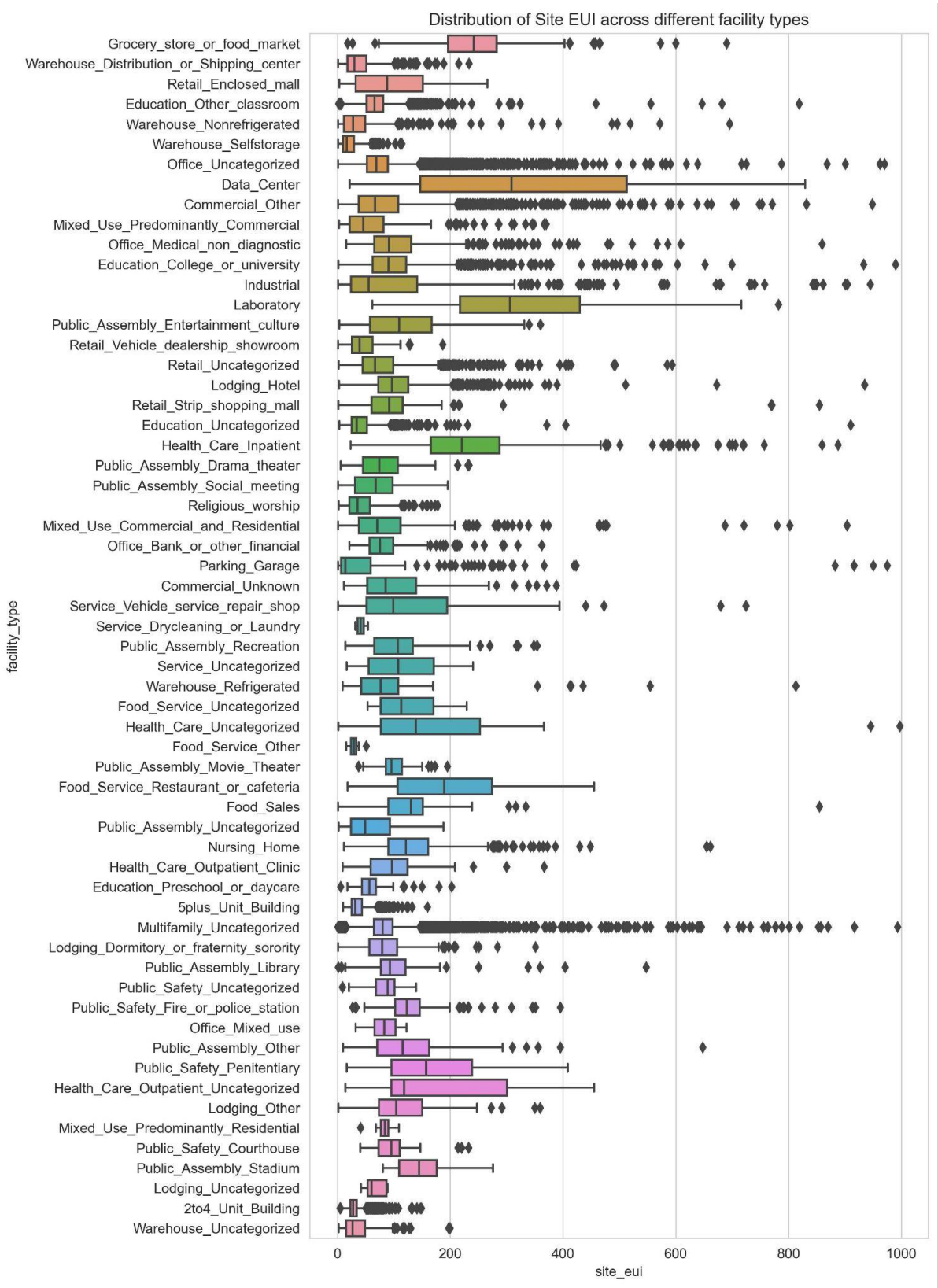
in the buildings.



Chapter 2.18: Contrasting the distribution of SITE_EUI across different facility types

On average, grocery_store/food_market, data_centers, and laboratories have higher site_eui. These facilities are usually larger than the rest of the facilities and/or have higher user concentration.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022



*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 3: Modeling Description

In the previous chapter, we have demonstrated and discussed some of the prominent features of our dataset. Most importantly, we discover that :

1. The percentage of missing or NaN values in the training data :
    - “days_with_fog”: 60.5%
    - “direction_peak_wind_speed”: 55.2%
    - “max_wind_speed”: 54.2%
    - “direction_max_wind_speed”: 54,2%
    - “energy_star_ratng”: 35.3%
    - “year_built”: 2.4%
2. There are three categorical columns in the dataset: facility_type, State_Factor and building_class :
    - facility_type has 60 unique values
    - building_class has 2 unique values
    - state_factor has 7 unique values
3. There are numerical features difference between train and test data:
    - The biggest difference in the distribution of the train and test datasets is found in wind speed data. Wind relate data have over 50% missing data.
    - For the test dataset, the only available year factor is 7 - so there is no overlap
    - In case of elevation there is a sharp right skewed spike in data distribution of training dataset, which is not present in test dataset.
    - Similar but opposite trend is observed in snow_depth and snowfall_inches data where there is a sharp right skewed spike in data distribution of testing dataset, which is not present in training dataset.
    - For rest of the factors, the training and test distributions mostly overlap with each other

Keeping these observations in mind, we proceed to build a computational model of our problem. Given 63 columns of climactic data, our objective is to predict the SITE EUI variable. We model our problem as a Regression task.
The steps are as followed :

- We preprocess our dataset to address the missing, NaN and categorical values. We then split the data into appropriate ratio and use them to evaluate our modeling.
- We build several well-known Statistical Regressors to predict our target variable. The models are optimized through feature selection and parameter search.
- A Neural Network model is developed and compared with the best performing Statistic Regressor

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

- Finally, we perform a simple weighted model ensemble of the best performing models to generate the final prediction.

Chapter 3.1: Dataset Preprocessing

As mentioned in the previous subsection, our dataset is constrained by several factors. To mitigate the effects of these factors, we need to preprocess our dataset to make them appropriate for feeding to the models.

Filling Missing or NaN values

Since 4 of our numerical columns have more than 50% missing or NaN entries, it is important to impute these values. For this task we use a SimpleImputer from the SkLearn Library, which uses a descriptive statistic (such as mean, median, or most common) or a constant number to fill in any empty data in each column. A ‘mean’ based strategy is used to run the imputer.

Label Encoding

The dataset contains 3 categorical columns. To feed these columns to the model, they must converted to numerical format. For this transformation, we again use the skLearn library to encode target labels with value between 0 and n_classes-1.

Data Scaling

A common requirement for many machine learning estimators is the standardisation of a dataset: if the individual features do not more or less resemble standard normally distributed data, they may behave poorly. Data scaling is applied to transform the features to a normal distribution. Sklearn’s StandardScaler is used for this stage which works by eliminating the mean and scaling the features to unit variance.

Chapter 3.2: Statistical Regressor

Regression is a method for determining how independent features or variables relate to a dependent feature or result. It is a technique for machine learning predictive modeling, where an algorithm is used to forecast continuous and real outcomes.
Regression is a typical application for supervised machine learning models in addition to classification. Since it is a crucial component of predictive modelling, it can be found in a wide range of machine learning applications. Regression analysis may provide organizations with crucial information for decision-making, whether it's used to power

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

financial forecasts or forecast healthcare trends. It is already utilised in various industries to map pay changes and forecast home prices as well as stock and share prices.

For our task, the target variable is also a continuous value representing a real number. Thus it is justifiable to apply statistical regression to this task.
Machine learning regression is accomplished using a variety of well-known techniques. The various methods could use various numbers of independent variables or handle data in distinct ways. Different kinds of machine learning regression models may also assume a different relationship between the independent and dependent variables. For instance, because linear regression approaches presuppose a linear relationship, they are ineffective for datasets with nonlinear interactions.

We will use in total 6 Regression model, starting from a simple Linear model and gradually building upto models capable of capturing more complex relationships.

Chapter 3.2.1 : Linear Regression

Linear Regression is the simplest type of Statistical Regression models. It represents a a linear equation that combines a particular set of input values (x) and yields the expected output for that set of input values serves as the representation (y). As a result, the output value (y) and the input values (x) are both numbers. Each column or input value is given one scale factor, known as a coefficient, which is denoted by the capital Greek letter Beta (B). The model fits the best line to predict the value of y for a given value of x during training. The bias, a further degree of freedom for the line, is also added as a further coefficient. By locating the best coefficients and bias values, the model produces the best regression fit line. Therefore, when we ultimately use our model to make a prediction, it will forecast the value of y based on the value of the input x.
With the least amount of error between the predicted value and the true value, the model seeks to predict the value of y. In order to achieve the optimal value that minimises the error between the predicted y value (pred) and true y value, it has to update the coefficients and bias values. The model employs gradient descent to update coefficients and bias settings in order to minimise the Cost function (i.e: minimising RMSE value) and obtain the best fit line. Starting with random coefficients and bias values, the goal is to iteratively update the values until the minimal cost is reached.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

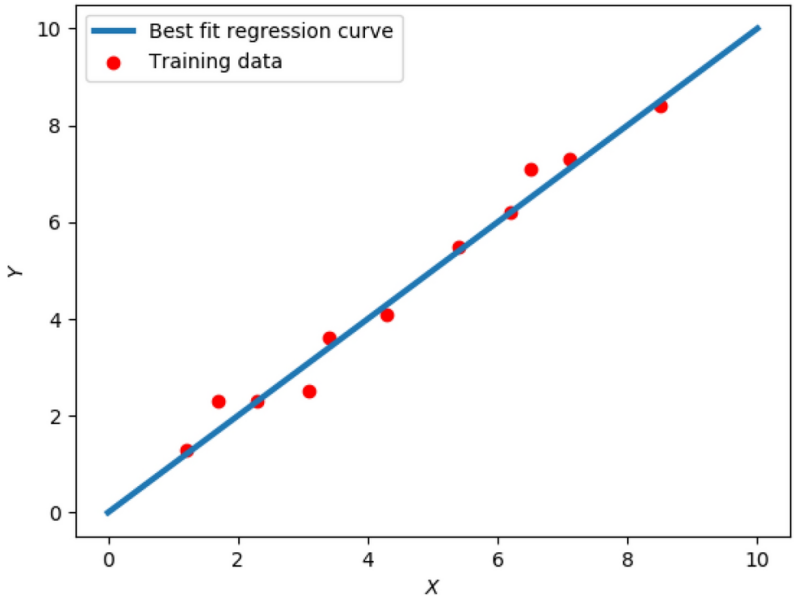


Figure: Linear Regression (source)

Chapter 3.2.2 :Decision Tree Regression

Decision trees are a sort of supervised machine learning technique that uses true or false responses to questions to regress the data. When the structure is visualized, it looks like a tree with distinct sorts of nodes at the root, internal, and leaf levels. The model attempts to mathematically fit a sine curve using additional noisy observations and thus, it learns local linear regressions that approximate the sine curve. Start by defining a feature that will serve as the decision tree's root node. Typically, no one feature can predict the final classes with absolute accuracy; this is known as impurity. This impurity is measured using techniques like Gini, entropy, and information gain, which show how well a feature categorizes the supplied data. For example, Information gain is simply the difference between the impurity of the parent node and the sum of the child node impurities — the lower the impurity of the child nodes, the larger the information gain. An objective function is constructed that the algorithm wishes to optimize using the tree learning technique to divide the nodes at the most informative attributes. The goal is to maximize information gain at each split.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Figure : Decision Tree as learning to fit a sine curve (source)

Chapter 3.2.3 :Random Forest Regressor

A random forest is a meta estimator that employs averaging to increase predicted accuracy and reduce overfitting after fitting numerous classification decision trees to different dataset subsamples. It is a supervised learning algorithm that does regression using ensemble learning. Without any interaction, the trees proceed parallel to one another. During training, a Random Forest builds many decision trees and outputs the mean of the classes as the prediction of all the trees. However, a significant disadvantage of this model is that it has no good interpretability of the outputs.

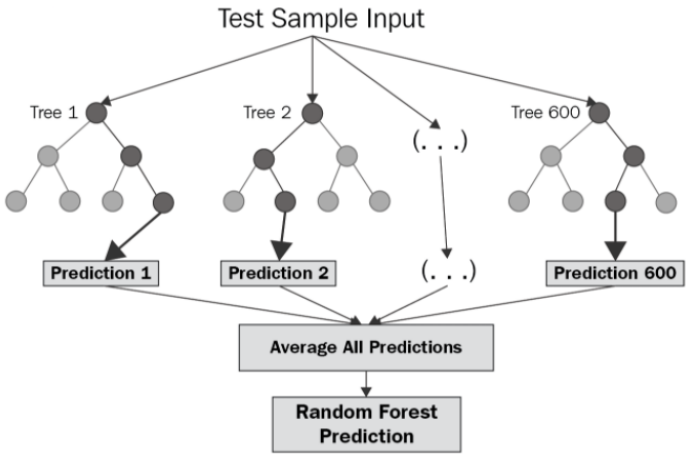


Figure: Random Forest Regressor (source)

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 3.2.4 : Gradient Boosting

Gradient Boosting is another ne of the ensemble technique variations that uses numerous weak models combined for greater overall performance. It computes the difference between the current forecast and the target value's known correct value (referred to as the "residual"). After that, a weak model that maps features to that residual is trained using gradient boosting regression. The goal is to use gradient descent to add weak learners and reduce this loss function. This algorithm moves the model closer to the desired outcome by adding the residual predicted by a weak model to the input of the current model. This process can be repeated numerous times to enhance the overall model forecast.

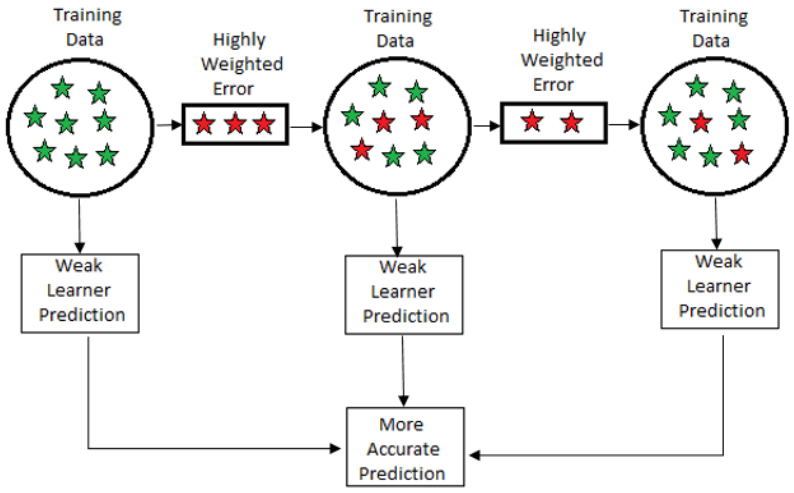


Figure: Gradient Boosting Algorithm (source)

Chapter 3.2.5 : Light Gradient Boosting

The Light Gradient Boosted Machine, or LightGBM for short, is an open-source gradient boosting system that aims to be as effective as existing implementations, if not more so. Light GBM is based on Decision tree methods, just as another gradient boosting method. Although other boosting algorithms divide the tree level- or depth-wise rather than leaf-wise, it divides the tree leaf-wise with the best fit. Light GBM chooses the leaf that creates the least mistake and the most efficiency when growing on the same leaf. As a result, the leaf-wise approach may minimise loss more than the level-wise technique and produces substantially greater accuracy when compared to the majority of other boosting algorithms now in use. Additionally, it model runs remarkably quickly, hence the name "Light". Light GBM can readily overfit a little quantity of data and is

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

susceptible to overfitting. If the data set contains multiple rows and has a larger sample size than 10,000, this light gradient boosting machine performs well.

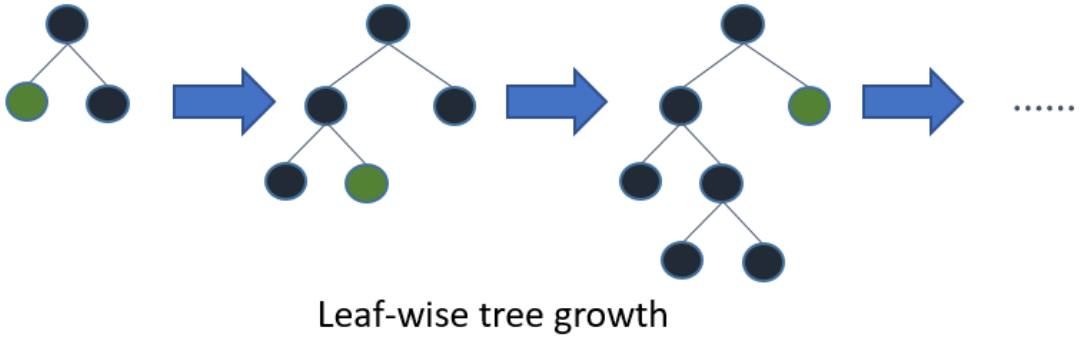


Figure: Light Gradient Boosting Algorithm (source)

Chapter 3.2.6 : Extreme Gradient Boosting (XGBM)

Although XGBoost is one of the Gradient Boosting implementations, what sets it apart from the others is the way it controls over-fitting by using a more regularised model formalisation. This allows XGBoost to perform better. The optimum node splits are determined by the Similarity Score and Gain, unlike standard Gradient Boosting, which employs a different method of tree construction. When working with large datasets, XGBoost has its own method for creating and pruning trees in addition to a number of built-in optimizations. In order to generate trees across all CPUs, XGBoost employs parallel computation during training. The traditional stopping criteria are replaced with the "max depth" option, and tree pruning is started from the backward direction. This significantly improves XGBoost's computational speed and efficiency in comparison to other GBM frameworks. It may then automatically choose the best missing value depending on training loss, which improves its ability to deal with different input data sparsity patterns.

Chapter 3.3 : Artificial Neural Networks (ANN)

Artificial Neural networks, or ANN are frequently used for predictions and pattern recognition due to the benefits of utilizing them, such as their capacity to learn complex behaviors. The input layer, hidden layer, and output layer are the three basic layers that make up the structure of an ANN model. In order to minimize the sum squared error (SSE), the original output and the desired output were compared, and the synaptic weight of each link connecting the neurons was adjusted until the difference was small.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

This provided some regularization for the mode. The weight is a graphical depiction of the neuron's importance.
The an ANN structure used for this study is a Multilayer Perceptron Model (MLP) with an error back propagation learning technique.

Chapter 3.3.1: Multi Layer Perceptron

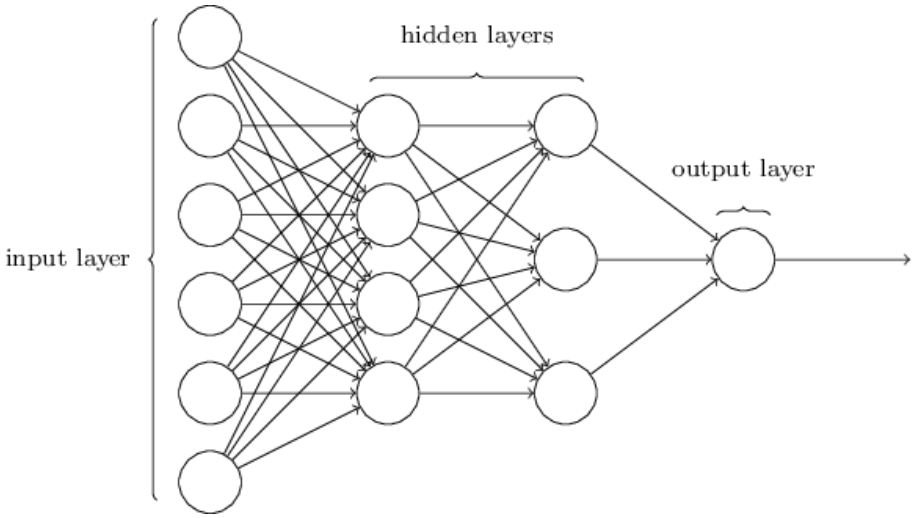


Figure: Architecture of a Multilayer Perceptron (source)

One of the most popular neural network models in the field of deep learning is the multi-layered perceptron (MLP). An MLP, often known as a "vanilla" neural network, is less sophisticated than modern complex models. Similar to the human brain, a multi-layered perceptron is made up of linked neurons that communicate with one another. There are three main layers that make up the network:

1. The input layer, which receives an input and uses it to create an output.
2. Hidden Layers, which use the incoming data's computations and operations to generate meaningful results.
3. The output layer, where the neurons produce information that is useful.

At first, the preprocessed input data is fed to the input layer for forward propagation. In the network, each hidden layer builds on the previous layer's output to create a particular representation of the input. The adaptable coefficient w_{ij}, also known as synaptic weight, multiplies each input x_i to a neuron j to reflect the degree of connection between neurons. Then, a non-linear function (referred to as the activation function) is used; typically, the function has a sigmoid shape (sigmoid or hyperbolic tangent function).. The network can provide a reliable representation of the input at large dimensions with this level of abstraction. The generated output is then compared with

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

the actual target outcomes using a loss function or error function. To adjust the MLP's coefficients (synaptic weights), the objective is to minimize the value of the loss function. For that reason, the error is propagated back through the neurons of the hidden layers again. The model uses the error to update itself by determining the error's derivative with respect to each neuron's weight. To learn the optimum weights, the process described above is repeated for a number of epochs.

Linear Layer

To build the Neural Models, different types of layers are used which performs distinct functions in manipulating and processing the input passed to them. In a Linear Layer, all the input units are connected to the following layers neurons. These layers use a weight matrix and matrix multiplication to linearly transform the input features into the output features, while accounting for a bias coefficient . The input characteristics are sent to a linear layer as a one-dimensional tensor that has been flattened, and they are then multiplied by the weight matrix.

ReLU Activation Unit

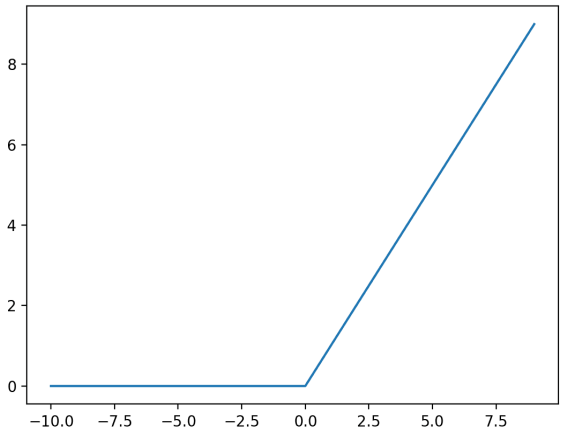


Figure: Input vs. the Output of a ReLU Activation Function (source)

In a neural network, the activation function specifies how the output of a node or nodes in a layer of the network is created from the weighted sum of the input. They are helpful because they give neural networks non-linearities, which let the neural networks learn effective operations. The neurons in a network are "fired" by a variety of activation functions or units. I utilized a piecewise linear activation function called the Rectified Linear Unit (ReLU) in my MLP model, which outputs zero if its input is negative and the input directly in all other cases.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 3.3.2: LSTM

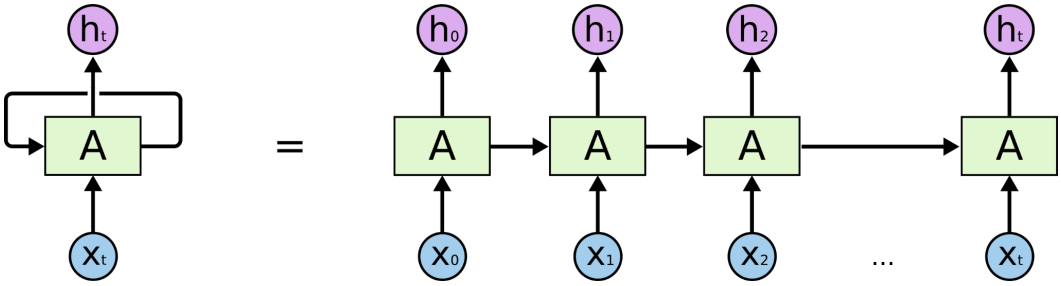


Figure: RNN Network

To understand LSTM, We need to know about RNN or recurrent neural networks. RNNs use multiple copies of the same Neural Network where each of the copies pass information from one cell to another. This chain structure allows RNNs to work on sequential data to learn their characteristics and use those patterns to predict the next likely scenario. Each copy of the RNN cell is a simple Neural network as described above.

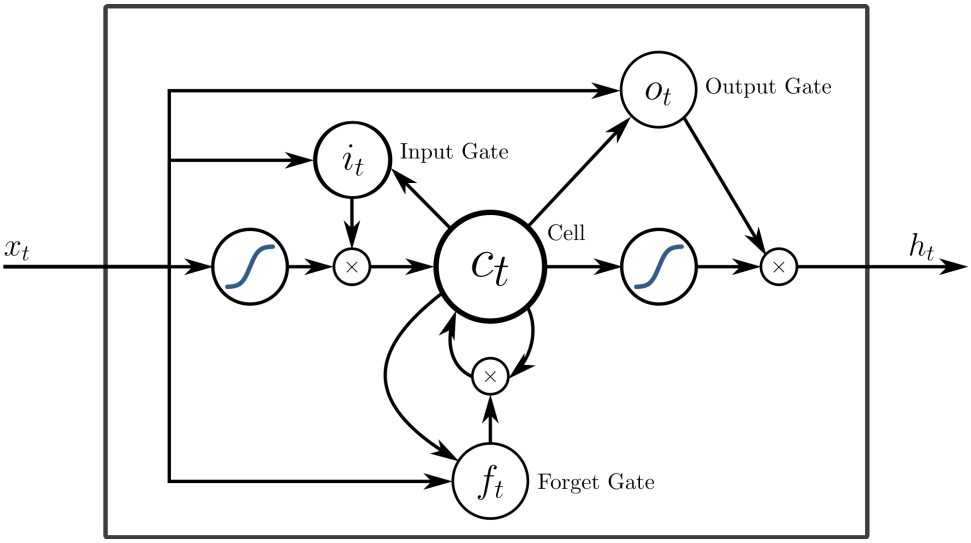


Figure: One cell of a LSTM Network (source)

LSTM stands for Long Short-Term Memory. It is a variant of RNN with an added capacity for learning long-term dependencies. RNNs, when looped over a long time, tend to experience vanishing gradients where important features get ‘lost’ after a certain level. LSTM uses a memory cell on top of the usual RNN cell, which ‘remembers’ this information of cell state over a long time. In the diagram above, the cell state is the horizontal line in the middle. Information in one can be regulated by the gates mentioned in the diagram. These gates allow the information to flow into and out of the cells. The gates contain pointwise multiplication operations and a sigmoid neural

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

network layer to assist this novel mechanism. While one RNN cell changes the input data entirely, LSTM can do this operation selectively using this gating mechanism. Below are a short description of the gates that are present in LSTM:

- Forget Gate: Forget gates applies a sigmoid function over the given input, hidden state and associated weight matrix to decide whether to keep the given information or not. The formula for the forget gate is as shown below:

$$F_t = \sigma(x_t * U_f + H_{t-1} * W_f)$$

  Where, X_t is the input, U_f is the weight associated with the input, H_{t-1} is the hidden state of the previous timestamp and W_f is the weight matrix associated with the hidden state.
- Input Gate: Input gates are quite similar to forget gates in structure except they quantify the importance of the information carried by the input.
- Output Gate: Output gates are also similar to forget gates. It is used to calculate the current hidden state using the formula below:

$$H_t = O_t * tanh(C_t)$$

  Where, H_t is the new hidden state and O_t is the value of output gate. C_t the current memory cell output.

  To calculate the output in the final state, Softmax activation is applied to the hidden state H_t

Optimizer

Optimizers are programmes or techniques that modify the neural network's properties, such as its weights and learning rate, in order to minimise losses. They are used to either maximise production efficiency or minimise an error function (loss function). These mathematical operations operate on the learnable parameters (Weights & Biases) of a model. Over the past few years, numerous optimizers have been studied, each with their own benefits and drawbacks. I used the ADAM optimizers, which compute adaptive learning rates for each parameter, in my project. Both the decaying average of the previous gradients—which is analogous to momentum—and the decaying average of the previous squared gradients are stored. It assists ADAM to converge quickly and makes it incredibly fast. Additionally, it corrects the issues with large variance and vanishing learning rate.

Dropout

The Dropout layer acts as a mask, eliminating some neurons' contributions to the subsequent layer while maintaining the functionality of all other neurons. It can be used on neurons that aren't visible in the main body of your network model. The dropout layer makes sure that the neurons can't depend on one input because it might be randomly dropped out. This lessens bias brought on by overly depending on a single input, which

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

is a primary contributor to overfitting. As a result, neurons won't pick up on unnecessary input details. A too-high dropout rate might impede the model's pace of convergence and frequently degrade final performance, while a too low rate produces little to no improvements in generalisation performance. Dropout rates should ideally be adjusted separately for each layer as well as across multiple training phases.

Chapter 3.4: Evaluation Metrics

Different mathematical functions are utilized to evaluate the performance of the Machine Learning models. For my experiment, I used R^2 score to measure the performance of the Statistical Regressor. L1 loss is used for the MLP model , as it is less sensitive to outliers and regularizes well. Finally, the LSTM regressor uses Mean Squared Error for it's loss calculation.

R^2 score

R^2, otherwise known as coefficient of determination, is the degree to which the variance of the target variable can be explained by the feature variables. The equation to calculate this value is as below:

$$R^2 = 1 - \frac{(\sum_{i=1}^{n} e_i^2)}{(\sum_{i=1}^{n} y_i - \bar{y})^2}$$

where, the dividend is the Mean Squared error of ground truth with the predicted value and the divider is the mean squared error between ground truth and mean of the target values.

The value of this ranges from -1 to 1. The negative value can exist because the regression can sometimes do worse than the sample mean in terms of tracking the dependent variable. While a 0 value of R squared is possible when the predicted value is always a constant value regardless of the input value.

L1 Loss

L1 Loss, otherwise known as Absolute Error Loss, is the absolute difference between the ground truth and the predicted value, calculated individually for each sample.

$$L1 = \left| y_{actual} - y_{target} \right|$$

The aggregation of all the loss values is called the cost function which, for L1 loss in known as Mean Absolute Error.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 4 : Model Building

Chapter 4.1 : Dataset

As described in the previous chapter, we process our dataset in several stages to make it appropriate for feeding to the different ML regression models. The preprocessing stages include using a mean-based imputer to fill up the missing or NaN values in various columns, encoding the categorical columns to numerical values, and scaling them to transform the input to a normalized distribution. The data processing is handled using the pandas and sklearn library.

The preprocessed data is chunked into three distinct parts. The training dataset represent the majority portion of the data used for fitting the model. Testing dataset is sample data which is used to evaluate the model. The data splitting can be done using the ‘train _fest_split' function which is supported by the sklearn package.
At first, a 75 : 25 ratio is used to generate the training and test data. We then further split the training data in an 80:20 ratio, to create a validation set as well. A dataset split is necessary for a fair assessment of prediction accuracy.

The loss function on the training data will continue to show decreasing values if our models hyper-specify to the training set, but the loss function on the held-out validation set will eventually rise. According to this, the models aren't learning effectively because they are essentially memorizing the training set. As a result, our models won't function properly on brand-new datapoints that they have never seen before.

Additionally, to prepare the dataset for the ANN model, we need to convert them to a specific TensorDataset format which is able to load and process each sample of the dataset lazily. It is also separately loaded by a DataLoader that takes care of shuffling/sampling/weighted sampling, batching, using multiprocessing to load the data, using pinned memory etc.

Chapter 4.2: Technology

To build our models and run our experiments, we have made use of the following frameworks, libraries and other tools.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Anaconda Navigator

Anaconda Navigator is Python's most frequently used graphical user interface (GUI). A desktop application is Anaconda Navigator. Programs may be accessed, and Anaconda Conda packages can be readily mitigated using the Anaconda Navigator distribution. Because of the environment provided by Anaconda Navigator, users may program without utilising command-line tools. Users can use a local Anaconda Repository or the Anaconda Cloud to install new packages found by Anaconda Navigator. Numerous operating systems, including Windows, Mac OS, and Linux, are compatible with the Anaconda Navigator.

Python

Popular open source programming language Python. Guido van Rossum created it, and it was delivered in 1991. It is used for system scripting, programme development, and the design of websites and mathematical applications. Because it uses an interpreter system, it implies that code can be executed as soon as it is created. This implies that prototyping can be quite energy-intensive. Python was created for simple coding and bears some similarities to the English language with influences

Numpy

Python's NumPy library is used to manipulate arrays. Additionally, it provides functions for working with matrices, the Fourier transform, and the area of linear algebra. Additionally known as Numerical Python.

MatplotLib

Data becomes relatively simple for people to understand because of data visualization techniques. An all-purpose data visualization library for Python is called Matplotlib. The widely used package for graph charting is called Matplotlib. I'm free to use it because it is open source. For platform portability, it is primarily written in Python, with a few pieces also written in C, Objective-C, and Javascript.

Scipy

This scientific computation package called SciPy is built on top of NumPy. It offers helpful functions for signal processing, statistics, and optimization. SciPy is open source, similar to NumPy.

Pandas

It is a straightforward data editing library that Python supports. It is built using the NumPy library, and the Data frame is its main data structure. Programmers can store and manipulate tabular data in rows of observations and columns of variables using data frames. It is a heterogeneous data structure that is offered by Pandas and is quite

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

adaptable. Pandas are one of the important open source libraries used by developers to build robust software applications. The Berkeley Software Distribution (BSD) licence governs its distribution.

Scikit-learn
This is a popular open-source machine learning package that Python supports. This library supports most of the prominent machine learning algorithms. It is compatible with Python's NumPy and SciPy scientific and numerical libraries and provides implementations of a lot of preprocessing and postprocessing methods.

Seaborn
Another Python library used for viewing the graphs is called Seaborn. These graphs convey information more effectively than conventional Matplotlib plots and are more visually appealing than Matplotlib graphs.

Tensorflow
Tensorflow is one of the earliest all-in-one python frameworks for deep neural networks. It uses a multidimensional array called Tensor as input and provides a system to write verbose dataflow graphs to specify the operations from input to output. This library is created and maintained by Google and comes with out-of-the-box support for GPU modeling.

Chapter 4.3: Software and Hardware

The software and hardware requirements for completing the investigations aimed by this thesis are listed below. Python has inbuilt libraries for implementing various ML algorithms rapidly. Therefore, it is justified to use python programming language for implementing the proposed ML model.

Software Requirements
The software versions used for developing this project are listed below:

conda version : 4.10.0
notebook : 6.0.3
python 3.7

Python Libraries:

matplotlib : 3.2.2
numpy : 1.18.5
pandas : 1.0.5

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

scikit-learn : 0.23.2
seaborn : 0.11.0
tensorflow : 2.1.0
tensorflow-gpu : 2.1.0
tensorboard : 2.2.1

Hardware Requirements
Minimum hardware requirements to satisfied for the successful development of this project are given below

1. Processor: Intel i3 Processor or its equivalent.
2. Hard Disk: 5 GB free space.
3. Memory: 8GB RAM.

Chapter 4.4 : Model Training (Statistical)

For the statistical models, the skLearn wrapper functions are used for we use a 10-fold cross validation to calculate the average r2 score for each of the models on the training split. The standard deviation score for each model is also reported.

Chapter 4.2.1 : Linear Regression Model

The first model we run is linear regression. The goal of this model is to minimize the residual sum of squares between the targets observed in the dataset and the targets anticipated by the linear approximation. The fitted linear model has coefficients w = (w1,..., wp). The obtained scores are :

Mean score: 0.19460321467434208
Standard Deviation: 0.014692150307216783
R2 scores: [0.21696847 0.17751011 0.19161379 0.16889957 0.20558141 0.19084525 0.18470733 0.21587233 0.19699711 0.19703676]

Chapter 4.2.2 : Decision Tree Regression Model

To implement this tree we use a “squared error” criterion to minimize the L2 loss using the mean of each terminal node. Log of the feature number is used to decide the number of features to look at while searching for the best split. The tree is also pruned, by using a max depth of 100. The obtained score are:

Mean score: 0.04486334775969299

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Standard Deviation: 0.1145643200586844
R2 scores: [-0.17312985 -0.10475699 0.12402266 0.01790495 0.07361837
0.22205683 0.15184263 0.05688908 0.11251705 -0.03233126]

Chapter 4.2.3: Random Forest Regression Model

Same as decision tree, for this model we also use a “squared error” criterion and logarithm based max sample size for pruning. 200 trees are bootstrapped into building the forest. The obtained score are:
Mean score: 0.4822273769268537
Standard Deviation: 0.034769054790786444
R2 scores: [0.50442544 0.43090987 0.48074661 0.42911349 0.53253273 0.52093959
0.51857077 0.46245681 0.46237054 0.48020792]

Chapter 4.2.4 : Gradient Boosting Regression Model

Opposed to the previous tree based models, Gradient boosting uses “squared error” as a loss function, with 0.1 learning rate. The criterion used in this model is called “Friedman MSE” which measures the quality of a split. This model is also pruned by a logarithmic scale to the sample size. Reported Score:

Mean score: 0.3970691302521846
Standard Deviation: 0.022529911310384256
R2 scores: [0.42924903 0.35881849 0.40656085 0.36955681 0.404104   0.42890806
0.40051161 0.40859264 0.38851778 0.37587204]

Chapter 4.2.5 : XGB Regression Model

This model uses the same parameters used for Gradient Boosting Regression Model. The score obtained here are:

Mean score: 0.3985000503334878
Standard Deviation: 0.02355027850656382
R2 scores: [0.42094017 0.36113329 0.40480052 0.36458533 0.41559376 0.43572382
0.40495909 0.41207096 0.38743912 0.37775444]

Chapter 4.6: Light GBM Regression Model

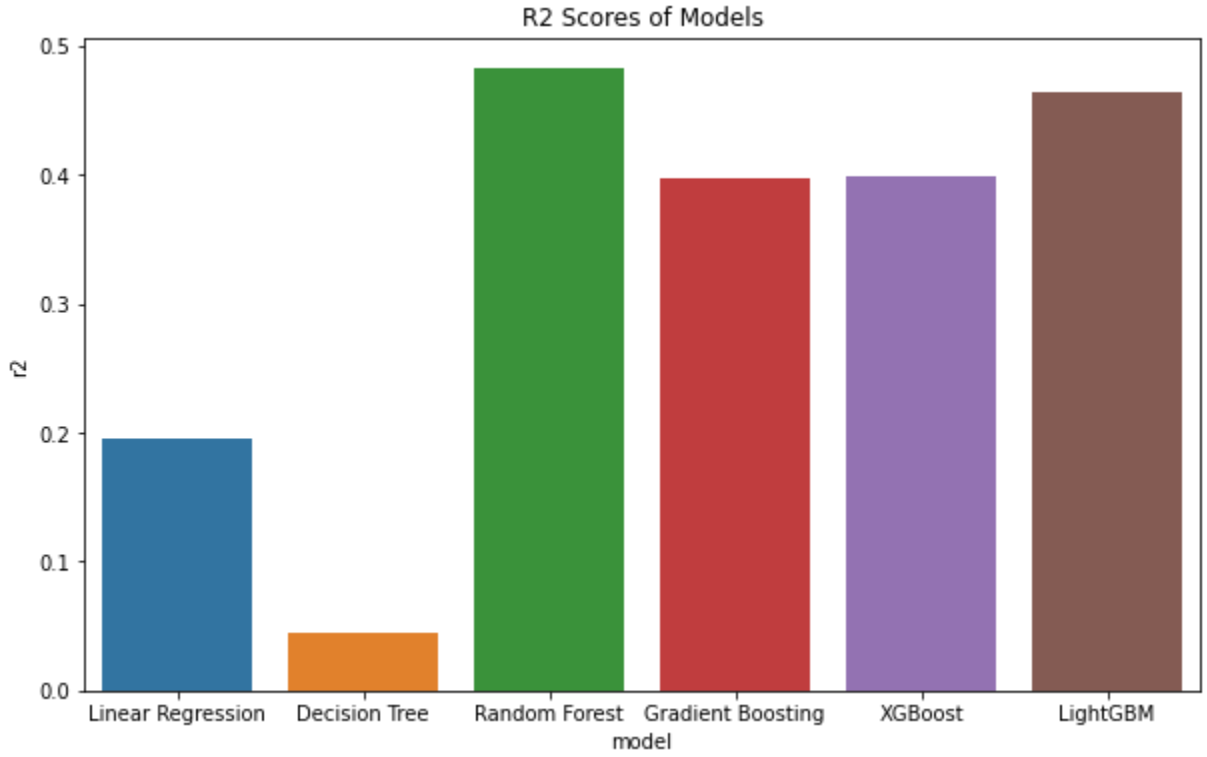
This model uses the same parameters used for Gradient Boosting Regression Model. The score obtained here are:
Mean score: 0.4635057569487426
Standard Deviation: 0.034795285868524616

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

R2 scores: [0.49658187 0.42129574 0.46617713 0.38966431 0.49021555 0.50284245 0.49236057 0.47760005 0.45499376 0.44332613]

Chapter 4.7: Model Selection

To visualize the performance of the models described above, we plot the average of their 10-fold validation r2 score as a barchart, given below :



From the above plot, we can see that lightBGM and Random Forest perform almost the same for this regression task.

So to bring further distinction between the two models, we fit our eval dataset to the Random Forest and LightBGM models and identify the most salient features from the large collection of columns. The top features with their importance score for each of the models are given below. These feature importance scores are computed as the mean and standard deviation of accumulation of the impurity decrease within each tree.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Feature	Importance
Floor_Area	0.232296
Facility_Type	0.214702
Energy_Star_Rating	0.199110
Year_Built	0.147042
Elevation	0.032150
Bulding_Class	0.016665
Days_with_Fog	0.014225
State_Factor	0.012875
Precipitation_Inches	0.005344
Max_Wind_Speed	0.004724

Figure: Top 10 important features of Random Forest Regressor

Feature	Importance
Facilty_Type	746
Floor_Area	572
Year_Built	557
Energy_Star_Rating	343
Elevation	98
Building_Class	81
State_Factor	72
Days_With_Fog	29
Year_Factor	28
November_Max_Temp	25

Figure: Top 10 important features of Light GBM Regressor

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

We then retrain the models using only these salient features for each, and compute r2 score on the test dataset. On the test set, Random Forest Regressor achieves an r2 score of 0.778 , while Light BGM scores much lower - approximately 0.52. Clearly, random forest can be used as our best model here.

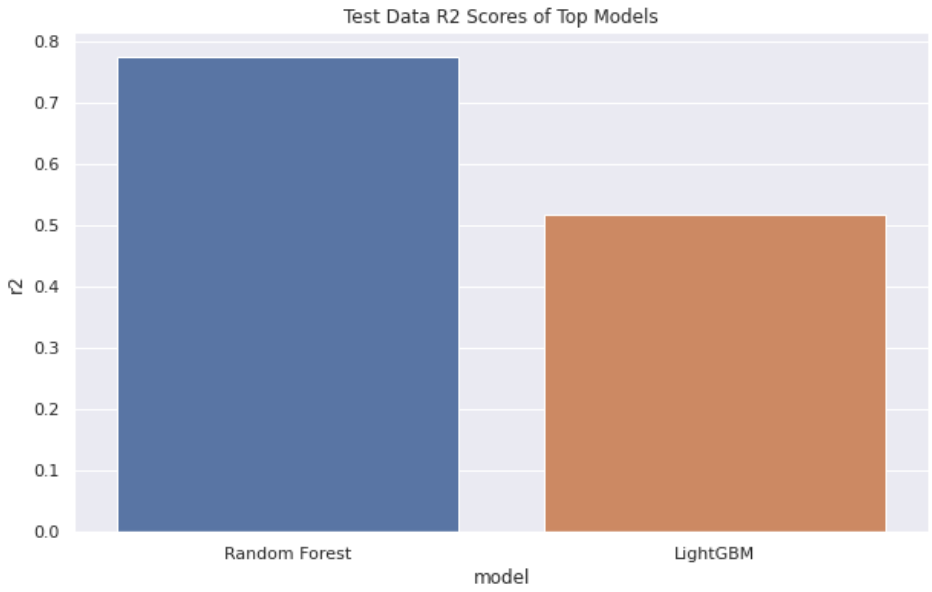


Figure : Test Data R2 Scores of Top Models

Chapter 4.5: Parameter Tuning

Hyperparameter values are predetermined before the training job is run, and they remain constant throughout the training. Exploring every possibility is impractical when creating sophisticated machine learning systems like deep learning neural networks. By automatically attempting many different model iterations, hyperparameter adjustment can increase training productivity. This is crucial for managing a machine learning model's behavior.

To choose the value that results in the best score, one strategy is to experiment with several values. This method is referred to as a grid search. In order to create a grid of values, we would examine all possible combinations of the sets of values if we had to choose values for two or more parameters. In our experiment, we loop over specified hyperparameters and fit our model to the training set with the aid of SkLearn's GridSearchCV function. So, from the list of hyperparameters, we may choose the best parameters in the end. Hyperparameters are passed to the GridSearchCV method with predetermined values. This is accomplished by constructing a dictionary in which a certain hyperparameter is mentioned along with the possible values. All possible combinations of the dictionary's values are tested by GridSearchCV, which then use the

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Cross-Validation method to evaluate the model for each combination. As a result, after employing this function, we can determine the accuracy and loss for each set of hyperparameters and select the combination that offers the best performance.
For our best model , i.e: the Random Forest Regression Model, we use the following hyperparameter dictionary :

```
param_grid = {
   'bootstrap': [True],
   'max_depth': [80, 90, 100, 110],
   'max_features': [2, 3],
   'min_samples_leaf': [3, 4, 5],
   'min_samples_split': [8, 10, 12],
   'n_estimators': [100, 200, 300, 1000]
}
```

Running the GridSearchCV process takes a significantly long time on all these parameters. After four hours of tuning, we get the following best parameters for the model

```
{
        max_depth=80,
        max_features=3,
        min_samples_leaf=3,
        min_samples_split=8,
        n_estimators=1000
}
```

However, in our experiment, the best parameter model ended up performing worse than the random RFR model and scored below .50 (i.e: worse than Light GBM). So we continue using our default model for the task.

Chapter 4.6 : Model Training (Neural Network)

For ANN training, I have used pytorch to build a simple Multi-layer Perceptron architecture. This is built using a combination of Linear transformation and ReLU activation units. An Adam optimizer with a learning rate scheduler is used with an L1 loss function. For each epoch, both the training loss and the validation loss are reported.

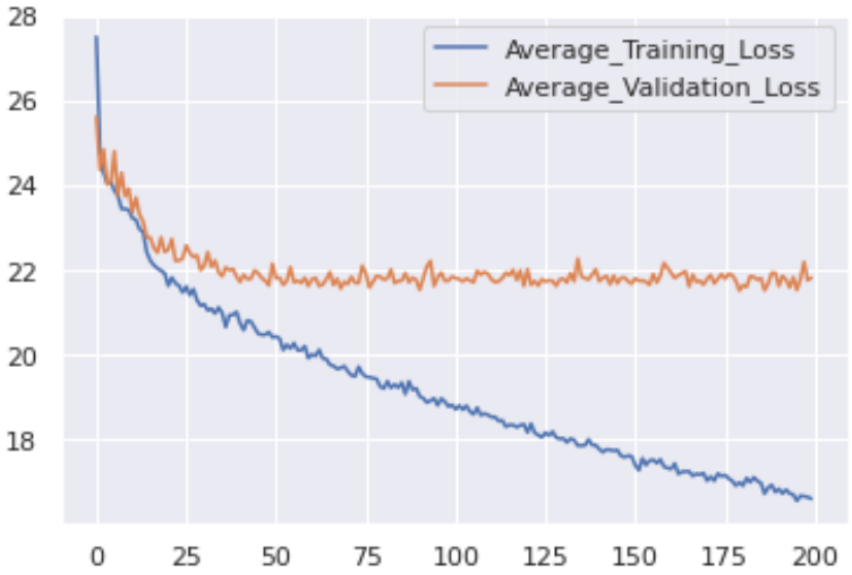
Chapter 4.2.1 : Multilayer Perceptron Model

At this point, I started experimenting with more complicated neural networks. These are generally very robust and have the capability to model very complicated problems. The MLP I have used had the following configuration:

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

- Number of Features = 10
- Hidden Size of the First Layer = 300
- Number of Linear Layers = 6
- Optimizer: AdamW
- Learning Rate: 1e-3
- Loss Function: *L1Loss* (Mean Absolute Error)

Training this Model for 200 Epochs yield the below results:



Average Train Loss: 16.6136
Average Validation Loss: 21.8243
Final Prediction Score: 0.5053

As the graph shows, the validation loss flatlined after 35 epochs. So the model might have overfitted after that point.

Chapter 4.2.2: LSTM Model

The LSTM model built for my experiment has 3 layers of each consisting of 30 units of LSTMcells. Each of these LSTM layers are followed by a dropout layer, which drops 20% of the units randomly at each iteration. A sigmoid activation function is used inside each LSTM cell and the layer kernels use a Glorot Uniform function to initialize. The final output layer of the model is Dense Layer - which is a regular densely-connected NN layer and gives us the final prediction of the model.
We run our LSTM model for 50 epochs, with a batch size of 30. The ADAM optimizer is used to optimize the parameters. However, the model shows a large loss value consistently, performs very poorly on the test set with a R2 score around .42 .

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Chapter 5: Results

As discussed in the previous Chapters, to predict the *Site_eui* value, I have trained several statistical regression models which tend to have a simpler architecture, and two neural network models, with a much more complicated architecture.
The training performance of the Statistical models allowed me to pick the top two best performing models. The table below shows the training performances of all the statistical models that I have trained so far:

Model Name	R^2 Score (Training)
Linear Regression	0.1946
Decision Tree	0.0449
Random Forest	**0.4822**
Gradient Boosting	0.3970
XGB Regression	0.3985
Light GBM Regression	0.4635

As seen from the table, Random Forest Regressor and Light GBM Regressor has a very similar training R2 score. Thus, we further evaluate these two models by identifying their most salient features, retraining them on those features only and comparing their performance on the test. It was observed that feature selection gave the Random Forest Regressor a significant boost, where it scored .7764 - as opposed to Light BGM's approximately .51 R2 Score. Thus, Random Forest was chosen as the best Statistical Regressor for our prediction task.

Training the Neural Models required much more Hardware resource and time, however the performance was barely on par with the Statistical Regressor. Below I present the Test data performance scores of the Best Performing Statistical Regressor, a simple MLP and a complex LSTM model.

Model Name	R^2 Score (Test)
Random Forest Regression	**0.776**
Multi Layer Perceptron	0.505
LSTM	0.411

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

It is clear that in both type of models, the Random Forest Regressor generalizes the dataset most efficiently - while requiring comparatively lower resources than the NNs.

Chapter 6: Discussion

Findings

In this research, I take an in-depth look into Site EUI and what goes into predicting it. The data analysis that I performed showed that we have more data for residential areas than commercial areas in the training dataset. However, from the other plots, I show that the site_eui value follows a similar trend regarding the state factor value, except for state_6. Moreover, the Energy Star rating has a reverse correlation with the site_eui, where a higher number of energy star rating usually means that the building will have a low site_eui. Later, I run a random forest regressor and a light GBM Regressor to find the essential features. For the former one, floor_area, facility_type, energy_star_rating, and year_built are the most salient features, while for Light GBM, these same four are essential, except with a different ranking. I strongly suggest maintaining a balance between floor area and energy star rating for specific facility types for future building plans.
While predicting the site_eui given various building features, I found that Random Forest Regression performed the best among all other approaches. While finding a much simpler model to perform better is quite unexpected, there can be several reasons behind it, such as overfitting, underfitting, and wrong parameter usage. However, Using GridSearchCV to find the best set of parameters failed to yield better results in my explorations.

Hypothesis Status

Beginning this research, I aimed to find out if there are significantly irresponsible consumer infrastructures at the electricity consumer space. From the data analysis in Chapter 2, I showed that grocery_store/food_market, data_centers, and laboratories have higher site_eui. Which means if we want to optimize our consumption, we need to focus more on these infrastructures. I also hypothesized that weather may have a significant effect on how electricity is consumed in each area, but searching for the most salient features in Dataset, I found out that, other building features such as, facility type, energy star rating, and floor area have a much more important effect on the yearly site_eui value of an infrastructure.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Commercial and Economical Context of this research

Predicting site_eui value using machine learning gives an approximate estimate of the future electrical cost of a building after it is built. So it will help us build a more efficient architecture designed for a specific set of variables. Furthermore, This will ensure the efficient use of electricity in the future, and this will also help us with tackling the climate problem we are experiencing now because the efficient and advanced understanding of energy usage allows us to design our grids efficiently while reducing energy usage and plan the production of electricity while incorporating renewable sources.

Limitations

The work has several limitations regarding how the dataset was collected and used. First, most of the data collected for this research were sensor data prone to anomalies because of malfunctions and technical errors. The dataset was also challenging because there was a lot of extra information, and selecting the most salient features became challenging while running the baseline models. Also, as a regression problem, the exact estimate of the site_eui is quite hard to predict, which is why the R2 score is used to find an approximate accuracy of a model. Due to time constraints, I couldn't experiment with more sophisticated machine learning mechanisms such as attention, which could've given us greater insight into how these features relate to or interact.

Management of This Work

Task Name	Task Goal	Final Progress
Literature Review	Explore different papers and aspects of the ideas to check for difficulties	I have conducted research on several relevant works and presented my findings based on them in the introduction section.
Request for Proposal	Develop a proposal consulting with the supervisor and setup targets and priorities for the challenge.	Developed DPP and IPR and consulted with the supervisor for feedback that I later incorporated in this report.
Gather Training Dataset	Review relevant datasets to find the most compatible one that aligns with the problem I will work on	Researched on several contemporary dataset and found the Building Energy Efficiency Dataset - 2022 to be the most comprehensive

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

		and latest one for our current research
Run Exploratory Data Analysis	Generate various plots from the selected dataset	Ran an extensive data analysis and showed my findings in Chapter 2
Identify Constraints and Exploitable Parameters	Identify outliers and constraints from the EDA and try to identify the intrinsic patterns in the dataset	Presented several interesting findings such as, site_eui depending on Energy Star rating and having no dependency on the year_factor. More details in chapter 2.
Experiment with Machine Learning Models	Try different Machine Learning models to predict energy consumption information	Experimented with Basic Statistical models and 2 neural model to predict Site_EUI value which is the energy consumption information of a building
Investigate and Suggest Policies according to Results	Suggest policies based on the findings from EDA and Machine Learning prediction to adopt a more efficient structural decision	Suggested policy based on my finding at the finding subsection of discussion section

Future Works

The current dataset that I am working on, contains data from a very limited number of geographical locations. If we can collect similar data from a larger geographic area, it may give us a more robust model for a wide range of consumers. Applying PCA and similar dimensionality reduction techniques can also improve the results of our models. Moreover, experimenting with more complicated and better interpretable models can also be explored to understand their potential.

Chapter 7: Conclusion

Efficient production and usage of electricity to reduce carbon-footprint globally has been a central focus in the modern research domains. In this research, I explore and show

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

the important building related feature variables that may affect the energy usages of a building in a passive manner. I also show that it is possible to predict the energy usage of a building given its features and facility types using simple statistical models. However, leveraging on more complicated models was not very successful for this specific task. If we can leverage the predictions of our best performing models for design consideration of a building, we can meaningfully reduce the long term electricity costs of a building while helping to reduce GHG emissions for a better future.

Future Plans

The project has large potentials to be expanded into performing more intricate tasks. In the coming days, I would like to gear the project in the following direction:

- **Experiment on Complex Models:** Although I have used many standard statistical and Neural Models in this project, more effort can be invested to build dedicated deep learning architectures to better capture the factors. It is also important to make these models explainable and as transparent as possible, to allow control.
- **Optimizing Infrastructures:** Designing Data Driven Optimized Infrastructure by observing the effect of different factors from extensive data analysis to assist both the consumers and providers in reducing carbon footprints and energy consumptions.
- **Assisting Legislations:** Assisting emission reduction policies with highly informative and detailed data analysis and probable consequences of plausible legislative measures. This can also assist the legislative bodies to garner enough public support to enforce the more seemingly restrictive laws for industries which may end up being helpful in the long run.

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

References

Aboelata, A. (2019, January 17). *Reducing outdoor air temperature, improving thermal comfort, and saving buildings' cooling energy demand in arid cities – Cool paving utilization*. Science Direct. Retrieved November 15, 2022, from https://www.sciencedirect.com/science/article/abs/pii/S2210670721000561

Andrews, C. J., & Krogmann, U. (2019, January 17). *Technology diffusion and energy intensity in US commercial buildings*. Science Direct. Retrieved November 15, 2022, from https://www.sciencedirect.com/science/article/abs/pii/S0301421508005636

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Department for Business, Energy and Industrial Strategy. (2021, July 29). *UK Energy in Brief 2021*. GOV.UK. Retrieved November 15, 2022, from https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1032260/UK_Energy_in_Brief_2021.pdf

eia. (2021, November 4). How much carbon dioxide is produced per kilowatthour of U.S. electricity generation? Retrieved November 15, 2022, from https://www.eia.gov/tools/faqs/faq.php?id=74&t=11

Eia.gov. (2018). *International Energy Outlook 2018.* EIA. Retrieved November 15, 2022, from https://www.eia.gov/outlooks/ieo/

IPCC. (2019). *AR4 Climate Change 2007: Synthesis Report — IPCC*. IPCC. Retrieved November 15, 2022, from https://www.ipcc.ch/report/ar4/syr/

Kharseh, M., & Al-Khawaja, M. (2019, January 17). *Retrofitting measures for reducing buildings cooling requirements in cooling-dominated environment: Residential house*. Science Direct. Retrieved November 15, 2022, from https://www.sciencedirect.com/science/article/abs/pii/S1359431115014441

Kulp, S. A. (2019, January 17). *New elevation data triple estimates of global vulnerability to sea-level rise and coastal flooding*. Nature Communications. Retrieved November 15, 2022, from https://www.nature.com/articles/s41467-019-12808-z

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

MIMURA, N. (n.d.). *Sea-level rise caused by climate change and its implications for society*. NCBI. Retrieved November 15, 2022, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3758961/

Sharma, S. (2022, January 7). *Retrofitting Existing Buildings to Improve Energy Performance*. MDPI. Retrieved November 15, 2022, from https://mdpi-res.com/d_attachment/sustainability/sustainability-14-00666/article_deploy/sustainability-14-00666-v2.pdf?version=1641811702

Tollefson, J. (2019, January 17). *IPCC climate report: Earth is warmer than it's been in 125,000 years*. IPCC climate report: Earth is warmer than it's been in 125,000 years. Nature,. Retrieved November 15, 2022, from https://doi.org/10.1038/d41586-021-02179-1

Yang, C., Choi, J.-H., Noble, D., & Schiler, M. (2015). *Method for estimating energy use intensity based on building façade*. Building Research Information Knowledgebase. Retrieved November 15, 2022, from https://www.brikbase.org/sites/default/files/ARCC2015_106_yang.pdf

Zafirah, M. F., & Mardiana, A. (2014). *Design, Efficiency and Recovered Energy of an Air-to-Air Energy Recovery System for Building Applications in Hot-Humid Climate*. International Journal of Science and Research (IJSR). Retrieved November 15, 2022, from https://www.ijsr.net/archive/v3i9/U0VQMTQ0OTU%3D.pdf

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

Appendix

EDA Notebook

```
# %%
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
sns.set_style("whitegrid")
import os

# %% [markdown]
# Importing Train and Test Set

# %%
train = pd.read_csv(r"data/train.csv")
test = pd.read_csv(r"data/test.csv")

# %% [markdown]
# Printing first 5 Row of the Train Dataset

# %%
train.head()

# %% [markdown]
# Printing first 5 Row of the Test Dataset

# %%
test.head()

# %% [markdown]
# Taking a peak the 62 columns, we can see the dataset contains several
significant information among them.
#
# #### Covarieties:
# - `id`: building id
# - `Year_Factor`: anonymized year in which the weather and energy usage
factors were observed
# - `State_Factor`: anonymized state in which the building is located
# - `building_class`: building classification
# - `facility_type`: building usage type
# - `floor_area`: floor area (in square feet) of the building
# - `year_built`: year in which the building was constructed
# - `energy_star_rating`: the energy star rating of the building
# - `ELEVATION`: elevation of the building location
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# - `january_min_temp`: minimum temperature in January (in Fahrenheit) at the
location of the building
# - `january_avg_temp`: average temperature in January (in Fahrenheit) at the
location of the building
# - `january_max_temp`: maximum temperature in January (in Fahrenheit) at the
location of the building
# - `cooling_degree_days`: cooling degree day for a given day is the number of
degrees where the daily average temperature exceeds 65 degrees Fahrenheit. Each
month is summed to produce an annual total at the location of the building.
# - `heating_degree_days`: heating degree day for a given day is the number of
degrees where the daily average temperature falls under 65 degrees Fahrenheit.
Each month is summed to produce an annual total at the location of the
building.
# - `precipitation_inches`: annual precipitation in inches at the location of
the building
# - `snowfall_inches`: annual snowfall in inches at the location of the
building
# - `snowdepth_inches`: annual snow depth in inches at the location of the
building
# - `avg_temp`: average temperature over a year at the location of the building
# - `days_below_30F`: total number of days below 30 degrees Fahrenheit at the
location of the building
# - `days_below_20F`: total number of days below 20 degrees Fahrenheit at the
location of the building
# - `days_below_10F`: total number of days below 10 degrees Fahrenheit at the
location of the building
# - `days_below_0F`: total number of days below 0 degrees Fahrenheit at the
location of the building
# - `days_above_80F`: total number of days above 80 degrees Fahrenheit at the
location of the building
# - `days_above_90F`: total number of days above 90 degrees Fahrenheit at the
location of the building
# - `days_above_100F`: total number of days above 100 degrees Fahrenheit at the
location of the building
# - `days_above_110F`: total number of days above 110 degrees Fahrenheit at the
location of the building
# - `direction_max_wind_speed`: wind direction for maximum wind speed at the
location of the building. Given in 360-degree
#     compass point directions (e.g. 360 = north, 180 = south, etc.).
# - `direction_peak_wind_speed`: wind direction for peak wind gust speed at the
location of the building. Given in 360-degree compass point directions (e.g.
360 = north, 180 = south, etc.).
# - `max_wind_speed`: maximum wind speed at the location of the building
# - `days_with_fog`: number of days with fog at the location of the building
#
# #### Target
# - `site_eui`: Site Energy Usage Intensity is the amount of heat and
electricity consumed by a building as reflected in utility bills
#
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# %%
train.columns

# %% [markdown]
# In the following cell, we drop the `id` column of the training dataset -
because it won't be relevant for the upcoming exploratory data analysis.

# %%
train.drop(columns=['id'],axis=1,inplace=True)

# %%
train.shape

# %%
train.info()

# %%
train.head()

# %% [markdown]
# The `describe()` method is used to view some basic statistical details like
percentile, mean, std etc. of a data frame or a series of numeric values.

# %%
train.describe()

# %%
test.describe()

# %%
print(f'\033[94mNumber of rows in train data: {train.shape[0]}')
print(f'\033[94mNumber of columns in train data: {train.shape[1]}')
print(f'\033[94mNumber of values in train data: {train.count().sum()}')
print(f'\033[94mNumber missing values in train data:
{sum(train.isna().sum())}')

# %% [markdown]
# ### Missing or Null Data present in Train Dataset
#  - Six columns in both train and test datasets have missing values
#  - Test dataset have more missing values (%) than train dataset
#  - Columns `days_with_fog`, `direction_max_wind_speed`,
`direction_peak_wind_speed`, and `max_wind_speed` have more than 50% missing
values in training dataset.
#
# To put it into perspective:
#  - “days_with_fog”: **60.5%** large
#  - “direction_peak_wind_speed”: **55.2%**, large!
#  - “max_wind_speed”: **54.2%**, large!
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
#  - “direction_max_wind_speed”: **54,2%**, large!
#  - “energy_star_ratng”: **35.3%**, somehow large!
#  - “year_built”: **2.4%**, Acceptable
#
# Missing the datapoints in this dataset meant two things:
# 1. Some of the data couldn't be collected for said data points i.e: specific
`year_built` was not available for some houses or `max_wind_speed` was not
collected for specific area
# 2. Some buildings were situated in a place where such points are not
applicable. Such as, not every house experience `days_with_fog` every month of
every year.

# %%
df = (train.isnull().sum() / len(train)) * 100
df = df.drop(df[df == 0].index).sort_values(ascending=False)
missing_data = pd.DataFrame({'Missing Ratio %' :df})
missing_data.plot(kind = "barh")
plt.xlabel("Percentage")
plt.ylabel("Variable Name")
plt.title("Percentage of Missing data present in Training Dataset")
plt.savefig("1.jpg", dpi=300 ,bbox_inches='tight')
plt.show()


# %%
df = (test.isnull().sum() / len(test)) * 100
df = df.drop(df[df == 0].index).sort_values(ascending=False)
missing_data = pd.DataFrame({'Missing Ratio %' :df})
missing_data.plot(kind = "barh")
plt.show()

# %% [markdown]
# ### Unique Values in the categorical columns of the Train Dataset
# There are three categorical columns in the dataset: `facility_type`,
`State_Factor` and `building_class`. From the plot below we find that :
# - `facility_type` has 60 unique values
# - `building_class` has 2 unique values
# - `state_factor` has 7 unique values
#
# We will need to convert these categorical values into numerical values for
the model to be able to understand them.

# %%
df_ = train.select_dtypes(exclude=['int', 'float'])
unique_data = {}
for col in df_.columns:
      unique_data[col] = len(df_[col].unique())
unique_data = pd.DataFrame.from_dict(unique_data, orient='index',
columns=['unique_values'])
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
unique_data.plot(kind = "barh")

plt.xlabel("Number of Unique Values")
plt.ylabel("Columns")
plt.title("Unique values in the Categorical Columns of the Train Dataset")
plt.savefig("2.jpg", dpi=300 ,bbox_inches='tight')

plt.show()


# %% [markdown]
# # Eda

# %% [markdown]
# ### Distribution of the data of Each given factors in training set
#  - The distribution of `floor_area` is skewed to the left with a long tail.
The floor area varies between 900 square feet to 63K square feet
#  - Most buildings were built around 1850 and 2000
#  - Building elevations is skewed to the right with a long tail and most
building are located at the elevation between 0 to 50.
#  - Both `heating_degree_days` and `precipitation_inches` are approximately
normally distributed.
#  - Average temperature is right skewed with a short tail.
#  - The rest of the factors are arbitrarily distributed.

# %%
import warnings
warnings.filterwarnings('ignore')
cols=['Year_Factor', 'floor_area', 'year_built', 'energy_star_rating',
      'ELEVATION',  'cooling_degree_days',
      'heating_degree_days', 'precipitation_inches', 'snowfall_inches',
      'snowdepth_inches', 'avg_temp', 'days_below_30F', 'days_below_20F',
      'days_below_10F', 'days_below_0F', 'days_above_80F', 'days_above_90F',
      'days_above_100F', 'days_above_110F', 'direction_max_wind_speed',
      'direction_peak_wind_speed', 'max_wind_speed', 'days_with_fog']
i=1
for col in cols:
      plt.figure(figsize=(15, 2))
      sns.distplot(train[col],kde=True)
      plt.title(col+" distribution")
      plt.show()


# %% [markdown]
# ### Numerical Features Difference between train and test data
#  - The biggest difference in the distribution of the train and test datasets
is found in `wind speed data`. Wind relate data have over 50% missing data.
#  - For the test dataset, the only available `year factor` is 7 - so there is
no overlap
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
#  - In case of `elevation` there is a sharp right skewed spike in data
distribution of training dataset, which is not present in test dataset.
#  - Similar but opposite trend is observed in `snow_depth` and
`snowfall_inches` data where there is a sharp right skewed spike in data
distribution of testing dataset, which is not present in training dataset.
#  - For rest of the factors, the training and test distributions mostly
overlap with each other

# %%

def dist_difference(train, test, features, title):

      L = len(features)
      ncol= 5
      nrow= int(np.ceil(L/ncol))
      remove_last= (nrow * ncol) - L

      fig, ax = plt.subplots(nrow, ncol,figsize=(16, 16), sharey=False,
facecolor='#dddddd')
      ax.flat[-remove_last].set_visible(False)
      fig.subplots_adjust(top=0.95)
      i = 1
      for feature in features:
      plt.subplot(nrow, ncol, i)
      ax = sns.kdeplot(train[feature], shade=False,  color='salmon',
alpha=0.85, label='train')
      ax = sns.kdeplot(test[feature], shade=False, color='gold',  alpha=0.85,
label='test')
      # ax.yaxis.set_major_formatter(FormatStrFormatter('%.0f'))
      ax.xaxis.set_label_position('top')
      ax.set_ylabel('')
      ax.set_yticks([])
      ax.set_xticks([])
      i += 1

      lines, labels = fig.axes[-1].get_legend_handles_labels()
      fig.legend(lines, labels, loc = 'upper center',borderaxespad= 4.0)

      plt.suptitle(title, fontsize=20)
      plt.show()

dist_difference(train, test, cols, title='Numerical Features Difference between
train and test data');

# %% [markdown]
# ### Contrasting the presence of each statefactor in Train and Test class
# Our test dataset and train dataset contains ratio of the data that are
largely different from each other with respect to `state factor`
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# %%
plt.figure(figsize=(18,6))
plt.subplot(1, 2, 1)
train['State_Factor'].value_counts().plot(kind='pie')

plt.title("State Factors in Train Dataset")

plt.subplot(1, 2, 2)
test['State_Factor'].value_counts().plot(kind='pie')

plt.title("State Factors in Test Dataset")
plt.savefig("3.jpg", dpi=300 ,bbox_inches='tight')

plt.show()

# %% [markdown]
# ### Comparison of State factors between Commercial and Residential building
class
#
# `state_factors` are mainly the anonymized states from which these data were
collected. We can see that `state_6` is the most data-rich state while
`state_10` have almost negligible amount of data comparing with `state_6`. So,
it is safe to assume that whatever we may be trying to predict here may give us
a more accurate result for buildings that are situated at similar states akin
to `state_6`.

# %%
sns.countplot(x='State_Factor',hue='building_class',data=train,order =
train['State_Factor'].value_counts().index)

plt.title("Comparison of State factors between Commercial and Residential
building class")
plt.savefig("4.jpg", dpi=300 ,bbox_inches='tight')

# %% [markdown]
# ### Correlation of all the rows among each other
#
# This plot shows the correlation of different factors that were presented in
the dataset where lightest point shows the highly correlated datapoints and
darkest point showed the least correlated datapoints. From here it is safe to
assume several assumption:
# 1. Elevation is not a huge factor behind the weather data collected for each
buildings.
# 2. `year_built` has a surprisingly high correlation with the
`days_above_110F`. This is a good example of correlation is not causation.
# 3. `energy_star_rating` has high correlation with the hotter areas.
# 4. Precipitation has a negative correlation with `avg_temp`, So this area has
to be on the northern hemisphere.
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# %%
x = train.corr()
plt.figure(figsize=(15,12))
sns.heatmap(x)

plt.title("Correlation of all the given variables")
plt.savefig("5.jpg", dpi=300 ,bbox_inches='tight')

# %% [markdown]
# ### The Number of buildings built in each year (In the dataset)
# While some year datapoints had unusually high number of presence in the
dataset, On average, from the plot below, it is safe to assume that the given
dataset has a good distribution of datapoints over several decades.

# %%
from collections import Counter
x = Counter(train.year_built)
x = sorted(x.items())

years = []
data = []

for i in x:
      if(i[1]>100):
      years.append(int(i[0]))
      data.append(i[1])
plt.figure(figsize=(24,6))
sns.barplot(x=years,y=data)
plt.xticks(rotation=90)
plt.xlabel("Year")
plt.ylabel("Frequency of Buildings built")
plt.title("The number of buildings built in each year")
plt.savefig("6.jpg", dpi=300 ,bbox_inches='tight')
plt.show()

# %% [markdown]
# ### Contrasting Site EUI with State Factor
# Site EUI represents Site Energy Usage Intensity which is the amount of heat
and electricity consumption of a building as reflected in Utility bills. This
plot shows the distribution of SITE EUI for each state factor. This was plotted
on log scale as the range for the `site_eui` data is pretty huge while
containing extreme outliers. From this figure below, if we can ignore the
extreme outliers, it can be seen that the `site_eui` data follows a normal
distribution pattern, although some `state_factor` experience a higher number
concentration of `site_eui` value in the center than the others.

# %%
df =
pd.DataFrame(train[train['building_class']=='Commercial'][['site_eui','floor_ar
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
ea','year_built','ELEVATION','State_Factor','facility_type']].sort_values(by=['
site_eui']).tail(200))
df.dropna()
plt.figure(figsize=(12,8))
sns.histplot(data = train, x = "site_eui",hue='State_Factor', log_scale=10)

plt.title("Distribution of Site EUI in each state factor")
plt.savefig("7.jpg", dpi=300 ,bbox_inches='tight')

plt.show()

# %% [markdown]
# ### Boxplot of SITE EUI with respect to STATE FACTOR
#
# Percentiles are good measure of explaining the skews a dataset may be
experiencing for certain datapoints. In the plot below we can see that
`state_2`, `state_1`, `state_11` have even distributions of collected
`site_eui` data for each states. Meaning, these areas contain an even
distribution of buildings with similar number of buildings that have a very
high number of energy usage intensity annd buildings that have a low number of
energy usage intensity. These states should also have a similar number of
residential and commercial buildings and this assumption is true if we observe
the plot for `Comparison of State factors between Commercial and Residential
building class`.
#
# On the other hand, `state_6`, `state_4` and `state_8` have a higher
concentration of buildings with comparably lower amount of energy usage
intensity. This signifies that the data collected for these areas are likely
residentials.

# %%
plt.figure(figsize=(10,6))
sns.boxplot(x="State_Factor",
            y="site_eui",
            data=df)

plt.title("Percentile distribution of Site EUI in different state factors")
plt.savefig("8.jpg", dpi=300 ,bbox_inches='tight')
plt.show()

# %% [markdown]
# ### Lineplot of minimum temperature per month with respect to State Factor
# - As expected, Electricity demand for cooling is significantly affected by
climate and weather, Cooling in the Summertime, heating in Wintertime, and
between we have Spring and fall.
# - State 4 seems to be having the lowest min temperature among other states
# - It appears that, in comparison to other states, state 11 has a moderate
temperature. The temperature trend line reveals that State 11's lowest and
```

```
maximum temperatures range from the lower 30s to the upper 50s and from the
upper 60s to the upper 90s, respectively.
# - Media temperature for state 10 is higher than the rest of the states

# %%
minimum_temp_columns = train[['State_Factor','january_min_temp',
'february_min_temp', 'march_min_temp', 'april_min_temp',
      'may_min_temp', 'june_min_temp', 'july_min_temp', 'august_min_temp',
'september_min_temp',
      'october_min_temp',
'november_min_temp','december_min_temp']].set_index('State_Factor')
min_temp_data = minimum_temp_columns.unstack().reset_index().rename(columns =
{0:'temp_min', 'level_0': 'month'})

statewise_min_temp = min_temp_data.groupby(['month',
'State_Factor'])['temp_min'].agg('median').reset_index()
# parse the month column
month = ['january', 'february', 'march', 'april','may', 'june',
      'july', 'august', 'september','october', 'november','december']
statewise_min_temp['month'] = statewise_min_temp['month'].apply(lambda x:
x.split('_')[0])
statewise_min_temp['month'] = pd.Categorical(statewise_min_temp['month'],
categories = month, ordered = True)
statewise_min_temp.sort_values(by = 'month', inplace=True)

plt.figure(figsize=(20, 10))
sns.lineplot(x='month', y='temp_min', hue='State_Factor',
            data=statewise_min_temp)

plt.title("Statewise Minimum Temperature in one year")
plt.savefig("9.jpg", dpi=300 ,bbox_inches='tight')

# %% [markdown]
# ### Contrasting the status of heating and cooling days with respect to SITE
EUI and State Factor

# %%
degree_days = train.groupby('State_Factor').agg({'cooling_degree_days':
'median',
                                                 'heating_degree_days': 'median',
                                                 'site_eui': 'median'}).reset_index()
plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
sns.scatterplot(x='cooling_degree_days', y='site_eui', hue='State_Factor',
data=degree_days)
plt.subplot(1, 2, 2)
sns.scatterplot(x='heating_degree_days', y='site_eui', hue='State_Factor',
data=degree_days)
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# %% [markdown]
# ### Distribtuion of Top 10 facilities in buildings
#
# 50% of our training dataset are for multifamily residents while only 1% data
are for Educational institutions, Retail shops and warehouses. It's safe to
assume that we will have a better capability of understanding the behavior of
energy usage of residential buildings from the given dataset.

# %%
df = pd.DataFrame(train['facility_type'].value_counts().head(10))
df['facility'] = df.index
df.rename(columns = {'facility_type':'count'}, inplace = True)
df['%share'] = df['count']/df['count'].sum()
df.drop(columns=['count'],inplace=True)
df.reset_index(drop=True)
plt.figure(figsize=(12,6))
ax = sns.barplot(y='facility',x='%share',data=df)
for i in ax.containers:
      ax.bar_label(i,)
plt.title("Distribution of Top 10 Facilities in Buildings in %")
plt.savefig("10.jpg", dpi=300 ,bbox_inches='tight')
plt.show()

# %% [markdown]
# ### Contrasting the distribution of facility type of each building class
#
# There are only 6 type of residential facility buildings available in the
dataset. The rest are all commercial buildings. This is a good sign as we can
assume that the dataset is more likely to have a better understanding of the
energy usage of commercial buildings.

# %%
plt.figure(figsize=(24,6))

plt.subplot(1, 2, 1)
train[train['building_class']=='Commercial'][['facility_type']].value_counts().
plot(kind='bar', logy=True)
plt.ylabel("Frequency")
plt.xlabel("Commercial Buildings")

plt.subplot(1, 2, 2)
train[train['building_class']=='Residential'][['facility_type']].value_counts()
.plot(kind='bar', logy=True)
plt.xlabel("Residential Buildings")
plt.ylabel("Frequency")

plt.title("Distribution of Facility Type of Each Building class")
plt.savefig("11.jpg", dpi=300 ,bbox_inches='tight')
plt.show()
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# %%
train[train['building_class']=='Residential'][['facility_type']].value_counts()

# %% [markdown]
# ### Contrasting the distribution of ENERGY STAR RATING of each building class
#
# The Building Energy Star rating is usually high for buildings that have low
eui. Another observation here is that state factor 10 is missing for
Residential buildings.

# %%
plt.figure(figsize=(18,6))
com_train = train[train['building_class']=='Commercial']
res_train = train[train['building_class']=='Residential']
plt.subplot(1, 2, 1)
sns.scatterplot(data=com_train, x="energy_star_rating", y="site_eui",
hue="State_Factor", alpha=0.5)
plt.xlabel("Energy Star Rating")

plt.title("Energy Star Rating of Commercial Building Class")

plt.subplot(1, 2, 2)
sns.scatterplot(data=res_train, x="energy_star_rating", y="site_eui",
hue="State_Factor", alpha=0.5)
plt.xlabel("Energy Star Rating")

plt.title("Energy Star Rating of Residential Building Class")

plt.savefig("12.jpg", dpi=300 ,bbox_inches='tight')

# %% [markdown]
# ### Scatterplot of SITE EUI Values in accordance of their year built  in the
training data
#
# From the range and percentile value of `site_eui` over years, we can see
that, over years the range of `site_eui` is greatly increased. This trend can
be attributed to the electrical appliances and machinaries that we have started
using over years. Especially since the 1980s, the increase of `site_eui` has
been observed significantly.

# %%
plt.figure(figsize=(20,6))
sns.scatterplot(x="year_built",
            y="site_eui",
            data=train)
plt.xlim(left=1600, right=2020)
plt.xticks(rotation=90)
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
plt.title("Relation between SiteEUI Value and the year that the buildings were
built")
plt.savefig("13.jpg", dpi=300 ,bbox_inches='tight')
plt.show()

# %% [markdown]
# ### Boxplot of SITE EUI Values in accordance of their year factor
# From the plot below it is evident that - year factor contributes very little
to the variation of SITE EUI values and remains the same across all values of
year factor.

# %%
plt.figure(figsize = (15,6))
df = train
sns.boxplot(x = df['Year_Factor'], y= df['site_eui'])
plt.xlabel("Year Factor", labelpad=10, fontsize=20)
plt.ylabel("site eui", labelpad=10, fontsize=20)
plt.xticks(size = 15)
plt.yticks(size = 15)
plt.grid()
plt.title("Year wise site eui", y=1.02, fontsize=20)

plt.savefig("14.jpg", dpi=300 ,bbox_inches='tight')

# %% [markdown]
# ### Floor area distribution of Each Building class
#
# The plot below shows a very significant trend. It shows the `site_eui`
generally ranges from extremely high to really low for both commerical and
residential areas but larger commercial areas always have a very low amount of
`site_eui`. This means that while residential areas are smaller than commercial
areas, due to usage of appliances, utility bills are always significantly
higher for residential areas and big commercial warehouses don't need that much
weather control inside. Additionally, the floor area outliers are mainly
related to commercial buildings

# %%
train['floor_area'].describe()

# %%
plt.figure(figsize=(12,6))
sns.scatterplot(data=train,x='floor_area',hue='building_class',y='site_eui')


plt.title("Floor area and site eui distribution of each building class")
plt.savefig("15.jpg", dpi=300 ,bbox_inches='tight')
plt.show()

# %% [markdown]
```

```
# ### Contrasting floor area with energy star rating for training and test set
#
# Energy stars signifies the efficiency of energy consumption in a building.
From the plots below, we first notice that, `State factor` 6 is missing in the
test data. Besides that the distribution of datapoint across floow area and
energy star rating seems to follow the same trends and range, with some
outliers.
#

# %%
plt.figure(figsize=(20,6))
plt.subplot(1,2,1)
sns.scatterplot(data=train,y='floor_area',hue='State_Factor',x='energy_star_rat
ing')
plt.title("Train Dataset")

plt.subplot(1,2,2)
sns.scatterplot(data=test,y='floor_area',hue='State_Factor',x='energy_star_rati
ng')
plt.title("Test Dataset")

plt.title("Relation of Floor Area with Energy Star Rating")
plt.savefig("16.jpg", dpi=300 ,bbox_inches='tight')
plt.show()

# %% [markdown]
# #### Floor area of 50K square feet and above tend to have high median
site_eui.

# %%
plt.figure(figsize=(20,6))
sns.scatterplot(data=train,x='floor_area',y='site_eui',hue='building_class')

plt.title("Relationship between Site EUI and Building Area")
plt.savefig("17.jpg", dpi=300 ,bbox_inches='tight')

# %% [markdown]
# ### Contrasting Elevation of different building classes
# Commercial buildings make up the majority of the structures located over 300
feet (elevation)! However, as there are no residential structures over 500 feet
(Elevation), commercial structures are what the Elevation variable's outliers
are tied to.

# %%
plt.figure(figsize=(12,6))
sns.barplot(x='building_class',y='ELEVATION',data=train)

plt.title("Elevation of each building class")
plt.savefig("18.jpg", dpi=300 ,bbox_inches='tight')
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# %% [markdown]
# ### Contrasting Elevation of different building classes with their EUI
#
# Elevation is also a noticably big factor for `site_eui`. Areas that are most
closely located to the ground have a higher amount of air circulation control
than the areas that are high above ground which incurs them a very low amount
of utility bills.
# From the plot we see that buildings below 300 (elevation) are using more
energy. So, more significant energy users (like Data warehouses) can be
suggested to be placed high up in the buildings.

# %%
plt.figure(figsize=(12,6))

sns.scatterplot(data=train, x="ELEVATION", y="site_eui",hue='building_class')

plt.title("Relationship of Elevation of different building class and their
SiteEUI")
plt.savefig("19.jpg", dpi=300 ,bbox_inches='tight')
plt.show()

# %% [markdown]
# ### Contrasting the distribution of SITE_EUI across different facility types
# On average, grocery_store/food_market, data_centers, and laboratory have
higher site_eui. These facilities are usually larger than the rest of the
facilities and/or have higher user concentration.

# %%
plt.figure(figsize=(8,16))
sns.boxplot(y='facility_type',x='site_eui',data=train)

plt.title("Distribution of Site EUI across different facility types")
plt.savefig("20.jpg", dpi=300 ,bbox_inches='tight')

# %%
```

Modeling Notebook

```
import numpy as np
import pandas as pd
from datetime import datetime
import random
import copy
import time
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
import matplotlib as plt
from sklearn.impute import KNNImputer
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold, RepeatedKFold,
GridSearchCV, cross_val_score, cross_validate, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor

import lightgbm
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor

import torch
import torch.nn.functional as F
import torch.nn as nn
import torch.optim as optim
from torch.utils import data
from torch.utils.data import DataLoader, TensorDataset
from tensorboardX import SummaryWriter

def fill_missing_values(df, test_df):
    missing_columns_train = [col for col in df.columns if
df[col].isnull().any()]
    print("Columns with Missing values in train set:",
missing_columns_train)
    missing_columns_test = [col for col in test_df.columns if
test_df[col].isnull().any()]
    print("Columns with Missing values in test set:",
missing_columns_test)

    df.loc[df["year_built"] == 0, "year_built"] = np.nan
    test_df.loc[test_df["year_built"] == 0, "year_built"] = np.nan

    imputer = SimpleImputer()
    imputer.fit(train[missing_columns_train])
    data_transformed =
imputer.transform(train[missing_columns_train])
    df[missing_columns_train] = pd.DataFrame(data_transformed)
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
    test_data_transformed =
imputer.transform(test[missing_columns_test])
    test_df[missing_columns_test] =
pd.DataFrame(test_data_transformed)

    print("Columns with Missing values in train set after
imputation:", df.columns[df.isnull().any()])
    print("Columns with Missing values in test set after
imputation:", test_df.columns[test_df.isnull().any()])

    return df, test_df


def categorical_label_encoding(df, test_df):
    object_cols = []
    int_cols = []
    float_cols = []
    for col in df.columns:
        if col!='site_eui':
            if df[col].dtype=='object':
                object_cols.append(col)
            elif df[col].dtype=='int64':
                int_cols.append(col)
            elif df[col].dtype=='float64':
                float_cols.append(col)
    print(f"Starting Label Encoding for {object_cols}")
    le = LabelEncoder()
    for col in object_cols:
        df[col] = le.fit_transform(df[col])
        test_df[col] = le.fit_transform(test_df[col])

    return df, test_df


def scale_data(df, test_df):
    print("Scaling Data with StandardScaler")
    import copy

    # code copied from
https://www.kaggle.com/usharengaraju/wids2022-lgbm-starter-w-b
    y_df = df["site_eui"]
    df = df.drop(["site_eui", "id"], axis=1)
    test_df = test_df.drop(["id"], axis=1)
    scaler = StandardScaler()
    df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
    test_df = pd.DataFrame(scaler.transform(test_df),
columns=test_df.columns)
    return df, test_df, y_df


def preprocess_data(df, test_df):
    df, test_df = fill_missing_values(df, test_df)
    df, test_df = categorical_label_encoding(df, test_df)
    df, test_df, y_df = scale_data(df, test_df)
    print("Preprocessing Done")
    print("Train Data Shape:", df.shape)
    print("Test Data Shape:", test_df.shape)
    return df, test_df, y_df


train = pd.read_csv(r"/content/drive/MyDrive/wids/train.csv")
test = pd.read_csv(r"/content/drive/MyDrive/wids/test.csv")
preprocessed_df, preprocessed_test_df,y_df = preprocess_data(train,
test)
# Split train set into train and test for model validation

X_train, X_test, y_train, y_test = train_test_split(
    preprocessed_df, y_df, test_size=0.25, random_state=42)

print("Train Data Shape:", X_train.shape)

print("Test Data Shape:", X_test.shape)
# keep track of the model r2-score in a list
df_r2_score = []

# A class which return cross-val mean R2 score

class CV_regression_model ():
    def __init__(self, model_name, model, X, y, folds=10):
        self.model_name = model_name
        self.model = model
        self.X = X
        self.y = y
        self.folds = folds
        self.results = None
        self.mean_score = None
        self.std_score = None

    def fit(self):
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
        scores = cross_val_score(self.model, self.X, self.y,
cv=self.folds, scoring='r2')
        self.results = scores
        self.mean_score = np.mean(scores)
        self.std_score = np.std(scores)

    def print_results(self):
        print(f"Model: {self.model_name}")
        print(f"Mean score: {self.mean_score}")
        print(f"Standard Deviation: {self.std_score}")
        print(f"R2 scores: {self.results}")

    def r2_scores(self):
        return round(self.mean_score, 4)
lr_model = LinearRegression()
lr = CV_regression_model("Linear Regression", lr_model, X_train,
y_train)
lr.fit()
lr.print_results()
df_r2_score.append(lr.r2_scores())
dt_model = DecisionTreeRegressor(random_state=42)
dt = CV_regression_model("Decision Tree", dt_model, X_train, y_train)
dt.fit()
dt.print_results()
df_r2_score.append(dt.r2_scores())
rfr_model = RandomForestRegressor(random_state=42)
rfr = CV_regression_model("Random Forest", rfr_model, X_train,
y_train)
rfr.fit()
rfr.print_results()
df_r2_score.append(rfr.r2_scores())
gbr_model = GradientBoostingRegressor(random_state=42)
gbr = CV_regression_model("Gradient Boosting", gbr_model, X_train,
y_train)
gbr.fit()
gbr.print_results()
df_r2_score.append(gbr.r2_scores())
xgbr_model =  XGBRegressor(random_state=42)
xgbr = CV_regression_model("XGBoost", xgbr_model, X_train, y_train)
xgbr.fit()
xgbr.print_results()
df_r2_score.append(xgbr.r2_scores())

lgbm_model = lightgbm.LGBMRegressor(random_state=42)
lgbm = CV_regression_model("LightGBM", lgbm_model, X_train, y_train)
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
lgbm.fit()
lgbm.print_results()
df_r2_score.append(lgbm.r2_scores())

# Define a data frame that includes R2 score and model numbers.
result = pd.DataFrame()
df_model = ["Linear Regression", "Decision Tree", "Random Forest",
"Gradient Boosting", "XGBoost", "LightGBM"]
# Assigne model name and R2 scores
result['model'] = df_model
result['r2'] = df_r2_score
plt.figure(figsize=(10, 6))
sns.barplot(x='model', y='r2', data=result)
plt.title('Mean R2 Scores of Models')
plt.show()

rfr = RandomForestRegressor(random_state=42)
rfr.fit(X_train, y_train)
# Create a dataframe which contains feature names and corresponding
feature importance scores
zipzip = zip(rfr.feature_importances_, X_train.columns)
df_feature = pd.DataFrame(data = zipzip, columns =
['importance','feature'])

# Sort the dataframe by importance scores
df_feature.sort_values('importance', ascending = False, inplace =
True)

# Top 10 important features
top10 = df_feature.head(10)
top_rfr_features = top10["feature"]
top10
lgbm = lightgbm.LGBMRegressor(random_state=42)
lgbm.fit(X_train, y_train)
# Create a dataframe which contains feature names and corresponding
feature importance scores
zipzip = zip(lgbm.feature_importances_, X_train.columns)
df_feature = pd.DataFrame(data = zipzip, columns =
['importance','feature'])

# Sort the dataframe by importance scores
df_feature.sort_values('importance', ascending = False, inplace =
True)

# Top 10 important features
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
top10 = df_feature.head(10)
top_lgbm_features = top10["feature"]
top10

top_2_r2 = []
# train on selected features
rfr_model = RandomForestRegressor(random_state=42)
rfr_model.fit( X_train[top_rfr_features], y_train)
rfr_y_pred = rfr_model.predict(X_test[top_rfr_features])
rfr_r2_score = r2_score(y_test.tolist(), rfr_y_pred)
print("Test R2 Score: ",  rfr_r2_score)
top_2_r2.append(rfr_r2_score)
# train on selected features
lgbm_model = lightgbm.LGBMRegressor(random_state=42)
lgbm_model.fit( X_train[top_lgbm_features], y_train)

lgbm_y_pred = lgbm_model.predict(X_test[top_lgbm_features])
lgbm_r2_score = r2_score(y_test.tolist(), lgbm_y_pred)
print("Test R2 Score: ",  lgbm_r2_score)
top_2_r2.append(lgbm_r2_score)
result = pd.DataFrame()
df_model = [ "Random Forest", "LightGBM"]
# Assigne model name and R2 scores
result['model'] = df_model
result['r2'] = top_2_r2
plt.figure(figsize=(10, 6))
sns.barplot(x='model', y='r2', data=result)
plt.title('Test Data R2 Scores of Top Models')
plt.show()
from sklearn.metrics import accuracy_score
rfr = RandomForestRegressor(random_state=42)
rfr.fit(X_train[top_rfr_features], y_train)
print(rfr.get_params())
print(rfr.score(X_test[top_rfr_features], y_test))

# define model
model = RandomForestRegressor(random_state=42)

# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

# define grid
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}

# define search
search =  GridSearchCV(estimator = model, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

# perform the search (runs for approximately 4 hours)
results = search.fit(X_train[top_rfr_features], y_train)

# summarize
print('R2: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
model_GS = RandomForestRegressor(random_state=42, bootstrap=True,
max_depth=80, max_features=3, min_samples_leaf=3,
min_samples_split=8, n_estimators=1000)
model_GS.fit(X_train[top_rfr_features], y_train)
print(model_GS.score(X_test[top_rfr_features], y_test))
# further split train data to train and eval
X_train, X_eval, y_train, y_eval = train_test_split(
    preprocessed_df, y_df, test_size=0.10, random_state=42)
print("Train Data Shape:", X_eval.shape)
print("Eval Data Shape:", X_eval.shape)
top_rfr_features = ["floor_area",
     "facility_type",
     "energy_star_rating",
     "year_built",
     "ELEVATION",
     "building_class",
     "days_with_fog",
     "State_Factor",
     "precipitation_inches",
     "max_wind_speed"]
x_train_tensor = torch.Tensor(X_train[top_rfr_features].values)
x_eval_tensor = torch.Tensor(X_eval[top_rfr_features].values)
x_test_tensor = torch.Tensor(X_test[top_rfr_features].values)
y_train_tensor = torch.Tensor(y_train.values)
y_eval_tensor = torch.Tensor(y_eval.values)
y_test_tensor = torch.Tensor(y_test.values)
# #Evaluation set for submission

# eval_target = torch.Tensor(test_f_id.to_numpy())
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# y_eval = torch.Tensor(test_f)

#Create Tensor datasets out of numpy dataset
train_dataset = TensorDataset(x_train_tensor, y_train_tensor)
eval_dataset = TensorDataset(x_eval_tensor, y_eval_tensor)
test_dataset = TensorDataset(x_test_tensor, y_test_tensor)
# eval_dataset = TensorDataset(y_eval, eval_target)

#Create the dataloaders
train_dataloader = DataLoader(train_dataset, batch_size=64,
shuffle=True, num_workers=2)
eval_dataloader = DataLoader(eval_dataset, batch_size=64,
shuffle=True, num_workers=2)
test_dataloader = DataLoader(test_dataset, batch_size=64,
shuffle=False, num_workers=2)

class SimpleMLP(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(SimpleMLP, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(inplace=True),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(inplace=True),
            nn.Linear(hidden_size, 256),
            nn.ReLU(inplace=True),
            nn.Linear(256, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, 64),
            nn.ReLU(inplace=True),
            nn.Linear(64, 1)

        )

    def forward(self, x):
        obs = self.net(x)
        return obs
n_features = 10
hidden_size = 300
model = SimpleMLP(n_features, hidden_size)
print("Model Architecture: ", model)
model.to(device)
# construct an optimizer
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
params = [p for p in model.parameters() if p.requires_grad]
optimizer = optim.Adam(params, lr=1e-3)
print("Optimizer: ", optimizer)
# and a learning rate scheduler
lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.1)
print("Learning Rate: ", lr_scheduler)

loss_function = nn.L1Loss()
print("Loss function: ", loss_function)

def train_one_epoch(epoch_index, tb_writer):
    running_loss = 0.
    last_loss = 0.

    # Here, we use enumerate(training_loader) instead of
    # iter(training_loader) so that we can track the batch
    # index and do some intra-epoch reporting
    for i, data in enumerate(train_dataloader):
        # Every data instance is an input + label pair
        inputs, labels = data
        # Zero your gradients for every batch!
        optimizer.zero_grad()

        # Make predictions for this batch
        outputs = model(inputs.to(device))
        # Compute the loss and its gradients
        loss = loss_function(outputs.squeeze(1), labels.to(device))
        loss.backward()
        # Adjust learning weights
        optimizer.step()

        # Gather data and report
        running_loss += loss.item()
        if i % 800 == 799:
            last_loss = running_loss / 800 # loss per batch
            tb_x = epoch_index * len(train_dataloader) + i + 1
            tb_writer.add_scalar('Loss/train', last_loss, tb_x)
            running_loss = 0.


    return last_loss
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
# Initializing in a separate cell so we can easily add more epochs to the same run
timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
writer = SummaryWriter('runs/MLP_trainer_{}'.format(timestamp))
epoch_number = 0
EPOCHS = 50
loss_list = []

best_vloss = 1_000_000.

for epoch in range(EPOCHS):
    print('EPOCH {}:'.format(epoch_number + 1))
    ts = time.time()
    # Make sure gradient tracking is on, and do a pass over the data
    model.train(True)
    avg_loss = train_one_epoch(epoch_number, writer)
    train_time = time.time() - ts
    # We don't need gradients on to do reporting
    model.train(False)
    model.eval()

    te = time.time()
    running_vloss = 0.0
    for i, vdata in enumerate(eval_dataloader):
        vinputs, vlabels = vdata
        voutputs = model(vinputs.to(device))
        vloss = loss_function(voutputs.squeeze(1), vlabels.to(device))
        running_vloss += vloss

    avg_vloss = running_vloss / (i + 1)
    print('Train Loss {} Validation Loss {}'.format(avg_loss, avg_vloss))
#     print('I was trained on {} for {} and evaluated for {} seconds'.format(device,
round(train_time, 3), round(time.time() - te, 3)))
    # Log the running loss averaged per batch
    # for both training and validation
    writer.add_scalars('Training vs. Validation Loss',
                { 'Training' : avg_loss, 'Validation' : avg_vloss },
                epoch_number + 1)
    writer.flush()
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
    # Track best performance, and save the model's state
    avg_mse = avg_vloss.detach().cpu().numpy() / 1.0
#     print(avg_mse)
    if avg_vloss < best_vloss:
        best_vloss = avg_vloss
        model_path = 'model_{}_{}'.format(timestamp, epoch_number)
        torch.save(model.state_dict(), model_path)

    loss_list.append([avg_loss, avg_mse])
#     lr_scheduler.step()

    epoch_number += 1


# print(loss_list)
mean_losses = pd.DataFrame(loss_list, columns=['Average_Training_Loss',
'Average_Validation_Loss'])

sns.set()
#Plot the train and validation losses.
# sns.relplot(x='epoch', y='Average_Validation_Loss',  kind='line', data=mean_losses)
# sns.relplot(x='epoch', y='Average_Training_Loss',  kind='line', data=mean_losses)

plt.figure(); mean_losses.plot(); plt.legend(loc='best')
predictions_eui = []
model.eval()

for i, vdata in enumerate(test_dataloader):
    vinputs, v_id = vdata
    voutputs = model(vinputs.to(device))
    predictions_eui.append(voutputs.detach().cpu().numpy())

out = np.concatenate(predictions_eui).ravel()
r2_score(y_test.tolist(), out)

from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022

```
xtrain, ytrain = np.array(X_train[top_rfr_features]), np.array(y_train)
xtrain= np.reshape(xtrain,(xtrain.shape[0],xtrain.shape[1],1))

#initialisizng the model
regression= Sequential()

regression.add(LSTM(units=30,return_sequences=True,kernel_initializer='glorot_uniform',input_shape=(xtrain.shape[1],1)))
regression.add(Dropout(0.2))

regression.add(LSTM(units=30,kernel_initializer='glorot_uniform',return_sequences=True))
regression.add(Dropout(0.2))

regression.add(LSTM(units=30,kernel_initializer='glorot_uniform',return_sequences=True))
regression.add(Dropout(0.2))

regression.add(LSTM(units=30,kernel_initializer='glorot_uniform'))
regression.add(Dropout(0.2))
regression.add(Dense(units=1))

#Compiling the network
regression.compile(optimizer='adam',loss='mean_squared_error')

regression.fit(xtrain,ytrain,batch_size=30,epochs=50)

xtest= np.array(X_test[top_rfr_features])
xtest= np.reshape(xtest,(xtest.shape[0],xtest.shape[1],1))
predicted_value= regression.predict(xtest).flatten().tolist()
print(predicted_value)
r2_score(y_test.tolist(), predicted_value)
```

*Public Dataset available at: https://www.kaggle.com/c/widsdatathon2022