ASHRAE - Great Energy Predictor III

Springboard - Capstone II

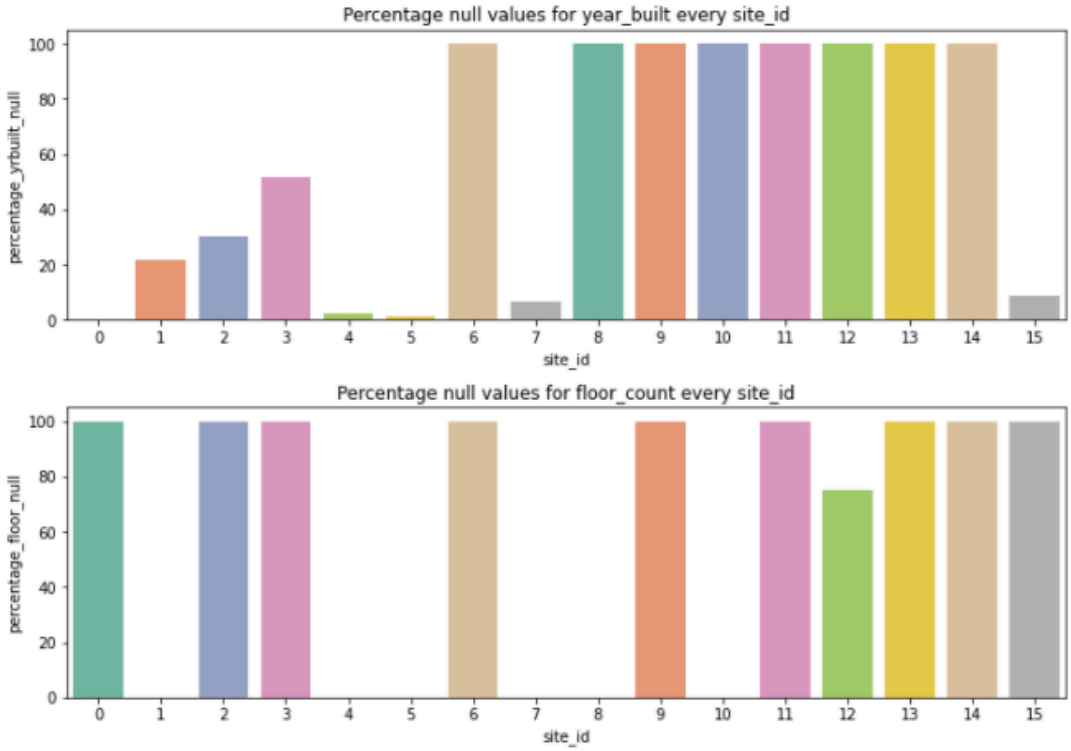
Problem Statement

Develop accurate models of metered building energy usage where data comes from over 1,000 buildings over a three-year timeframe across various locations to get a better estimate of energy-saving investments.

Data Wrangling & EDA

- Data sources: https://www.kaggle.com/c/ashrae-energy-prediction/data

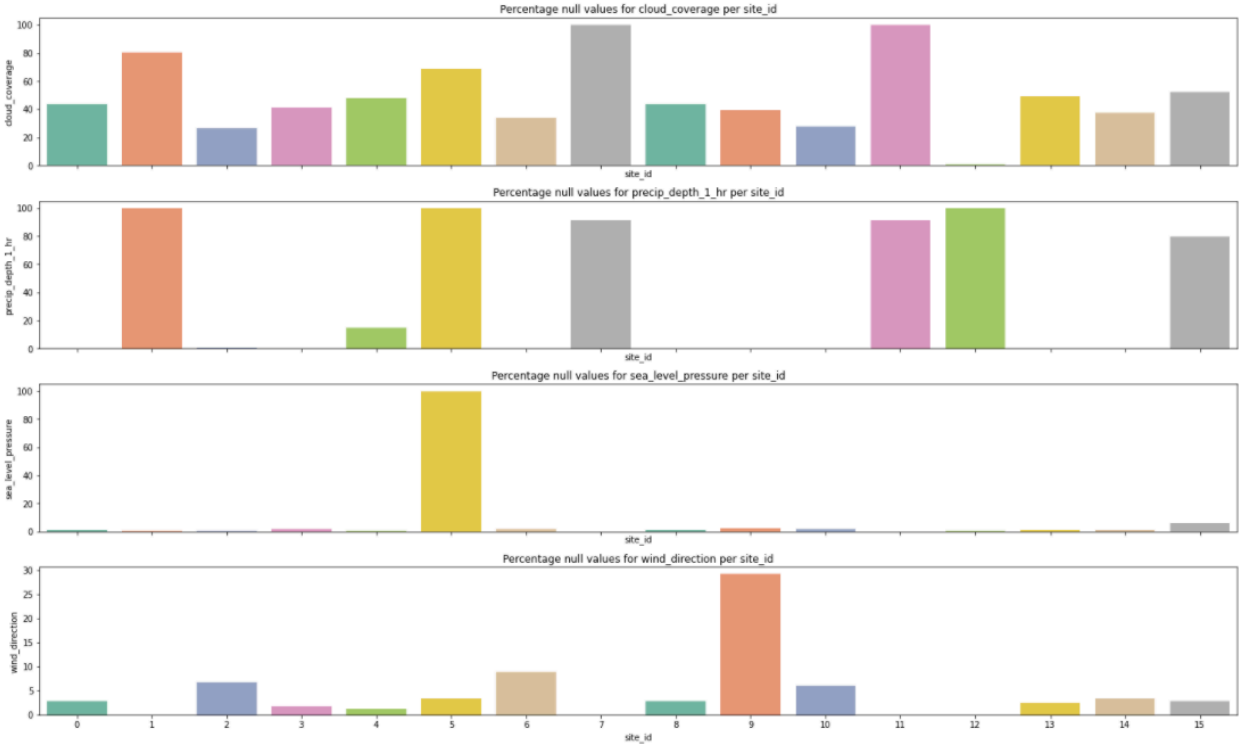
1. Missing data analysis of Building metadata



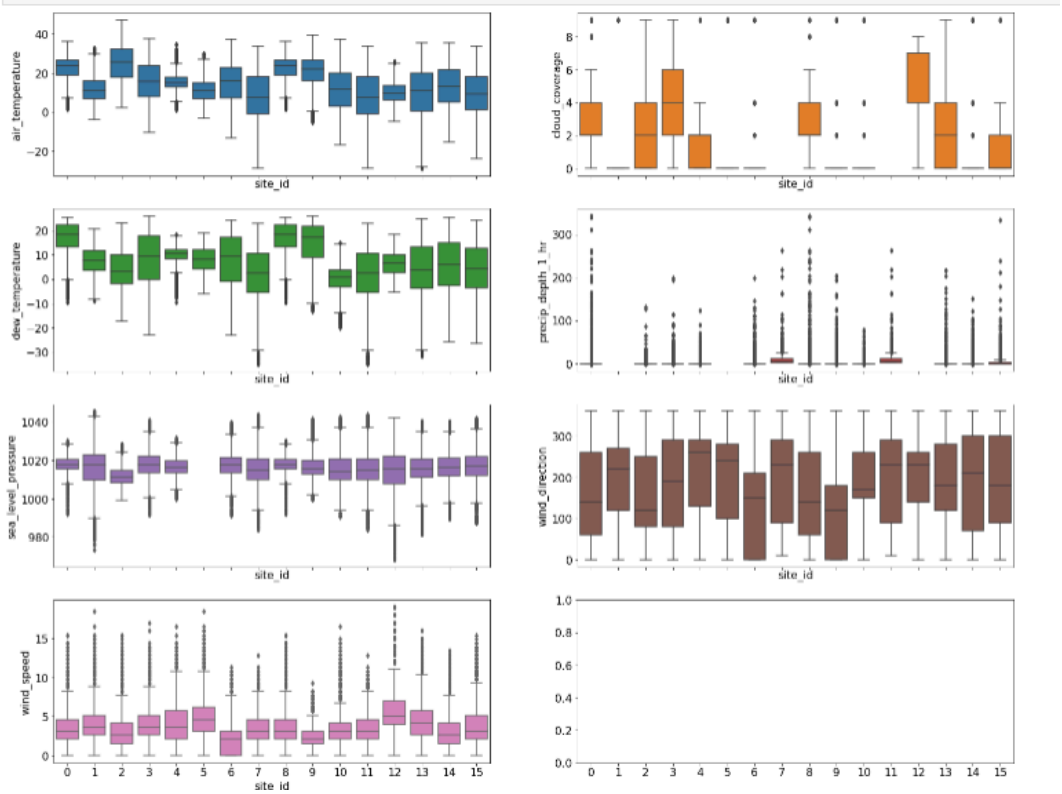
Only for site id 12 can we infer some information from other buildings .
Rest all sites have either null values for all buildings or none at all.

Since too many buildings have year_built NaN we are going to leave the column as is.
Same is the case with floor_count

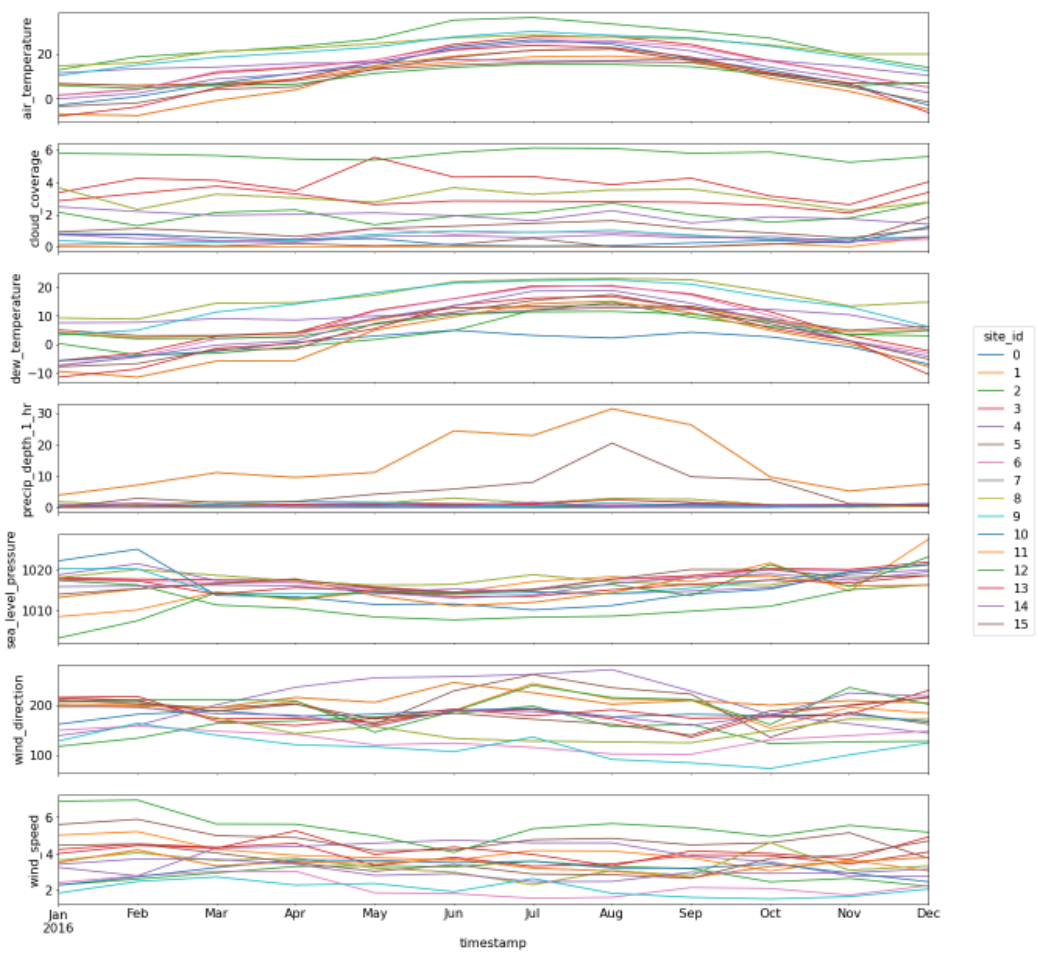
2. Weather data - outliers & missing data interpolation



Looks like we are dealing with sparse data especially with wind_speed and percip_depth_1hr.
sparsity = count zero elements / total elements.



We converted weather data to time series and interpolated missing data linearly.



Before interpolation

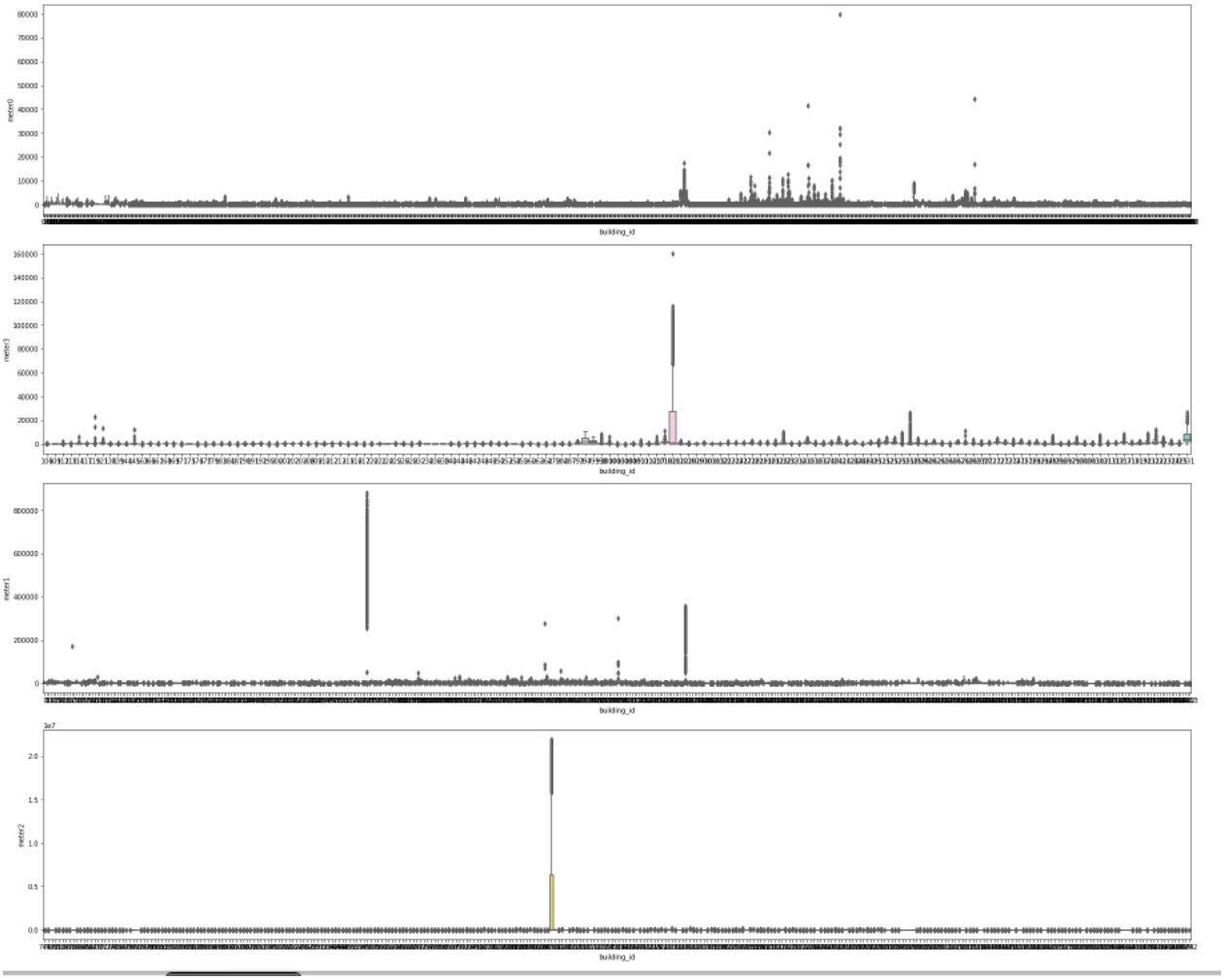
```
Percentage null values for the weather_train_df:
site_id                0.000000
timestamp              0.000000
air_temperature        0.039350
dew_temperature        0.080845
wind_speed             0.217496
wind_direction         4.484414
sea_level_pressure     7.596603
precip_depth_1_hr     35.979052
cloud_coverage        49.489529
dtype: float64
```

After interpolation

```
Percentage null values for the weather_train_df:
air_temperature        0.00
dew_temperature        0.00
wind_direction         0.00
wind_speed             0.00
sea_level_pressure     6.25
cloud_coverage        12.50
precip_depth_1_hr     18.75
dtype: float64
```

3. Meter data - outliers & missing data interpolation

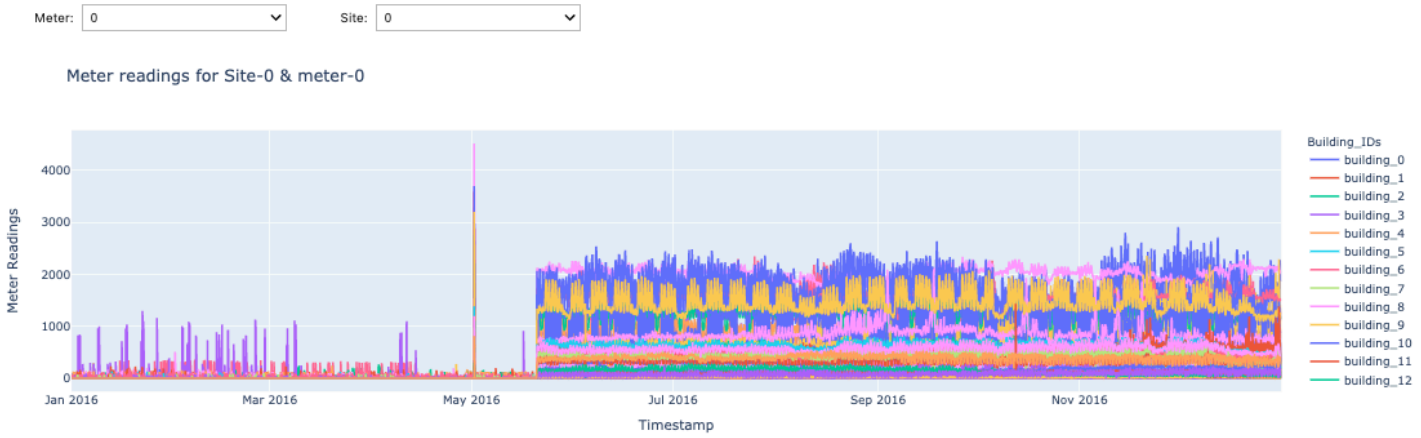
As we can see below , there are some clear outliers clear outliers for meter 0 & meter 3.



This data needs to be analysed at the site level.

With 15 sites & 4 meter types it is difficult to analyse this with a static visualisation.

Hence plotly express & ipywidgets is used to spot outliers or incorrect readings in the data.



Based of the finding from the above plotly visualisation we find outliers and sites with missing meter data. We remove such incorrect data before feature engineering.

3.3.0.1. remove site 0 incorrect data

```
len(train_build_df[(train_build_df.site_id == 0) & (train_build_df.meter == 0) & (train_build_df.timestamp < '2016-05-20')\
& (train_build_df.meter_reading == 0.0)])/\
len(train_build_df[(train_build_df.site_id == 0) & (train_build_df.meter == 0) & (train_build_df.timestamp < '2016-05-20')])
```

```
0.9902719350961539
```

As seen above 99% of the meter reading for all buildings before may20,2016 at site 0 is 0.0.
we need to drop this incorrect data

```
train_build_df = train_build_df[~((train_build_df.site_id == 0) & (train_build_df.meter == 0) & (train_build_df.timestamp < '2016-05-20')\
& (train_build_df.meter_reading == 0.0))]
```

...

3.3.0.2. Site1 meter 1 building 60

```
indx = train_df.query('(building_id == 60) & (meter == 1) & (meter_reading > 150000)').index
```

```
train_df.drop(indx,inplace=True)
```

3.3.0.3. Site6 meter1 building778

```
indx_list = train_df.query('(building_id == 778) & (meter == 1) & (meter_reading ==0)').index.to_list()
```

```
train_df.drop(indx_list,inplace=True)
```

3.3.0.4. Site13 meter2 building1099

```
indx_list = train_df.query('(building_id == 1099) & (meter == 2)').index.to_list()
```

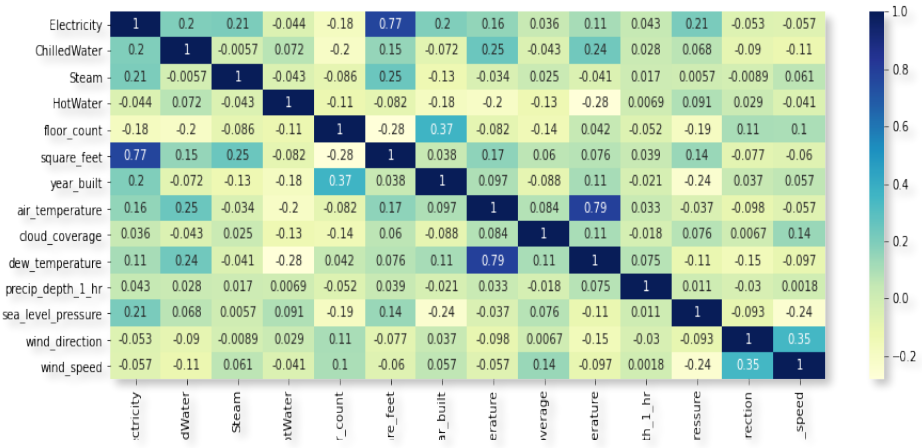
```
train_df.drop(indx_list,inplace=True)
```

3.3.0.5. Site13 meter1 building1088

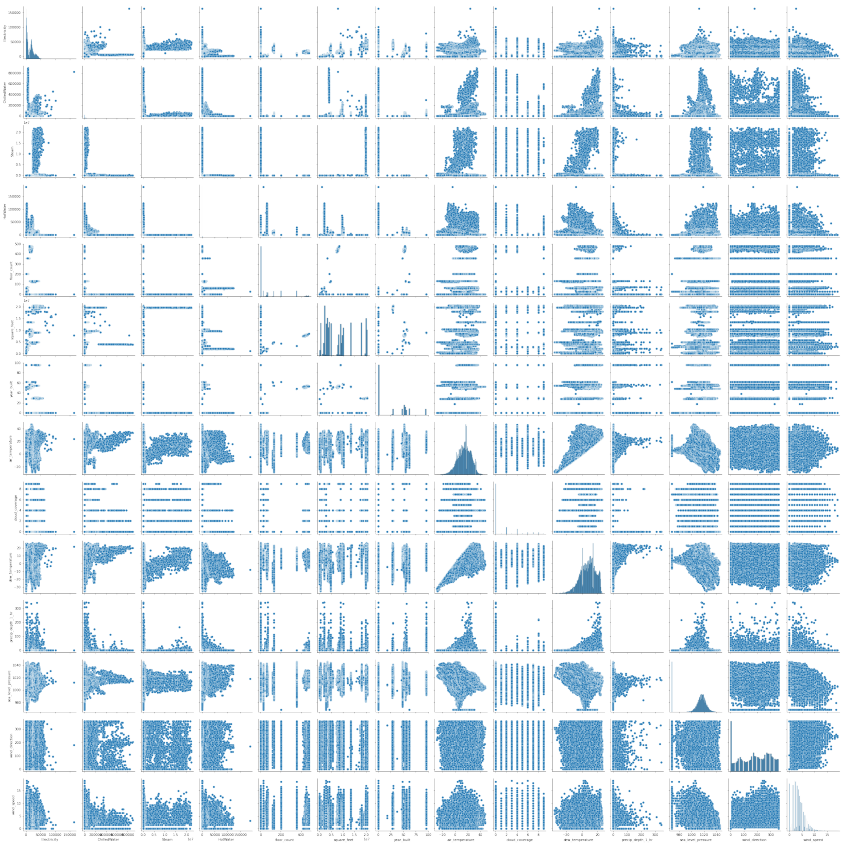
```
indx_list = train_df.query('(building_id == 1088) & (meter == 1)').index.to_list()
```

```
train_df.drop(indx_list,inplace=True)
```

4. Meter and weather Correlation Plots



5. Meter and weather Pair-Plot



Feature Engineering

1. Leak Data

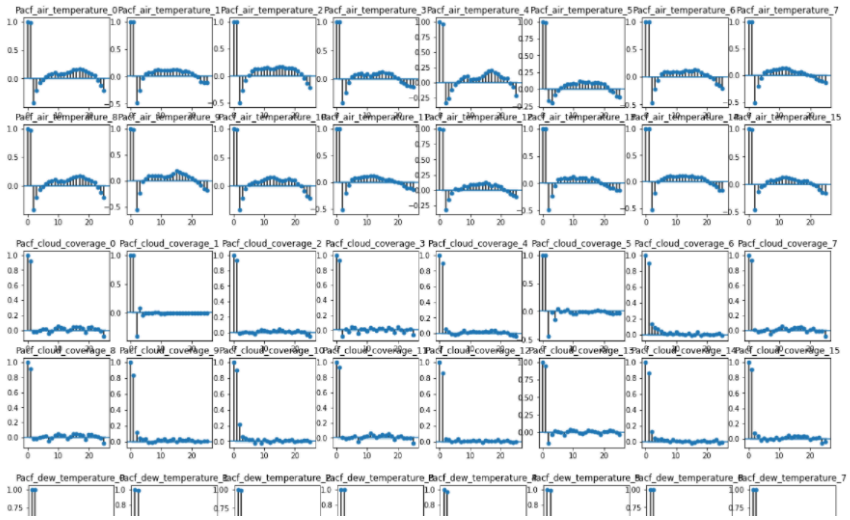
The dataset provided in Kaggle isn't complete and supplementary leak data is found which needs to be combined with the original dataset before proceeding to modelling.

2. Add Holiday data

Add column "Is_holiday" to site location according to their location timestamp and holiday calendars

3. Add lag and date features to weather data.

We create partial Autocorrelation graphs like below. (Partial snapshot)



After analysing the statsmodels pack graphs we create additional features with lag data and rolling mean features for weather data.

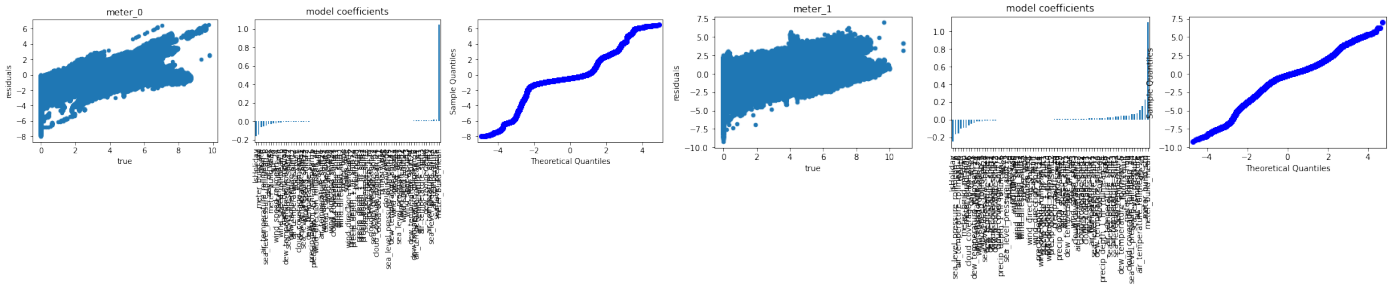
4. Add aggregated meter features for train and test data.

We aggregate meter readings by building id per meter id and add these additional features.
These include the mean, median, min, mac and std.

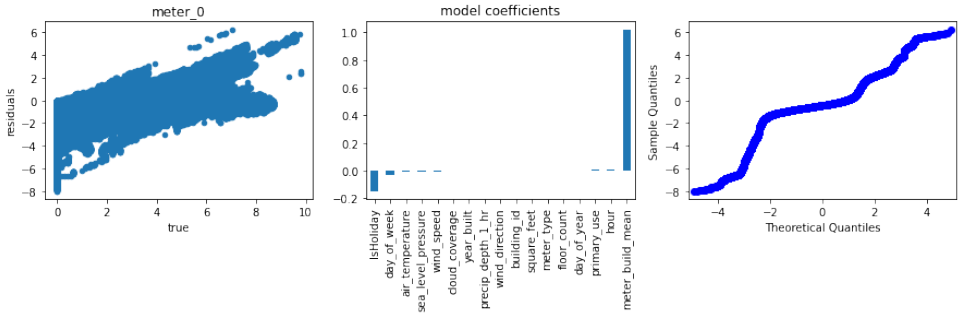
Modeling

As the different meters behavior is independent we have modelled the data separately on each meter id for better performance and accuracy

1. Linear Regression

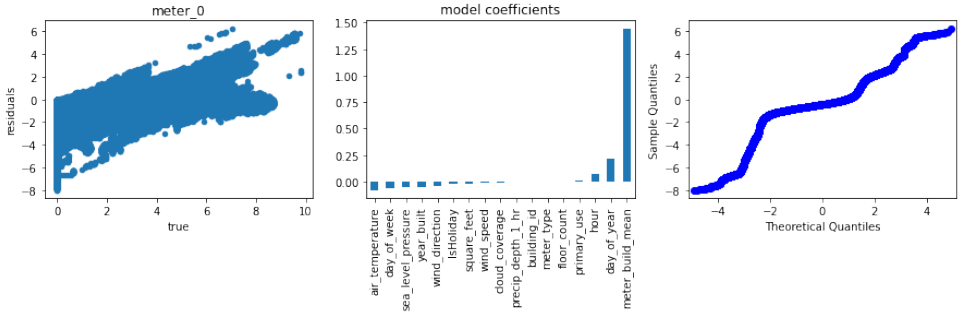


2. Linear Regression after dropping correlated columns

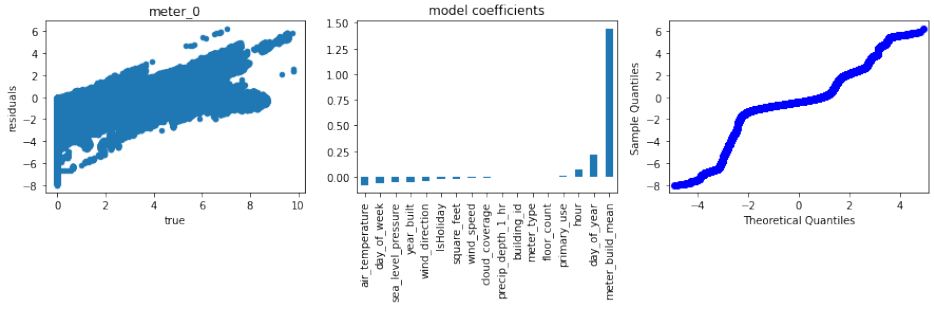


Similar graphs for meter ids 1-3 are also created.

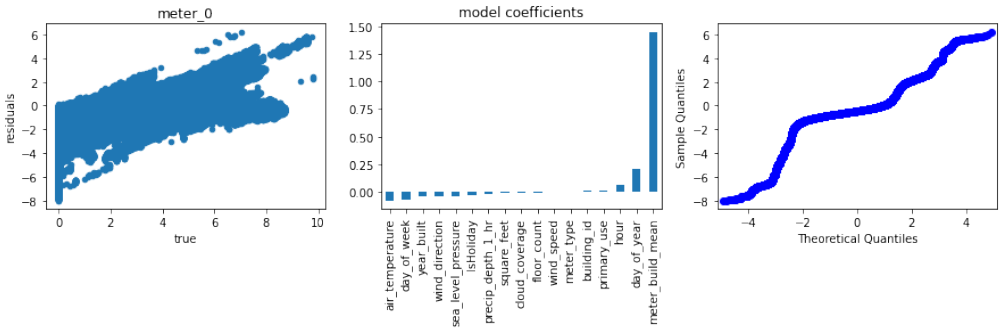
3. Linear Regression using Standard Scaler on features.



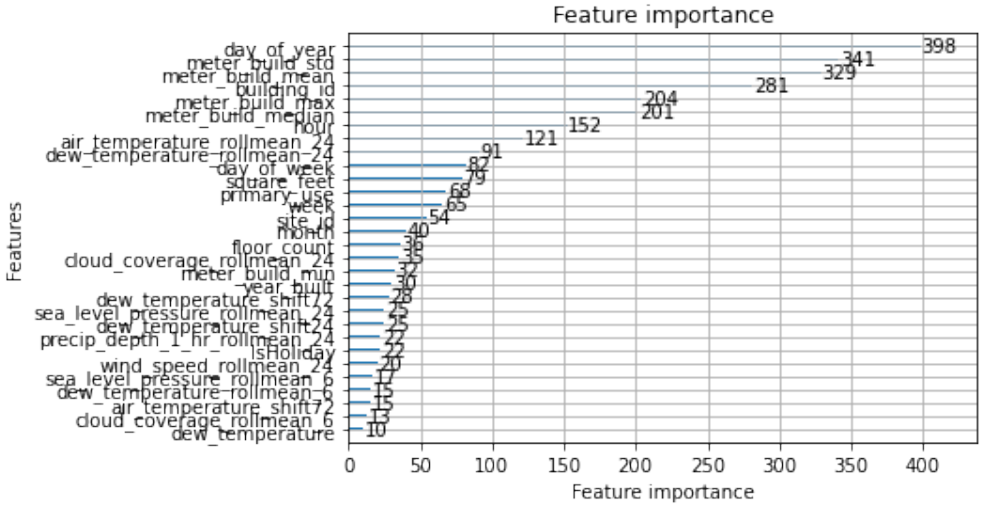
4. Linear regression with Ridge regularization



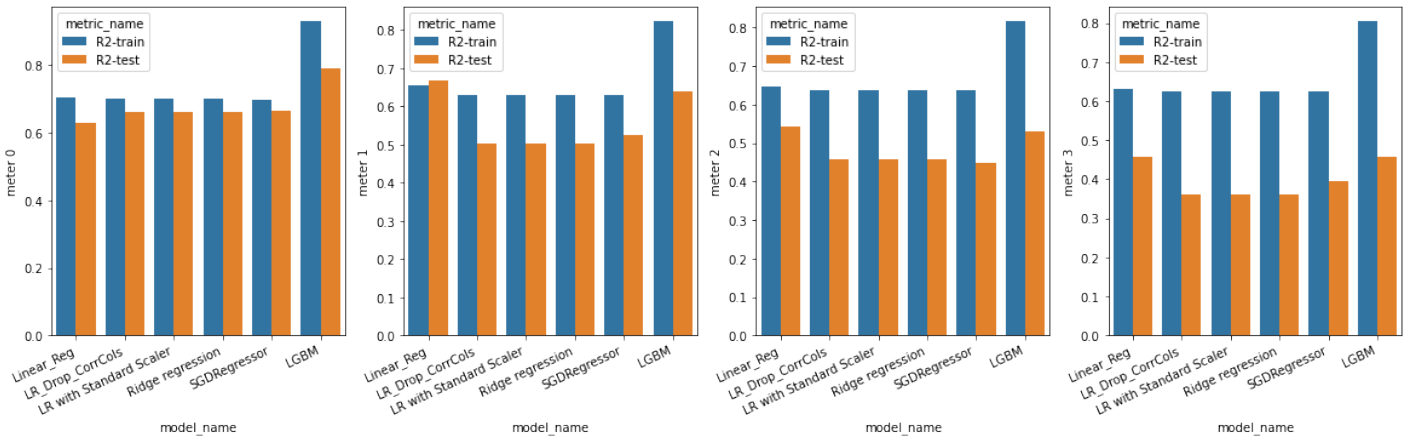
5. SDG Regression



6. LGBM Regression



Comparing accuracy of models with metric MSE and R2 values:



Since it is clear that LGBM is outperforming the other models we choose it and further play with hyper parameter tuning.

We use TimeSeriesSplit for Crossfold validation as the data is time sensitive.

We try performances with both RandomizedSearch and GridSearch.

The final model is trained on the entire training set with the best performing parameters of the previous CV tests.

```
lgbm_compare_df
```

	lgbm_name	target_meter	mse_train	mse_test	R2_train	R2_test	Exec_time	learning_rate	max_depth	min_data_in_leaf	...	min
0	LGBM_RS_TS_model1	0	0.162093	0.583497	0.946386	0.758373	9974.977779	0.10	10.0	400.0	...	
1	LGBM_RS_TS_model1	1	0.833992	2.186633	0.867070	0.648268	38156.528994	0.10	10.0	400.0	...	
2	LGBM_RS_TS_model1	2	0.987921	2.185719	0.859603	0.584620	670.725857	0.10	30.0	100.0	...	
3	LGBM_RS_TS_model1	3	1.356063	3.585909	0.799746	0.452534	331.601888	0.10	10.0	400.0	...	
4	LGBM_RS_TS_model2	1	0.736280	2.321672	0.882644	0.626546	1560.060600	0.10	30.0	700.0	...	
5	LGBM_RS_TS_model2	2	0.911444	2.508518	0.870472	0.523274	1486.101876	0.10	30.0	100.0	...	
6	LGBM_RS_TS_model2	3	1.346353	3.594948	0.801180	0.451154	459.645866	0.10	10.0	100.0	...	
7	LGBM_RS_TS_model3	1	0.717737	2.230456	0.885600	0.641219	1862.915258	0.05	30.0	400.0	...	
8	LGBM_RS_TS_model3	2	0.945831	2.230992	0.865585	0.576016	1045.339270	0.05	30.0	400.0	...	
9	LGBM_RS_TS_model3	3	1.352400	3.721019	0.800287	0.431906	601.976977	0.05	50.0	400.0	...	
10	LGBM_GS_TS_model1	0	0.152967	0.579509	0.949404	0.760025	8831.637197	0.05	30.0	400.0	...	
11	LGBM_GS_TS_model1	1	0.761740	2.156217	0.878586	0.653161	31412.346265	0.05	30.0	100.0	...	
12	LGBM_GS_TS_model1	2	0.971347	2.377841	0.861959	0.548108	4397.291591	0.05	20.0	100.0	...	
13	LGBM_GS_TS_model1	3	1.365097	3.652228	0.798412	0.442409	1051.189330	0.05	10.0	400.0	...	
14	LGBM_final_model1	0	0.152967	0.577209	0.949404	0.760977	291.239806	0.05	30.0	400.0	...	
15	LGBM_final_model1	0	0.152967	0.577209	0.949404	0.760977	330.158287	0.05	30.0	400.0	...	
16	LGBM_final_model1	1	0.761740	2.139326	0.878586	0.655878	129.593751	0.05	30.0	100.0	...	
17	LGBM_final_model1	2	0.987921	2.182671	0.859603	0.585199	41.138348	0.10	30.0	100.0	...	
18	LGBM_final_model1	3	1.356063	3.588026	0.799746	0.452210	15.283795	0.10	10.0	400.0	...	

19 rows × 28 columns

Conclusion & Further steps:

LGBM performed the best out of all models we tried , yet it needs to be evaluated against the actual test data provided by Kaggle. Upon submission to the Kaggle competition we can find out where the performance lies in comparison with all other submissions.