Contents

9. **References**

MACHINE LEARNING BASED DATA-DRIVEN APPROACH FOR BUILDING ENERGY STAR SCORE PREDICTION

1. INTRODUCTION

The Energy Star score gives a comprehensive depiction of your building's vitality execution, considering the building's physical assets. It is a simple but powerful tool expressed as a number on a simple scale of 1-100. Based on genuine measured data, energy Star Score evaluates how the buildings is performed by considering its physical properties, operations, and how individuals utilize it. In this work, we predicted the scores

sing regression (linear regression, Random Forest regression and interpret regression) and classification methodologies (Random Forest Classifier and Interpret Classification). One of the key features is that we can interpret an ensemble of decision trees is through the key features, these can be interpreted as the variable which are most predictive of the target. As we can try to peer into the black box of the model we build, we can know the accurate and predict the score to within 9.1 points. The random forest regressor trained on the training data was able to achieve an average absolute error of 10 points on a hold out testing set, which is significantly better than the baseline measure.

1.1 Abstract: An energy star score is a percentile comparison of your building against other comparative buildings within the U.S. An energy performance score gives an outside reference that makes a difference vitality directions survey how effectively a building employments vitality relative to comparable buildings. As a quick understanding this Energy star score permits everybody as how building is performing as the score of 50 speaks to middle vitality execution, whereas a score of 75 or higher demonstrate building may be a beat entertainer and may be qualified for certification. With the assistance of water-normalized vitality utilize escalated, costs, water utilize, nursery gas emanations, or another execution pointer of your choice. So, these factors help us to compare and predict a better score. As per the fire stations report on Energy utilize comparison portfolio director empowers us to choose from 18 wide categories and 80 distinctive capacities to portray the sort of building. This makes it simple for us to recognize and compare for superior understanding.

In this work, we depict the investigation of NYC benchmarking dataset, which degree 60 factors over 11,000 unmistakable buildings, in this study. Agreeing to nearby NYC Benchmarking Law (neighbourhood law 84 of 2009), annually benchmarking and distribution of vitality and water utilizing information is required. A single building with a net floor space more prominent than 50,000 square feet (Sq. ft) and assess parcels having more than one building with a net floor zone of more than 100,000 sq. ft that incorporates charges. The NYC Benchmarking Law in the year 2018 expands to encompass properties larger than 25,000 sq. ft. We utilized the dataset incorporates data provided to the City by Admirable 1, 2016, for calendar year 2015 vitality and water utilization measurements. The metrices are calculate by Natural Assurance Agency's and taken after by Vitality STAR portfolio Director, with information is self-reporting by building proprietors. Information accessibility is open, permitting building's execution to compare between nearby and national information, incentivizing the foremost exact benchmarking of vitality utilization, and illuminating vitality administration choices.

Our interest is to calculate the Energy star Score which is often used for aggregate measure of overall

efficiency of a building by considering some key factors like Gross floor Area, Energy and water, Site EUI, source EUI, Fuel oil, Natural Gas use, Electricity use, GHG Emission, Property GFA etc. The energy star score could be a percentile evaluation of a building's vitality execution based on self- detailed vitality utilize. In spite of the fact that the precise usage points of interest may change, the common structure as takes after: information cleaning and arrangement, Exploratory information examination, Include building and choice, building up a standard and execution metric, performing hyperparameter optimization, testing information set, deciphering demonstrate comes about to the most noteworthy degree conceivable and at long last drawing conclusions.

1.2 Contribution:

The major contribution of this work is as follows:

i) Find the predictors within the dataset for the Energy Star Score

ii) To predict the Score, we use Regression/Classification models that can predict the Energy Star Score of the building with the given dataset of buildings energy

iii) Now we interpret the results of the model and use it as the training model and try to infer with the new buildings and get the score of Energy Star Score of new buildings

2. LITERATURE REVIEW

In this section, we present the broad literature pertaining to energy star score prediction systems.

Author	Algorithm	Performance	Inference
Clayton Mille (2018) [8]	Multiple linear regression and gradient boosted trees	We utilize two recommended datadriven displaying procedures to illustrate an blunder rate diminished when compared to the current state of benchmarking with the Vitality Star framework and with a extraordinary work..	The result and methods are as it were important to the tried information input. As we will see in this case, the method was carried out on three huge sets of information from different parts of the Joined together States
E kontokosata (2018)	Strategy utilized: pandas, NumPy, XGBoost, scikit-learn (demonstrate determination, K- implies clustering), We utilize Python's matplotlib, seaborn,	Particularly, we utilize XGBoost as a variation of slope tree boosting, with the set of subjective highlights which	The approach is powerfully upgraded with the foremost later information streams and may be utilized to cities with heterogeneous characteristics. execution measurements, and

	and plotly libraries for visualizations.	contains an exceptionally great result.	the suitable benchmarks for building vitality grading
DongWookSohn (2018) [10]	technique is in utilizing the taking after bundles: pandas, NumPy (information preprocessing)	The performance is not accurate and working model and given data are just fine and can have a better output performance by changing the techniques.	The effects of numerous urban form elements on building energy use are exceedingly intricate. Based on climatic variations in several places. This leads to a continuing investigation of urban from in many cities.
Gary Pivo (2018) [11]	a statistical analysis, s linear regression, REGRESSION EQUATION	Performance of this paper is completely based on formula and arranging the data works perfectly fine but can use the ML techniques for more understanding	Pros: Easy in implementation and easy with calculations Cons: not accurate in results
Jillian Doane (2018) [12]	Statistics and basic math	Performance is not to the mark and it's a new Statistical study on this Particular Study	Pros: First statistically-significant Study of energy performance for LEED
Nofri YenitaDahlan (2022) [13]	MLR model, Data checking, cleansing and grouping, Parameter optimization	It's a good technique for analysis with a little drawback but we can proceed with the techniques	Cons: only require huge data sets and rejects less dataset

Hanaa Talei (2022) [14]	K-Means Clustering Algorithm	In terms of outside temperature and user count, all three scaling algorithms generated satisfactory clustering results. It works perfectly	Pros: The findings suggest that time series analysis and an unsupervised learning approach may be used to differentiate timeslots.
Seyed Milad Parvaneh (2022) [15]	Artificial Neural Networks Multiple Linear Regression Boosting Trees Least Squares regression	two ML algorithms, including OLS and ANN, were employed to train prediction models in order to estimate energy consumption in residential buildings so this output obtained result	Pros: probably need to analyze each building type independently because consumption patterns seem highly dependent on building types and their application cons: The model performance and discuss possible sources of prediction failure that were not considered
Araham Jesus Martinez Lagunas et al. (2022) [16]	data preprocessing method applied is a combination of supervised and unsupervised learning	Works fine able to predict their data for up to 30 years based on data provided	Pros: Have good ML techniques and have a prediction accuracy of 73%
SimonWenninger (2022) [17]	MLR, XGB, ANN, and the QLattice	Works amazing, high prediction performance, and explain ability by design.	Pros: In terms of CV, all algorithms have almost the same standard deviation. One of the inner folds algorithms may overfit, or the method may be sensitive to hyper adjustments.
John H. Scofield (2022) [18]	Regression, linear regression.	Not a very good model based of medical buildings Shows a good result	Pros: Shows a very good result Cons: Only shows the result of medical buildings and failed in residential buildings

CalebRobinson (2022) [19]	XGBoost Extra Trees Regressor Random Forest Regressor MLP Regressor Ridge Regressor Linear Regression	The consumption mainly focus on improving model effectiveness in predicting building energy usage.	Pros: The models can highlight options for future development that we are interested in exploring, such as an investigation of the possible effect of future climatic scenarios and alternative patterns of urban expansion.

3. PROPOSED MODEL The first task is to process the dataset into an organized form and with the help of the preliminary data cleaning and exploration we have done the task and processed further the organized data.

3.1 Missing values

First, we imported the packages required and checked the new dataset which has over 11000 buildings and 60 energy related features each. We found most of the columns empty with a significant portion of missing values represented as ‘Not Available’, so we filled and changed the with nan and converted the appropriate columns to numerical values.

We can begin analysis now that we have the right column datatypes by looking at the percentage of missing values in each column. Missing values are acceptable when performing exploratory data analysis, but they must be filled in when using machine learning methods. We will eliminate any columns with more than 50% missing values for this operation. In general, avoid removing any information because it may not be present for all observations, it might still be helpful in estimating the goal value. Before doing machine learning, the remaining missing data must be imputed using an acceptable approach

3.2 Data Exploration

As we concerned about the Energy star Score and so we made a distribution of measures across all the buildings into the dataset that have a score (9642).

3.2.1 Potential Issue with Energy Star Scores

As we see in the Fig1 the scores w measured is based on self-reported energy usage of a buildings that are supposed to present percentile energy performance rank of the building as the above distribution is extraordinary efficient buildings or the calculate is not objective. As we know around 7% of the buildings have a score of exactly 100, but we expect a uniform distribution as per the percentile measure.

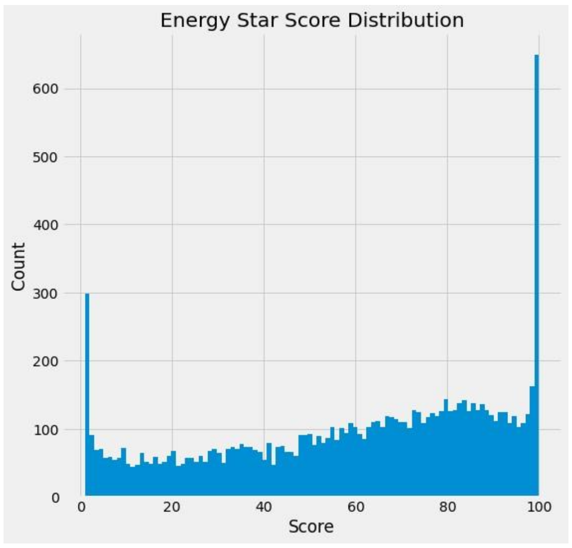


Fig1: Score Distribution according to efficiency

In addition to Energy Star Score, we check the Score based on Energy Use Intensity (EUI) on bases of utility. So, to calculate by taking annual energy use and square footage of building, the EUI is meant to normalize the use comparisons between buildings of the same buildings. So, on based on graph we see the EUI is normal which is what we except from a random variable.

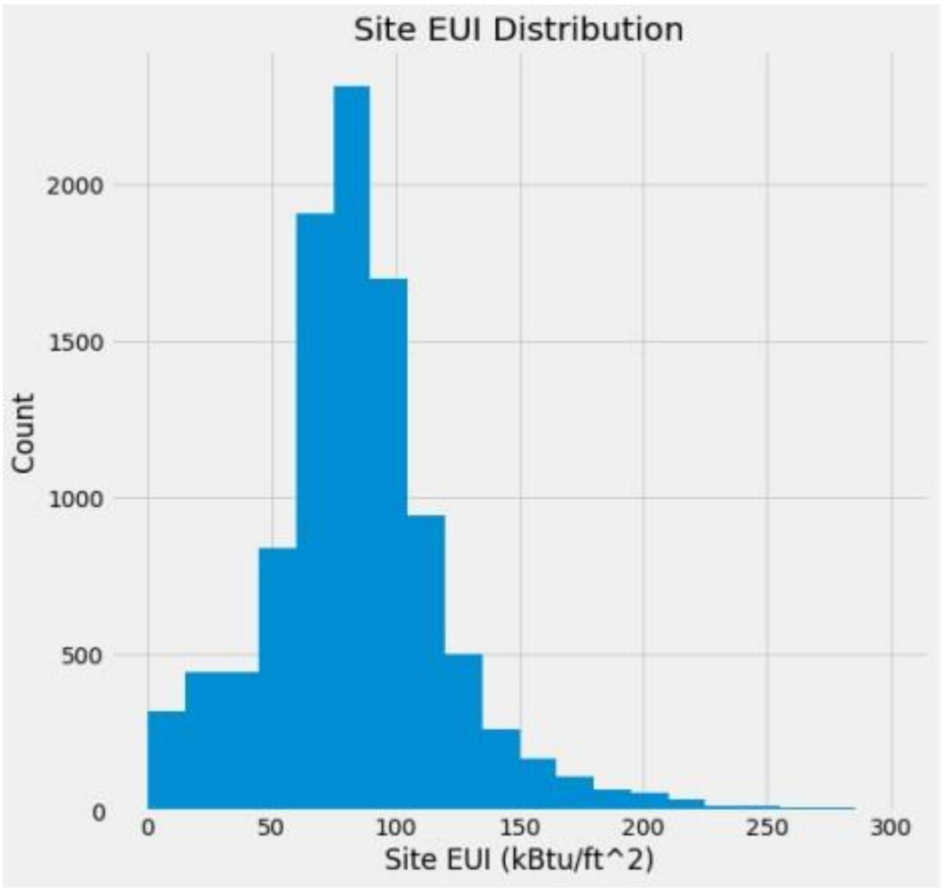


Fig2: Site EUI where distribution is unusual

3.2.2 Energy Star Score by Building Type

We can map the distribution of energy star scores by building type to see if specific building types tend to score higher or lower on the Energy Star Score. The density map below depicts the distribution of scores for building types with more than 80 measurements. Because of the, the x-limits of the kernel density plot stretch outside the range of real scores. The probability density distribution was created using the kernel density estimation method. The numbers in a density plot might be difficult to read, thus it is better to concentrate on the figure's distribution/shape.

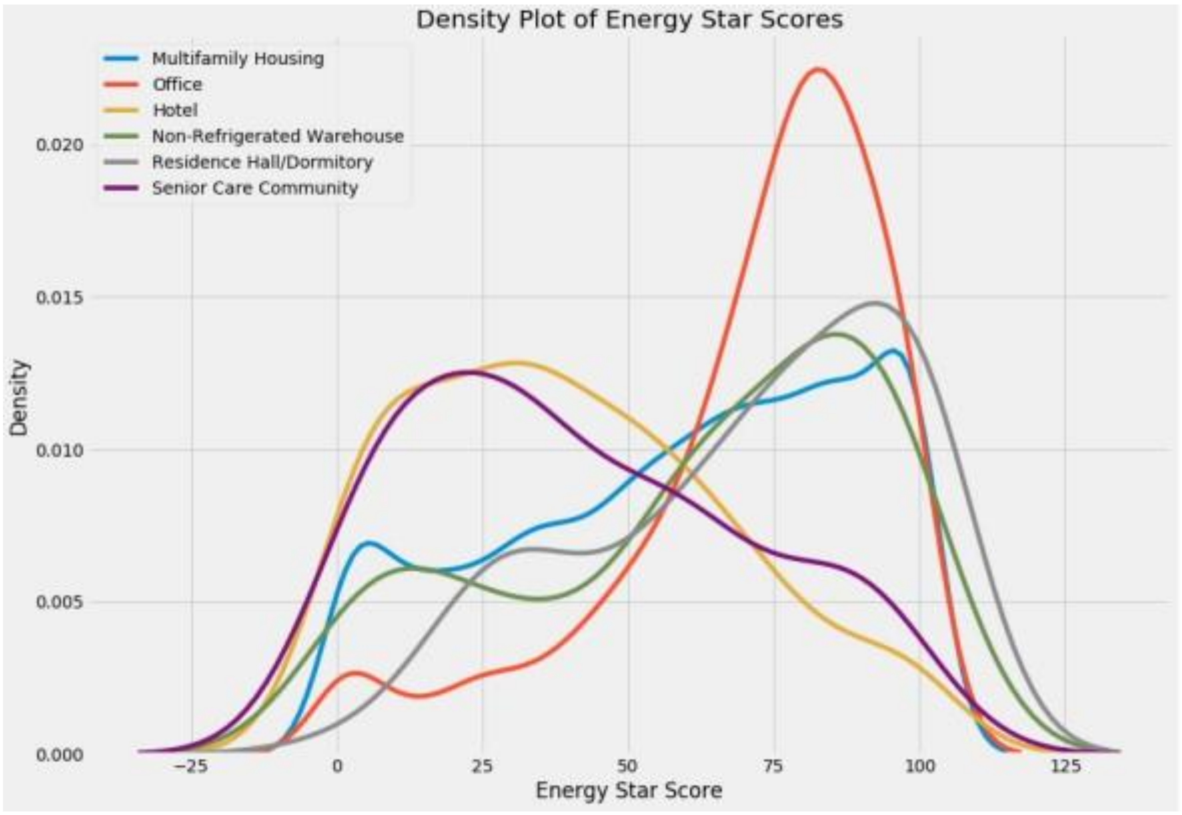


Fig4: Density plot of Energy Star Score

This gives us our to begin with conclusion: office buildings tend to have much higher Vitality Star Scores than inns and senior care communities. Home corridors and non-refrigerated stockrooms also have higher scores whereas multifamily lodging incorporates a bi-modal dispersion of scores comparative to the overall conveyance. It would be valuable to see on the off chance that this holds over a bigger test measure, and to get more information to figure out why a few building sorts tend to do better. In the exploration note pad, I recognized that there was a negative relationship between the Site EUI and the Vitality Star Score. Ready to plot the Location EUI by building sort to see on the off chance that there's moreover a difference in Location EUI between building sorts.

3.2.3 Site EUI by Building Type

Based on analysing form Fig5 the two distributions of building types, we may draw another conclusion: there appears to be a negative link between the Site EUI and the Energy Star Score. Lower Site EUI building types tend to have better Energy Star Scores. The higher the energy use intensity (energy use / area), the worse the energy efficiency performance of the structure. In a scatterplot, we can see the link between the Energy Star Score and the Site EUI.

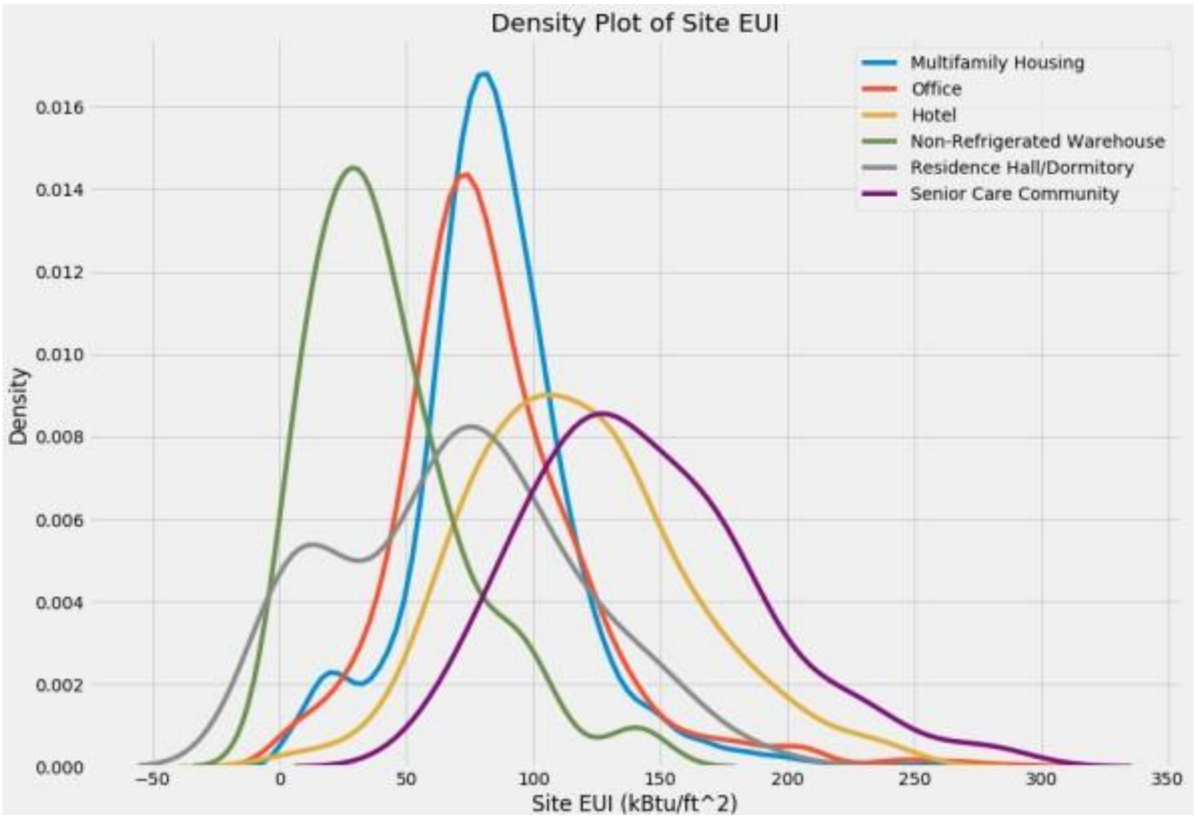


Fig5: Density Plot of site EUI based on Building Types

3.2.4 Energy Star Score VS Stie EUI

The plot in the Fig6 shows the expected negative relationship between Energy Star Score and Site EUI which holds the relation across building types. We can calculate the Pearson Correlation Coefficient between the two variables. Nature of linear correlation which shows both the strength and the direction of the relationship. We will look at the correlation coefficient between Energy Star Scores and Several measures.

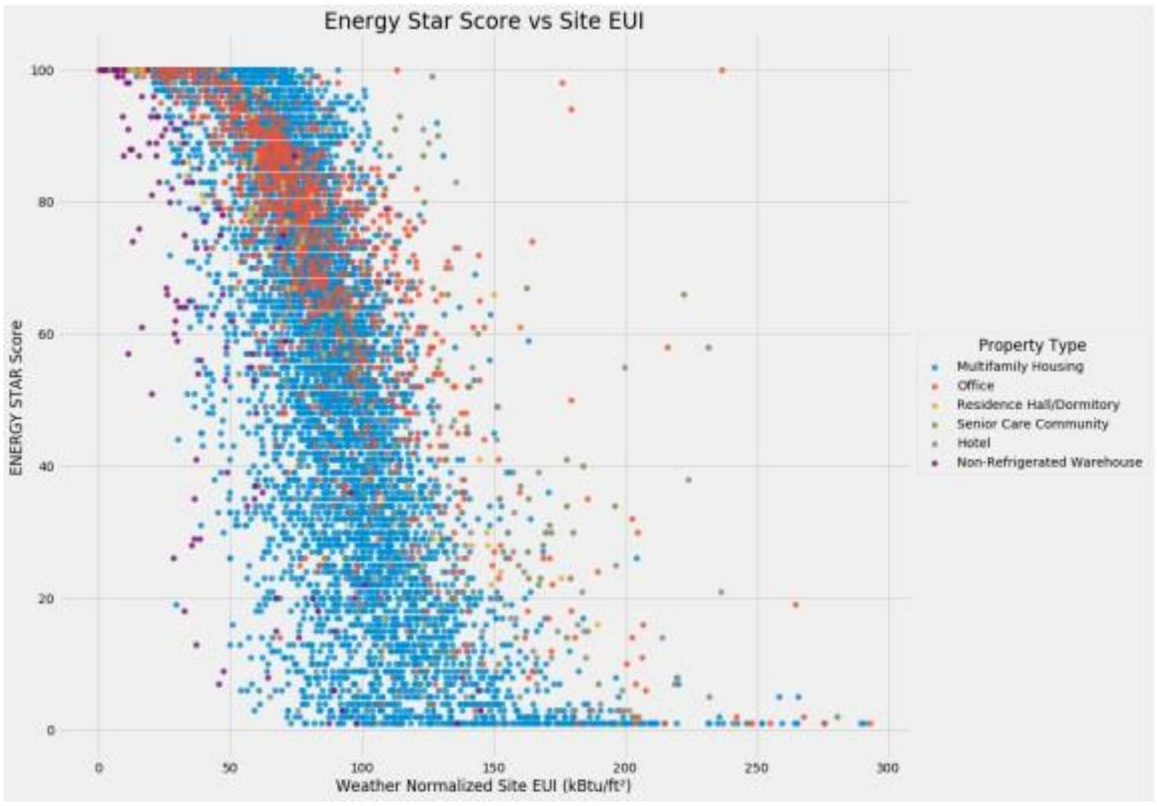
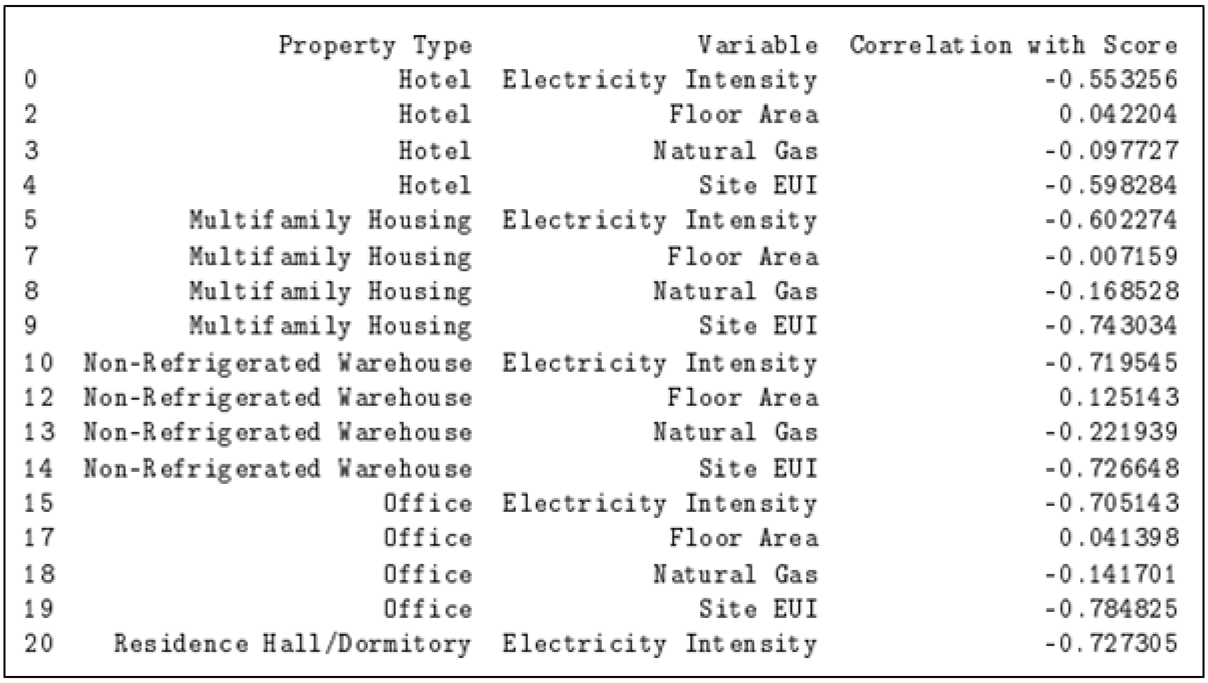


Fig6: Energy Star Score vs EUI

3.2.5 Linear Correlations with Energy Star Score



```
                Property Type                Variable  Correlation with Score
0                       Hotel  Electricity Intensity               -0.553256
2                       Hotel              Floor Area               0.042204
3                       Hotel             Natural Gas              -0.097727
4                       Hotel                Site EUI              -0.598284
5         Multifamily Housing  Electricity Intensity               -0.602274
7         Multifamily Housing              Floor Area              -0.007159
8         Multifamily Housing             Natural Gas              -0.168528
9         Multifamily Housing                Site EUI              -0.743034
10 Non-Refrigerated Warehouse  Electricity Intensity               -0.719545
12 Non-Refrigerated Warehouse              Floor Area               0.125143
13 Non-Refrigerated Warehouse             Natural Gas              -0.221939
14 Non-Refrigerated Warehouse                Site EUI              -0.726648
15                     Office  Electricity Intensity               -0.705143
17                     Office              Floor Area               0.041398
18                     Office             Natural Gas              -0.141701
19                     Office                Site EUI              -0.784825
20   Residence Hall/Dormitory  Electricity Intensity               -0.727305
```

Fig7: Relation between Energy Star Score and three different measures by building type.

The table describes the correlation between Energy Star Score and Three different measures by building type. We see the following relationships: Energy Star Score is strongly negatively correlated with the Electricity Intensity and the site EUI. The strength of the natural gas correlation depends on the dataset, but natural gas usage tends to be negatively correlated with the Energy Star Score as well. The relationship between floor area and the Energy Star score is weak for all building types.

3.2.6 Remove Collinear Features

We want to eliminate features that are highly correlated with one another (not with the Energy Star Score) prior to developing a machine learning model. We don't want to use multiple collinear variables in our model because the goal is to see if independent variables can predict the Energy Star Score. Site EUI and Weather Normalized Site EUI, for instance, have a very high correlation (nearly 1.0), so we only need to incorporate one of these metrics into our model. Calculating the correlation coefficient between every pair of variables and eliminating those with a magnitude greater than a predetermined threshold is an easy way to eliminate collinear features. Here, this procedure is carried out with a correlation coefficient magnitude threshold of 0.5.

18 highly collinear columns were eliminated from the data by the procedure. The next thing we need to do is figure out if we can create machine learning models that can use these features to predict the Energy Star Score that we want to achieve.

4 Machine Learning Approach

Our main objective is to predict the Energy Star Score of the building with the help of the variables given in the data. As we go through the process first we separate the buildings that have score and those without, next step the buildings with the Energy Star Score, We use 25% for evaluating our model and 75% as training model. So we proceed with two different methods:

Regression: Using regression we predict the numerical value of the Energy Star Score. The metrices used include Mean Absolute Error, the average deviation between predictions and true values and R^2, the percentage of variation in the response.

Classification: Based on Energy Star Score intervals we divide the buildings into 5 groups and predict the class for a building. The metrices used include Accuracy, confusion matrix, f1_score.

We can try clustering on the buildings, as of now we stick to supervised machine learning methods. Unsupervised methods would be an interesting avenue for further.

4.1 Feature Pre-processing

As we needed the data for machine learning, we will follow some standard pre-processing steps:

Subset to 3 categorical columns and numerical columns. We focus on using the numeric values to predict the score.

One-hot encoding of categorical variables.

Split the data into training and testing sets. In which we will use 75% training and rest for testing

Next step is to impute the missing values. As there are many methods to imputation we going to follow with straightforward median values.

We will train the imputer only on the training data and make transformation of both the training data and testing data

At the end we return the values for training and testing for the model.

5. Regression of Energy Star Scores

In this stage we will train the model with over 7000 buildings using 77 features. The testing and training data have the same number of features which is a good sanity check.

5.1 Establishing a regression baseline

So we calculate a common baseline For establishing a regression baseline so if a model cannot be beaten this mark the Mission learning may not be approved for the task the single baseline is predicted with the help of the meanwhile you of the target in using the training data for all the testing. So, our baseline error is 25.9698. As with the baseline error is 25.9698 so our model cannot beat an average error of 26 then we'll need to rethink our approach.

5.2 Linear Regression

Now you did linear aggression is extremely straightforward and produces the good result but it is a good method to start a machine learning technique which helps us to interrupt with the results however it is not a linear So our model will not be very useful. The linear regression error is 27.9440. As we see the result which is not great so will for the move with the series of machine learning models so where will increase in the complexity it will increase in the complexity via our accuracy increases.

5.3 Random Forest Regression

Next, we will implement the model of random forest Regression. The random forest regression helps the variables to make more predictions. As we know the random forest suggestion is generally very accurate and performance well on nonlinear problems with many features because it does not implement the feature selection. The random forest error is around 10.6761. As we see the record on the random forest is around 10. And this sustainability is lower than the baseline we can conclude that the machine learning approved that we used to predict the energy star score is possible with the given data set. There's to the shoes our model can predict the energy start copies in the 10 points of the true value on average.

5.4 Interpret Regression Results

As we see the difference between the predicted distribution and actual distribution How model can predict the mode of score such 100 which does not mean Able to predict the loose course very well. This is the major problem we need to fix the imbalance for future work.

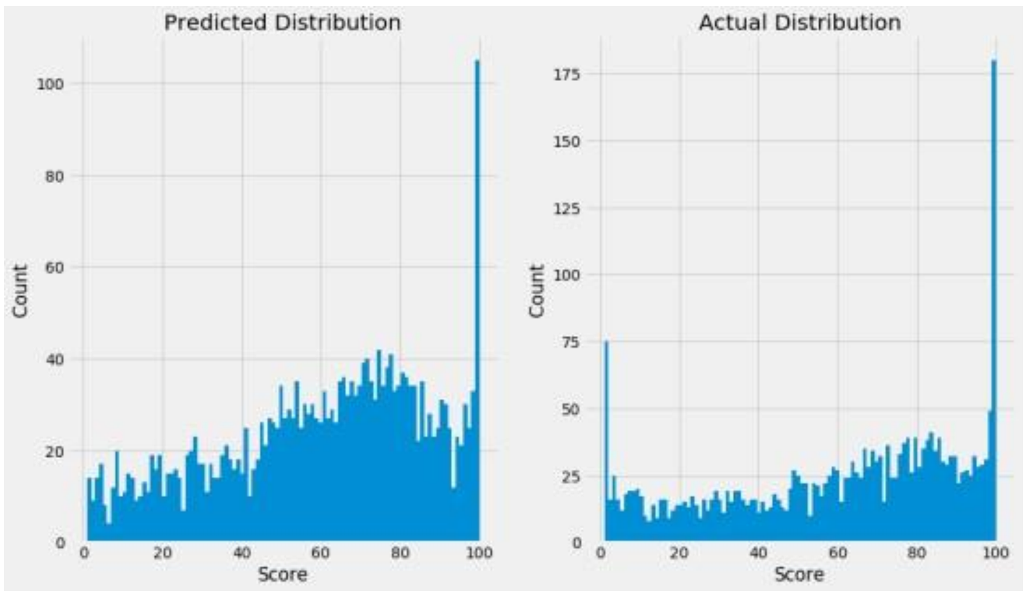


Fig 8: Predicted vs Actual Distribution

5.4.1 Predicted Scores of New Buildings

As we see the model to generate new predictions for the buildings with missing values. Now we make the predictions for the new building as long it has the measurements recorded for the original data now, we use the crowded model to infer this course and we estimate 10 points of the true values with the data we acquired from Fig 9.

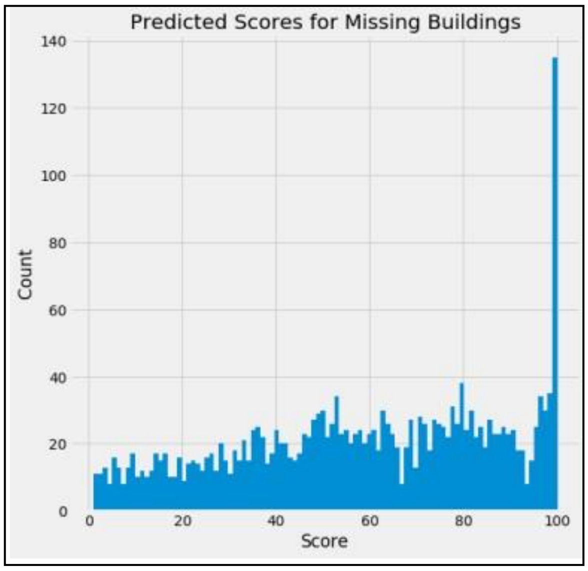


Fig 9: Predicted Scores for missing Buildings

5.4.2 Interpret Feature Importance

After understanding the random forest model, we look into the specific features he considered most important the absolute values of the feature importance are not as crucial as they lead to ranking of variables which tells us which variable are most useful for predicting the energy star score.

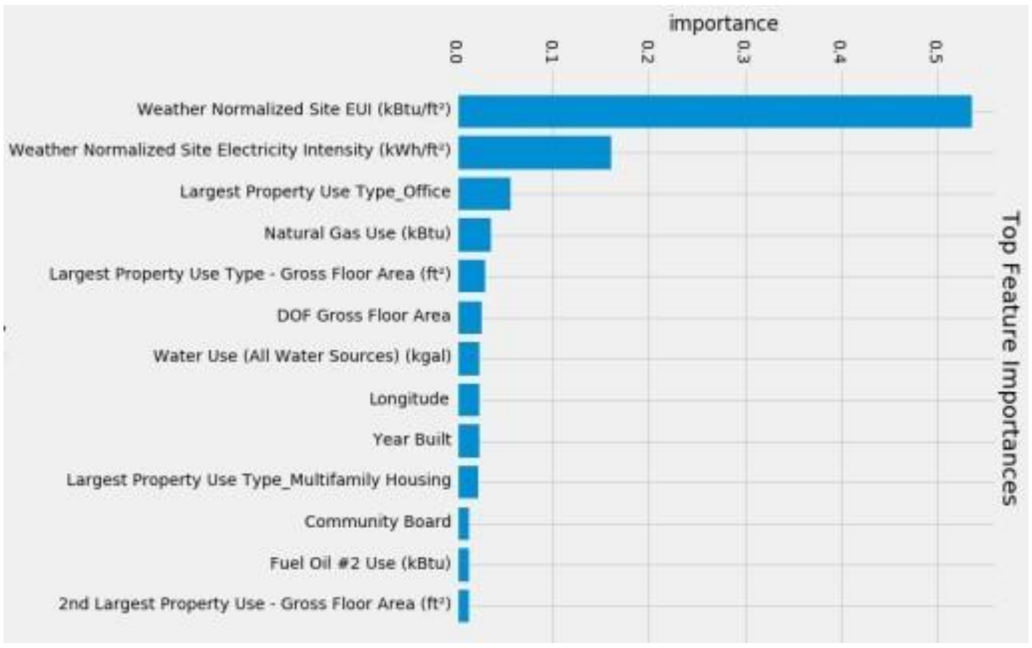


Fig 10: Top features among all the factors

According to the random forest model I'm born features for the predicting the energy star score of the building are electric intensity, EUI, Property used type, the natural gas in the area these variables come into the notification and help us to predict the energy star score Please notice. All these relations are closer to the log transformation of the variable taken we apply the transformation and look at the resulting relationships.

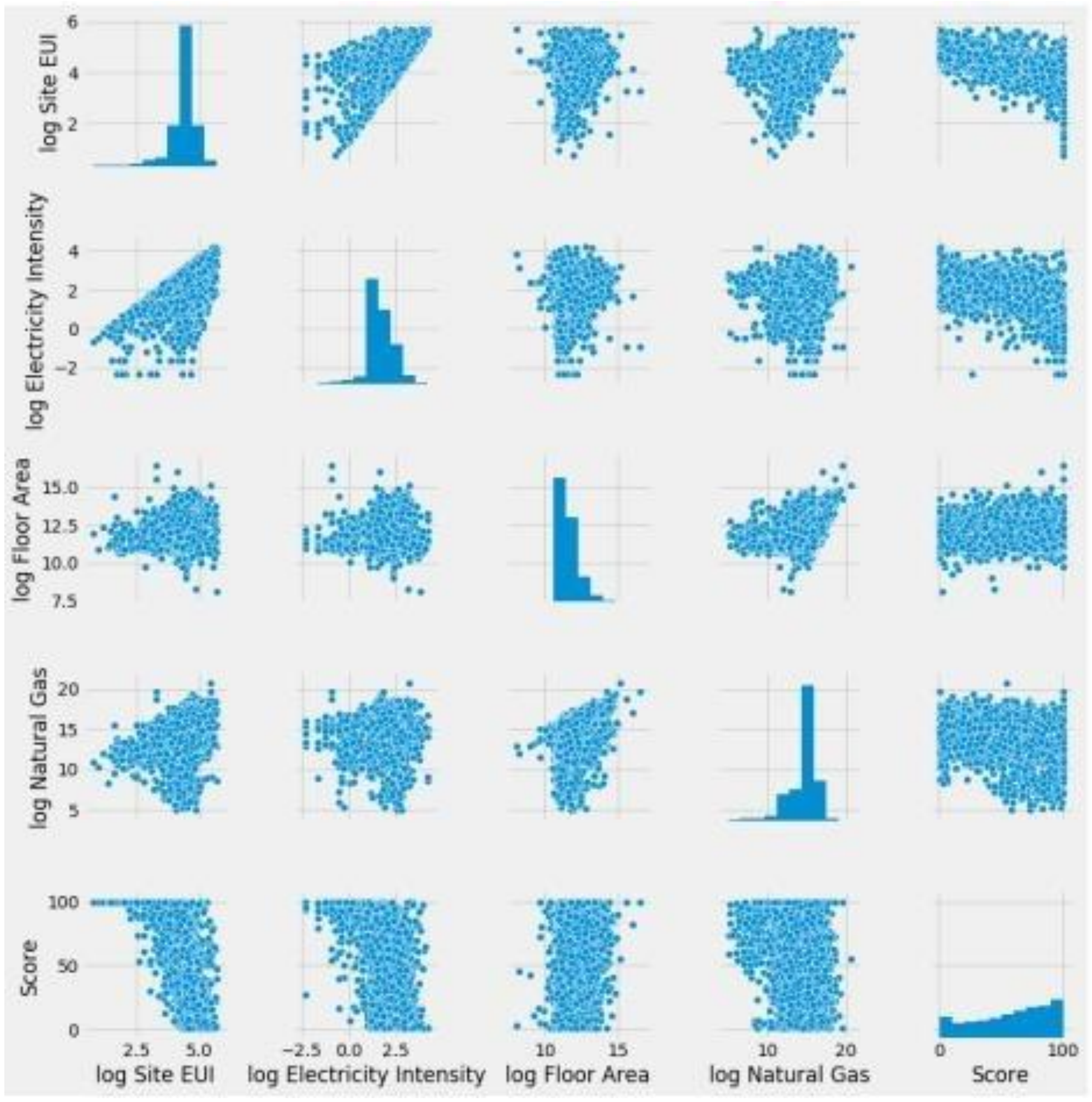


Fig 11: Regression on various variables

This graph is not that informative, but the overall result of the regression shows us that it is possible to interfere the energy star score of the building with the given data available. As the model result or not easy to explain but the important features in the line with those found to be correlated with the energy star score.

6. Classification of Energy Star Scores

In here we see the possibility of place building into classes based on their energy star score. Which is a simple grading scheme where every 20-point interval is a grade giving us 5 total.

i) *Classification Pre-processing:* We generate a training and testing dataset for classification with the 5 unique classes.

ii) *Random Forest Classifier:* We use the random forest classifier and make the prediction using the training and testing data and make sure the results have some accurate predictions.

iii) *Interpret Classification Results:* As we knew there are many metrices for classification and we have an accuracy and confusion matrix which is used to define performance of a classification algorithm. The weighted average parameter for the f1 score to account for class imbalances and this confusion matrix allows us to see the errors that our classifier makes. The accuracy is 0.59 and F1 score is 0.58.

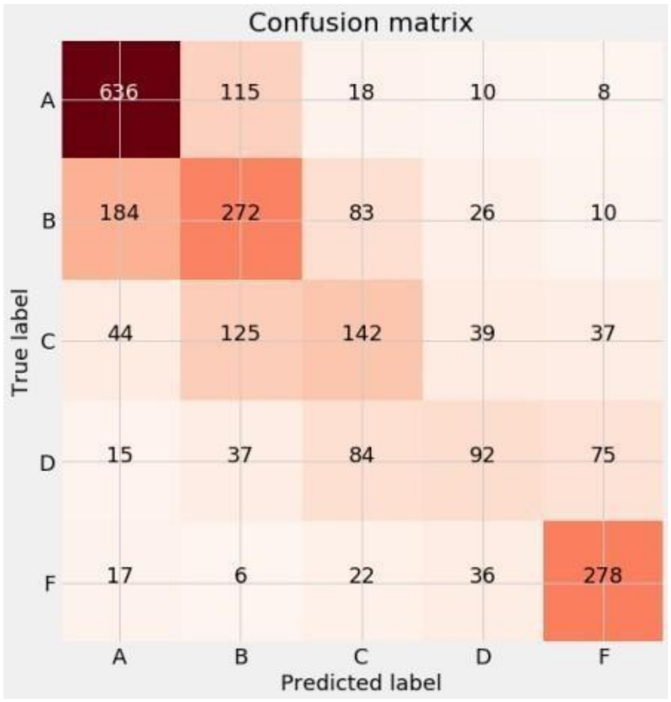


Fig 12: Prediction vs Accuracy using confusion matrix

We see the accuracy with the help of the straightforward metric and with the help of this our model can correctly choose among the five classes 60% of the time. The random forest classifier appears to do very well and can accurately infer the energy Star Score of the building if provided with the building data as in the figure 12.

6.1 Predicted Classification of New Data:

These classifiers can also be used to assign grades to buildings which were missing a score. We don't have the true values, but we at least look at the distribution of predictions of the buildings. Now, any new building can be classified by the model if it is provided with the given data and the prediction of the model as shown by the high performance on the testing data.

CODE:

Energy Star Score Prediction

In this project we are gonna predict the Energy start score of a building unsing Machine learning techniques.

we are working on a supervised, regression machine learning problem. Using real-world New York City building energy data, we want to predict the Energy Star Score of a building and determine the factors that in uence the score.

We are using the general outline of the machine learning pipeline to structure our project:

1. Data cleaning and formatting
2. Exploratory data analysis
3. Feature engineering and selection
4. Compare several machine learning models on a performance metric
5. Perform hyperparameter tuning on the best model to optimize it for the problem . Evaluate the best model on the testing set

Importing packages and setting a coloum limit

```
import pandas as pd
import numpy as np

pd.options.mode.chained_assignment = None
pd.set_option('display.max_columns', 60)

import matplotlib.pyplot as plt
%matplotlib inline

plt.rcParams['font.size'] = 24 from
IPython.core.pylabtools import figsize
import seaborn as sns sns.set(font_scale =
2)
from sklearn.model_selection import train_test_split
```

Loading the data and Examine

```
# Read in data into a dataframe data =
pd.read_csv('/content/Energy_and_Water_Data_Disclosure_for_Local_Law_84_2017__Data_for_Calendar_Year_2016_.csv')
data.head()
```

```
                                                                  NYC
                                                              Borough,
                                    Parent Parent NYC Building Address 1 Property Property BBL - 10 Block
          and
   Order          Property Property        Identification     (selfId   Name      digits Lot (BBL)
                                      Id       Name                                Number (BIN) reported)
                                                                      selfreported

                                                                                                  201/205
 0   1          13286     201/205    13286     201/205    1013160001          1013160001         1037549
     East 42nd
                                                                                                      st.

                          NYP                  NYP                                        1084198;
                       Columbia             Columbia                        1-02138-   1084387;1084385;    622 168th
 1       2      28400    28400       1021380040
                    (West (West 0040 1084386; Street Campus) Campus) 1084388; 10...

                                             NYP
                     MSCHoNY                Columbia                        1-02138-                          3975
 2       3      4778226  28400       1021380030         1063380 North          (West        0030        Broadway
                                            Campus)

                        Herbert
                                             NYP
                         Irving                                                                         161 Fort
                                            Columbia                        1-02139-
 3       4      4778267  Pavilion &  28400    1021390001              1087281; 1076746        Washington
                                             (West                            0001
                        Millstein                                                                            Ave
                                            Campus)
                        Hospital

                                             NYP
                                                                                                     710 West
                          Neuro             Columbia                        1-02139-
 4       5      4778288  28400       1021390085             1063403    168th Institute          (West       0085
                                                                                                      Street
                                            Campus)
```

Verifying the data types and checking missing values

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11746 entries, 0 to 11745 Data columns (total 60 columns):
 #   Column                                                       Non-Null Count  Dtype
---  ------                                                       --------------  -----
     0       Order                                                        11746 non-null  int64
     1       Property Id                                                  11746 non-null  int64
     2       Property Name                                                11746 non-null  object
     3       Parent Property Id                                           11746 non-null  object
     4       Parent Property Name                                         11746 non-null  object
     5       BBL - 10 digits                                              11735 non-null  object
     6       NYC Borough, Block and Lot (BBL) self-reported               11746 non-null  object
     7       NYC Building Identification Number (BIN)                     11746 non-null  object
     8       Address 1 (self-reported)                                    11746 non-null  object
     9       Address 2                                                    11746 non-null  object
     10      Postal Code                                                  11746 non-null  object
     11      Street Number                                                11622 non-null  object
     12      Street Name                                                  11624 non-null  object
     13      Borough                                                      11628 non-null  object
     14      DOF Gross Floor Area                                         11628 non-null  float64
     15      Primary Property Type - Self Selected                        11746 non-null  object
     16      List of All Property Use Types at Property                   11746 non-null  object
     17      Largest Property Use Type                                    11746 non-null  object
     18      Largest Property Use Type - Gross Floor Area (ft²)           11746 non-null  object
     19      2nd Largest Property Use Type                                11746 non-null  object
     20      2nd Largest Property Use - Gross Floor Area (ft²)            11746 non-null  object
     21      3rd Largest Property Use Type                                11746 non-null  object
     22      3rd Largest Property Use Type - Gross Floor Area (ft²)       11746 non-null  object
     23      Year Built                                                   11746 non-null  int64
     24      Number of Buildings - Self-reported                          11746 non-null  int64
     25      Occupancy                                                    11746 non-null  int64
     26      Metered Areas (Energy)                                       11746 non-null  object
     27      Metered Areas  (Water)                                       11746 non-null  object
     28      ENERGY STAR Score                                            11746 non-null  object
     29      Site EUI (kBtu/ft²)                                          11746 non-null  object
     30      Weather Normalized Site EUI (kBtu/ft²)                       11746 non-null  object
     31      Weather Normalized Site Electricity Intensity (kWh/ft²)      11746 non-null  object
     32      Weather Normalized Site Natural Gas Intensity (therms/ft²)   11746 non-null  object
     33      Weather Normalized Source EUI (kBtu/ft²)                     11746 non-null  object
     34      Fuel Oil #1 Use (kBtu)                                       11746 non-null  object
     35      Fuel Oil #2 Use (kBtu)                                       11746 non-null  object
     36      Fuel Oil #4 Use (kBtu)                                       11746 non-null  object
     37      Fuel Oil #5 & 6 Use (kBtu)                                   11746 non-null  object
     38      Diesel #2 Use (kBtu)                                         11746 non-null  object
     39      District Steam Use (kBtu)                                    11746 non-null  object
     40      Natural Gas Use (kBtu)                                       11746 non-null  object
     41      Weather Normalized Site Natural Gas Use (therms)             11746 non-null  object
     42      Electricity Use - Grid Purchase (kBtu)                       11746 non-null  object
     43      Weather Normalized Site Electricity (kWh)                    11746 non-null  object
     44      Total GHG Emissions (Metric Tons CO2e)                       11746 non-null  object
     45      Direct GHG Emissions (Metric Tons CO2e)                      11746 non-null  object
     46      Indirect GHG Emissions (Metric Tons CO2e)                    11746 non-null  object
     47      Property GFA - Self-Reported (ft²)                           11746 non-null  int64
     48      Water Use (All Water Sources) (kgal)                         11746 non-null  object
     49      Water Intensity (All Water Sources) (gal/ft²)                11746 non-null  object
     50      Source EUI (kBtu/ft²)                                        11746 non-null  object
     51      Release Date                                                 11746 non-null  object
     52      Water Required?                                              11628 non-null  object
```

replacing the missing values with NA and converting data to correct types

```
data = data.replace({'Not Available': np.nan})

for col in list(data.columns):
    if ('ft²' in col or 'kBtu' in col or 'Metric Tons CO2e' in col or 'kWh' in
col or 'therms' in col or 'gal' in col or 'Score' in col):
        # Convert the data type to float
data[col] = data[col].astype(float)
```

```
data.describe() #Statistics for each column
```

	Order	Property Id	DOF Gross Floor Area	Largest Property Use Type - Gross Floor Area (ft²)	2nd Largest Property Use - Gross Floor Area (ft²)	3rd Largest Property Use Type - Gross Floor Area (ft²)	Year Built
count	11746.000000	1.174600e+04	1.162800e+04	1.174400e+04	3741.000000	1484.000000	11746.000000

mean	7185.759578	3.642958e+06	1.732695e+05	1.605524e+05	22778.682010	12016.825270	1948.738379
std	4323.859984	1.049070e+06	3.367055e+05	3.095746e+05	55094.441422	27959.755486	30.576386
min	1.000000	7.365000e+03	5.002800e+04	5.400000e+01	0.000000	0.000000	1600.000000
25%	3428.250000	2.747222e+06	6.524000e+04	6.520100e+04	4000.000000	1720.750000	1927.000000
50%	6986.500000	3.236404e+06	9.313850e+04	9.132400e+04	8654.000000	5000.000000	1941.000000
75%	11054.500000	4.409092e+06	1.596140e+05	1.532550e+05	20000.000000	12000.000000	1966.000000
max	14993.000000	5.991312e+06	1.354011e+07	1.421712e+07	962428.000000	591640.000000	2019.000000

Checking missing values by column

```
def missing_values_table(df):
        mis_val = df.isnull().sum()
                mis_val_percent = 100 * df.isnull().sum() / len(df) #percentage of
missing values
                mis_val_table = pd.concat([mis_val, mis_val_percent],
axis=1)
                mis_val_table_ren_columns = mis_val_table.rename(
columns = {0 : 'Missing Values', 1 : '% of Total Values'})
                mis_val_table_ren_columns =
mis_val_table_ren_columns[
mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
                # Print some summary information         print ("Your selected
dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")
                # Return the dataframe with missing
information         return mis_val_table_ren_columns


missing_values_table(data)
```

```
Your selected dataframe has 60 columns.
There are 46 columns that have missing values.
```

	Missing Values	% of Total Values
Fuel Oil #1 Use (kBtu)	11737	99.9
Diesel #2 Use (kBtu)	11730	99.9
Address 2	11539	98.2
Fuel Oil #5 & 6 Use (kBtu)	11152	94.9
District Steam Use (kBtu)	10810	92.0
Fuel Oil #4 Use (kBtu)	10425	88.8
3rd Largest Property Use Type - Gross Floor Area (ft²)	10262	87.4
3rd Largest Property Use Type	10262	87.4

if a column has a high percentage of missing values, then it probably will not be of much use.

we will remove any columns with more than 50% missing values. In general, be careful about dropping any information because even if it is not there for all the observations, it may still be useful for predicting the target value.

```
# Get the columns with > 50% missing
missing_df = missing_values_table(data);
missing_columns = list(missing_df[missing_df['% of Total Values'] > 50].index)
print('We will remove %d columns.' % len(missing_columns))
```

```
Your selected dataframe has 60 columns.
There are 46 columns that have missing values.
We will remove 11 columns.
```

The rest of the missing values will have to be imputed (filled-in) using an appropriate strategy before doing machine learning.

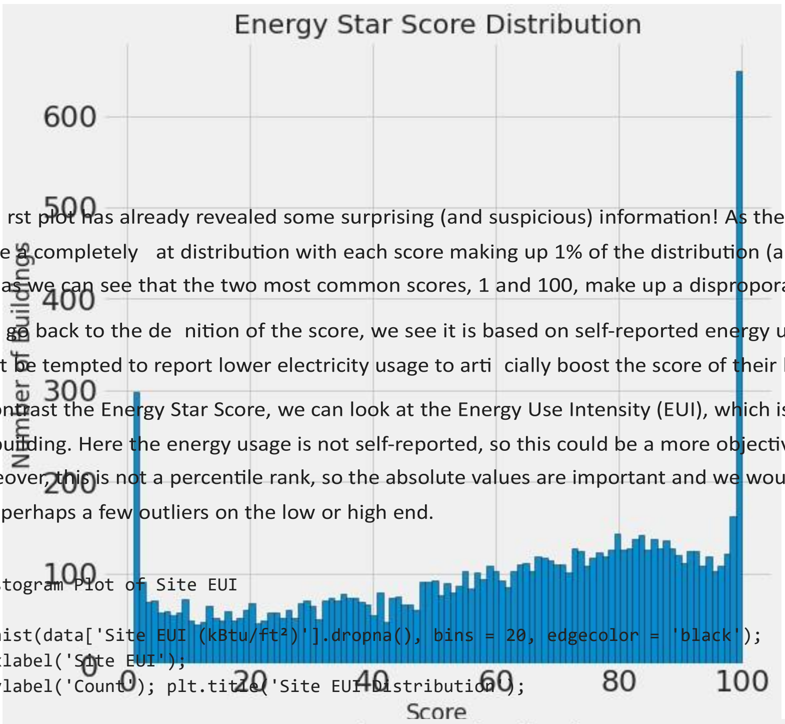
Census Tract	2263	19.3

```
data = data.drop(columns = list(missing_columns))
print('we have remvoved 11 colums from the data set')
```

```
we have remvoved 11 colums from the data set
```

Exploratory Data Analysis (EDA) is an open-ended process where we make plots and calculate statistics in order to explore our data. The purpose is to to find anomalies, patterns, trends, or relationships. These may be interesting by themselves (for example finding a correlation between two variables) or they can be used to inform modeling decisions such as which features to use. In short, the goal of EDA is to determine what our data can tell us! EDA generally starts out with a high-level overview, and then narrows in to specific parts of the dataset once as we find interesting areas to examine.

```
figsize(8, 8)

# Rename the score
data = data.rename(columns = {'ENERGY STAR Score': 'score'})

# Histogram of the Energy Star Score
plt.style.use('fivethirtyeight')
plt.hist(data['score'].dropna(), bins = 100, edgecolor = 'k');
plt.xlabel('Score'); plt.ylabel('Number of Buildings');
plt.title('Energy Star Score Distribution');
```



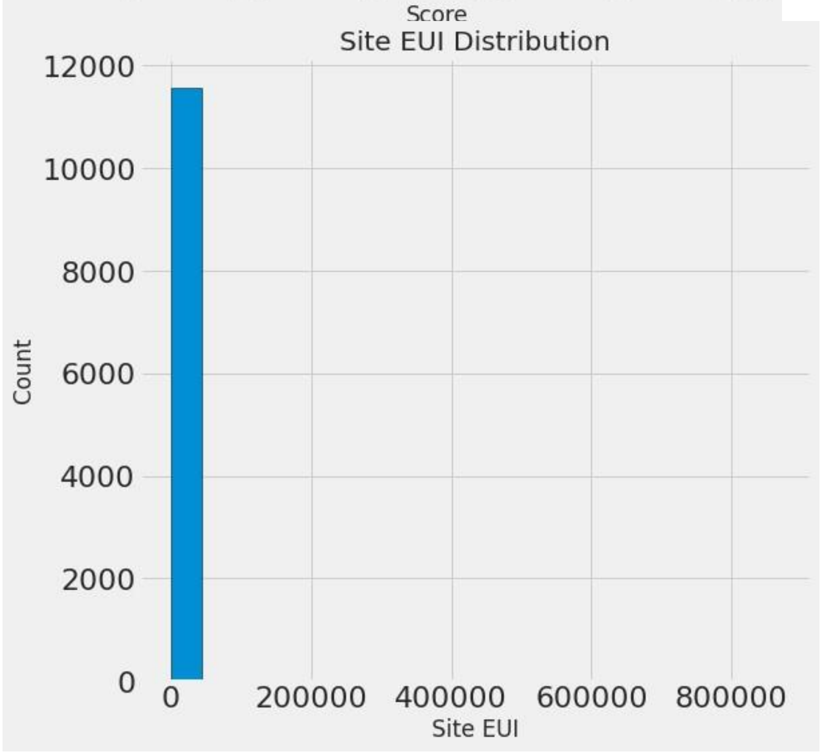
Our rst plot has already revealed some surprising (and suspicious) information! As the Energy Star Score is a percentile rank, we would expect to see a completely at distribution with each score making up 1% of the distribution (about 90 buildings). However, this is de nitely not the case as we can see that the two most common scores, 1 and 100, make up a disproporationate number of the overall scores.

If we go back to the de nition of the score, we see it is based on self-reported energy usage. This poses a problem, because a building owner might be tempted to report lower electricity usage to arti cially boost the score of their building.

To contrast the Energy Star Score, we can look at the Energy Use Intensity (EUI), which is the total energy use divided by the square footage of the building. Here the energy usage is not self-reported, so this could be a more objective measure of the energy e

Moreover, this is not a percentile rank, so the absolute values are important and we would expect them to be approximately normally distributed with perhaps a few outliers on the low or high end.

```
# Histogram Plot of Site EUI

plt.hist(data['Site EUI (kBtu/ft²)'].dropna(), bins = 20, edgecolor = 'black');
plt.xlabel('Site EUI');
plt.ylabel('Count'); plt.title('Site EUI Distribution');
```



ciency of a building.

```
figsize(8, 8)
```

Well this shows us we have another problem: outliers! The graph is incredibly skewed because of the presence of a few buildings with very high scores. It looks like we will have to take a slight detour to deal with the outliers. Let's look at the stats for this feature.

```
data['Site EUI (kBtu/ft²)'].describe()
```

```
count     11583.000000
mean        280.071484
std        8607.178877
min           0.000000
25%          61.800000
50%          78.500000 75%
97.600000 max      869265.000000 Name:
Site EUI (kBtu/ft²), dtype: float64
```

```
data['Site EUI (kBtu/ft²)'].dropna().sort_values().tail(10)
```

```
3173     51328.8
3170     51831.2
3383     78360.1
8269     84969.6
3263     95560.2
8268    103562.7
8174    112173.6
3898    126307.4
7       143974.4
8068    869265.0
Name: Site EUI (kBtu/ft²), dtype: float64
```

```
data.loc[data['Site EUI (kBtu/ft²)'] == 869265, :]
```

```
                                                                 NYC
                                                             Borough,
                                  Parent         Parent   NYC Building Address Property Property         BBL
                 - 10 Block and
         Order                  Property   Property                         Identification     (self
                   Id      Name                                digits Lot (BBL)
                                      Id       Name                            Number (BIN) reported
                                                                      selfreported
________________________________________________________________________________________

                                     Not          Not
                                                                                                  23
                          234   Applicable:   Applicable:                           3390250 ,
8068 9984 4414323 3028937502 3028937502 Skillma Skillman Standalone Standalone 3338313
                                                                                               avenu
                                   Property Property
```

Removing Outliers

Outliers can occur for many reasons: typos, malfunctions in measuring devices, incorrect units, or they can be legitimate but extreme values.
Outliers can throw off a model because they are not indicative of the actual distribution of data.

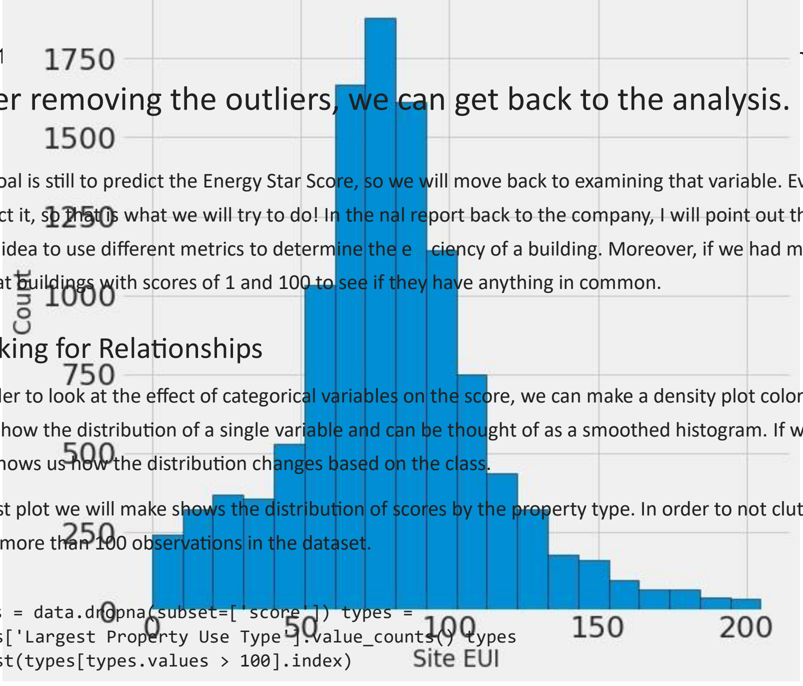
When we remove outliers, we want to be careful that we are not throwing away measurements just because they look strange. They may be the result of actual phenomenon that we should further investigate. When removing outliers, I try to be as conservative as possible, using the de nition of an extreme outlier:

- On the low end, an extreme outlier is below $\text{First Quartile} - 3 * \text{Interquartile Range}$
- On the high end, an extreme outlier is above $\text{Third Quartile} + 3 * \text{Interquartile Range}$

In this case, I will only remove the single outlying point and see how the distribution looks.

```
first_quartile = data['Site EUI (kBtu/ft²)'].describe()['25%']

third_quartile = data['Site EUI (kBtu/ft²)'].describe()['75%']

iqr = third_quartile - first_quartile

data = data[(data['Site EUI (kBtu/ft²)'] > (first_quartile - 3 * iqr)) &
(data['Site EUI (kBtu/ft²)'] < (third_quartile + 3 * iqr))]
```

```
# Histogram Plot of Site EUI
figsize(8, 8)
plt.hist(data['Site EUI (kBtu/ft²)'].dropna(), bins = 20, edgecolor = 'black');
plt.xlabel('Site EUI'); plt.ylabel('Count'); plt.title('Site EUI
Distribution');
```



After removing the outliers, we can get back to the analysis.

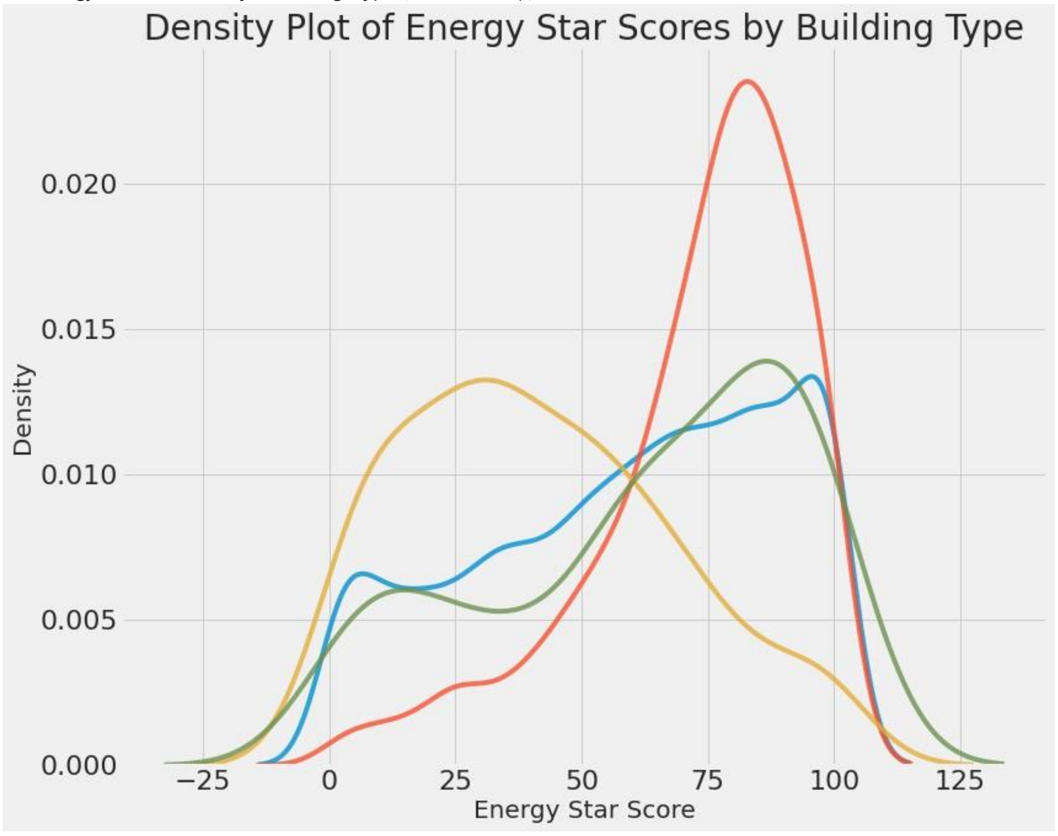
our goal is still to predict the Energy Star Score, so we will move back to examining that variable. Even if the score is not a good measure, it is still our task to predict it, so that is what we will try to do! In the nal report back to the company, I will point out this might not be an objective measure, and it would be a good idea to use different metrics to determine the e ciency of a building. Moreover, if we had more time for this project, it might be interesting to take a look at buildings with scores of 1 and 100 to see if they have anything in common.

Looking for Relationships

In order to look at the effect of categorical variables on the score, we can make a density plot colored by the value of the categorical variable. Density plots also show the distribution of a single variable and can be thought of as a smoothed histogram. If we color the density curves by a categorical variable, this will shows us how the distribution changes based on the class.

The rst plot we will make shows the distribution of scores by the property type. In order to not clutter the plot, we will limit the graph to building types that have more than 100 observations in the dataset.

```
types = data.dropna(subset=['score']) types =
types['Largest Property Use Type'].value_counts() types
= list(types[types.values > 100].index)


figsize(12, 10)

for b_type in types:
    subset = data[data['Largest Property Use Type'] == b_type]


    sns.kdeplot(subset['score'].dropna(),
                label = b_type, shade = False, alpha = 0.8);


plt.xlabel('Energy Star Score', size = 20); plt.ylabel('Density', size = 20); plt.title('Density
Plot of Energy Star Scores by Building Type', size = 28);
```

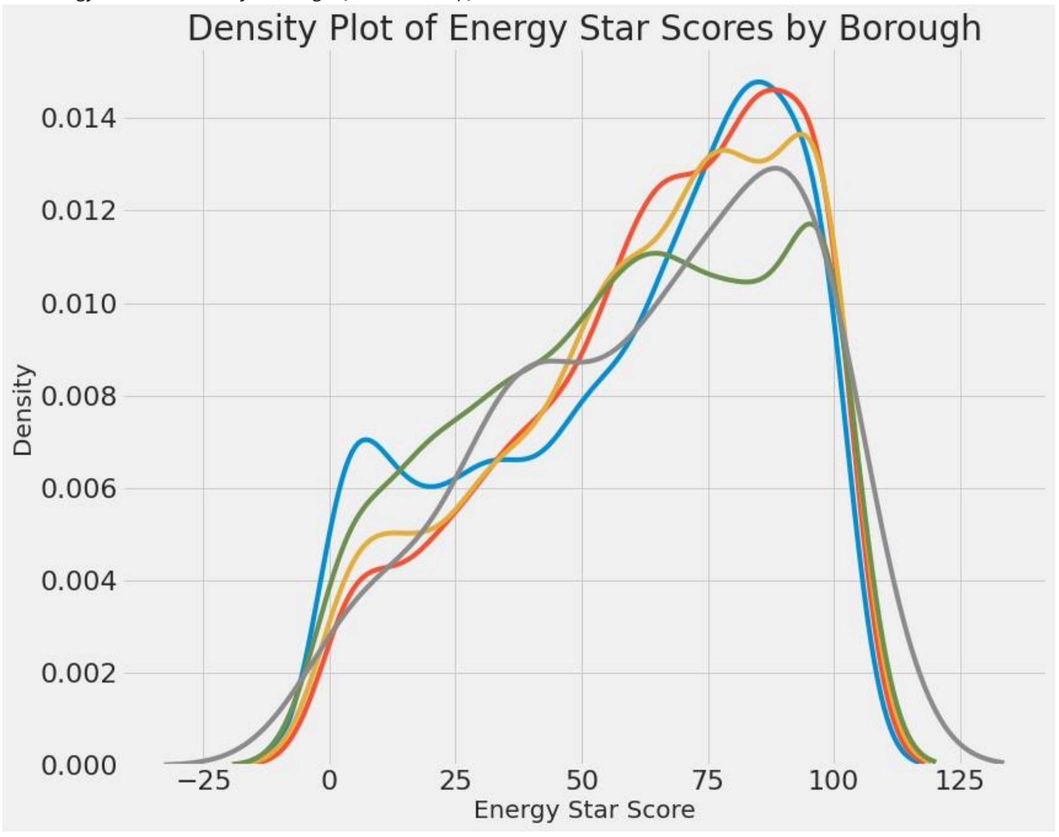


this graph tells us that we should include the property type because this information can be useful for determining the score. As building type is a categorical variable, it will have to be one-hot encoded before we can feed it into a machine learning model.

To examine another categorical variable, borough, we can make the same graph, but this time colored by the borough.

```
boroughs = data.dropna(subset=['score']) boroughs =
boroughs['Borough'].value_counts() boroughs =
list(boroughs[boroughs.values > 100].index)
```

```
figsize(12, 10) for
borough in boroughs:
    subset = data[data['Borough'] == borough]


sns.kdeplot(subset['score'].dropna(),
label = borough);
plt.xlabel('Energy Star Score', size = 20); plt.ylabel('Density', size = 20); plt.title('Density
Plot of Energy Star Scores by Borough', size = 28);
```



The borough of the building does not seem to make as signi cant a difference in the distribution of the score as does the building type. Nonetheless, it might make sense to include the borough as a categorical variable.

Correlations between Features and Target

In order to quantify correlations between the features (variables) and the target, we can calculate the Pearson correlation coe cient. This is a measure of the strength and direction of a linear relationship between two variables: a value of -1 means the two variables are perfectly negatively linearly correlated and a value of +1 means the two variables are perfectly positively linearly correlated. The gure below shows different values of the correlation coe ent and how they appear graphically.

Although there can be non-linear relationships between the features and targets and correlation coe cients do not account for interactions between features, linear relationships are a good way to start exploring trends in the data. We can then use these values for selecting the features to employ in our model.

The code below calculates the correlation coe cients between all the variables and the score.

```
correlations_data = data.corr()['score'].sort_values()

print(correlations_data.head(15), '\n')

print(correlations_data.tail(15))
```

```
    Site EUI (kBtu/ft²)                                          -0.723864
    Weather Normalized Site EUI (kBtu/ft²)                       -0.713993
    Weather Normalized Source EUI (kBtu/ft²)                     -0.645542
    Source EUI (kBtu/ft²)                                        -0.641037
    Weather Normalized Site Electricity Intensity (kWh/ft²)      -0.358394
    Weather Normalized Site Natural Gas Intensity (therms/ft²)   -0.346046
    Direct GHG Emissions (Metric Tons CO2e)                      -0.147792
    Weather Normalized Site Natural Gas Use (therms)             -0.135211
    Natural Gas Use (kBtu)                                       -0.133648
    Year Built                                                   -0.121249
```

```
Total GHG Emissions (Metric Tons CO2e)                      -0.113136
Electricity Use - Grid Purchase (kBtu)                      -0.050639
Weather Normalized Site Electricity (kWh)                   -0.048207
Latitude                                                    -0.048196
Property Id                                                 -0.046605
Name: score, dtype: float64

Property Id                                         -0.046605
Indirect GHG Emissions (Metric Tons CO2e)           -0.043982
Longitude                                           -0.037455
Occupancy                                           -0.033215
Number of Buildings - Self-reported                 -0.022407
Water Use (All Water Sources) (kgal)                -0.013681
Water Intensity (All Water Sources) (gal/ft²)       -0.012148
Census Tract                                        -0.002299
DOF Gross Floor Area                                 0.013001
Property GFA - Self-Reported (ft²)                   0.017360
Largest Property Use Type - Gross Floor Area (ft²)   0.018330
Order                                                0.036827
Community Board                                      0.056612
Council District                                     0.061639
score                                                1.000000
Name: score, dtype: float64
```

There are several strong negative correlations between the features and the target. The most negative correlations with the score are the different categories of Energy Use Intensity (EUI), Site EUI (kBtu/ft²) and Weather Normalized Site EUI (kBtu/ft²) (these vary slightly in how they are calculated). The EUI is the amount of energy used by a building divided by the square footage of the buildings and is meant to be a measure of the e ciency of a building with a lower score being better. Intuitively, these correlations then make sense: as the EUI increases, the Energy Star Score tends to decrease.

To account for possible non-linear relationships, we can take square root and natural log transformations of the features and then calculate the correlation coe cients with the score. To try and capture any possible relationships between the borough or building type (remember these are categorical variables) and the score we will have to one-hot encode these columns.

In the following code, we take log and square root transformations of the numerical variables, one-hot encode the two selected categorical variables (building type and borough), calculate the correlations between all of the features and the score, and display the top 15 most positive and top 15 most negative correlations. This is a lot, but with pandas, it is straightforward to do each step!

```
# Select the numeric columns numeric_subset
= data.select_dtypes('number')

# Create columns with square root and log of numeric
columns for col in numeric_subset.columns:     # Skip the
Energy Star Score column     if col == 'score':
        next
else:
        numeric_subset['sqrt_' + col] = np.sqrt(numeric_subset[col])
numeric_subset['log_' + col] = np.log(numeric_subset[col])

# Select the categorical columns categorical_subset =
data[['Borough', 'Largest Property Use Type']]

# One hot encode
categorical_subset = pd.get_dummies(categorical_subset)

# Join the two dataframes using concat
# Make sure to use axis = 1 to perform a column bind features =
pd.concat([numeric_subset, categorical_subset], axis = 1)

# Drop buildings without an energy star score
features = features.dropna(subset = ['score'])

# Find correlations with the score
correlations = features.corr()['score'].dropna().sort_values()
```

```
/usr/local/lib/python3.8/dist-packages/pandas/core/arraylike.py:364: RuntimeWarning: divide by zero encountered in log
result = getattr(ufunc, method)(*inputs, **kwargs)
/usr/local/lib/python3.8/dist-packages/pandas/core/arraylike.py:364: RuntimeWarning: invalid value encountered in sqrt
result = getattr(ufunc, method)(*inputs, **kwargs)
/usr/local/lib/python3.8/dist-packages/pandas/core/arraylike.py:364: RuntimeWarning: invalid value encountered in log
result = getattr(ufunc, method)(*inputs, **kwargs)
```

```
# Display most negative correlations
correlations.head(15)
```

```
Site EUI (kBtu/ft²)                                        -0.723864
Weather Normalized Site EUI (kBtu/ft²)                     -0.713993
sqrt_Site EUI (kBtu/ft²)                                   -0.699817
sqrt_Weather Normalized Site EUI (kBtu/ft²)                -0.689019
sqrt_Weather Normalized Source EUI (kBtu/ft²)              -0.671044
sqrt_Source EUI (kBtu/ft²)                                 -0.669396
Weather Normalized Source EUI (kBtu/ft²)                   -0.645542
Source EUI (kBtu/ft²)                                      -0.641037
```

```
     log_Source EUI (kBtu/ft²)                                     -0.622892
     log_Weather Normalized Source EUI (kBtu/ft²)                  -0.620329
     log_Site EUI (kBtu/ft²)                                       -0.612039
     log_Weather Normalized Site EUI (kBtu/ft²)                    -0.601332
     log_Weather Normalized Site Electricity Intensity (kWh/ft²)   -0.424246
     sqrt_Weather Normalized Site Electricity Intensity (kWh/ft²)  -0.406669
     Weather Normalized Site Electricity Intensity (kWh/ft²)       -0.358394
     Name: score, dtype: float64
```

```
# Display most positive correlations
correlations.tail(15)
```

```
     sqrt_Order                                                        0.028662
     Borough_Queens                                                    0.029545
     Largest Property Use Type_Supermarket/Grocery Store               0.030038
     Largest Property Use Type_Residence Hall/Dormitory                0.035407
     Order                                                             0.036827
     Largest Property Use Type_Hospital (General Medical & Surgical)   0.048410
     Borough_Brooklyn                                                  0.050486
     log_Community Board                                               0.055495
     Community Board                                                   0.056612
     sqrt_Community Board                                              0.058029
     sqrt_Council District                                             0.060623
     log_Council District                                              0.061101
     Council District                                                  0.061639
     Largest Property Use Type_Office                                  0.158484
     score                                                             1.000000
     Name: score, dtype: float64
```

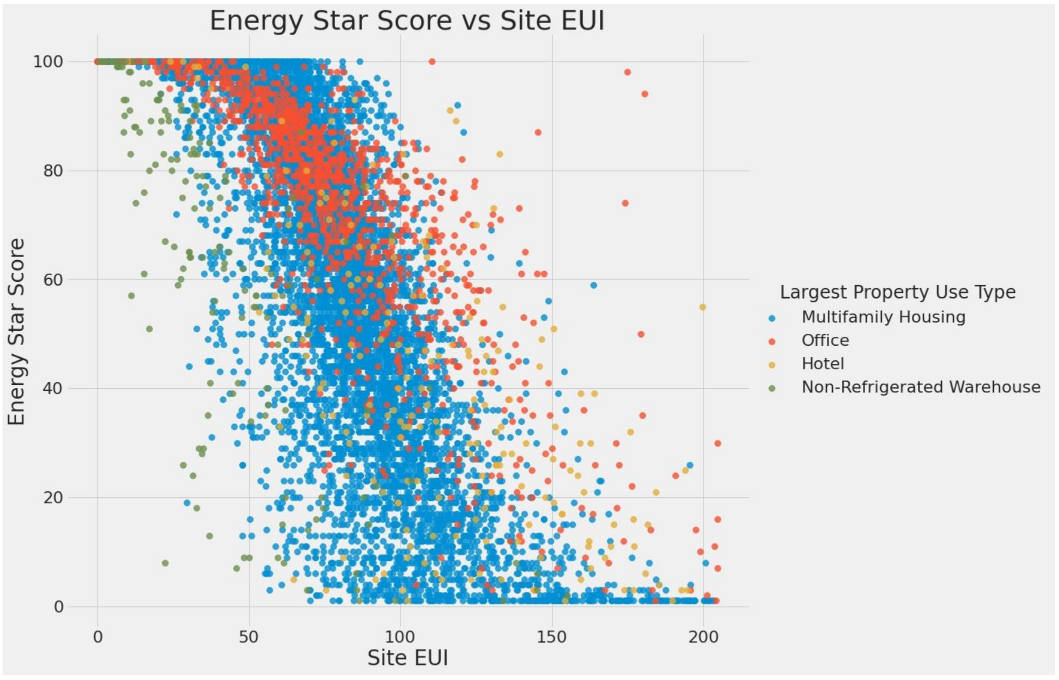
After transforming the features, the strongest relationships are still those related to Energy Use Intensity (EUI). The log and square root transformations do not seem the have resulted in any stronger relationships. There are no strong positive linear relationships although we do see that a building type of o
ce (Largest Property Use Type_O ce) is slightly positively correlated with the score. This variable is a one-hot encoded representation of the categorical variables for building type.

We can use these correlations in order to perform feature selection (coming up in a little bit). Right now, let's graph the most signi cant correlation (in terms of absolute value) in the dataset which is Site EUI (kBtu/ft^2). We can color the graph by the building type to show how that affects the relationship.

Two-Variable Plots

In order to visualize the relationship between two variables, we use a scatterplot. We can also include additional variables using aspects such as color of the markers or size of the markers. Here we will plot two numeric variables against one another and use color to represent a third categorical variable.

```
figsize(12, 10)

# Extract the building types features['Largest Property Use Type'] = data.dropna(subset =
['score'])['Largest Property Use Type']

 # Limit to building types with more than 100 observations (from previous code)
features = features[features['Largest Property Use Type'].isin(types)]

# Use seaborn to plot a scatterplot of Score vs Log Source EUI
sns.lmplot('Site EUI (kBtu/ft²)', 'score',           hue =
'Largest Property Use Type', data = features,
scatter_kws = {'alpha': 0.8, 's': 60}, fit_reg = False,
size = 12, aspect = 1.2);

# Plot labeling plt.xlabel("Site EUI", size = 28)
plt.ylabel('Energy Star Score', size = 28)
plt.title('Energy Star Score vs Site EUI', size =
36); /usr/local/lib/python3.8/dist-
packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following var   warnings.warn(
     /usr/local/lib/python3.8/dist-packages/seaborn/regression.py:581: UserWarning: The `size` parameter has
     warnings.warn(msg, UserWarning)
```



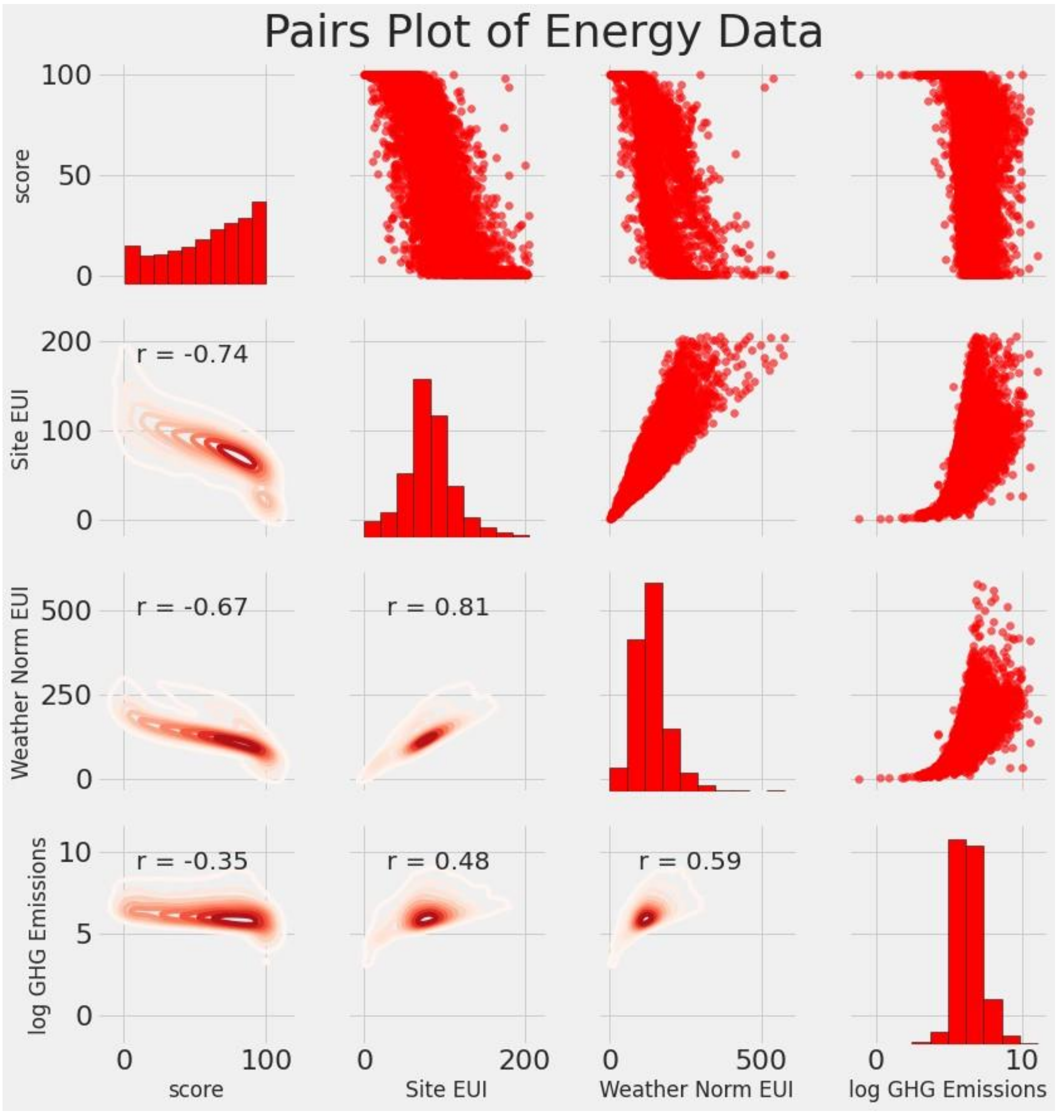
There is a clear negative relationship between the Site EUI and the score. The relationship is not perfectly linear (it looks with a correlation coe cient of -0.7, but it does look like this feature will be important for predicting the score of a building.

Pairs Plot

As a nal exercise for exploratory data analysis, we can make a pairs plot between several different variables. The Pairs Plot is a great way to examine many variables at once as it shows scatterplots between pairs of variables and histograms of single variables on the diagonal.

Using the seaborn PairGrid function, we can map different plots on to the three aspects of the grid. The upper triangle will have scatterplots, the diagonal will show histograms, and the lower triangle will show both the correlation coe cient between two variables and a 2-D kernel density estimate of the two variables.

```
# Extract the columns to  plot
plot_data = features[['score', 'Site EUI (kBtu/ft²)',
                      'Weather Normalized Source EUI (kBtu/ft²)',
                      'log_Total GHG Emissions (Metric Tons CO2e)']]

# Replace the inf with nan
plot_data = plot_data.replace({np.inf: np.nan, -np.inf: np.nan})

# Rename columns plot_data = plot_data.rename(columns = {'Site EUI
(kBtu/ft²)': 'Site EUI',
                                        'Weather Normalized Source EUI (kBtu/ft²)': 'Weather Norm EUI',
                                        'log_Total GHG Emissions (Metric Tons CO2e)': 'log GHG Emissions'})

# Drop na values
plot_data = plot_data.dropna()

def corr_func(x, y, **kwargs):     r =
np.corrcoef(x, y)[0][1]     ax = plt.gca()
ax.annotate("r = {:.2f}".format(r),
xy=(.2, .8), xycoords=ax.transAxes,
size = 20)

grid = sns.PairGrid(data = plot_data, size = 3)
grid.map_upper(plt.scatter, color = 'red', alpha = 0.6)
grid.map_diag(plt.hist, color = 'red', edgecolor = 'black')
grid.map_lower(corr_func);
grid.map_lower(sns.kdeplot, cmap = plt.cm.Reds)

# Title for entire plot plt.suptitle('Pairs Plot of Energy
Data', size = 36, y = 1.02); /usr/local/lib/python3.8/dist-
packages/seaborn/axisgrid.py:1209: UserWarning: The `size`
parameter has   warnings.warn(UserWarning(msg))
```



To interpret the relationships in the plot, we can look for where the variables in one row intersect with the variables in one column. For example, to nd the relationship between score and the log of GHG Emissions, we look at the score column and nd the log GHG Emissions row. At the intersection (the lower left plot) we see that the score has a -0.35 correlation coe cient with this varible. If we look at the upper right plot, we can see a scatterplot of this relationship.

Feature Engineering and Selection

Now that we have explored the trends and relationships within the data, we can work on engineering a set of features for our models. We can use the results of the EDA to inform this feature engineering. In particular, we learned the following from EDA which can help us in engineering/selecting features:

The score distribution varies by building type and to a lesser extent by borough. Although we will focus on numerical features, we should also include these two categorical features in the model. Taking the log transformation of features does not result in signi cant increases in the linear correlations between features and the score Before we get any further, we should de ne what feature engineering and selection are!
These de nitions are informal and have considerable overlap, but I like to think of them as two separate processes:

Feature Engineering: The process of taking raw data and extracting or creating new features that allow a machine learning model to learn a mapping beween these features and the target. This might mean taking transformations of variables, such as we did with the log and square root, or one-hot encoding categorical variables so they can be used in a model. Generally, I think of feature engineering as adding additional features derived from the raw data.
Feature Selection: The process of choosing the most relevant features in your data. "Most relevant" can depend on many factors, but it might be something as simple as the highest correlation with the target, or the features with the most variance. In feature selection, we remove features that do not help our model learn the relationship between features and the target. This can help the model generalize better to new data and results in a more interpretable model. Generally, I think of feature selection as subtracting features so we are left with only those that are most important. Feature engineering and selection are iterative processes that will usually require several attempts to get right. Often we will use the results of modeling, such as the feature importances from a random forest, to go back and redo feature selection, or we might later discover relationships that necessitate creating new variables. Moreover, these processes usually incorporate a mixture of domain knowledge and statistical qualitites of the data.

Feature engineering and selection often has the highest returns on time invested in a machine learning problem. It can take quite a while to get right, but is often more important than the exact algorithm and hyperparameters used for the model. If we don't feed the model the correct data, then we are setting it up to fail and we should not expect it to learn!

In this project, we will take the following steps for feature engineering:

Select only the numerical variables and two categorical variables (borough and property use type) Add in the log transformation of the numerical variables One-hot encode the categorical variables For feature selection, we will do the following:

Remove collinear features We will discuss collinearity (also called multicollinearity) when we get to that process!

```
# Copy the original data features =

data.copy() numeric_subset =

data.select_dtypes('number')

for col in numeric_subset.columns:
if col == 'score':
        next     else:         numeric_subset['log_' + col] =
np.log(numeric_subset[col])
        categorical_subset = data[['Borough', 'Largest Property Use
Type']] categorical_subset = pd.get_dummies(categorical_subset)
features = pd.concat([numeric_subset, categorical_subset], axis =
1) features.shape
```

```
/usr/local/lib/python3.8/dist-packages/pandas/core/arraylike.py:364: RuntimeWarning: divide by zero encountered in log
result = getattr(ufunc, method)(*inputs, **kwargs)
/usr/local/lib/python3.8/dist-packages/pandas/core/arraylike.py:364: RuntimeWarning: invalid value encountered in log
result = getattr(ufunc, method)(*inputs, **kwargs) (11319, 110)
```

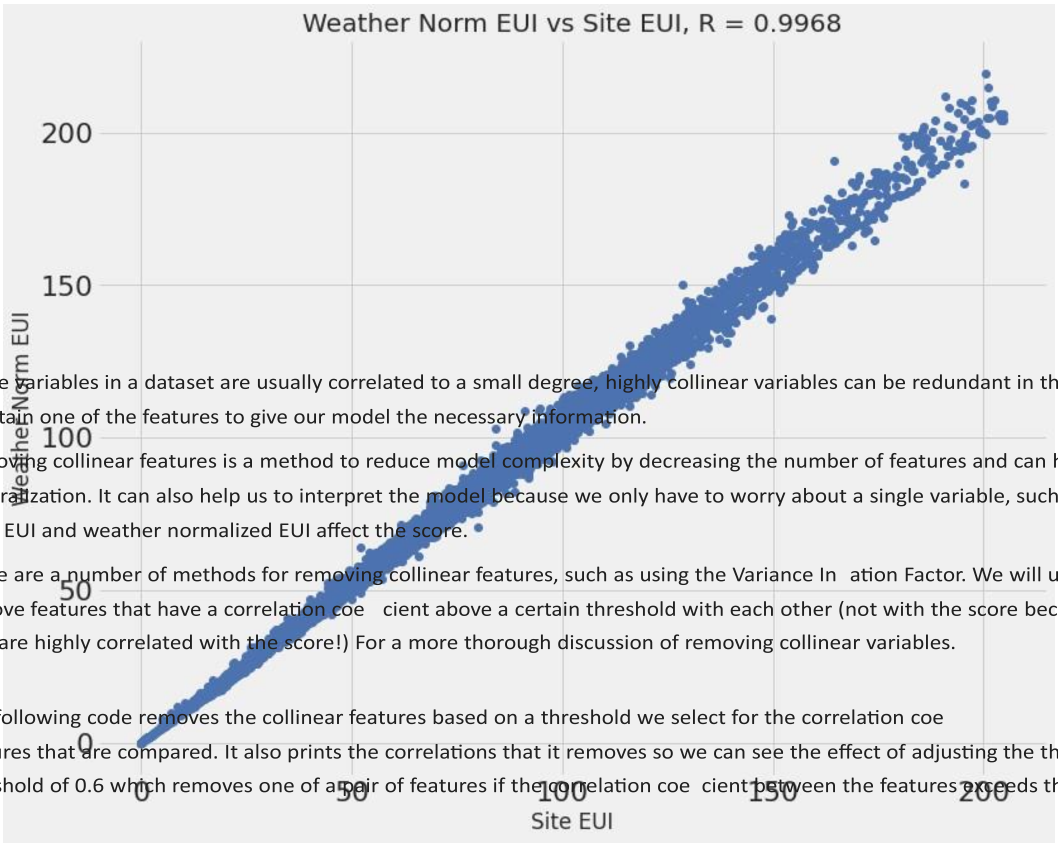
At this point, we have 11319 observations (buildings) with 109 different features (one column is the score). Not all of these features are likely to be important for predicting the score, and several of these features are also redundant because they are highly correlated. We will deal with this second issue below.

Remove Collinear Features

Highly collinear features have a signi cant correlation coe   cent between them. For example, in our dataset, the Site EUI and Weather Norm EUI are highly correlated because they are just slightly different means of calculating the energy use intensity.

Double-click (or enter) to edit

```
plot_data = data[['Weather Normalized Site EUI (kBtu/ft²)', 'Site EUI (kBtu/ft²)']].dropna()

plt.plot(plot_data['Site EUI (kBtu/ft²)'], plot_data['Weather Normalized Site EUI (kBtu/ft²)'], 'bo') plt.xlabel('Site EUI');
plt.ylabel('Weather Norm EUI') plt.title('Weather Norm EUI vs Site EUI, R = %0.4f' % np.corrcoef(data[['Weather Normalized Site EUI
(kBtu/ft²)', 'Site EUI (kBtu/ft²)']]
```



While variables in a dataset are usually correlated to a small degree, highly collinear variables can be redundant in the sense that we only need to retain one of the features to give our model the necessary information.

Removing collinear features is a method to reduce model complexity by decreasing the number of features and can help to increase model generalization. It can also help us to interpret the model because we only have to worry about a single variable, such as EUI, rather than how both EUI and weather normalized EUI affect the score.

There are a number of methods for removing collinear features, such as using the Variance In ation Factor. We will use a simpler metric, and remove features that have a correlation coe cient above a certain threshold with each other (not with the score because we want variables that are highly correlated with the score!) For a more thorough discussion of removing collinear variables.

The following code removes the collinear features based on a threshold we select for the correlation coe

features that are compared. It also prints the correlations that it removes so we can see the effect of adjusting the threshold. We will use a threshold of 0.6 which removes one of a pair of features if the correlation coe cient between the features exceeds this value.

cients by removing one of the two

```
def remove_collinear_features(x, threshold):
    '''
    Objective:
        Remove collinear features in a dataframe with a correlation coefficient
greater than the threshold. Removing collinear features can help a model
to generalize and improves the interpretability of the model.

Inputs:
        threshold: any features with correlations greater than this value are removed
        Output:         dataframe that contains only the non-highly-
collinear features
    '''         y = x['score']
x = x.drop(columns = ['score'])


    # Calculate the correlation matrix
corr_matrix = x.corr()
    iters = range(len(corr_matrix.columns) - 1)
drop_cols = []

    # Iterate through the correlation matrix and compare correlations
for i in iters:         for j in range(i):
            item = corr_matrix.iloc[j:(j+1),
(i+1):(i+2)]             col = item.columns
row = item.index             val = abs(item.values)
                        if val >= threshold:
drop_cols.append(col.values[0])

    # Drop one of each pair of correlated columns     drops =
set(drop_cols)     x = x.drop(columns = drops)     x =
x.drop(columns = ['Weather Normalized Site EUI (kBtu/ft²)',
'Water Use (All Water Sources) (kgal)',
                          'log_Water Use (All Water Sources) (kgal)',
                          'Largest Property Use Type - Gross Floor Area (ft²)'])

x['score'] = y
return x


features = remove_collinear_features(features, 0.6);
features  = features.dropna(axis=1, how = 'all')
features.shape
```

```
     (11319, 65)
```

Our nal dataset now has 64 features (one of the columns is the target). This is still quite a few, but mostly it is because we have one-hot encoded the categorical variables. Moreover, while a large number of features may be problematic for models such as linear regression, models such as the random forest perform implicit feature selection and automatically determine which features are important during traning. There are other feature selection steps to take, but for now we will keep all the features we have and see how the model performs.

Additional Feature Selection

There are plenty of more methods for feature selection. Some popular methods include principal components analysis (PCA) which transforms the features into a reduced number of dimensions that preserve the greatest variance, or independent components analysis (ICA) which aims to nd the independent sources in a set of features. However, while these methods are effective at reducing the number of features, they create new features that have no physical meaning and hence make interpreting a model nearly impossible.

These methods are very helpful for dealing with high-dimensional data

Split Into Training and Testing Sets

In machine learning, we always need to separate our features into two sets:

Training set which we provide to our model during training along with the answers so it can learn a mapping between the features and the target. Testing set which we use to evaluate the mapping learned by the model. The model has never seen the answers on the testing set, but instead, must make predictions using only the features. As we know the true answers for the test set, we can then compare the test predictions to the true test targets to ghet an estimate of how well our model will perform when deployed in the real world. For our problem, we will rst extract all the buildings without an Energy Star Score (we don't know the true answer for these buildings so they will not be helpful for training or testing). Then, we will split the buildings with an Energy Star Score into a testing set of 30% of the buildings, and a training set of 70% of the buildings.

Splitting the data into a random training and testing set is simple using scikit-learn. We can set the random state of the split to ensure consistent results.

```
# Extract the buildings with no score and the buildings with a
score no_score = features[features['score'].isna()] score =
features[features['score'].notnull()]

print(no_score.shape)
print(score.shape)

    (1858, 65)
    (9461, 65)
```

Double-click (or enter) to edit

```
# Separate out the features and targets features =
score.drop(columns='score') targets =
pd.DataFrame(score['score']) features =
features.replace({np.inf: np.nan, -np.inf: np.nan})

# Split into 70% training and 30% testing set
X, X_test, y, y_test = train_test_split(features, targets, test_size = 0.3, random_state = 42)

print(X.shape)
print(X_test.shape)
print(y.shape)
print(y_test.shape)

    (6622, 64)
    (2839, 64)
    (6622, 1)
    (2839, 1)
```

We have 1858 buildings with no score, 6622 buildings with a score in the training set, and 2839 buildings with a score in the testing set. We have one nal step to take in this notebook: determining a naive baseline for our models to beat!

Establish a Baseline

It's important to establish a naive baseline before we beginning making machine learning models. If the models we build cannot outperform a naive guess then we might have to admit that machine learning is not suited for this problem. This could be because we are not using the right models, because we need more data, or because there is a simpler solution that does not require machine learning. Establishing a baseline is crucial so we do not end up building a machine learning model only to realize we can't actually solve the problem.

For a regression task, a good naive baseline is to predict the median value of the target on the training set for all examples on the test set. This is simple to implement and sets a relatively low bar for our models: if they cannot do better than guessing the medin value, then we will need to rethink our approach.

Metric: Mean Absolute Error

There are a number of metrics used in machine learning tasks and it can be di cult to know which one to choose. Most of the time it will depend on the particular problem and if you have a speci c goal to optimize for. I like Andrew Ng's advice to use a single real-value performance metric in order to compare models because it simpli es the evaluate process. Rather than calculating multiple metrics and trying to determine how important each one is, we should use a single number. In this case, because we doing regression, the mean absolute error is an appropriate metric. This is also interpretable because it represents the average amount our estimate if off by in the same units as the target value.

The function below calculates the mean absolute error between true values and predictions.

```
# Function to calculate mean absolute
error def mae(y_true, y_pred):     return
np.mean(abs(y_true - y_pred))
```

Now we can make the median guess and evaluate it on the test set.

```
baseline_guess = np.median(y)

print('The baseline guess is a score of %0.2f' % baseline_guess)
print("Baseline Performance on the test set: MAE = %0.4f" % mae(y_test, baseline_guess))
```

```
     The baseline guess is a score of 66.00
     Baseline Performance on the test set: MAE = 24.5164
```

This shows our average estimate on the test set is off by about 25 points. The scores are between 1 and 100 so this means the average error from a naive method if about 25%. The naive method of guessing the median training value provides us a low baseline for our models to beat!

Conclusions

1. Cleaned and formatted the raw data
2. Performed an exploratory data analysis
3. Developed a set of features to train our model using feature engineering and feature selection

We also completed the crucial task of establishing a baseline metric so we can determine if our model is better than guessing!

In part two, we will focus on implementing several machine learning methods, selecting the best model, and optimizing it for our problem using hyperparameter tuning with cross validation. As a nal step here, we will save the datasets we developed to use again in the part 2.

part 2

we are working on a supervised, regression machine learning problem. Using real-world New York City building energy data, we want to predict the Energy Star Score of a building and determine the factors that in uence the score.

in part 2:

1. Compare several machine learning models on a performance metric
2. Perform hyperparameter tuning on the best model to optimize it for the problem 3. Evaluate the best model on the testing set

Imports

We will use the standard data science and machine learning libraries in this project.

```
 import pandas as pd
import numpy as np

pd.options.mode.chained_assignment = None
pd.set_option('display.max_columns', 60)

import matplotlib.pyplot as plt
%matplotlib inline

plt.rcParams['font.size'] = 24 from

IPython.core.pylabtools import figsize

import seaborn as sns
 sns.set(font_scale = 2)
```

```
from sklearn.impute import SimpleImputer from
sklearn.preprocessing import MinMaxScaler from
sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR from sklearn.neighbors import
KNeighborsRegressor

# Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
```

Read in Data

First let's read in the formatted data from the previous notebook.

```
# Read in data into dataframes train_features =
pd.read_csv('/content/training_features.csv') test_features =
pd.read_csv('/content/testing_features.csv') train_labels =
pd.read_csv('/content/training_labels.csv') test_labels =
pd.read_csv('/content/testing_labels.csv')

# Display sizes of data print('Training Feature Size:
', train_features.shape) print('Testing Feature Size:
', test_features.shape) print('Training Labels Size:
', train_labels.shape) print('Testing Labels Size:
', test_labels.shape)
```

```
     Training Feature Size:  (6622, 64)
     Testing Feature Size:   (2839, 64)
     Training Labels Size:   (6622, 1)
     Testing Labels Size:    (2839, 1)
```

As a reminder, here is what the formatted data looks like. In the rst notebook, we engineered a number features by taking the natural log of the variables, included two categorical variables, and selected a subset of features by removing highly collinear features.

```
train_features.head(12)
```

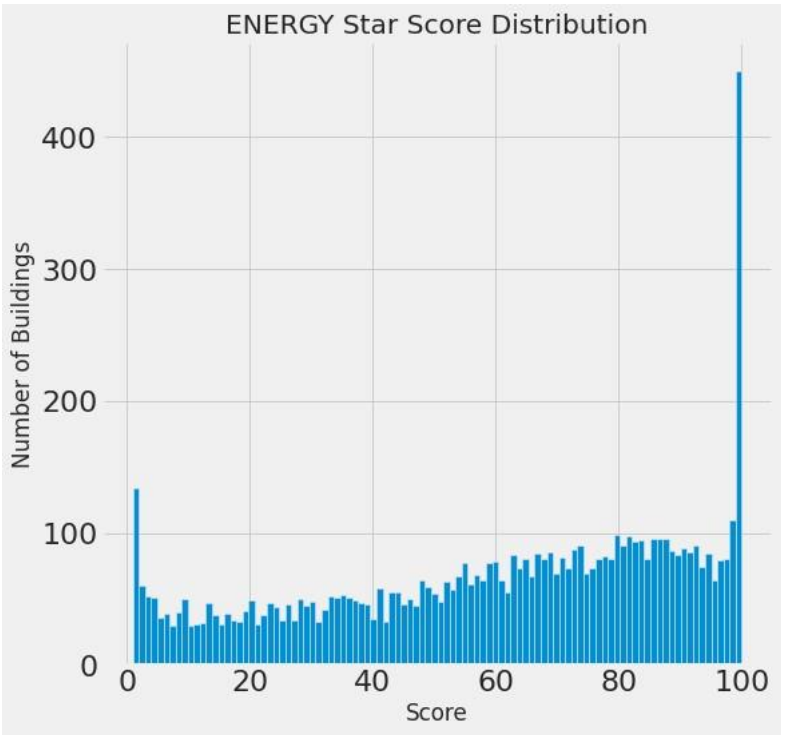
```
                                                                   Weather       Weather      W
                              DOF Number of Normalized Normalized Inten Property Gross Year Buildings Site EUI
                   Site Site Natural
            Order                                        Occupancy
                         Id     Floor Built    - Self-            (kBtu/ft²) Electricity           Gas      W
                                Area          reported                           Intensity      Intensity Sour
                                                                                 (kWh/ft²) (therms/ft²) (gal/

        0 13276     5849784    90300.0    1950              1         100          126.0            5.2             1.2        9
```

The score column contains the Energy Star Score, the target for our machine learning problem. The Energy Star Score is supposed to be a comparitive measurement of the energy e**1** 73774398442 52000.0 1926ciency of a building, although we saw there may be issues with how this is calculated in part one!1 10095.4 4.7 0.9

```
          2     9479  4665374    104700.0   1954        1          100          40.4           3.8            0.3
```

Here's the distribution of the Energy Star Score.

```
      3     14774                3393340    129333.0   1992        1     100          157.1          16.9           1.1

                                                                  100           62.3           3.5            0.0
figsize(48, 32868)2704325  109896.0    1927              1
                                                                   90           52.9           9.7            0.2
        5     1060     2430725    182655.0     1929             1
# Histogram of the Energy Star Score                              100           66.8           3.0            0.6
                                                                                                                   2
plt.style.use(6 10846'fivethirtyeight'5737475   65400.0)          100           78.4           5.7            NaN
        1942       1
plt.hist(train_labels['score'].dropna(), bins = 100);             100           63.0           3.4            0.5
plt.xlabel(7 4280'Score'2670505); plt.ylabel113150.0('Number                                                       6
of   Buildings'1938  1  );   plt.title('ENERGY   Star   Score
Distribution');                                                                                                    3
        8 12974     2964670   137700.0    1959             1                                                       4
```



4.3	0.8	8
4.5	0.0	
3.6	1.1	

Evaluating and Comparing Machine Learning Models

In this section we will build, train, and evalute several machine learning methods for our supervised regression task. The objective is to determine which model holds the most promise for further development (such as hyperparameter tuning).

We are comparing models using the mean absolute error. A baseline model that guessed the median value of the score was off by an average of 25 points.

Imputing Missing Values

Standard machine learning models cannot deal with missing values, and which means we have to nd a way to ll these in or disard any features with missing values. Since we already removed features with more than 50% missing values in the rst part, here we will focus on lling in these missing values, a process known as imputation. There are a number of methods for imputation but here we will use the relatively simple method of replacing missing values with the median of the column.

In the code below, we create a Scikit-learn Imputer object to ll in missing values with the median of the column. Notice that we train the imputer (using the Imputer. t method) on the training data but not the testing data. We then transform (using Imputer.transform) both the training data and testing data. This means that the missing values in the testing set are lled in with the median value of the corresponding columns in the training set. We have to do it this way rather than because at deployment time, we will have to impute the missing values in new observations based on the previous training data. This is one way to avoid the problem known as data leakage where information from the testing set "leaks" into the training process.

```
# Create an imputer object with a median filling strategy
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(train_features)

# Transform both training data and testing data
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)
```

```
 print('Missing values in training features: ', np.sum(np.isnan(X)))
print('Missing values in testing features:  ', np.sum(np.isnan(X_test)))

     Missing values in training features:  0
     Missing values in testing features:   0
```

After imputation, all of the features are real-valued. For more sophisticated methods of imputation (although median values usually works well)

Scaling Features

The nal step to take before we can build our models is to scale the features. This is necessary because features are in different units, and we want to normalize the features so the units do not affect the algorithm. Linear Regression and Random Forest do not require feature scaling, but other methods, such as support vector machines and k nearest neighbors, do require it because they take into account the Euclidean distance between observations. For this reason, it is a best practice to scale features when we are comparing multiple algorithms.

There are two ways to scale features:

1. For each value, subtract the mean of the feature and divide by the standard deviation of the feature. This is known as standardization and results in each feature having a mean of 0 and a standard deviation of 1.
2. For each value, subtract the minimum value of the feature and divide by the maximum minus the minimum for the feature (the range). This assures that all the values for a feature are between 0 and 1 and is called scaling to a range or normalization.

As with imputation, when we train the scaling object, we want to use only the training set. When we transform features, we will transform both the training set and the testing set.

```
# Create the scaler object with a range of 0-1
scaler = MinMaxScaler(feature_range=(0, 1))

# Fit on the training data
scaler.fit(X)

# Transform both the training and testing data
 X = scaler.transform(X)
 X_test = scaler.transform(X_test)


# Convert y to one-dimensional array (vector)
y = np.array(train_labels).reshape((-1, ))
y_test = np.array(test_labels).reshape((-1,
))
```

Models to Evaluate

We will compare ve different machine learning models using the great Scikit-Learn library:

1. Linear Regression
2. Support Vector Machine Regression
3. Random Forest Regression
4. Gradient Boosting Regression
5. K-Nearest Neighbors Regression

Again, here I'm focusing on implementation rather than explaining how these work. To compare the models, we are going to be mostly using the Scikit-Learn defaults for the model hyperparameters. Generally these will perform decently, but should be optimized before actually using a model. At rst, we just want to determine the baseline performance of each model, and then we can select the best performing model for further optimization using hyperparameter tuning. Remember that the default hyperparameters will get a model up and running, but nearly always should be adjusted using some sort of search to nd the best settings for your problem!

Here is what the Scikit-learn documentation says about the defaults:

```
__Sensible defaults__: Whenever an operation requires a user-defined
parameter, an appropriate default value is defined by the library. The default
value


should cause the operation to be performed in a sensible way (giving a
baseline solution for the task at hand.)
```

One of the best parts about scikit-learn is that all models are implemented in an identical manner: once you know how to build one, you can implement an extremely diverse array of models. Here we will implement the entire training and testing procedures for a number of models in just a few lines of code.

```
# Function to calculate mean absolute
error def mae(y_true, y_pred):     return
np.mean(abs(y_true - y_pred))

# Takes in a model, trains the model, and evaluates the model on the test set
def fit_and_evaluate(model):
        model.fit(X, y)
model_pred = model.predict(X_test)
model_mae = mae(y_test, model_pred)
return model_mae


lr = LinearRegression() lr_mae = fit_and_evaluate(lr) print('Linear
Regression Performance on the test set: MAE = %0.4f' % lr_mae)

     Linear Regression Performance on the test set: MAE = 13.4901
```

```
svm = SVR(C = 1000, gamma = 0.1) svm_mae = fit_and_evaluate(svm) print('Support Vector
Machine Regression Performance on the test set: MAE = %0.4f' % svm_mae)
```

```
     Support Vector Machine Regression Performance on the test set: MAE = 11.1165
```

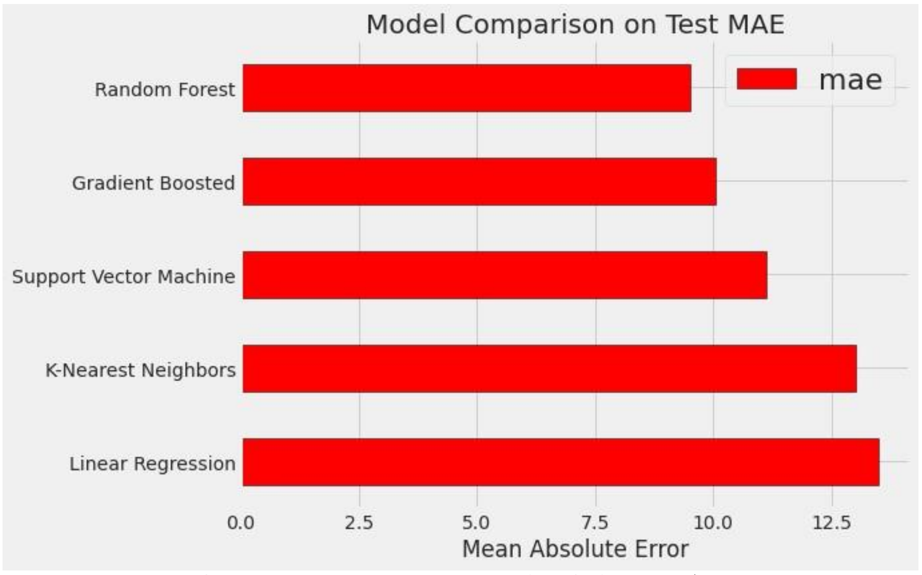
```
random_forest = RandomForestRegressor(random_state=60) random_forest_mae =
fit_and_evaluate(random_forest) print('Random Forest Regression Performance on the test set:
MAE = %0.4f' % random_forest_mae)
```

```
     Random Forest Regression Performance on the test set: MAE = 9.5266
```

```
gradient_boosted = GradientBoostingRegressor(random_state=60) gradient_boosted_mae =
fit_and_evaluate(gradient_boosted) print('Gradient Boosted Regression Performance on the test set:
MAE = %0.4f' % gradient_boosted_mae)
```

```
     Gradient Boosted Regression Performance on the test set: MAE = 10.0542
```

```
knn = KNeighborsRegressor(n_neighbors=10) knn_mae = fit_and_evaluate(knn) print('K-Nearest
```



```
Neighbors Regression Performance on the test set: MAE = %0.4f' % knn_mae)
```

```
     K-Nearest Neighbors Regression Performance on the test set: MAE = 13.0343
```

```
plt.style.use('fivethirtyeight')
figsize(8, 6)

# Dataframe to hold the results model_comparison = pd.DataFrame({'model': ['Linear
Regression', 'Support Vector Machine',
                                           'Random Forest', 'Gradient Boosted',
                                            'K-Nearest Neighbors'],
                                 'mae': [lr_mae, svm_mae, random_forest_mae,
gradient_boosted_mae, knn_mae]})

model_comparison.sort_values('mae', ascending = False).plot(x = 'model', y = 'mae', kind = 'barh',
color = 'red', edgecolor = 'black')

# Plot formatting
plt.ylabel(''); plt.yticks(size = 14); plt.xlabel('Mean Absolute Error'); plt.xticks(size = 14)
plt.title('Model Comparison on Test MAE', size = 20);
```

Depending on the run (the exact results change slighty each time), the gradient boosting regressor performs the best followed by the random forest. I have to admit that this is not the most fair comparison because we are using mostly the default hyperparameters. Especially with the Support Vector Regressor, the hyperparameters have a signi cant in uence on performance. (the random forest and gradient boosting methods are great for starting out because the performance is less dependent on the model settings). Nonetheless, from these results, we can conclude that machine learning is applicable because all the models signi cantly outperform the baseline!

From here, I am going to concentrate on optimizing the best model using hyperparamter tuning. Given the results here, I will concentrate on using the GradientBoostingRegressor. This is the Scikit-Learn implementation of Gradient Boosted Trees which has won many Kaggle competitions in the past few years. The Scikit-Learn version is generally slower than the XGBoost version, but here we'll stick to Scikit-Learn because the syntax is more familiar. Here's a guide to using the implementation in the XGBoost package.

Model Optimization

In machine learning, optimizing a model means nding the best set of hyperparameters for a particular problem.

Hyperparameters First off, we need to understand what model hyperparameters are in contrast to model parameters :

Model hyperparameters are best thought of as settings for a machine learning algorithm that are tuned by the data scientist before training. Examples would be the number of trees in the random forest, or the number of neighbors used in K Nearest Neighbors Regression. Model parameters are what the model learns during training, such as the weights in the linear regression. We as data scientists control a model by choosing the hyperparameters, and these choices can have a signi cant effect on the nal performance of the model (although usually not as great of an effect as getting more data or engineering features).

Tuning the model hyperparameters controls the balance of under vs over tting in a model. We can try to correct for under- tting by making a more complex model, such as using more trees in a random forest or more layers in a deep neural network. A model that under ts has high bias, and occurs when our model does not have enough capacity (degrees of freedom) to learn the relationship between the features and the target. We can try to correct for over tting by limiting the complexity of the model and applying regularization. This might mean decreasing the degree of a polynomial regression, or adding dropout layers to a deep neural network. A model that over ts has high variance and in effect has memorized the training set. Both under tting and over tting lead to poor generalization performance on the test set.

The problem with choosing the hyperparameters is that no set will work best across all problems. Therefore, for every new dataset, we have to nd the best settings. This can be a time-consuming process, but luckily there are several options for performing this procedure in Scikit-Learn. Even better, new libraries, such as TPOT by epistasis labs, is aiming to do this process automatically for you! For now, we will stick to doing this manually (sort of) in Scikit-Learn, but stay tuned for an article on automatic model selection!

Hyperparameter Tuning with Random Search and Cross Validation

We can choose the best hyperparameters for a model through random search and cross validation.

- Random search refers to the method in which we choose hyperparameters to evaluate: we de ne a range of options, and then randomly select combinations to try. This is in contrast to grid search which evaluates every single combination we specify. Generally, random search is better when we have limited knowledge of the best model hyperparameters and we can use random search to narrow down the options and then use grid search with a more limited range of options.
- Cross validation is the method used to assess the performance of the hyperparameters. Rather than splitting the training set up into separate training and validation sets which reduces the amount of training data we can use, we use K-Fold Cross Validation. This means dividing the training data into K folds, and then going through an iterative process where we rst train on K-1 of the folds and then evaluate performance on the kth fold. We repeat this process K times so eventually we will have tested on every example in the training data with the key that each iteration we are testing on data that we **did not train on**. At the end of K-fold cross validation, we take the average error on each of the K iterations as the nal performance measure and then train the model on all the training data at once. The performance we record is then used to compare different combinations of hyperparameters.

Here we will implement random search with cross validation to select the optimal hyperparameters for the gradient boosting regressor. We rst de ne a grid then peform an iterative process of: randomly sample a set of hyperparameters from the grid, evaluate the hyperparameters using 4-fold cross-validation, and then select the hyperparameters with the best performance.

Of course we don't actually do this iteration ourselves, we let Scikit-Learn and `RandomizedSearchCV` do the process for us!

```
loss = ['ls', 'lad', 'huber']

# Number of trees used in the boosting process
n_estimators = [100, 500, 900, 1100, 1500]

# Maximum depth of each tree
max_depth = [2, 3, 5, 10, 15]

# Minimum number of samples per leaf
 min_samples_leaf = [1, 2, 4, 6, 8]

# Minimum number of samples to split a node

min_samples_split = [2, 4, 6, 10]

max_features = ['auto', 'sqrt', 'log2',

None]

# Define the grid of hyperparameters to search
hyperparameter_grid = {'loss': loss,
                       'n_estimators': n_estimators,
                       'max_depth': max_depth,
                       'min_samples_leaf': min_samples_leaf,
                       'min_samples_split': min_samples_split,
                       'max_features': max_features}
```

We selected 6 different hyperparameters to tune in the gradient boosting regressor. These all will affect the model in different ways that are hard to determine ahead of time, and the only method for nding the best combination for a speci c problem is to test them out! To read about the hyperparameters

For now, just know that we are trying to nd the best combination of hyperparameters and because there is no theory to tell us which will work best, we just have to evaluate them, like runnning an experiment!

In the code below, we create the Randomized Search Object passing in the following parameters:

estimator: the model param_distributions: the distribution of parameters we de ned cv the number of folds to use for k-fold cross validation n_iter: the number of different combinations to try scoring: which metric to use when evaluating candidates n_jobs: number of cores to run in parallel (-1 will use all available) verbose: how much information to display (1 displays a limited amount) return_train_score: return the training score for each cross-validation fold random_state: xes the random number generator used so we get the same results every run The Randomized Search Object is trained the same way as any other scikit-learn model. After training, we can compare all the different hyperparameter combinations and nd the best

```
# Create the model to use for hyperparameter tuning
model = GradientBoostingRegressor(random_state = 42)

# Set up the random search with 4-fold cross validation
random_cv = RandomizedSearchCV(estimator=model,
                               param_distributions=hyperparameter_grid,
cv=4, n_iter=25,
                               scoring = 'neg_mean_absolute_error',
n_jobs = -1, verbose = 1,
return_train_score = True,
random_state=42)


# Fit on the training data
random_cv.fit(X, y)
```

```
Fitting 4 folds for each of 25 candidates, totalling 100 fits
/usr/local/lib/python3.8/dist-packages/sklearn/ensemble/_gb.py:293: FutureWarning: The loss 'lad' was deprecated in v1.0 and will b
warnings.warn(
RandomizedSearchCV(cv=4, estimator=GradientBoostingRegressor(random_state=42),
n_iter=25, n_jobs=-1,
                   param_distributions={'loss': ['ls', 'lad', 'huber'],
'max_depth': [2, 3, 5, 10, 15],
                                        'max_features': ['auto', 'sqrt', 'log2',
                                                         None],
                                        'min_samples_leaf': [1, 2, 4, 6, 8],
                                        'min_samples_split': [2, 4, 6, 10],
                                        'n_estimators': [100, 500, 900, 1100,
                                                         1500]},
                   random_state=42, return_train_score=True,
scoring='neg_mean_absolute_error', verbose=1)
```

Scikit-learn uses the negative mean absolute error for evaluation because it wants a metric to maximize. Therefore, a better score will be closer to 0. We can get the results of the randomized search into a dataframe, and sort the values by performance.

```
# Get all of the cv results and sort by the test performance
random_results = pd.DataFrame(random_cv.cv_results_).sort_values('mean_test_score', ascending = False)

random_results.head(10)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_min_samples
12	27.855932	0.390030	0.024367	0.000635	500	
3	30.023900	0.647228	0.027485	0.001071	500	
9	17.840448	0.088589	0.017201	0.000184	500	
0	4.617086	0.038179	0.007538	0.000160	100	
7	14.507146	0.092443	0.017689	0.000664	500	
10	137.214633	2.175003	0.107665	0.011102	1100	
2	85.546104	2.961274	0.055253	0.001409	500	
16	44.070959	0.863041	0.046740	0.000547	1500	
19	36.892311	0.181149	0.030055	0.000308	1100	
21	10.258651	0.553783	0.033655	0.008664	500	

```
random_cv.best_estimator_
```

```
GradientBoostingRegressor(loss='lad', max_depth=5, min_samples_leaf=6,
min_samples_split=6, n_estimators=500,
random_state=42)
```

The best gradient boosted model has the following hyperparameters:

- `loss = lad`
- `n_estimators = 500`
- `max_depth = 5`
- `min_samples_leaf = 6`
- `min_samples_split = 6`
- `max_features = None` (This means that `max_features = n_features` according to the docs)

Using random search is a good method to narrow down the possible hyperparameters to try. Initially, we had no idea which combination would work the best, but this at least narrows down the range of options.

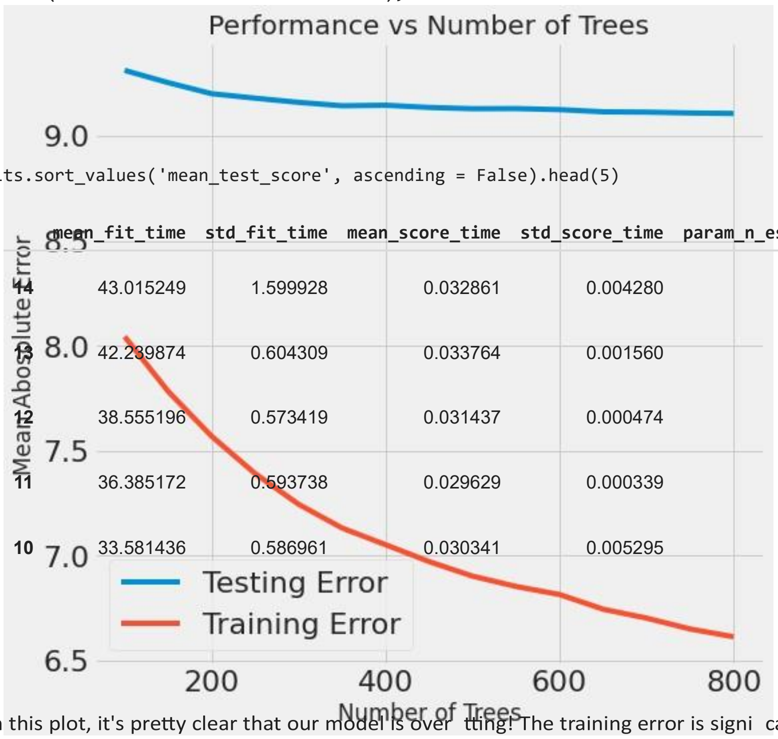
We could use the random search results to inform a grid search by creating a grid with hyperparameters close to those that worked best during the randomized search. However, rather than evaluating all of these settings again, I will focus on a single one, the number of trees in the forest (`n_estimators`). By varying only one hyperparameter, we can directly observe how it affects performance. In the case of the number of trees, we would expect to see a signi cant affect on the amount of under vs over tting.

Here we will use grid search with a grid that only has the `n_estimators` hyperparameter. We will evaluate a range of trees then plot the training and testing performance to get an idea of what increasing the number of trees does for our model. We will x the other hyperparameters at the best values returned from random search to isolate the number of trees effect.

```
# Create a range of trees to evaluate trees_grid = {'n_estimators': [100, 150, 200, 250, 300, 350, 400,
450, 500, 550, 600, 650, 700, 750, 800]}

model = GradientBoostingRegressor(loss = 'lad', max_depth = 5,
min_samples_leaf = 6,
min_samples_split = 6,
max_features = None,
random_state = 42)

# Grid Search Object using the trees range and the random forest model
grid_search = GridSearchCV(estimator = model, param_grid=trees_grid, cv = 4,
scoring = 'neg_mean_absolute_error', verbose = 1,
n_jobs = -1, return_train_score = True)


# Fit the grid search
grid_search.fit(X, y)
```

```
Fitting 4 folds for each of 15 candidates, totalling 60 fits
/usr/local/lib/python3.8/dist-packages/sklearn/ensemble/_gb.py:293: FutureWarning: The loss 'lad' was deprecated in v1.0 and will b
warnings.warn( GridSearchCV(cv=4,
             estimator=GradientBoostingRegressor(loss='lad', max_depth=5,
min_samples_leaf=6,
min_samples_split=6,
random_state=42),              n_jobs=-1,
param_grid={'n_estimators': [100, 150, 200, 250, 300, 350, 400,
450, 500, 550, 600, 650, 700, 750,
                                          800]},
             return_train_score=True, scoring='neg_mean_absolute_error',
verbose=1)
```

```
# Get the results into a dataframe results =
pd.DataFrame(grid_search.cv_results_)

# Plot the training and testing error vs number of trees
figsize(8, 8)
plt.style.use('fivethirtyeight')
plt.plot(results['param_n_estimators'], -1 * results['mean_test_score'], label = 'Testing Error')
plt.plot(results['param_n_estimators'], -1 * results['mean_train_score'], label = 'Training Error')
plt.xlabel('Number of Trees'); plt.ylabel('Mean Abosolute Error'); plt.legend();
plt.title('Performance vs Number of Trees');
```



```
results.sort_values('mean_test_score', ascending = False).head(5)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	params	spl
14	43.015249	1.599928	0.032861	0.004280	800	{'n_estimators': 800}	
13	42.239874	0.604309	0.033764	0.001560	750	{'n_estimators': 750}	
12	38.555196	0.573419	0.031437	0.000474	700	{'n_estimators': 700}	
11	36.385172	0.593738	0.029629	0.000339	650	{'n_estimators': 650}	
10	33.581436	0.586961	0.030341	0.005295	600	{'n_estimators': 600}	

From this plot, it's pretty clear that our model is over tting! The training error is signi cantly lower than the testing error, which shows that the model is learning the training data very well but then is not able to generalize to the test data as well. Moveover, as the number of trees increases, the amount of over tting increases. Both the test and training error decrease as the number of trees increase but the training error decreases more rapidly.

There will always be a difference between the training error and testing error (the training error is always lower) but if there is a signi cant difference, we want to try and reduce over tting, either by getting more training data or reducing the complexity of the model through hyperparameter tuning or regularization. For the gradient boosting regressor, some options include reducing the number of trees, reducing the max depth of each tree, and increasing the minimum number of samples in a leaf node. For anyone who wants to go further into the gradient boosting regressor, here is a great article. For now, we will use the model with the best performance and accept that it may be over tting to the training set.

Based on the cross validation results, the best model using 800 trees and achieves a cross validation error under 9. This indicates that the average cross-validation estimate of the Energy Star Score is within 9 points of the true answer!

Evaluate Final Model on the Test Set

We will use the best model from hyperparameter tuning to make predictions on the testing set. Remember, our model has never seen the test set before, so this performance should be a good indicator of how the model would perform if deployed in the real world.

For comparison, we can also look at the performance of the default model. The code below creates the nal model, trains it (with timing), and evaluates on the test set.

```
# Default model default_model =
GradientBoostingRegressor(random_state = 42)

# Select the best model final_model =

grid_search.best_estimator_ final_model
```

```
    GradientBoostingRegressor(loss='lad', max_depth=5, min_samples_leaf=6,
    min_samples_split=6, n_estimators=800,
    random_state=42)
```

```
%%timeit -n 1 -r 5
default_model.fit(X, y)
```

```
    2.51 s ± 18 ms per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

```
%%timeit -n 1 -r 5
final_model.fit(X, y)
```

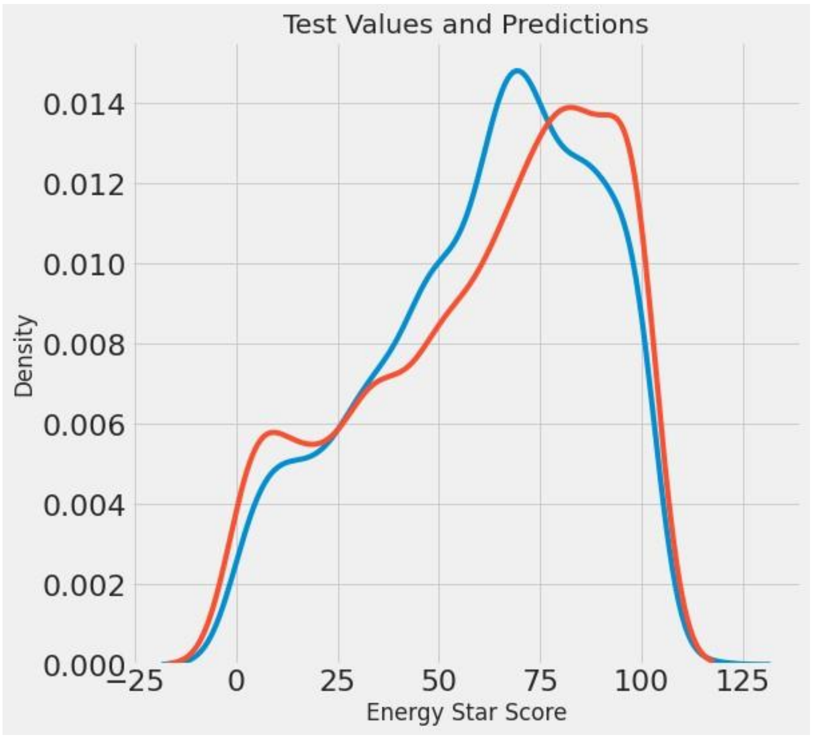
```
    /usr/local/lib/python3.8/dist-packages/sklearn/ensemble/_gb.py:293: FutureWarning: The loss 'lad' was deprecated in v1.0 and will b
    warnings.warn(
    /usr/local/lib/python3.8/dist-packages/sklearn/ensemble/_gb.py:293: FutureWarning: The loss 'lad' was deprecated in v1.0 and will b
    warnings.warn(
    /usr/local/lib/python3.8/dist-packages/sklearn/ensemble/_gb.py:293: FutureWarning: The loss 'lad' was deprecated in v1.0 and will b
    warnings.warn(
```

```
default_pred = default_model.predict(X_test) final_pred = final_model.predict(X_test)
print('Default model performance on the test set: MAE = %0.4f.' % mae(y_test, default_pred))
print('Final model performance on the test set:   MAE = %0.4f.' % mae(y_test, final_pred))
```

The nal model does out-perform the baseline model by about 10%, but at the cost of signi cantly increased running time (it's about 12 times slower on my machine). Machine learning is often a eld of tradeoffs: bias vs variance, acccuracy vs interpretability, accuracy vs running time, and the nal decision of which model to use depends on the situation. Here, the increase in run time is not an impediment, because while the relative difference is large, the absolute magnitude of the training time is not signi cant. In a different situation, the balance might not be the same so we would need to consider what we are optimizing for and the limitations we have to work with.

To get a sense of the predictions, we can plot the distribution of true values on the test set and the predicted values on the test set.

```
figsize(8, 8)

# Density plot of the final predictions and the test values
sns.kdeplot(final_pred, label = 'Predictions')
sns.kdeplot(y_test, label = 'Values')

# Label the plot plt.xlabel('Energy Star Score');
plt.ylabel('Density'); plt.title('Test Values and
Predictions');
```



The distribution looks to be nearly the same although the density of the predicted values is closer to the median of the test values rather than to the actual peak at 100. It appears the model might be less accurate at predicting the extreme values and instead predicts values closer to the median.

Another diagnostic plot is a histogram of the residuals. Ideally, we would hope that the residuals are normally distributed, meaning that the model is wrong the same amount in both directions (high and low).

```
figsize = (6, 6)

# Calculate the residuals
residuals = final_pred - y_test

plt.hist(residuals, color = 'red', bins = 20,
edgecolor = 'black') plt.xlabel('Error');
plt.ylabel('Count') plt.title('Distribution of
Residuals');
```

The residuals are close to normally disributed, with a few noticeable outliers on the low end. These indicate errors where the model estimate was far below that of the true value.

Conclusions

In this part 2, we covered crucial concepts in the machine learning pipeline:

Imputing missing values Evaluating and comparing several machine learning methods Hyperparameter tuning a machine learning model using random search and cross validation Evaluating the best model on the testing set

The results showed us that machine learning is applicable to our problem, with the nal model able to the predict the Energy Star Score of a building to within 9.1 points. We also saw that hyperparamter tuning was able to improve the performance of the model although at a considerable cost. This is a good reminder that proper feature engineering and gathering more data (if possible!) has a much larger pay-off than ne-tuning the model. We also observed the trade-off in run-time versus accuracy, which is one of many considerations we have to take into account when designing machine learning models.

We know our model is accurate, but do we know why it makes the predictions it does? The next step in the machine learning process is crucial: trying to understand how the model makes predictions. Achieveing high accuracy is great, but it would also be helpful if we could gure out why the model is able to predict accurately so we could use this information to better understand the problem. For example, what features does the model rely on to infer the Energy Star Score? It is possible to use this model for feature selection and implement a simpler model that is more interpretable?

In the nal part, we will try to answer these questions and draw nal conclusions from the project!

FINAL PART

we will concentrate on the last two steps, and try to peer into the black box of the model we built. We know is it accurate, as it can predict the Energy Star Score to within 9.1 points of the true value, but how exactly does it make predictions? We'll examine some ways to attempt to understand the gradient boosting machine and then draw conclusions

Recreate nal Model

```
imputer = Imputer(strategy='median')

imputer.fit(train_features)

# Transform both training data and testing data
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)

# Sklearn wants the labels as one-dimensional
vectors y = np.array(train_labels).reshape((-1,))
y_test = np.array(test_labels).reshape((-1,))


# Function to calculate mean absolute
error def mae(y_true, y_pred):     return
np.mean(abs(y_true - y_pred))


#  Make predictions on the test set model_pred = model.predict(X_test) print('Final

Model Performance on the test set: MAE = %0.4f' % mae(y_test, model_pred))
```

Interprete the Model

Machine learning is often criticized as being a black-box: we put data in on one side and it gives us the answers on the other. While these answers are often extremely accurate, the model tells us nothing about how it actually made the predictions. This is true to some extent, but there are ways in which we can try and discover how a model "thinks" such as the Locally Interpretable Model-agnostic Explainer (LIME). This attemps to explain model predictions by learning a linear regression around the prediction, which is an easily interpretable model!

We will explore several ways to interpret our model:

- Feature importances
- Locally Interpretable Model-agnostic Explainer (LIME) Examining a
- single decision tree in the ensemble.

Feature Importances

One of the basic ways we can interpret an ensemble of decision trees is through what are known as the feature importances. These can be interpreted as the variables which are most predictive of the target. While the actual details of the feature importances are quite complex (here is a Stack Over ow question on the subject, we can use the relative values to compare the features and determine which are most relevant to our problem.

Extracting the feature importances from a trained ensemble of trees is quite easy in scikit-learn. We will store the feature importances in a dataframe to analyze and visualize them.

```
# Extract the feature importances into a dataframe feature_results =
pd.DataFrame({'feature': list(train_features.columns),
                                'importance': model.feature_importances_})

# Show the top 10 most important
feature_results = feature_results.sort_values('importance', ascending = False).reset_index(drop=True)

feature_results.head(10)
```

The Site Energy Use Intensity, `Site EUI (kBtu/ft²)` , and the Weather Normalized Site Electricity Intensity, `Weather Normalized Site Electricity Intensity (kWh/ft²)` are the two most important features by quite a large margin. After that, the relative importance drops off considerably which indicates that we might not need to retain all of the features to create a model with nearly the same performance.

Let's graph the feature importances to compare visually.

```
figsize(12, 10)
plt.style.use('fivethirtyeight')

# Plot the 10 most important features in a horizontal bar chart
feature_results.loc[:9, :].plot(x = 'feature', y = 'importance',
edgecolor = 'k',
                                 kind='barh', color = 'blue');
plt.xlabel('Relative Importance', size = 20); plt.ylabel('')
plt.title('Feature Importances from Random Forest', size = 30);
```

Use Feature Importances for Feature Selection

Given that not every feature is important for nding the score, what would happen if we used a simpler model, such as a linear regression, with the subset of most important features from the random forest? The linear regression did outperform the baseline, but it did not perform well compared to the model complex models. Let's try using only the 10 most important features in the linear regression to see if performance is improved. We can also limit to these features and re-evaluate the random forest.

```
# Extract the names of the most important features
most_important_features = feature_results['feature'][:10]

# Find the index that corresponds to each feature name indices =
[list(train_features.columns).index(x) for x in most_important_features]

# Keep only the most important features
X_reduced = X[:, indices]
X_test_reduced = X_test[:, indices]

print('Most important training features shape: ', X_reduced.shape)
print('Most important testing  features shape: ', X_test_reduced.shape) lr =
LinearRegression()

# Fit on full set of features
lr.fit(X, y)
lr_full_pred = lr.predict(X_test)

# Fit on reduced set of features
lr.fit(X_reduced, y)
lr_reduced_pred = lr.predict(X_test_reduced)

# Display results
print('Linear Regression Full Results: MAE =    %0.4f.' % mae(y_test, lr_full_pred))
print('Linear Regression Reduced Results: MAE = %0.4f.' % mae(y_test, lr_reduced_pred))
```

Well, reducing the features did not improve the linear regression results! It turns out that the extra information in the features with low importance do actually improve performance.

Let's look at using the reduced set of features in the gradient boosted regressor. How is the performance affected?

```
# Create the model with the same hyperparamters
model_reduced = GradientBoostingRegressor(loss='lad', max_depth=5, max_features=None,
min_samples_leaf=6, min_samples_split=6,
n_estimators=800, random_state=42)

# Fit and test on the reduced set of features model_reduced.fit(X_reduced, y)
model_reduced_pred = model_reduced.predict(X_test_reduced) print('Gradient Boosted
Reduced Results: MAE = %0.4f' % mae(y_test, model_reduced_pred))
```

The model results are slightly worse with the reduced set of features and we will keep all of the features for the nal model. The desire to reduce the number of features is because we are always looking to build the most parsimonious model: that is, the simplest model with adequate performance. A model that uses fewer features will be faster to train and generally easier to interpret. In this case, keeping all of the features is not a major concern because the training time is not signi cant and we can still make interpretations with many features.

Locally Interpretable Model-agnostic Explanations

We will look at using LIME to explain individual predictions made the by the model. LIME is a relatively new effort aimed at showing how a machine learning model thinks by approximating the region around a prediction with a linear model.

We will look at trying to explain the predictions on an example the model gets very wrong and an example the model gets correct. We will restrict ourselves to using the reduced set of 10 features to aid interpretability. The model trained on the 10 most important features is slightly less accurate, but we generally have to trade off accuracy for interpretability!

```
# Find the residuals
residuals = abs(model_reduced_pred - y_test)
    # Exact the worst and best prediction
wrong = X_test_reduced[np.argmax(residuals),
:] right =
X_test_reduced[np.argmin(residuals), :]
```

```
# Create a lime explainer object
explainer = lime.lime_tabular.LimeTabularExplainer(training_data = X_reduced,
mode = 'regression',
training_labels = y,
                                                   feature_names = list(most_important_features))


# Display the predicted and true value for the wrong instance
print('Prediction: %0.4f' % model_reduced.predict(wrong.reshape(1, -1)))
print('Actual Value: %0.4f' % y_test[np.argmax(residuals)])

# Explanation for wrong prediction
wrong_exp = explainer.explain_instance(data_row = wrong,
                                       predict_fn = model_reduced.predict)

# Plot the prediction explaination
wrong_exp.as_pyplot_figure();
plt.title('Explanation of Prediction', size = 28);
plt.xlabel('Effect on Prediction', size = 22);


wrong_exp.show_in_notebook(show_predicted_value=False)
```

In this example, our gradient boosted model predicted a score of 12.86 and the actual value was 100.

The plot from LIME is showing us the contribution to the nal prediction from each of the features for the example. We can see that the Site EUI singi cantly decreased the prediction because it was above 95.50. The Weather Normalized Site Electricity Intensity on the other hand, increased the prediction because it was lower than 3.80.

We can interpret this as saying that our model thought the Energy Star Score would be much lower than it actually was because the Site EUI was high. However, in this case, the score was 100 despite the high value of the EUI. While this signi cant mistake (off by 88 points!) might initially have been confusing, now we can see that in reality, the model was reasoning through the problem and just arrived at the incorrect value! A human going over the same process probably would have arrived at the same conclusion (if they had the patience to go through all the data).

```
# Display the predicted and true value for the wrong instance
print('Prediction: %0.4f' % model_reduced.predict(right.reshape(1, -1)))
print('Actual Value: %0.4f' % y_test[np.argmin(residuals)])

# Explanation for wrong prediction
right_exp = explainer.explain_instance(right, model_reduced.predict, num_features=10)
right_exp.as_pyplot_figure();
plt.title('Explanation of Prediction', size = 28);
plt.xlabel('Effect on Prediction', size = 22);


right_exp.show_in_notebook(show_predicted_value=False)
```

The correct value for this case was 100 which our gradient boosted model got right on!

The plot from LIME again shows the contribution to the prediciton of each of feature variables for the example. For instance, because the Site EUI was less than 62.70, that contributed signi cantly to a higher estimate of the score. Likewise, the year built being less than 1927 also positively contributed to the nal prediction.

Observing break down plots like these allow us to get an idea of how the model makes a prediction. This is probably most valuable for cases where the model is off by a large amount as we can inspect the errors and perhaps engineer better features or adjust the hyperparameters of the model to improve predictions for next time. The examples where the model is off the most could also be interesting edge cases to look at manually. The model drastically underestimated the Energy Star Score for the rst building because of the elevated Site EUI. We might therefore want to ask why the building has such a high Energy Star Score even though it has such a high EUI. A process such as this where we try to work with the machine learning algorithm to gain understanding of a problem seems much better than simply letting the model make predictions and completely trusting them! Although LIME is not perfect, it represents a step in the right direction towards explaining machine learning models.

Examining a Single Decision Tree

One of the coolest parts about a tree-based ensemble is that we can look at any individual estimator. Although our nal model is composed of 800 decision trees, and looking at a single one is not indicative of the entire model, it still allows us to see the general idea of how a decision tree works. From there, it is a natural extension to imagine hundreds of these trees building off the mistakes of previous trees to make a nal prediction (this is a signi cant oversimpli cation of how gradient boosting regression works!)

We will rst extract a tree from the forest and then save it using `sklearn.tree.export_graphviz` . This saves the tree as a `.dot` le which can be converted to a png using command line instructions in the Notebook.

```
# Extract a single tree single_tree =
model_reduced.estimators_[105][0]

tree.export_graphviz(single_tree, out_file = '/content/tree.dot',
rounded = True,
                             feature_names =
most_important_features,                              filled = True)
single_tree
```

That's one entire tree in our regressor of 800! It's a little di   cult to make out because the maximum depth of the tree is 5. To improve the readability, we can limit the max depth in the call to export our tree.

```
tree.export_graphviz(single_tree, out_file = '/content/tree_small.dot',
rounded = True, feature_names = most_important_features,
filled = True, max_depth = 3)
```

Now we can take a look at the tree and try to intrepret it's decisions! The best way to think of a decision tree is as a series of yes/no questions, like a owchart. We start at the top, called the root, and move our way down the tree, with the direction of travel determined by the answer to each equation.

For instance, here the rst question we ask is: is the `Site EUI` less than or equal to 15.95? If the answer is yes, then we move to the left and ask the question: is the `Weather Normalized Site Electricity Intensity` less than or equal to 3.85? If the answer to the rst question was no, we move to the right and ask the question is the `Weather Normalized Site Electricity Intensity` less than or equal to 26.85?

We continue this iterative process until we reach the bottom of the tree and end up in a leaf node. Here, the value we predict corresponds to the value shown in the node (the values in this tree appear to be the actual predictions divided by 100).

Each node has four different pieces of information:

1. The question: based on this answer we move right or left to the next node a layer down in the tree
2. The friedman_mse: a measure of the error for all of the examples in a given node
3. The samples: number of examples in a node
4. The value: the prediction of the target for all examples in a node

We can see that as we increase the depth of the tree, we will be better able to t the data. With a small tree, there will be many examples in each leaf node, and because the model estimates the same value for each example in a node, there will probably be a larger error (unless all of the examples have the same target value). Constructing too large of a tree though can lead to over tting. We can control a number of hyperparameters that determine the depth of the tree and the number of examples in each leaf. We saw how to select a few of these hyperaparameters in the second part when we performed optimimation using cross validation.

CONCLUSION

7. CONCLUSIONS

Given the exploration I can conclude that we can create a model that is accurately infer the Energy Star Score of the building and determine the EUI, Electricity Integrity, floor area, building type and natural gas are the powerful measures for determining the energy star score. The highlights from the work are the Scores of the building are skewed high with a disproportionate global maximum at 100 and a secondary maximum at 1. Senior care communications and hotels have lesser scores than offices, residential halls, warehouses where the multi-family housing falling. Random forest regression is significantly better than the baseline measure when we tested on testing and training data, we achieved an average error of 10 points. So, we provide the data of the new building, our training model can infer and make the accurate prediction of energy star score. This work includes fining an objective measurement of overall performance of building energy and determine why the scores differ between building types.

8. REFERENCES

[1] Scofield, J. H. (2016). Building Energy Star Scores: Good Idea, Bad Science.

[2] Ma, H., Yang, X., Mao, J., & Zheng, H. (2018, October). The energy efficiency prediction method based on gradient boosting regression tree. In *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)* (pp. 1-9). IEEE.

[3] Arjunan, P., Poolla, K., & Miller, C. (2020). EnergyStar++: Towards more accurate and explanatory building energy benchmarking. *Applied Energy*, *276*, 115413.

[4] Papadopoulos, S., & Kontokosta, C. E. (2019). Grading buildings on energy performance using city benchmarking data. *Applied Energy*, *233*, 244-253.

[5] Xi Cheng, P. E. (2020). Applying Machine Learning Based Data-Driven Approach in Commercial Building Energy Prediction. *ASHRAE Transactions*, *126*, 403-411.

[6] Dahlan, N. Y., Mohamed, H., Kamaluddin, K. A., Abd Rahman, N. M., Reimann, G., Chia, J., & Ilham, N. I. (2022). Energy Star based benchmarking model for Malaysian Government hospitals-A qualitative and quantitative approach to assess energy performances. *Journal of Building Engineering*, *45*, 103460.

[7] Jin, X., & Xiao, F. (2021). Synthetic minority oversampling based machine learning method for urban level building EUI prediction and benchmarking. In *Presented at the Applied Energy Symposium 2021: low carbon cities and urban energy systems*.

[8] Ahn, Y., & Sohn, D. W. (2019). The effect of neighbourhood-level urban form on residential building energy use: A GIS-based model using building energy benchmarking data in Seattle. *Energy and Buildings*, *196*, 124-133.

[9] Ma, H., Yang, X., Mao, J., & Zheng, H. (2018, October). The energy efficiency prediction method based on gradient boosting regression tree. In *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)* (pp. 1-9). IEEE.

[10]Scofield, J. H. (2014, August). Energy star building benchmarking scores: good idea, bad science. In *ACEEE Summer Study on Energy Efficiency in Buildings* (pp. 267-82).

[11]Meng, T., Hsu, D., & Han, A. (2017). Estimating energy savings from benchmarking policies in New York City. *Energy*, *133*, 415-423.

[12]Gao, X., & Malkawi, A. (2014). A new methodology for building energy performance benchmarking: An approach based on intelligent clustering algorithm. *Energy and Buildings*, *84*, 607-616.

[13]Chekroud, A. M., Zotti, R. J., Shehzad, Z., Gueorguieva, R., Johnson, M. K., Trivedi, M. H., ... & Corlett, P. R. (2016). Cross-trial prediction of treatment outcome in depression: a machine learning approach. *The Lancet Psychiatry*, *3*(3), 243-250.

[14]Hossain, M. S., Rahaman, S., Kor, A. L., Andersson, K., & Pattinson, C. (2017). A belief rule based expert system for datacenter pue prediction under uncertainty. *IEEE Transactions on Sustainable Computing*, *2*(2), 140-153.