jupyter Applied AI Last Checkpoint: 3 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [1]:
```
#Applied AI final Project
```

In [2]:
```
conda install lightgbm
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
import os
os.system('brew install libomp')
```

Out[3]:
```
0
```

In [4]:
```
import numpy as np
import pandas as pd
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from time import time
import datetime
import gc
from sklearn.model_selection import train_test_split,KFold
from sklearn.preprocessing import LabelEncoder
import lightgbm as lgb
```

```
/Users/vidhyasagarudayakumar/anaconda3/lib/python3.7/site-packages/lightgbm/__init__.py:48: UserWarning: Starting fro
m version 2.2.1, the library file in distribution wheels for macOS is built by the Apple Clang (Xcode_8.3.3) compile
r.
This means that in case of installing LightGBM from PyPI via the ``pip install lightgbm`` command, you don't need to
install the gcc compiler anymore.
Instead of that, you need to install the OpenMP library, which is required for running LightGBM on the system with th
e Apple Clang compiler.
You can install the OpenMP library by the following command: ``brew install libomp``.
  "You can install the OpenMP library by the following command: ``brew install libomp``.", UserWarning)
```

In [5]:
```
pd.set_option('display.max_columns',100)
pd.set_option('display.max_rows',1500)
pd.set_option('display.float_format', lambda x: '%.5f' % x)
```

In [6]:
```
metadata_dtype = {'site_id':"uint8",'building_id':'uint16','square_feet':'float32','year_built':'float32','floor_count
metadata = pd.read_csv("building_metadata.csv",dtype=metadata_dtype)
metadata.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1449 entries, 0 to 1448
Data columns (total 6 columns):
site_id          1449 non-null uint8
building_id      1449 non-null uint16
primary_use      1449 non-null object
square_feet      1449 non-null float32
year_built       675 non-null float32
floor_count      355 non-null float16
dtypes: float16(1), float32(2), object(1), uint16(1), uint8(1)
memory usage: 117.2 KB
```

In [7]:
```
dtype = {"site_id":"uint8",'air_temperature':"float16",'cloud_coverage':"float16",'dew_temperature':"float16",'precip_de
         'sea_level_pressure':"float32",'wind_direction':"float16",'wind_speed':"float16"}

train = pd.read_csv("weather_train.csv",parse_dates=['timestamp'],dtype=weather_dtype)
test = pd.read_csv("weather_test.csv",parse_dates=['timestamp'],dtype=weather_dtype)
eather_train.info(memory_usage='deep'))
eather_test.info(memory_usage='deep'))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 139773 entries, 0 to 139772
Data columns (total 9 columns):
site_id               139773 non-null uint8
timestamp             139773 non-null datetime64[ns]
air_temperature       139718 non-null float16
cloud_coverage        70600 non-null float16
dew_temperature       139660 non-null float16
precip_depth_1_hr     89484 non-null float16
sea_level_pressure    129155 non-null float32
wind_direction        133505 non-null float16
wind_speed            139469 non-null float16
dtypes: datetime64[ns](1), float16(6), float32(1), uint8(1)
memory usage: 3.3 MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277243 entries, 0 to 277242
Data columns (total 9 columns):
site_id               277243 non-null uint8
timestamp             277243 non-null datetime64[ns]
air_temperature       277139 non-null float16
cloud_coverage        136795 non-null float16
dew_temperature       276916 non-null float16
precip_depth_1_hr     181655 non-null float16
sea_level_pressure    255978 non-null float32
wind_direction        264873 non-null float16
wind_speed            276783 non-null float16
dtypes: datetime64[ns](1), float16(6), float32(1), uint8(1)
memory usage: 6.6 MB
None
```

In [8]:
```
train_dtype = {'meter':"uint8",'building_id':'uint16','meter_reading':"float32"}
train = pd.read_csv("train.csv",parse_dates=['timestamp'],dtype=train_dtype)
test_dtype = {'meter':"uint8",'building_id':'uint16'}
test_cols_to_read = ['building_id','meter','timestamp']
test = pd.read_csv("test.csv",parse_dates=['timestamp'],usecols=test_cols_to_read,dtype=test_dtype)
```

In [9]:
```
Submission = pd.DataFrame(test.index,columns=['row_id'])
```

In [10]:
```
train.head()
```

Out[10]:

	building_id	meter	timestamp	meter_reading
0	0	0	2016-01-01	0.00000
1	1	0	2016-01-01	0.00000
2	2	0	2016-01-01	0.00000
3	3	0	2016-01-01	0.00000
4	4	0	2016-01-01	0.00000

In [11]:
```
test.head()
```

Out[11]:

	building_id	meter	timestamp
0	0	0	2017-01-01
1	1	0	2017-01-01
2	2	0	2017-01-01
3	3	0	2017-01-01

4	4	0	2017-01-01

In [12]:
```
metadata.head()
```

Out[12]:

	site_id	building_id	primary_use	square_feet	year_built	floor_count
0	0	0	Education	7432.00000	2008.00000	nan
1	0	1	Education	2720.00000	2004.00000	nan
2	0	2	Education	5376.00000	1991.00000	nan
3	0	3	Education	23685.00000	2002.00000	nan
4	0	4	Education	116607.00000	1975.00000	nan

In [13]:
```
weather_train.head()
```

Out[13]:

	site_id	timestamp	air_temperature	cloud_coverage	dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_direction	wind_speed
0	0	2016-01-01 00:00:00	25.00000	6.00000	20.00000	nan	1019.70001	0.00000	0.00000
1	0	2016-01-01 01:00:00	24.40625	nan	21.09375	-1.00000	1020.20001	70.00000	1.50000
2	0	2016-01-01 02:00:00	22.79688	2.00000	21.09375	0.00000	1020.20001	0.00000	0.00000
3	0	2016-01-01 03:00:00	21.09375	2.00000	20.59375	0.00000	1020.09998	0.00000	0.00000
4	0	2016-01-01 04:00:00	20.00000	2.00000	20.00000	-1.00000	1020.00000	250.00000	2.59961

In [14]:
```
weather_test.head()
```

Out[14]:

	site_id	timestamp	air_temperature	cloud_coverage	dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_direction	wind_speed
0	0	2017-01-01 00:00:00	17.79688	4.00000	11.70312	nan	1021.40002	100.00000	3.59961
1	0	2017-01-01 01:00:00	17.79688	2.00000	12.79688	0.00000	1022.00000	130.00000	3.09961
2	0	2017-01-01 02:00:00	16.09375	0.00000	12.79688	0.00000	1021.90002	140.00000	3.09961
3	0	2017-01-01 03:00:00	17.20312	0.00000	13.29688	0.00000	1022.20001	140.00000	3.09961
4	0	2017-01-01 04:00:00	16.70312	2.00000	13.29688	0.00000	1022.29999	130.00000	2.59961

In [15]:
```
missing_weather = pd.DataFrame(weather_train.isna().sum()/len(weather_train),columns=["Weather_Train_Missing_Pct"])
missing_weather["Weather_Test_Missing_Pct"] = weather_test.isna().sum()/len(weather_test)
missing_weather
```

Out[15]:

	Weather_Train_Missing_Pct	Weather_Test_Missing_Pct
site_id	0.00000	0.00000
timestamp	0.00000	0.00000
air_temperature	0.00039	0.00038
cloud_coverage	0.49490	0.50659
dew_temperature	0.00081	0.00118
precip_depth_1_hr	0.35979	0.34478
sea_level_pressure	0.07597	0.07670
wind_direction	0.04484	0.04462
wind_speed	0.00217	0.00166

In [16]:
```
metadata.isna().sum()/len(metadata)
```

Out[16]:
```
site_id        0.00000
building_id    0.00000
primary_use    0.00000
square_feet    0.00000
year_built     0.53416
floor_count    0.75500
dtype: float64
```

In [17]:
```
metadata['floor_count_isNa'] = metadata['floor_count'].isna().astype('uint8')
metadata['year_built_isNa'] = metadata['year_built'].isna().astype('uint8')
# Dropping floor_count variable as it has 75% missing values
metadata.drop('floor_count',axis=1,inplace=True)
```

In [18]:
```
missing_train_test = pd.DataFrame(train.isna().sum()/len(train),columns=["Missing_Pct_Train"])
missing_train_test["Missing_Pct_Test"] = test.isna().sum()/len(test)
missing_train_test
```

Out[18]:

	Missing_Pct_Train	Missing_Pct_Test
building_id	0.00000	0.00000
meter	0.00000	0.00000
timestamp	0.00000	0.00000
meter_reading	0.00000	nan

In [ ]:
```
dataset.fillna(dataset.mean(), inplace=True)
```

In [19]:
```
train.head()
```

Out[19]:

	building_id	meter	timestamp	meter_reading
0	0	0	2016-01-01	0.00000
1	1	0	2016-01-01	0.00000
2	2	0	2016-01-01	0.00000
3	3	0	2016-01-01	0.00000
4	4	0	2016-01-01	0.00000

In [20]:
```
train.describe(include='all')
```

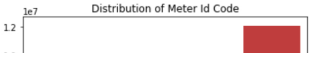
Out[20]:

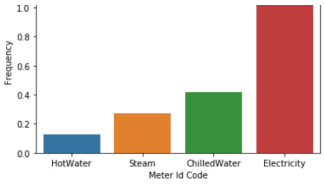
	building_id	meter	timestamp	meter_reading
count	20216100.00000	20216100.00000	20216100	20216100.00000
unique	nan	nan	8784	nan
top	nan	nan	2016-12-27 22:00:00	nan
freq	nan	nan	2370	nan
first	nan	nan	2016-01-01 00:00:00	nan
last	nan	nan	2016-12-31 23:00:00	nan
mean	799.27800	0.66244	NaN	1988.70618
std	426.91331	0.93099	NaN	153215.93750
min	0.00000	0.00000	NaN	0.00000
25%	393.00000	0.00000	NaN	18.30000
50%	895.00000	0.00000	NaN	78.77500
75%	1179.00000	1.00000	NaN	267.98401
max	1448.00000	3.00000	NaN	21904700.00000

In [21]:
```
train['meter'].replace({0:"Electricity",1:"ChilledWater",2:"Steam",3:"HotWater"},inplace=True)
test['meter'].replace({0:"Electricity",1:"ChilledWater",2:"Steam",3:"HotWater"},inplace=True)
```

In [22]:
```
sns.countplot(train['meter'],order=train['meter'].value_counts().sort_values().index)
plt.title("Distribution of Meter Id Code")
plt.xlabel("Meter Id Code")
plt.ylabel("Frequency")
```

Out[22]:
```
Text(0, 0.5, 'Frequency')
```





In [23]:
```
print ("There are {} unique Buildings in the training data".format(train['building_id'].nunique()))
```

```
There are 1449 unique Buildings in the training data
```

In [24]:
```
train['building_id'].value_counts(dropna=False).head(20)
```

Out[24]:
```
1298    35136
1249    35136
1301    35128
1241    35116
1296    35115
1297    35115
1295    35108
1294    35108
1259    35099
1232    34131
1293    33714
1258    31967
1331    29608
1094    26352
1260    26352
1254    26352
1252    26352
1251    26352
1262    26352
1263    26352
Name: building_id, dtype: int64
```

In [25]:
```
train[train['building_id'] == 1094]['meter'].unique()
```

Out[25]:
```
array(['Electricity', 'ChilledWater', 'Steam'], dtype=object)
```

In [26]:
```
train.groupby('meter')['meter_reading'].agg(['min','max','mean','median','count','std'])
```

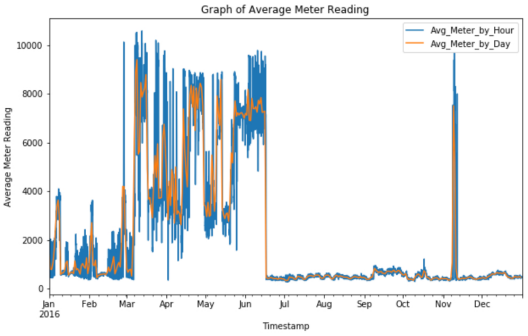
Out[26]:

meter	min	max	mean	median	count	std
ChilledWater	0.00000	880374.00000	633.36395	120.48700	4182440	7988.21289
Electricity	0.00000	79769.00000	170.82564	62.83190	12060910	380.83429
HotWater	0.00000	160187.00000	385.86679	39.62370	1264037	2508.17261
Steam	0.00000	21904700.00000	13882.18750	257.81201	2708713	418313.50000

In [27]:
```
for df in [train, test]:
    df['Month'] = df['timestamp'].dt.month.astype("uint8")
    df['DayOfMonth'] = df['timestamp'].dt.day.astype("uint8")
    df['DayOfWeek'] = df['timestamp'].dt.dayofweek.astype("uint8")
    df['Hour'] = df['timestamp'].dt.hour.astype("uint8")
```

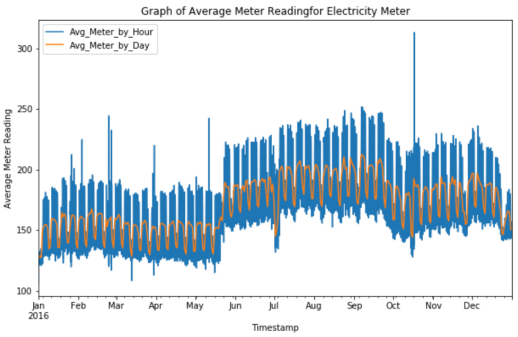
In [28]:
```
ain[['timestamp','meter_reading']].set_index('timestamp').resample("H")['meter_reading'].mean().plot(kind='line',figsize
ain[['timestamp','meter_reading']].set_index('timestamp').resample("D")['meter_reading'].mean().plot(kind='line',figsize
:.legend()
:.xlabel("Timestamp")
:.ylabel("Average Meter Reading")
:.title("Graph of Average Meter Reading")
```

Out[28]:
```
Text(0.5, 1.0, 'Graph of Average Meter Reading')
```



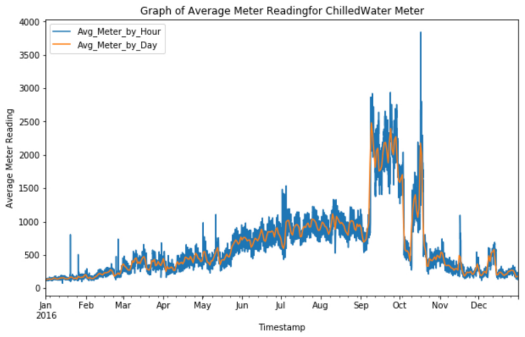
In [29]:
```
meter_Electricity = train[train['meter'] == "Electricity"]
meter_Electricity[['timestamp','meter_reading']].set_index('timestamp').resample("H")['meter_reading'].mean().plot(kinc
meter_Electricity[['timestamp','meter_reading']].set_index('timestamp').resample("D")['meter_reading'].mean().plot(kinc
plt.legend()
plt.xlabel("Timestamp")
plt.ylabel("Average Meter Reading")
plt.title("Graph of Average Meter Readingfor Electricity Meter")
```

Out[29]:
```
Text(0.5, 1.0, 'Graph of Average Meter Readingfor Electricity Meter')
```



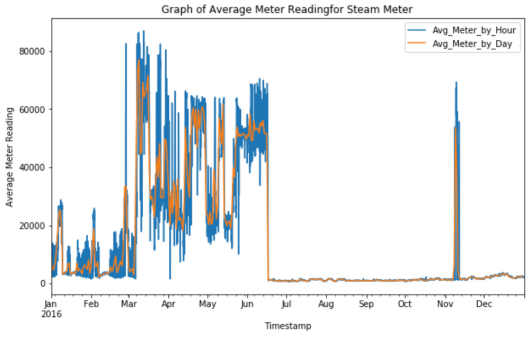
In [30]:
```
meter_ChilledWater = train[train['meter'] == "ChilledWater"]
meter_ChilledWater[['timestamp','meter_reading']].set_index('timestamp').resample("H")['meter_reading'].mean().plot(kin
meter_ChilledWater[['timestamp','meter_reading']].set_index('timestamp').resample("D")['meter_reading'].mean().plot(kin
plt.legend()
plt.xlabel("Timestamp")
plt.ylabel("Average Meter Reading")
plt.title("Graph of Average Meter Readingfor ChilledWater Meter")
```

Out[30]: Text(0.5, 1.0, 'Graph of Average Meter Readingfor ChilledWater Meter')



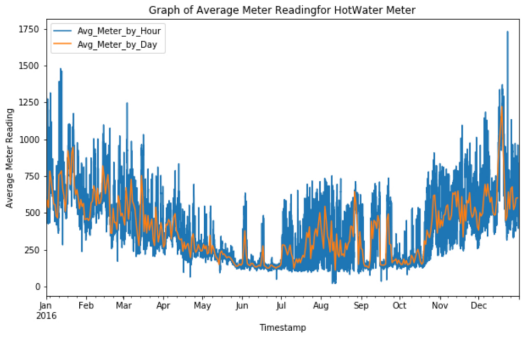
In [31]:
```
meter_Steam = train[train['meter'] == "Steam"]
meter_Steam[['timestamp','meter_reading']].set_index('timestamp').resample("H")['meter_reading'].mean().plot(kind='line
meter_Steam[['timestamp','meter_reading']].set_index('timestamp').resample("D")['meter_reading'].mean().plot(kind='line
plt.legend()
plt.xlabel("Timestamp")
plt.ylabel("Average Meter Reading")
plt.title("Graph of Average Meter Readingfor Steam Meter")
```

Out[31]: Text(0.5, 1.0, 'Graph of Average Meter Readingfor Steam Meter')



In [32]:
```
meter_HotWater = train[train['meter'] == "HotWater"]
meter_HotWater[['timestamp','meter_reading']].set_index('timestamp').resample("H")['meter_reading'].mean().plot(kind='
meter_HotWater[['timestamp','meter_reading']].set_index('timestamp').resample("D")['meter_reading'].mean().plot(kind='
plt.legend()
plt.xlabel("Timestamp")
plt.ylabel("Average Meter Reading")
plt.title("Graph of Average Meter Readingfor HotWater Meter")
```

Out[32]: Text(0.5, 1.0, 'Graph of Average Meter Readingfor HotWater Meter')



In [79]:
```
%%time
train = pd.merge(train,metadata,on='building_id',how='left')
test  = pd.merge(test,metadata,on='building_id',how='left')
print ("Training Data Shape {}".format(train.shape))
print ("Testing Data Shape {}".format(test.shape))
gc.collect()
```

```
Training Data Shape (20216100, 14)
Testing Data Shape (41697600, 13)
CPU times: user 21.6 s, sys: 5.52 s, total: 27.1 s
Wall time: 15.2 s
```

Out[79]: 50399

In [80]:
```
%%time
train = pd.merge(train,weather_train,on=['site_id','timestamp'],how='left')
test  = pd.merge(test,weather_test,on=['site_id','timestamp'],how='left')
print ("Training Data Shape {}".format(train.shape))
print ("Testing Data Shape {}".format(test.shape))
gc.collect()
```

```
Training Data Shape (20216100, 21)
Testing Data Shape (41697600, 20)
CPU times: user 30.4 s, sys: 9.64 s, total: 40.1 s
Wall time: 21.3 s
```

Out[80]: 62

In [81]:
```
holidays = ["2016-01-01", "2016-01-18", "2016-02-15", "2016-05-30", "2016-07-04",
            "2016-09-05", "2016-10-10", "2016-11-11", "2016-11-24", "2016-12-26",
            "2017-01-02", "2017-01-16", "2017-02-20", "2017-05-29", "2017-07-04",
            "2017-09-04", "2017-10-09", "2017-11-10", "2017-11-23", "2017-12-25",
            "2018-01-01", "2018-01-15", "2018-02-19", "2018-05-28", "2018-07-04",
            "2018-09-03", "2018-10-08", "2018-11-12", "2018-11-22", "2018-12-25",
            "2019-01-01"]
for df in [train,test]:
    df['square_feet'] = df['square_feet'].astype('float16')
    df["Holiday"] = (df.timestamp.isin(holidays)).astype(int)
```

In [91]:
```
%%time
le = LabelEncoder()

train['meter']= le.fit_transform(train['meter']).astype("uint8")
test['meter']= le.fit_transform(test['meter']).astype("uint8")
train['primary_use']= le.fit_transform(train['primary_use']).astype("uint8")
test['primary_use']= le.fit_transform(test['primary_use']).astype("uint8")
```

```
CPU times: user 17.7 s, sys: 2.14 s, total: 19.9 s
Wall time: 18.3 s
```

In [92]:
```
%%time
# Let's check the correlation between the variables and eliminate the one's that have high correlation
# Threshold for removing correlated variables
threshold = 0.9

# Absolute value correlation matrix
corr_matrix = train.corr().abs()
corr_matrix.head()
```

```
CPU times: user 1min 10s, sys: 1min 11s, total: 2min 21s
Wall time: 3min 3s
```

Out[92]:

	building_id	meter	meter_reading	Month	DayOfMonth	DayOfWeek	Hour	site_id	primary_use	square_feet	year_built	floor_count_isNa
building_id	1.00000	0.15473	0.08498	0.02306	0.00366	0.00023	0.00010	0.98030	0.08434	0.15811	0.42348	0.16197
meter	0.15473	1.00000	0.09882	0.00451	0.00026	0.00036	0.00034	0.15063	0.01463	0.07066	0.07039	0.03262
meter_reading	0.08498	0.09882	1.00000	0.01320	0.00310	0.02757	0.03513	0.10446	0.05749	0.46145	0.01264	0.15638
Month	0.02306	0.00451	0.01320	1.00000	0.01216	0.00324	0.00011	0.01919	0.00000	0.00533	0.04640	0.01871
DayOfMonth	0.00366	0.00026	0.00310	0.01216	1.00000	0.00364	0.00055	0.00339	0.00043	0.00144	0.00147	0.00010

In [93]:
```
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
upper.head()
```

Out[93]:

	building_id	meter	meter_reading	Month	DayOfMonth	DayOfWeek	Hour	site_id	primary_use	square_feet	year_built	floor_count_isNa
building_id	nan	0.15473	0.08498	0.02306	0.00366	0.00023	0.00010	0.98030	0.08434	0.15811	0.42348	0.16197
meter	nan	nan	0.09882	0.00451	0.00026	0.00036	0.00034	0.15063	0.01463	0.07066	0.07039	0.03262
meter_reading	nan	nan	nan	0.01320	0.00310	0.02757	0.03513	0.10446	0.05749	0.46145	0.01264	0.15638
Month	nan	nan	nan	nan	0.01216	0.00324	0.00011	0.01919	0.00000	0.00533	0.04640	0.01871
DayOfMonth	nan	nan	nan	nan	nan	0.00364	0.00055	0.00339	0.00043	0.00144	0.00147	0.00010

In [94]:
```
to_drop = [column for column in upper.columns if any(upper[column] > threshold)]

print('There are %d columns to remove.' % (len(to_drop)))
print ("Following columns can be dropped {}".format(to_drop))
```

```
There are 7 columns to remove.
Following columns can be dropped ['site_id', 'year_built_isNa', 'median_meter_reading_per_building', 'median_meter_re
ading_on_year_built', 'median_meter_reading_per_meter', 'median_meter_reading_per_primary_usage', 'median_meter_readi
ng_per_site_id']
```

In [95]:
```
train.drop(to_drop,axis=1,inplace=True)
test.drop(to_drop,axis=1,inplace=True)
```

In [96]:
```
y = train['meter_reading']
train.drop('meter_reading',axis=1,inplace=True)
```

In [97]:
```
categorical_cols = ['building_id','Month','meter','Hour','primary_use','DayOfWeek','DayOfMonth','floor_count_isNa','Hol
```

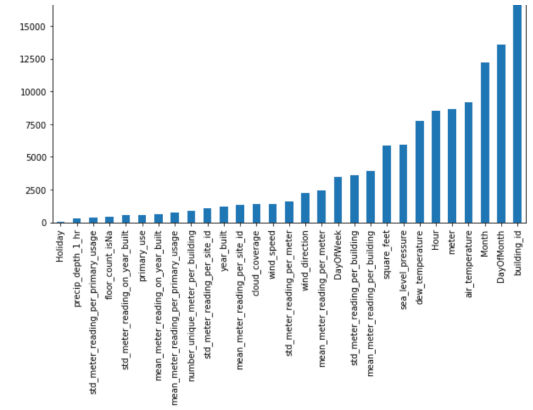
In [98]:
```
%time
_train,x_test,y_train,y_test = train_test_split(train,y,test_size=0.25,random_state=42)
print (x_train.shape)
print (y_train.shape)
print (x_test.shape)
print (y_test.shape)

gb_train = lgb.Dataset(x_train, y_train,categorical_feature=categorical_cols)
gb_test = lgb.Dataset(x_test, y_test,categorical_feature=categorical_cols)
del x_train, x_test , y_train, y_test

params = {'feature_fraction': 0.75,
          'bagging_fraction': 0.75,
          'objective': 'regression',
          'max_depth': -1,
          'learning_rate': 0.15,
          "boosting_type": "gbdt",
          "bagging_seed": 11,
          "metric": 'rmse',
          "verbosity": -1,
          'reg_alpha': 0.5,
          'reg_lambda': 0.5,
          'random_state': 47,
          "num_leaves": 41}

reg = lgb.train(params, lgb_train, num_boost_round=3000, valid_sets=[lgb_train, lgb_test], early_stopping_rounds=100, ve
```

```
(14725725, 29)
(14725725,)
(4908575, 29)
(4908575,)
/Users/vidhyasagarudayakumar/anaconda3/lib/python3.7/site-packages/lightgbm/basic.py:1243: UserWarning: Using categor
ical_feature in Dataset.
  warnings.warn('Using categorical_feature in Dataset.')
Training until validation scores don't improve for 100 rounds
[100]   training's rmse: 0.831112       valid_1's rmse: 0.832347
[200]   training's rmse: 0.777703       valid_1's rmse: 0.779641
[300]   training's rmse: 0.732993       valid_1's rmse: 0.735477
[400]   training's rmse: 0.705201       valid_1's rmse: 0.70832
[500]   training's rmse: 0.687411       valid_1's rmse: 0.691192
[600]   training's rmse: 0.669919       valid_1's rmse: 0.674286
[700]   training's rmse: 0.656737       valid_1's rmse: 0.66178
[800]   training's rmse: 0.646599       valid_1's rmse: 0.652315
[900]   training's rmse: 0.637149       valid_1's rmse: 0.643427
[1000]  training's rmse: 0.629843       valid_1's rmse: 0.636669
[1100]  training's rmse: 0.622305       valid_1's rmse: 0.629744
[1200]  training's rmse: 0.614012       valid_1's rmse: 0.622085
[1300]  training's rmse: 0.605981       valid_1's rmse: 0.614566
[1400]  training's rmse: 0.599792       valid_1's rmse: 0.608931
[1500]  training's rmse: 0.594222       valid_1's rmse: 0.603913
[1600]  training's rmse: 0.588553       valid_1's rmse: 0.598748
[1700]  training's rmse: 0.583777       valid_1's rmse: 0.594489
[1800]  training's rmse: 0.578677       valid_1's rmse: 0.590026
[1900]  training's rmse: 0.574244       valid_1's rmse: 0.586159
[2000]  training's rmse: 0.570205       valid_1's rmse: 0.582597
[2100]  training's rmse: 0.566235       valid_1's rmse: 0.579182
[2200]  training's rmse: 0.562615       valid_1's rmse: 0.576142
[2300]  training's rmse: 0.559195       valid_1's rmse: 0.573183
[2400]  training's rmse: 0.555818       valid_1's rmse: 0.570318
[2500]  training's rmse: 0.552631       valid_1's rmse: 0.567672
[2600]  training's rmse: 0.549603       valid_1's rmse: 0.565197
[2700]  training's rmse: 0.54697        valid_1's rmse: 0.563026
[2800]  training's rmse: 0.544  valid_1's rmse: 0.560598
[2900]  training's rmse: 0.541127       valid_1's rmse: 0.558242
[3000]  training's rmse: 0.538751       valid_1's rmse: 0.556471
Did not meet early stopping. Best iteration is:
[3000]  training's rmse: 0.538751       valid_1's rmse: 0.556471
CPU times: user 2h 6min 57s, sys: 54.4 s, total: 2h 7min 52s
Wall time: 19min 42s
```

In [99]:
```
del lgb_train,lgb_test
```

In [100]:
```
ser = pd.DataFrame(reg.feature_importance(),train.columns,columns=['Importance']).sort_values(by='Importance')
ser['Importance'].plot(kind='bar',figsize=(10,6))
```

Out[100]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a50129f60>
```



In [104]:
```
%%time
prediction = []
step = 100000
for i in range(0, len(test), step):
    prediction.extend(np.expm1(reg.predict(test.iloc[i: min(i+step, len(test)), :], num_iteration=reg.best_iteration)))
```

```
CPU times: user 5h 7min 33s, sys: 1min 43s, total: 5h 9min 16s
Wall time: 44min 19s
```

In [106]:
```
%%time
Submission['meter_reading'] = prediction
Submission['meter_reading'].clip(lower=0,upper=None,inplace=True)
Submission.to_csv("output.csv",index=None)
```

```
CPU times: user 2min 4s, sys: 19.5 s, total: 2min 24s
Wall time: 2min 26s
```

In [ ]: