

1. DEFINITION

1.1. Project Overview

I've been working data science and machine learning as a self-learner enthusiast to improve my skills and toolbox as a business development professional and team leader.

Self-learning with online education has great merits but also has some drawbacks. One of the main ones is showing your skills on real world applications and data sets. Therefore, for my capstone project I've decided to enter an ongoing Kaggle competition to contest with experts and see my score against real people instead of only metrics. Also, before nanodegree my first attempt to enter a Kaggle competition was interrupted by my health and family issues and this time I want to earn a respectful leaderboard place.

So, I've decided to enter ASHRAE - Great Energy Predictor III Kaggle Competition. Competition is a complex regression problem on energy consumption. Energy consumption predictions has a rich context since it has been studied all around the world for some time.

Current global warming debate and problems originated because of global warming has increased importance of energy efficiency. There are many regulations for energy consumption on buildings both sides of the pond. ASHRAE - Great Energy Predictor III is a competition consists both consumption prediction and its use to determine efficiency of energy efficiency retrofits against actual data. This real-life using purpose made it perfect candidate for my project.

In this huge context I tried to find researches similar to my project here and used key words such as:

- *"measuring efficiency of building retrofits with predicted consumption"*
- *"using regression to predict energy consumption"*
- *"measuring efficiency of building retrofits with machine learning"*
- *"measuring energy consumption of building retrofits with machine learning"*

I have found 2 of related research paper I'd like to mention here:

- <https://viejournal.springeropen.com/track/pdf/10.1186/s40327-018-0064-7>: This paper has given me idea of using Energy Performance Indicator which is energy consumption of building during a definite period normalized by floor area. With this new feature I will have one more feature to enrich my data set. Also, it uses the same mythology of applying various ML models as my solution plan.
- <https://www.ashrae.org/File%20Library/Conferences/Specialty%20Conferences/2018%20Building%20Performance%20Analysis%20Conference%20and%20SimBuild/Papers/C013.pdf> : Even though paper aims to solve energy classification of buildings and recommend retrofits, it has also supported yearly EPI as an important feature.

Url: <https://www.kaggle.com/c/ashrae-energy-prediction>

1.2. Problem Statement

ASHRAE - Great Energy Predictor III Kaggle Competition is all about creating models to predict hourly consumption data on 4 types of energy consumption meters (electricity, chilled-water, steam, hot-water).

In the competition overview, competition host describes they have 3 years of data over 1000 buildings and their 4 types of consumption meters. Host has given 1 year of data as training data with labels and 2-year data for predictions as test. Final outcome will be predicted hourly meter readings which makes our problem as a complex regression problem.

As a problem statement host states that: building owners, financial intuitions and all related parties such as tenants has invested energy saving investments/retrofits. Therefore, buildings actual consumption data for last 2 years (scope of test data) has been changed. On the other hand, for comparison, buildings owners need estimation what would have been their consumption data if they hadn't invested efficiency retrofits. This is where competition predictions become important. Predictions will be used to measure efficiency of improvements on buildings. Which will lead to measure accuracy of investment decisions.

To solve a complex regression problem first I wanted to explore data sets given by host. As it will be described more thoroughly Data Exploration section, there are 5 data sets from and merging operations one training and one test data has been generated. It seems there are train and test data separately, but test data given by host is unlabeled competition test data. Although, it will be processed with train data in codes, unlabeled test data will be used only model Justification section in my project. I will exploratory analyze training data, preprocess it. Also, since our training data has over 20 million rows, I will be choosing a respectful size of sample from it and create train and test splits.

I am planning to choose 2 algorithms for feature importance determine most important features and use 3 or more regression algorithms as one for benchmark the others for model selection. After selection of most promising model, I will tune it and get final results according to metrics chosen. Finally, I will be comparing my final solution to benchmark model Linear Regression and expecting the solution I've came up will have better comparison metrics then benchmark model.

1.3. Metrics

Competition host uses root-mean-squared-logarithmic-error as its final evaluation metric. Also, some regression models do not support natively RMSLE, therefore I will consider root-mean-squared-error on implementation but final evaluation metric will be RMSLE as competition suggests.

$$\text{Root Mean Squared Logarithmic Error}^5 = \sqrt{\frac{1}{n} \sum_1^n (\log(y_{\text{pred}} + 1) - \log(y_{\text{test}} + 1))^2}$$

Root Mean Squared Logarithmic Error is a more robust metric for large data sets since it is more capable handling varied predictions which caused by outliers in data. It also favors overestimation to under estimations and handles large values for predictions which can be a useful feature for regression problems^{1, 2}. Therefore it has been using in data science competitions more and more and I think a very good metric for my project too.

2. ANALYSIS

2.1. Data Exploration

Like all Kaggle Competitions problem data sets are available at Kaggle ASHRAE - Great Energy Predictor III web page for Kaggle contestants. There are 5 csv of folders:

- train: Host's given training data. Shape: (20216100, 4)
- test: Host's given training data. Shape: (41697600, 4)
- building_metadata: building definitions data. Shape: (1449, 6)
- weather_train: weather data for training data points. Shape: (139773, 9)
- weather_test: weather data for test data points. Shape: (277243, 9)

Our data set has different csv files with different features. For EDA first thing to do is merging additional information to train and test data. After merging, train and test data will have 16 original features, 15 of which are identical for train and test data set. There is a target column in train data and index column in unlabeled test data which cause difference in data sets.

As stated in problem statement, I will use train data set for my project, I've decided to explore train data and apply same procedures to test data too. Unlabeled-test data will be used at 4.2. Justification section.

At first glance there are 1-datetime, 3-categorical, 11-numerical features and 1-numerical target variable at our data set. These features are:

- building_id: Buildings identification numbers. Although, it seems a numerical variable, for our solution I've considered it as categorical variable since it is only identification feature and there isn't any order between values in column. [0, 1448] (int32)
- meter: Meter type feature. Values defines electricity meter, chilled-water meter, steam meter, hot-water meter types. This feature is also a categorical variable for my solution. [0, 3] (int32)
- meter_reading: Exact value of every meter reading instance. This feature is target variable for my solution. [0, 3.57×10^6] (float64)
- Timestamp: Measurement time of target variable meter_reading. [01-01.2016, 31.12.2016] (Datetime)
- site_id: Regional identification number of buildings in building_id feature. This feature is also a categorical variable for my solution. [0, 15] (int32)
- primary_use: Describes use of the building educational, residential etc. Values are in string format but this is also a categorical feature for my solution. It will be encoded with label encoder for string to number conversion. (string)
- square_feet: Buildings' use area metric. It is a numerical feature. [273, 8.75×10^5] (int32)
- year_built: Buildings' opening year. It is a numerical feature. [1900, 2017] (int32)
- floor_count: Floor number of given building. It is a numerical feature. [1, 26] (int32)
- air_temperature: Air temperature in Celsius. It is a numerical feature. [-16.70, 28.29] (float64)
- cloud_coverage: Portion of sky covered by clouds in octas (a meteorology measurement unit). This feature is also a categorical variable for my solution because methodology of measurement is classifying sky as completely clear to completely cloudy with numbers 0 to 9 as degrees of cloudiness³. [0, 8] (int32)
- dew_temperature: the point where free moisture condenses⁴. It is a numerical feature. [-20, 21.09] (float64)

- precip_depth_1_hr: Amount of rain in one hour in millimeters. It is a numerical feature. [-1, 5] (float64)
- sea_level_pressure: Air pressure. It is a numerical feature. [993, 1038] (float64)
- wind_direction: Direction of wind according to compass degrees. Although it seems a numerical feature it describes direction of wind. Therefore, it is a categorical feature for my solution. [0, 360] (float64)
- wind_speed: Wind speed in meter per second. It is a numerical feature. [0, 14] (float64)

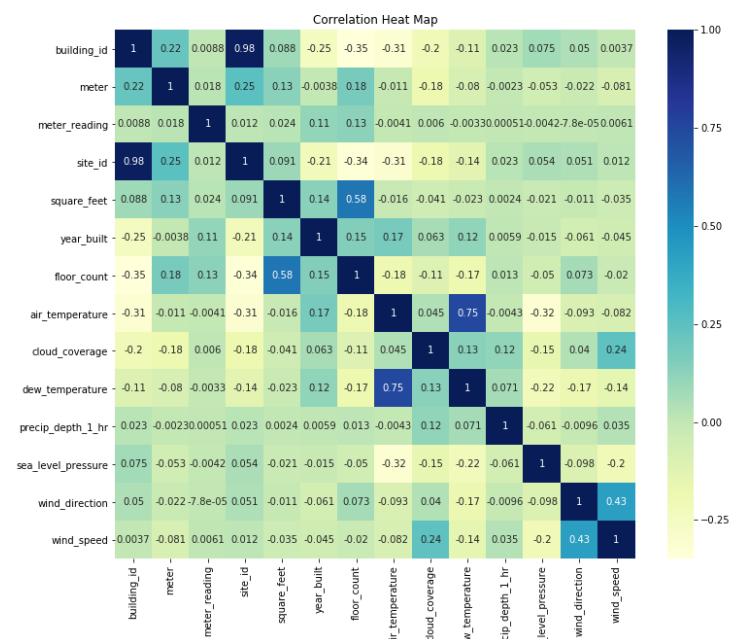
Finally, below table show my assignments to features and I will use these feature type in my solution.

Feature Name	Assigned Feature Type	Feature Name	Assigned Feature Type
building_id	Categorical	floor_count	Numerical
meter	Categorical	air_temperature	Numerical
timestamp	Datetime	cloud_coverage	Categorical
meter_reading	Target	dew_temperature	Numerical
site_id	Categorical	precip_depth_1_hr	Numerical
primary_use	Categorical	sea_level_pressure	Numerical
square_feet	Numerical	wind_direction	Categorical
year_built	Numerical	wind_speed	Numerical

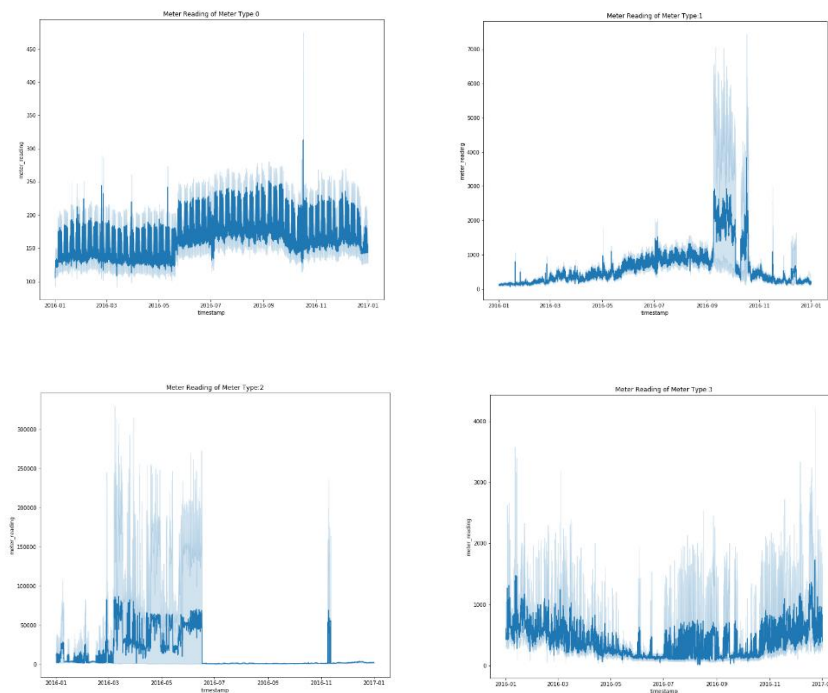
2.2. Exploratory Visualization

After categorizing my features, exploring their attributes and trying to find conclusions are very important before data preprocessing via visualization.

The below image shows our features heatmap. It indicates there are several relatively corelated features in data set. However, there isn't any important conclusions in it.

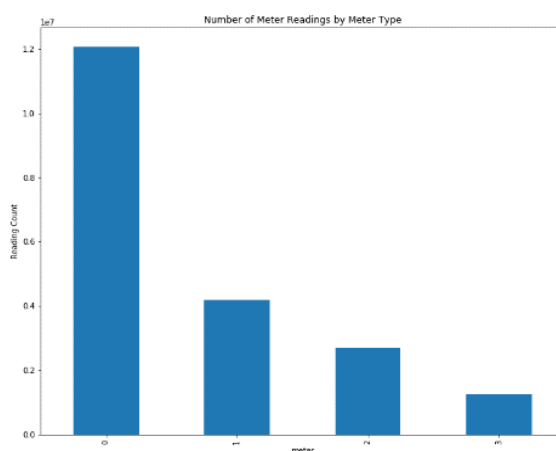


Our target variable `meter_reading` is simply reading instances from given meters on given timestamps. Therefore, I've decided to plot `meter_readings` according to meter types over time as shown below.



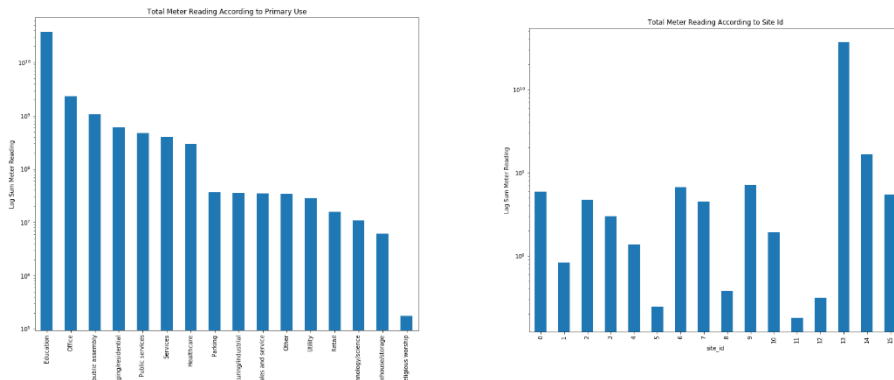
At first glance, except meter type 2, meter reading values are normal time series data with some outliers and seasonality. So it must be cleaned from outliers.

For meter type 2 there are some outlier reading but more importantly there are too much zero readings. So at preprocessing maybe zero values should be eliminated. Finally, our graphs display there is serious value difference our readings according to their types and our predictions too maybe varied numerically.



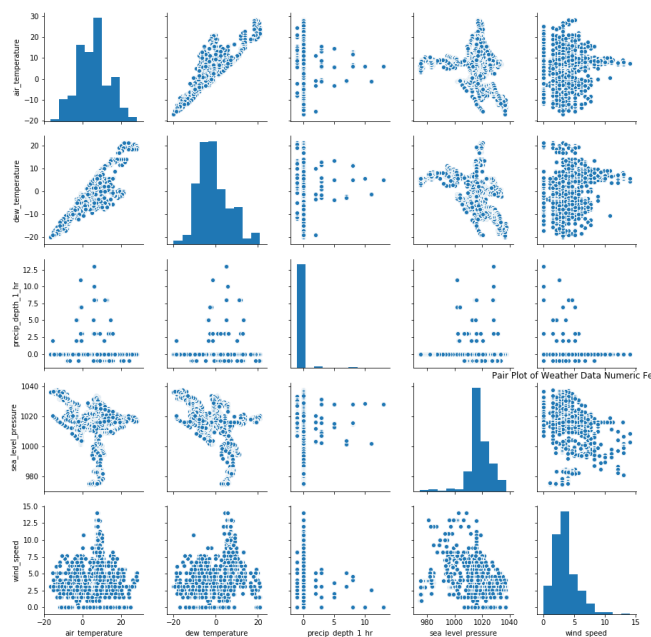
Number of Meter Readings by Meter Type graph shows `meter_reading` counts also varied over meter type and meter type will be an important categorical feature.

For our categorical variables I've decided to examine primary_use and site_id's relations with meter readings. Below graphs were plotted for this reason.



For both categorical feature it seems there are some clear insight which can be useful for candidate solutions, such as educational buildings has highest total meter_reading sum or site_id 13 has more meter_reading instances than any other site.

Finally I've decided to examine numerical weather data. To do this I've plotted a gridgraph with %1 of the data since it is very hard to plot 20 million data points. Therefore I've only used first 200.000 data points for below graph:



Grid plot showed similar results with heat map. Air_temperature and dew_temperature data are correlated. There isn't any other correlation. Also, besides precip_depth_1_hr feature distributions seems relatively normal.

2.3. Algorithm and Techniques

I've decided 3 parts for my implementation which requires different algorithms for different purposes.

First, I will use Random Forest and LightGBM regression models to determine feature importance by averaging their solutions. These algorithms are chosen for their capability to handle both categorical and numerical data well. At last, I will use most important features at my second part of implementation.

For second part, I will choose %10 of my original data, which has more than 20 million of data points, for decreasing computational complexity. Also, I will only use selected features at the first step of implementation.

Then I will train, Linear Regression as benchmark, LightGBM Regressor as candidate 1, XGBoost Regressor as candidate 2 and a MLP Regressor for candidate 3.

Below summarizes the table of algorithms and purpose of use first 2 steps:

Implementation Stage	Linear Regression	Random Forest	LightGMB	XGBoost	MLP
1	-	Feature Importance	Feature Importance	-	-
2	Benchmark	-	Candidate 1	Candidate 2	Candidate 3

Finally, I will be selecting most promising algorithm and tune it for final prediction as third/justification part of my implementation plan.

2.4. Benchmark

My benchmark algorithm for this project is Linear Regression. It is simplest and basic but widely used algorithm for regression problems.

I think it will be a good competitor for my candidate algorithms which are relatively new on machine learning scene such as LightGMB or XGBoost regressors.

3. METHODOLOGY

3.1. Data Preprocessing

During data preprocessing I've made several conversions, created new features and dropped a few one. These preprocessing steps are:

- Meter Reading Site ID 0 and Meter 0 Unit Conversion

As stated at competition discussion part, site_id 0 and meter 0 has wrong type of meter reading values. Therefore, I've corrected it as seen below:

```
# Correction on Site 0's electric meters units.  
# https://www.kaggle.com/c/ashrae-energy-prediction/discussion/119261  
  
df_X.loc[(df_X['site_id'] == 0) & (df_X['meter'] == 0), 'meter_reading'] *= 0.2931
```

- Adding Time and Holiday/Weekend Features

Our data has a timestamp column. I've decided to derive year, month, day of week and hour features from time stamp. Also holidays and weekends can affect consumptions. So, I've added two binary columns is_holiday and is_weekend for UK and USA since most of the buildings from these countries. Finally dropped timestamp column as seen below:

```
# Let's break down our timestamp data and drop datetime type timestamp
# Holidays for us and uk will be considered to create a is a holiday or not binary column. Since at discussion posts
# sites were disclosed as usa and uk based.
# An is_weekend binary column was created for differentiate bussiness days and weekends.

import holidays

us_holidays = holidays.US(years=[2016, 2017, 2018])
uk_holidays = holidays.UK(years=[2016, 2017, 2018])

us_uk_total_holidays = us_holidays + uk_holidays

df_X['year'] = df_X['timestamp'].dt.year
df_X['month'] = df_X['timestamp'].dt.month
df_X['dayofweek'] = df_X['timestamp'].dt.dayofweek
df_X['hour'] = df_X['timestamp'].dt.hour
df_X['is_holiday'] = df_X['timestamp'].map(us_uk_total_holidays).fillna(0).replace(list(us_uk_total_holidays.values()),1)
df_X['is_weekend'] = df_X['dayofweek'].map({0:0, 1:0, 2:0, 3:0, 4:0, 5:1, 6:1})

df_X_given_test['year'] = df_X_given_test['timestamp'].dt.year
df_X_given_test['month'] = df_X_given_test['timestamp'].dt.month
df_X_given_test['dayofweek'] = df_X_given_test['timestamp'].dt.dayofweek
df_X_given_test['hour'] = df_X_given_test['timestamp'].dt.hour
df_X_given_test['is_holiday'] = df_X_given_test['timestamp'].map(us_uk_total_holidays).fillna(0).replace(list(us_uk_total_holidays.values()),1)
df_X_given_test['is_weekend'] = df_X_given_test['dayofweek'].map({0:0, 1:0, 2:0, 3:0, 4:0, 5:1, 6:1})

df_X.drop('timestamp', axis=1, inplace=True)
df_X_given_test.drop('timestamp', axis=1, inplace=True)

#print(df_X.head())
#print(df_X_given_test.head())
```

- Adding Energy Use Intensity (EUI) Feature

From papers discussed at 1.1. Project Overview energy consumption of building during a definite period normalized by floor area used to express the performance (kWh/m2/period) known as Energy Performance Indicator (EPI) or Energy Use Intensity (EUI). Also, EUI is expressed as energy per square foot per year more commonly. I've created an EUI estimation column for yearly meter readings divided by square_foot column to enrich our feature list as below:

```
# From paper given in proposal: Generally, the energy consumption of building during a definite period normalised
# by floor area is used to express the performance(kWh/m2/period) known as Energy Performance Indicator (EPI)
# or Energy Use Intensity(EUI). Also EUI is expressed as energy per square foot per year more commonly.
# We can create an EUI estimation column for yearly meter readings divided by square_foot column to enrich our feature list.

df_X['eui_2016'] = df_X.groupby(['building_id', 'year'])['meter_reading'].transform('sum')/df_X['square_feet']
d_eui = df_X.set_index('building_id')['eui_2016'].to_dict()

df_X_given_test['eui_2016'] = df_X_given_test['building_id'].map(d_eui)

#df_X.head()
#df_X_given_test.head()
```

- Add Semester Feature

My data has primary_use equals to education feature points. Therefore, I've decided to create a feature which will separate summer break to semester. But our data has sites only as id's I used general assumption that June, July and August are summer break months and others are semester months as below:

```
# I've decided ad a new column as is_education_semester

df_X['is_semester_on'] = df_X['month'].map({1:1, 2:1, 3:1, 4:1, 5:1, 6:0, 7:0, 8:0, 9:1, 10:1, 11:1, 12:1})
df_X_given_test['is_semester_on'] = df_X_given_test['month'].map({1:1, 2:1, 3:1, 4:1, 5:1, 6:0, 7:0, 8:0, 9:1, 10:1, 11:1, 12:1})
```


- Add Season Feature

I've decided to add a season feature {0: Winter, 1: Spring, 2: Summer, 3: Autumn} as below:

```
# I've decided ad a new column as season

df_X['season'] = df_X['month'].map({12:0, 1:0, 2:0, 3:1, 4:1, 5:1, 6:2, 7:2, 8:2, 9:3, 10:3, 11:3})
df_X_given_test['season'] = df_X_given_test['month'].map({12:0, 1:0, 2:0, 3:1, 4:1, 5:1, 6:2, 7:2, 8:2, 9:3, 10:3, 11:3})
```

- Dropped floor_count Feature

floor_count feature had 82% empty rows so I've decided to drop our data as below:

```
# floor_count will be dropped since it has %82 nan values and binarizing it seems has no gain.

df_X.drop('floor_count', axis=1, inplace=True)
df_X_given_test.drop('floor_count', axis=1, inplace=True)
```

- Creating an Age Column and Dropping year_built Feature

year_built feature has near 60% percent emptiness. Also I've thought an age feature would suit my algorithms since it suggests an increasing order between buildings. So I've decided to create an age column with filling missing values with mode value to equalize unknown ages to most common value and then dropped year_built feature as below:

```
# year built has near %60 null values. Since it is a year variable i've decided to impute it with mode and
# create an age column. Then i dropped year_built.

df_X['year_built'].fillna(int(df_X['year_built'].mode()), inplace=True)
df_X_given_test['year_built'].fillna(int(df_X_given_test['year_built'].mode()), inplace=True)

df_X['age'] = df_X['year'] - df_X['year_built']
df_X_given_test['age'] = df_X_given_test['year'] - df_X_given_test['year_built']

df_X.drop('year_built', axis=1, inplace=True)
df_X_given_test.drop('year_built', axis=1, inplace=True)

print(df_X['age'].describe())
print(df_X_given_test['age'].unique())
```

- Dropping year Feature

Since our data set only covers 2016 and unlabeled test data given by host covers 2017 and 2018, I've decided to drop year column not to affect our regression algorithms as below:

```
# Since we break down hour time stamp into categorical variables and there is a year difference in train and test data
# i've decided to drop year column from two data sets.

df_X.drop('year', axis=1, inplace=True)
df_X_given_test.drop('year', axis=1, inplace=True)
```

- Outlier Elimination with Confidence Level Definition and Value Equalization

Our numerical features and target variable meter_reading have outlier values on both ends of their quantiles. Therefore, I've decided to chose a confidence level then change values of rows below and upper this confidence level to given quantile value as below:

```
# After feature add and drop operations i've decided to choose %99 confidence interval of our numerical features
# to eliminate zero values and some outliers by setting values lower of higher to quantile values.

confidence_interval = 0.99
quantile_low_end = (1-confidence_interval)/2
quantile_high_end = 1 - quantile_low_end

for feature in l_numerical_features:
    df_X.loc[(df_X[feature]<=df_X[feature].quantile(quantile_low_end)), feature] = df_X[feature].quantile(quantile_low_end)
    df_X.loc[(df_X[feature]>=df_X[feature].quantile(quantile_high_end)), feature] = df_X[feature].quantile(quantile_high_end)

# Also i've applied confidence interval high and low end value correction according to meter type for our target variable.
d_meter_le = df_X.groupby('meter')['meter_reading'].quantile(quantile_low_end).to_dict()
d_meter_he = df_X.groupby('meter')['meter_reading'].quantile(quantile_high_end).to_dict()

for key, value in d_meter_le.items():
    df_X.loc[(df_X['meter'] == key) & (df_X['meter_reading'] <= value), 'meter_reading'] = value

for key, value in d_meter_he.items():
    df_X.loc[(df_X['meter'] == key) & (df_X['meter_reading'] >= value), 'meter_reading'] = value
```

- Imputation of Categorical Features

I've add a new category unknown to all my categorical features as below:

```
# Let's mark categorical features and fill them with 'unknown'

for feature in l_categorical_features:
    df_X[feature].fillna('unknown', inplace=True)
    df_X_given_test[feature].fillna('unknown', inplace=True)

df_X[l_categorical_features] = df_X[l_categorical_features].astype('object')
df_X_given_test[l_categorical_features] = df_X_given_test[l_categorical_features].astype('object')

print(df_X.info())
print(df_X['is_holiday'].unique())
```

- Imputation of Numerical Features

I've decided to impute numerical features with median values as below:

```
# Let's fill our numerical features with mean values. Because they have relatively low empty percentage.
for column in df_X.columns:
    if df_X[column].dtype in ['int64', 'float64'] and df_X[column].isna().sum() != 0:
        df_X[column] = df_X[column].fillna(df_X[column].median())

for column in df_X_given_test.columns:
    if df_X_given_test[column].dtype == 'float64' and df_X_given_test[column].isna().sum() != 0:
        df_X_given_test[column] = df_X_given_test[column].fillna(df_X_given_test[column].median())

print(df_X.info())
print(df_X['wind_speed'].unique())
```

- Label Encoding

I've used Label Encoding to convert string data in categorical features to numerical ones as below:

```
# for memory saving i've decided to encode dfs to int from string
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
for feature in l_categorical_features:
    df_X[feature] = encoder.fit_transform(df_X[feature].astype(str))
    df_X_given_test[feature] = encoder.fit_transform(df_X_given_test[feature].astype(str))

print(df_X['primary_use'].unique())
```

3.2. Implementation

I've divided my implementation plan into 3 parts. Feature Selection, Model Selection and Refinement. Feature and Model Selection are covered at this section.

Feature Selection

Result of preprocessing, my problem data have 21 features and 1 target column. Also as covered data analysis part my problem data set consists more than 20 million data points. This is a very complex and heavy data set for any regression algorithm. Therefore, I've decided to made a feature selection process to feed regression algorithms with most important features before predictions.

For feature selection, I've chosen 2 algorithms Random Forest Regressor and LightGBM Regressor. The reason behind my choice is these two algorithms are well equipped to handle both numerical and categorical data which are two type of my features.

I had a simple coding approach for feature selection operation. After reading preprocessed data, I did create train and test splits with 25% test set and fit RF and LGBM Regressors. Then, I've used RMSLE metric to look at is there overfitting. Finally, I've compiled RF and LGMB Feature Importance values and take their average as final feature importance.

The results are shown in below Pycharm IDE output.

```
Run: aep-feature_selection x
C:\Users\bilge\Anaconda3\python.exe "C:/Users/bilge/Desktop/Bilge Python
C:\Users\bilge\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:
"10 in version 0.20 to 100 in 0.22.", FutureWarning)
RF RMSLE Train: 1.998958138331942
RF RMSLE Test: 1.989438419815646

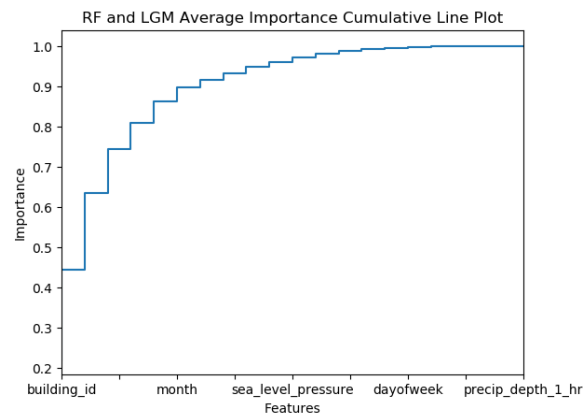
LGBM RMSLE Train: 1.6494077823588387
LGBM RMSLE Test: 1.6490141852293436

          RF      LGBM  Average_Importance
features
building_id  0.066266  0.378333      0.222300
meter        0.296607  0.146667      0.221637
square_feet  0.321734  0.059333      0.190534
eui_2016     0.187933  0.031667      0.109800
air_temperature 0.038148  0.092667      0.065407
month        0.009094  0.099333      0.054214
dew_temperature 0.007444  0.058333      0.032889
season       0.023462  0.015667      0.019564
hour         0.000000  0.034667      0.017333
primary_use  0.030189  0.000333      0.015261
sea_level_pressure 0.000000  0.024000      0.012000
site_id      0.018924  0.003667      0.011295
wind_direction 0.000000  0.020333      0.010167
is_weekend   0.000000  0.011333      0.005667
is_semester_on 0.000000  0.009667      0.004833
dayofweek    0.000000  0.007333      0.003667
wind_speed   0.000000  0.004667      0.002333
age          0.000199  0.001000      0.000599
cloud_coverage 0.000000  0.001000      0.000500
is_holiday   0.000000  0.000000      0.000000
precip_depth_1_hr 0.000000  0.000000      0.000000

Process finished with exit code 0
```

As seen from the figure train and test RMSLE metrics both of algorithms performed well. There is no clear sign for over or under fitting.

Examining feature importance results also seems consistent for most of the features. Inconsistencies are resolved by our average importance calculations and its cumulative graph shown below:



There are two significant drops on our average importance column. First one is between 4th and 5th, second one is between 13th and 14th features. Their cumulative importance percentages are 71.43% for first 4 feature and 98.24% for first 13 feature. So, I've decided to select first 13 features as most important features and use them in model selection part.

Final result of this feature selection part are two lists as follows:

```
I_features_will_be_used = [building_id, meter, square_feet, eui_2016, air_temperature,
month, dew_temperature, season, hour, primary_use, sea_level_pressure, site_id,
wind_direction]
```

```
I_features_will_be_dropped = [is_weekend, is_semester_on, dayofweek, wind_speed, age,
cloud_coverage, is_holiday, precip_depth_1_hr]
```

Model Selection

From the beginning, I had the idea of using new generation algorithms for my final solution. For this purpose, I've chosen LightGBM, XGBoost and MLP regression algorithms. LightGBM and XGBoost are relatively new and hyped algorithms in data science and machine learning community. MLP Regressor is also a new implementation in sklearn toolkit as a supervised neural network model.

Apart from algorithm selection in my implementation code, I used results of feature selection models. An important decision I've made for this section was sampling. Even though I decreased feature number from 21 to 13 by average importance, my data is still computationally expensive. I decided to take 10% of my data as model selection sample. So my model selection data was 2 million data points to 13 features and 1 target variable.

My coding approach was again simple. I've read the original data from preprocessing results. Dropped least important features. Create 3 sets of train and test data since:

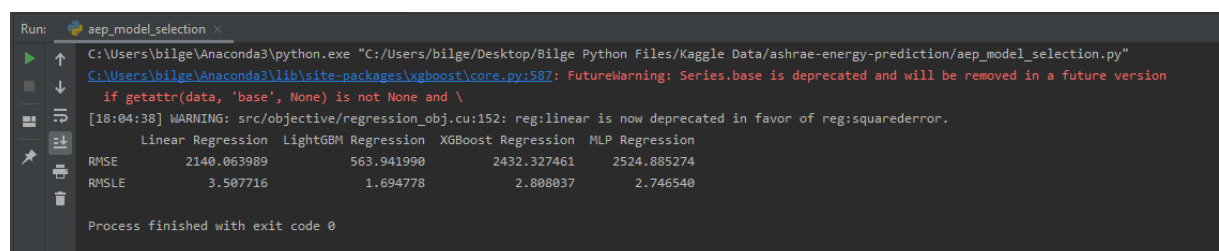
- LightGBM Regressor uses unencoded and unscaled data
- XGBoost Regressor best performs on encoded but unscaled data
- Linear Regression (Benchmark) and MLP needs both encoded and scaled data.

Also I want to mention that I've used two different techniques for encoding not to increase data complexity. For all categorical features I've used pandas get_dummies method besides building_id feature.

building_id feature is simply id number of building. It is a categorical variable in my solution. But get_dummies method creates over 1000 new columns. To avoid this increase in data size, I encoded building_id column with weight of evidence (WOE) encoding. It is simply encoding feature values with mean of target feature data points associated with that feature value. Coding was as follows:

```
39 # Categorical Encoding with WoE Encoding building_id and dummy encoding other categorical features
40 X_encoded = X
41 X_encoded['building_id'] = X_encoded['building_id'].map(df_X.groupby('building_id')['meter_reading'].mean())
42 X_encoded = pd.get_dummies(X_encoded, drop_first=True)
43 X_encoded['building_id'] = X_encoded['building_id'].astype('category')
44 del df_X
```

After created 3 train test splits for algorithms, I defined regressor instances and decided not to tune any parameters. Rest was pretty straightforward. I've trained algorithms and use RMSE and RMSLE for comparison metric. The results are shown in below Pycharm IDE output.



```
Run: aep_model_selection
C:\Users\bilge\Anaconda3\python.exe "C:\Users\bilge\Desktop\Bilge Python Files\Kaggle Data\ashrae-energy-prediction\aep_model_selection.py"
C:\Users\bilge\Anaconda3\lib\site-packages\xgboost\core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
if getattr(data, 'base', None) is not None and \
[18:04:38] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Linear Regression  LightGBM Regression  XGBoost Regression  MLP Regression
RMSE      2140.063989      563.941990      2432.327461      2524.885274
RMSLE      3.507716      1.694778      2.808037      2.746540
Process finished with exit code 0
```

The results show that LightGBM has lower results on both RMSE and RMSLE against Linear, XGBoost and MLP regressors. Especially lower result on RMSLE is very favorable since my regression problem has both very high and very low target values.

After considering these results, I've chosen LightGBM algorithms because of best performance on model selection run and its on-built categorical data handling. Then I've started 3rd part of my solution plan tuning algorithm and getting final results.

3.3. Refinement

3rd part of my problem solution plan is tuning final model and training in my entire data set which has over 20 million data points. At model selection part I've chosen LightGBM algorithm as final model.

For this part my coding approach was similar to model selection. I've read the preprocessed data. Dropped least important features and create train and test data as 75%-25% respectively. Also I've decided to select tuning parameters of algorithm as follows:

```
params = {'task': 'train', 'objective': 'regression', 'metric': 'rmse', 'boosting_type': 'gbdt',
          'num_boost_round': 256, 'early_stopping_rounds': 50, 'verbose': 1,
          'learning_rate': 0.05, 'num_leaves': 256, 'feature_fraction': 0.8, 'bagging_fraction': 0.8}
```

The results are shown in below Pycharm IDE output.

```

Run: aep_lightgbm_model
[254] train's rmse: 362.011 eval's rmse: 412.561
[255] train's rmse: 361.854 eval's rmse: 412.5
[256] train's rmse: 361.691 eval's rmse: 412.36
Did not meet early stopping. Best iteration is:
[256] train's rmse: 361.691 eval's rmse: 412.36
Lightgbm Regression RMSE Train: 361.17393837023457
Lightgbm Regression RMSLE Train: 1.3708334066849859
Lightgbm Regression RMSE Test: 411.8319917842606
Lightgbm Regression RMSLE Test: 1.3716338231344307
importance
building_id 10382
air_temperature 9353
sea_level_pressure 9028
dew_temperature 8764
square_feet 5144
month 5144
wind_direction 4162
hour 3869
eui_2016 3080
season 2649
meter 2644
site_id 597
primary_use 464
['importance']
Process finished with exit code 0

```

LGBM regressor worked for 256 boosting rounds and did not meet an early stopping point. Final metrics are:

- LGBM RMSE Train = 361.1739
- LGBM RMSLE Train = 1.3708
- LGBM RMSE Test = 411.8320
- LGBM RMSLE Test = 1.3716

These results are better than our model selection run. Also, since it did not meet early stopping it has still potential for improvement by increasing boost round number. But for my problem solution these results are acceptable and my solution is ready for the comparison with benchmark model Linear Regression.

4. RESULTS

4.1. Model Evaluation and Validation

My final solution to ASHRAE - Great Energy Predictor III Kaggle Competition, is using LightGBM Regression as discussed on 3.2. Refinement section. With tuned parameters and train and testing on (20216100, 13) shaped data its test metrics are:

- LG BM RMSE Test= 411.8320
- LG BM RMSLE Test = 1.3716

To achieve these results:

num_boost_round was chosen as 256 to keep computation time relatively low. On the other hand, since our final run did not meet an early stopping it can be increased by carefully considering overfitting possibility.

learning_rate was determined as 0.05 since lower learning rates result more robust improvements during iterations.

num_leaves was set to 256 to increase accuracy since our data set is very large.

feature_fraction and bagging_fraction were decided as 0.8 to lower overfitting possibility by choosing 80% percent of data before each iteration.

To test robustness of my solution I've decided to change num_boost_round parameter increase/decrease and compare metric results as shown below table:

	NBR = 128	NBR = 256	NBR = 512
RMSLE Train	1.5007	1.3708	1.3019
RMSLE Test	1.5005	1.3716	1.3035

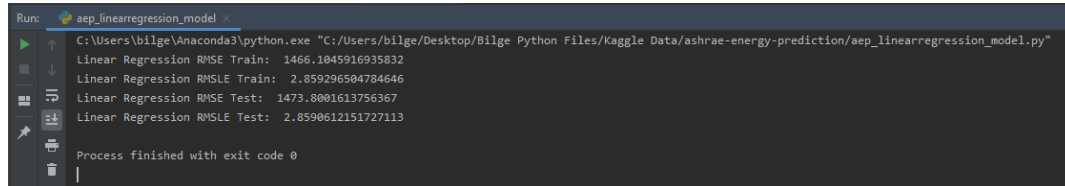
After examining above results I can conclude that decreasing num_boost_round metric results increase validation metric and worseness our predictions. On the other hand, increasing parameter to 512 decreases our validation metric but it increased my computation time significantly. Therefore, I've concluded num_boost_round 256 is my final solution.

4.2. Justification

Final part of my project is comparing my tuned model and metrics to selected benchmark of the project.

At section 2.4 I've chosen my benchmark algorithm for the project as Linear Regression. After tuning and finalizing my solution I've applied Linear Regression to problem data set.

The results are shown in below Pycharm IDE output.

A screenshot of the PyCharm IDE's Run output window. The window title is 'Run: aep_linearregression_model'. The output text shows the execution of a Python script using the Anaconda3 interpreter. The results displayed are: Linear Regression RMSE Train: 1466.1045916935832, Linear Regression RMSLE Train: 2.859296504784646, Linear Regression RMSE Test: 1473.8001613756367, and Linear Regression RMSLE Test: 2.8590612151727113. The window concludes with 'Process finished with exit code 0'.

```
Run: aep_linearregression_model
C:\Users\bilge\Anaconda3\python.exe "C:/Users/bilge/Desktop/Bilge Python Files/Kaggle Data/ashrae-energy-prediction/aep_linearregression_model.py"
Linear Regression RMSE Train: 1466.1045916935832
Linear Regression RMSLE Train: 2.859296504784646
Linear Regression RMSE Test: 1473.8001613756367
Linear Regression RMSLE Test: 2.8590612151727113
Process finished with exit code 0
```

Linear regression algorithm also performed well on our data. It did not indicate any overfitting on training data and its RSMLE metric was reasonably low.

On the other hand, when compared to my final solution it has almost three times higher on root mean squared error and more than two times higher than root mean squared error.

	Benchmark	Final Solution
RSME	1473.8001	411.8320
RSMLE	2.8591	1.3716

In summary, these results clearly show that my solution performs really well against other algorithms both model selection and benchmarking and it solves my defined problem.

Next step on this part is making prediction on unlabeled competition train data and submitting it to competition. However, competition's Kaggle due date has expired during my solution studies. I am waiting to Kaggle leaderboard late submission to open to see my solutions robustness against other competitors.

APPENDICES

- 1- <https://medium.com/analytics-vidhya/root-mean-square-log-error-rmse-vs-rmlse-935c6cc1802a>
- 2- <https://www.quora.com/What-is-the-difference-between-an-RMSE-and-RMSLE-logarithmic-error-and-does-a-high-RMSE-imply-low-RMSLE>
- 3- <https://en.wikipedia.org/wiki/Okta>
- 4- https://en.wikipedia.org/wiki/Dew_point
- 5- https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-log-error
- 6- <https://scikit-learn.org>
- 7- <https://pandas.pydata.org/pandas-docs/stable/>
- 8- <https://seaborn.pydata.org/index.html>