In [6]:
```
!pip install isotree
```

```
Requirement already satisfied: isotree in /opt/conda/lib/python3.6/site-packa
ges (0.1.9)
```

In [54]:
```
import pandas as pd
import numpy as np
import category_encoders
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error
from isotree import IsolationForest
from sklearn.metrics import r2_score
import datetime
import gc

#can be removed if files are local
DATA_PATH = "../input/ashrae-energy-prediction/"
```

In [36]:
```
train_df = pd.read_csv(DATA_PATH + 'train.csv')

# Remove any outliers as it will imbalance data
train_df = train_df [ train_df['building_id'] != 1099 ]
train_df = train_df.query('not (building_id <= 104 & meter == 0 & timestamp <=
"2016-05-20")')

#change path if DATA_PATH is not needed
building_df = pd.read_csv(DATA_PATH + 'building_metadata.csv')
weather_df = pd.read_csv(DATA_PATH + 'weather_train.csv')
```

In [37]:
```
#Original code from https://www.kaggle.com/aitude/ashrae-missing-weather-data-handling by @aitude

def fill_weather_dataset(weather_df):

    # Find Missing Dates
    time_format = "%Y-%m-%d %H:%M:%S"
    start_date = datetime.datetime.strptime(weather_df['timestamp'].min(),time_format)
    end_date = datetime.datetime.strptime(weather_df['timestamp'].max(),time_format)
    total_hours = int(((end_date - start_date).total_seconds() + 3600) / 3600)
    hours_list = [(end_date - datetime.timedelta(hours=x)).strftime(time_format) for x in range(total_hours)]

    missing_hours = []
    for site_id in range(16):
        site_hours = np.array(weather_df[weather_df['site_id'] == site_id]['timestamp'])
        new_rows = pd.DataFrame(np.setdiff1d(hours_list,site_hours),columns=['timestamp'])
        new_rows['site_id'] = site_id
        weather_df = pd.concat([weather_df,new_rows])

        weather_df = weather_df.reset_index(drop=True)

    # Add new Features that are beneficial
    weather_df["datetime"] = pd.to_datetime(weather_df["timestamp"])
    weather_df["day"] = weather_df["datetime"].dt.day
    weather_df["week"] = weather_df["datetime"].dt.week
    weather_df["month"] = weather_df["datetime"].dt.month

    # Reset Index for Fast Update
    weather_df = weather_df.set_index(['site_id','day','month'])

    air_temperature_filler = pd.DataFrame(weather_df.groupby(['site_id','day','month'])['air_temperature'].mean(),columns=["air_temperature"])
    weather_df.update(air_temperature_filler,overwrite=False)

    # Step 1
    cloud_coverage_filler = weather_df.groupby(['site_id','day','month'])['cloud_coverage'].mean()
    # Step 2
    cloud_coverage_filler = pd.DataFrame(cloud_coverage_filler.fillna(method='ffill'),columns=["cloud_coverage"])

    weather_df.update(cloud_coverage_filler,overwrite=False)

    due_temperature_filler = pd.DataFrame(weather_df.groupby(['site_id','day','month'])['dew_temperature'].mean(),columns=["dew_temperature"])
    weather_df.update(due_temperature_filler,overwrite=False)

    # Step 1
    sea_level_filler = weather_df.groupby(['site_id','day','month'])['sea_level_pressure'].mean()
    # Step 2
```

```
    sea_level_filler = pd.DataFrame(sea_level_filler.fillna(method='ffill'),co
lumns=['sea_level_pressure'])

    weather_df.update(sea_level_filler,overwrite=False)

    wind_direction_filler =  pd.DataFrame(weather_df.groupby(['site_id','day',
'month'])['wind_direction'].mean(),columns=['wind_direction'])
    weather_df.update(wind_direction_filler,overwrite=False)

    wind_speed_filler =  pd.DataFrame(weather_df.groupby(['site_id','day','mon
th'])['wind_speed'].mean(),columns=['wind_speed'])
    weather_df.update(wind_speed_filler,overwrite=False)

    # Step 1
    precip_depth_filler = weather_df.groupby(['site_id','day','month'])['preci
p_depth_1_hr'].mean()
    # Step 2
    precip_depth_filler = pd.DataFrame(precip_depth_filler.fillna(method='ffil
l'),columns=['precip_depth_1_hr'])

    weather_df.update(precip_depth_filler,overwrite=False)

    weather_df = weather_df.reset_index()
    weather_df = weather_df.drop(['datetime','day','week','month'],axis=1)

    return weather_df

# Original code from https://www.kaggle.com/gemartin/load-data-reduce-memory-u
sage by @gemartin

from pandas.api.types import is_datetime64_any_dtype as is_datetime
from pandas.api.types import is_categorical_dtype

def reduce_mem_usage(df, use_float16=False):
    """
    Iterate through all the columns of a dataframe and modify the data type to
reduce memory usage.
    """
    #decrease memory usage if processing
    start_mem = df.memory_usage().sum() / 1024**2
    print("Memory usage of dataframe is {:.2f} MB".format(start_mem))

    for col in df.columns:
        if is_datetime(df[col]) or is_categorical_dtype(df[col]):
            continue
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == "int":
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8)
.max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.in
t16).max:
                    df[col] = df[col].astype(np.int16)
```

```
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.in
t32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.in
t64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if use_float16 and c_min > np.finfo(np.float16).min and c_max
< np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.
float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype("category")

    end_mem = df.memory_usage().sum() / 1024**2
    print("Memory usage after optimization is: {:.2f} MB".format(end_mem))
    print("Decreased by {:.1f}%".format(100 * (start_mem - end_mem) / start_me
m))

    return df


def features_engineering(df):

    # Sort by timestamp
    df.sort_values("timestamp")
    df.reset_index(drop=True)

    # Add more features
    df["timestamp"] = pd.to_datetime(df["timestamp"],format="%Y-%m-%d %H:%M:%
S")
    df["hour"] = df["timestamp"].dt.hour
    df["weekend"] = df["timestamp"].dt.weekday
    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", "2
016-12-26",
                    "2017-01-02", "2017-01-16", "2017-02-20", "2017-05-29", "2
017-07-04",
                    "2017-09-04", "2017-10-09", "2017-11-10", "2017-11-23", "2
017-12-25",
                    "2018-01-01", "2018-01-15", "2018-02-19", "2018-05-28", "2
018-07-04",
                    "2018-09-03", "2018-10-08", "2018-11-12", "2018-11-22", "2
018-12-25",
                    "2019-01-01"]
    df["is_holiday"] = (df.timestamp.isin(holidays)).astype(int)
    df['square_feet'] =  np.log1p(df['square_feet']**0.5)

    # Remove Unused Columns
    drop = ["timestamp","sea_level_pressure", "wind_direction", "wind_speed",
"year_built","floor_count"]
    df = df.drop(drop, axis=1)
```

```
    gc.collect()

    # Encode Categorical Data
    le = LabelEncoder()
    df["primary_use"] = le.fit_transform(df["primary_use"])

    return df
```

In [40]:
```
weather_df = fill_weather_dataset(weather_df)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:17: FutureWarnin
g: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

In [41]:
```
train_df = reduce_mem_usage(train_df,use_float16=True)
building_df = reduce_mem_usage(building_df,use_float16=True)
weather_df = reduce_mem_usage(weather_df,use_float16=True)
```

```
Memory usage of dataframe is 757.31 MB
Memory usage after optimization is: 322.24 MB
Decreased by 57.4%
Memory usage of dataframe is 0.07 MB
Memory usage after optimization is: 0.02 MB
Decreased by 73.8%
Memory usage of dataframe is 9.65 MB
Memory usage after optimization is: 2.66 MB
Decreased by 72.5%
```

In [42]:
```
train_df = train_df.merge(building_df, left_on='building_id',right_on='buildin
g_id',how='left')
train_df = train_df.merge(weather_df,how='left',left_on=['site_id','timestamp'
],right_on=['site_id','timestamp'])
del weather_df
gc.collect()
```

Out[42]: 0

In [43]:
```
train_df = features_engineering(train_df)
```

In [44]:
```
target = np.log1p(train_df["meter_reading"])
train = train_df.drop(['meter_reading', 'is_holiday', 'hour', 'weekend'], axis
= 1)
del train_df
gc.collect()
train.head()
```

Out[44]:

	building_id	meter	site_id	primary_use	square_feet	air_temperature	cloud_coverage	dew_t
0	105	0	1	0	5.420515	3.800781	0.0	
1	106	0	1	0	4.308213	3.800781	0.0	
2	106	3	1	0	4.308213	3.800781	0.0	
3	107	0	1	0	5.747165	3.800781	0.0	
4	108	0	1	0	5.658165	3.800781	0.0	

In [45]:
```
categorical_features = ["building_id", "site_id", "meter", "primary_use"]
```

In [46]:
```
ce = category_encoders.CountEncoder(cols=categorical_features)
ce.fit(train)
train = ce.transform(train)
train.head()
```

Out[46]:

	building_id	meter	site_id	primary_use	square_feet	air_temperature	cloud_coverage	de
0	8784	11706016	553357	8050507	5.420515	3.800781	0.0	
1	17564	11706016	553357	8050507	4.308213	3.800781	0.0	
2	17564	1264037	553357	8050507	4.308213	3.800781	0.0	
3	8784	11706016	553357	8050507	5.747165	3.800781	0.0	
4	8784	11706016	553357	8050507	5.658165	3.800781	0.0	

In [47]:
```
N_train = train.shape[0]
for feature in categorical_features:
    train[feature] = train[feature]/N_train

train.head()
```

Out[47]:

	building_id	meter	site_id	primary_use	square_feet	air_temperature	cloud_coverage	d
0	0.000442	0.589652	0.027874	0.405518	5.420515	3.800781	0.0	
1	0.000885	0.589652	0.027874	0.405518	4.308213	3.800781	0.0	
2	0.000885	0.063672	0.027874	0.405518	4.308213	3.800781	0.0	
3	0.000442	0.589652	0.027874	0.405518	5.747165	3.800781	0.0	
4	0.000442	0.589652	0.027874	0.405518	5.658165	3.800781	0.0	

In [48]:
```
'''%%time
iso = IsolationForest(build_imputer=True, prob_pick_pooled_gain=1, ntry=10)
iso.fit(train)
train = iso.transform(train)'''
```

Out[48]:
```
'%%time\niso = IsolationForest(build_imputer=True, prob_pick_pooled_gain=1, n
try=10)\niso.fit(train)\ntrain = iso.transform(train)'
```

In [49]:
```
#helps fill nan values
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit(train)
train = imp.transform(train)
```

In [61]:
```
kf = KFold(n_splits=3)
models = []
count = 0
for train_index, val_index in kf.split(train):
    train_features = train[train_index]
    train_target = target[train_index]
    count+=1
    val_features = train[val_index]
    val_target = target[val_index]

    model = LinearRegression()
    model.fit(train_features, train_target)
    models.append(model)
    val_pred = model.predict(val_features)
    print("Split ", count)
    print("================================")
    print("MSE: ", mean_squared_error(val_target, val_pred))
    print("RMSE: ",np.sqrt(mean_squared_error(val_target, val_pred)))
    print("R2: ",r2_score(val_target, val_pred))
    print("================================")
    del train_features, train_target, val_features, val_target
    gc.collect()
```

```
Split  1
================================
MSE:  3.5161722475359487
RMSE:  1.8751459269976694
R2:  0.20670128963545897
================================
Split  2
================================
MSE:  3.3798425244499897
RMSE:  1.8384348028826014
R2:  0.20712207139233563
================================
Split  3
================================
MSE:  3.4452563038241215
RMSE:  1.8561401627636103
R2:  0.21286843945354816
================================
```

In [62]:
```
del train, target
gc.collect()
```

Out[62]: 0

In [63]:
```
#create test dataframe and apply low mem usage
test_df = pd.read_csv(DATA_PATH + 'test.csv')
row_ids = test_df["row_id"]
test_df.drop("row_id", axis=1, inplace=True)
test_df = reduce_mem_usage(test_df)
```

```
Memory usage of dataframe is 954.38 MB
Memory usage after optimization is: 199.59 MB
Decreased by 79.1%
```

In [64]:
```
test_df = test_df.merge(building_df,left_on='building_id',right_on='building_id',how='left')
del building_df
gc.collect()
```

Out[64]:
```
0
```

In [65]:
```
weather_df = pd.read_csv(DATA_PATH + 'weather_test.csv')
weather_df = fill_weather_dataset(weather_df)
weather_df = reduce_mem_usage(weather_df)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:17: FutureWarning: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.


Memory usage of dataframe is 19.25 MB
Memory usage after optimization is: 9.05 MB
Decreased by 53.0%
```

In [66]:
```
test_df = test_df.merge(weather_df,how='left',on=['timestamp','site_id'])
del weather_df
gc.collect()
```

Out[66]:
```
0
```

In [67]:
```
test_df = features_engineering(test_df)
```

In [68]:
```
test = test_df.drop(['is_holiday', 'hour', 'weekend'], axis = 1)
del test_df
gc.collect()
test.head()
```

Out[68]:

	building_id	meter	site_id	primary_use	square_feet	air_temperature	cloud_coverage	dew_t
0	0	0	0	0	4.468308	17.799999	4.0	
1	1	0	0	0	3.973186	17.799999	4.0	
2	2	0	0	0	4.308396	17.799999	4.0	
3	3	0	0	0	5.042775	17.799999	4.0	
4	4	0	0	0	5.836206	17.799999	4.0	

In [69]:
```
test = ce.transform(test)
for feature in categorical_features:
    test[feature] = test[feature]/N_train
test = imp.transform(test)
```

In [70]:
```
results = 0
for model in models:
    results += np.expm1(model.predict(test))/ len(models)
    del model
    gc.collect()
```

In [71]:
```
del test, models
gc.collect()
```

Out[71]: 0

In [ ]:
```
results_df = pd.DataFrame({"row_id": row_ids, "meter_reading": np.clip(results
, 0, a_max=None)})
del row_ids,results
gc.collect()
results_df.to_csv("results.csv", index=False)
```

In [ ]:
```

```