Kaggle Project Report

Reshma, Yang,

Michael, Vincent,

Manaswi, Q

ASHRAE - Great Energy Predictor III

- Assessing the value of energy efficiency improvements can be challenging as there's no way to truly know how much energy a building would have used without the improvements. The best we can do is to build counterfactual models. Once a building is overhauled the new (lower) energy consumption is compared against modeled values for the original building to calculate the savings from the retrofit. More accurate models could support better market incentives and enable lower cost financing.
- This competition challenges you to build these counterfactual models across four energy types based on historic usage rates and observed weather. The dataset includes three years of hourly meter readings from over one thousand buildings at several different sites around the world.

Datasets (train and test)

building_metadata	site_id, building_id, primary_use, square_feet, year_built, floor_count
test	row_id, building_id, meter, timestamp
train	'building_id', 'meter', 'timestamp', 'meter_reading'
weather_train	'site_id', 'timestamp', 'air_temperature', 'cloud_coverage', 'dew_temperature', 'precip_depth_1_hr', 'sea_level_pressure', 'wind_direction', 'wind_speed'
weather_test	site_id, timestampe, air_temperature, cloud_coverage, 'dew_temperature', 'precip_depth_1_hr', 'sea_level_pressure', 'wind_direction', 'wind_speed'

Missing Values

#Check the missing values in each table and dataframe it

```
def cal_missing_values(df):
    data_dict = {}
    for i in df.columns:
        data_dict[i] = df[i].isnull().sum()/len(df[i])
    Dataframe = pd.DataFrame.from_dict(data_dict, orient = "index", columns = ['MissingValues'])
    return (Dataframe)
```

```
cal_missing_values(weather_test)
```

	MissingValues
site_id	0.000000
timestamp	0.000000
air_temperature	0.000375
cloud_coverage	0.506588
dew_temperature	0.001179
precip_depth_1_hr	0.344781
sea_level_pressure	0.076702
wind_direction	0.044618
wind_speed	0.001659

```
cal_missing_values(weather_train)
```

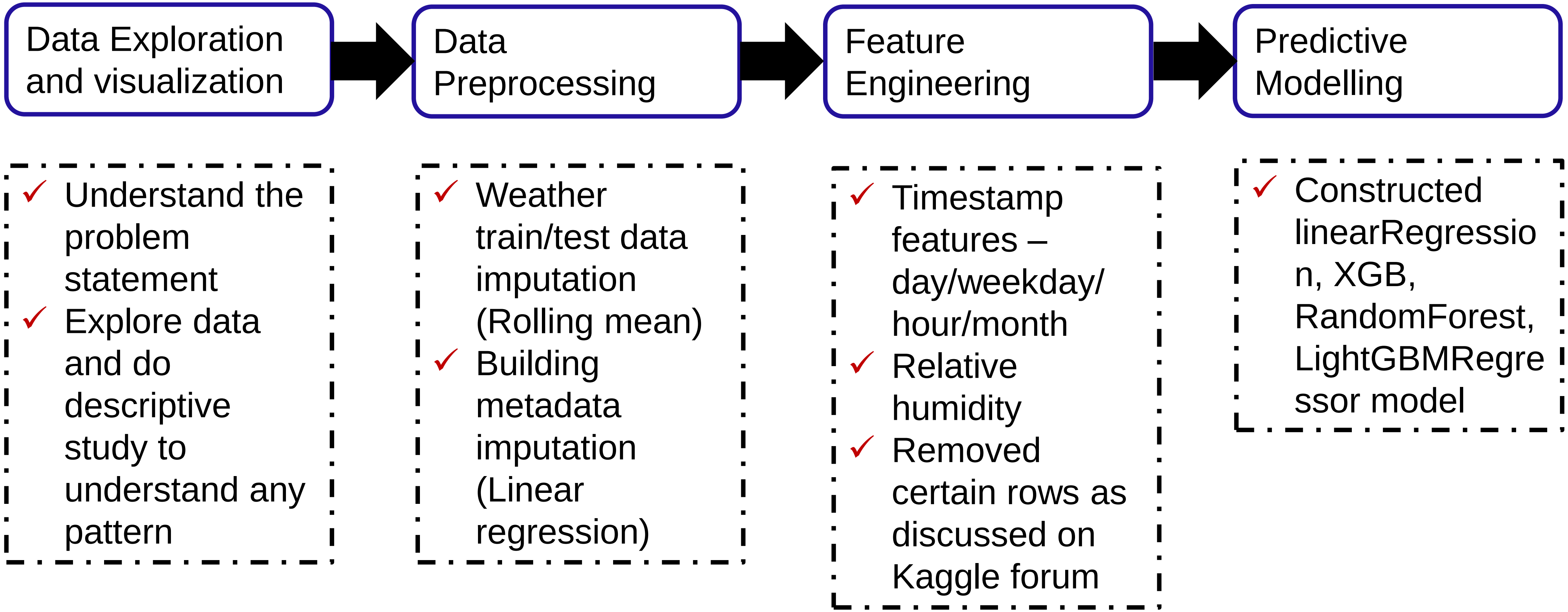
	MissingValues
site_id	0.000000
timestamp	0.000000
air_temperature	0.000393
cloud_coverage	0.494895
dew_temperature	0.000808
precip_depth_1_hr	0.359791
sea_level_pressure	0.075966
wind_direction	0.044844
wind_speed	0.002175

```
cal_missing_values(building_metadata)
```

	MissingValues
site_id	0.000000
building_id	0.000000
primary_use	0.000000
square_feet	0.000000
year_built	0.534161
floor_count	0.755003

Memory Reduce

```python
# Reducing memory
# Function to reduce the DF size
import numpy as np

def reduce_memory(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            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.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if 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)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:.2f} Mb ({:.1f}% reduction)'.format(end_mem,
                                                                                       100*(start_mem-end_mem)/start_mem))
    return df
```


Data Exploration and visualization
Data Preprocessing
Feature Engineering
Predictive Modelling
✓ Understand the problem statement
✓ Explore data and do descriptive study to understand any pattern
✓ Weather train/test data imputation (Rolling mean)
✓ Building metadata imputation (Linear regression)
✓ Timestamp features – day/weekday/hour/month
✓ Relative humidity
✓ Removed certain rows as discussed on Kaggle forum
✓ Constructed linearRegression, XGB, RandomForest, LightGBMRegressor model

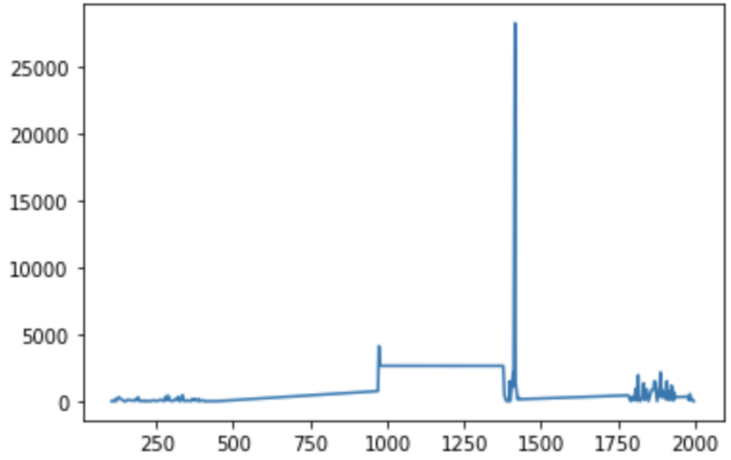
Replacing missing values and Filling with Rolling dates

- **Example of replacing missing values with mean/median/max/min:**

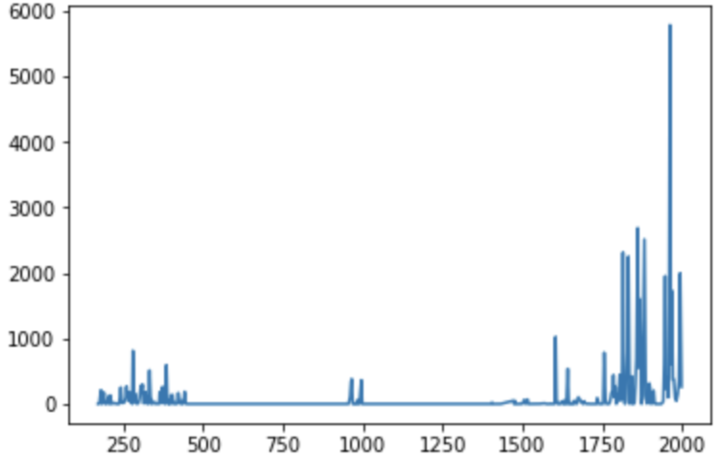
```
# Adding columns of mean, median, max and min of cloud coverage to replace the missing values

weather_train['cloud_coverage_mean'] = weather_train['cloud_coverage']

weather_train['cloud_coverage_median'] = weather_train['cloud_coverage']

weather_train['cloud_coverage_max'] = weather_train['cloud_coverage']

weather_train['cloud_coverage_min'] = weather_train['cloud_coverage']

weather_train['cloud_coverage_mean'] = weather_train['cloud_coverage_mean'].fillna(np.mean(weather_train.cloud_coverage))

weather_train['cloud_coverage_median'] = weather_train['cloud_coverage_median'].fillna(np.median(weather_train.cloud_coverage))

weather_train['cloud_coverage_max'] = weather_train['cloud_coverage_max'].fillna(np.max(weather_train.cloud_coverage))

weather_train['cloud_coverage_min'] = weather_train['cloud_coverage_min'].fillna(np.min(weather_train.cloud_coverage))

print(weather_train)
```

Missing values based on different meter type

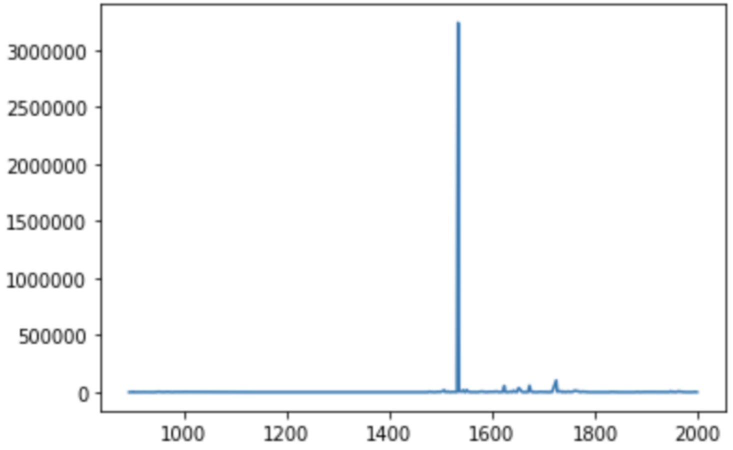
Meter = 3



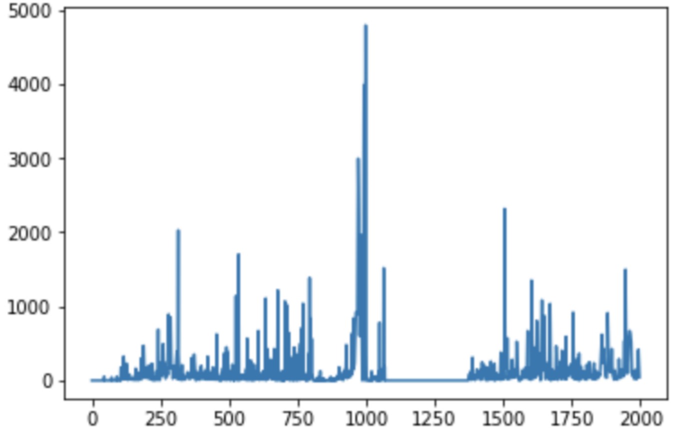
Meter = 1



Meter = 2



Meter = 0

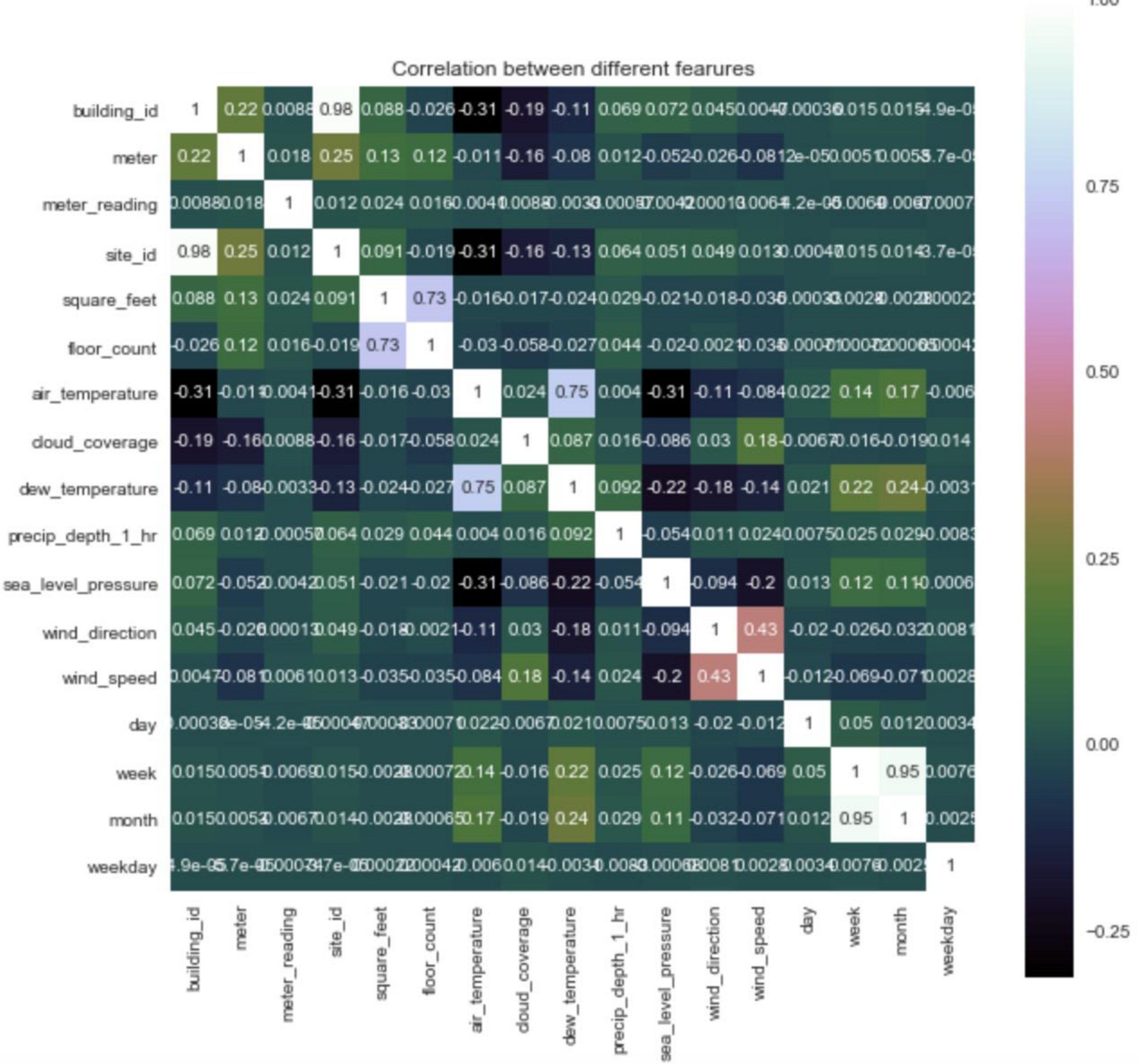


Imputation of missing value with rolling means

```
def rolling_mean_weather(df, cols, num_of_days):

    #assumes that functions will be mean, median, min, max

    for col in cols:

        for day in num_of_days:

            print ("processing missing values for", col, day)

            df[col+'_'+'R'+str(day)+'_'+'mean']   = df.groupby('site_id')[col].apply(lambda x: x.fillna(x.rolling(day, center=True, min_periods=1).mean()))

            df[col+'_'+'R'+str(day)+'_'+'median'] = df.groupby('site_id')[col].apply(lambda x: x.fillna(x.rolling(day, center=True, min_periods=1).median()))

            df[col+'_'+'R'+str(day)+'_'+'min']    = df.groupby('site_id')[col].apply(lambda x: x.fillna(x.rolling(day, center=True, min_periods=1).min()))

            df[col+'_'+'R'+str(day)+'_'+'max']    = df.groupby('site_id')[col].apply(lambda x: x.fillna(x.rolling(day, center=True, min_periods=1).max()))


    return df

cols_to_impute = ['air_temperature','cloud_coverage','dew_temperature','precip_depth_1_hr', 'sea_level_pressure', 'wind_direction', 'wind_speed' ]

weather_train  = rolling_mean_weather(weather_train, cols_to_impute, [3,5,7,14,30])

weather_test   = rolling_mean_weather(weather_test, cols_to_impute, [3, 5, 7, 14, 30])
```

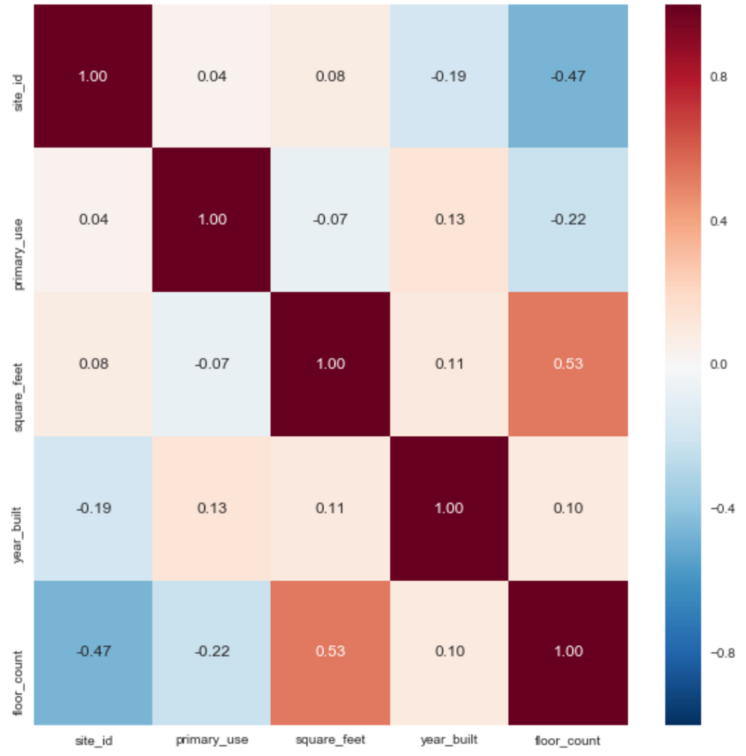
For building & weather

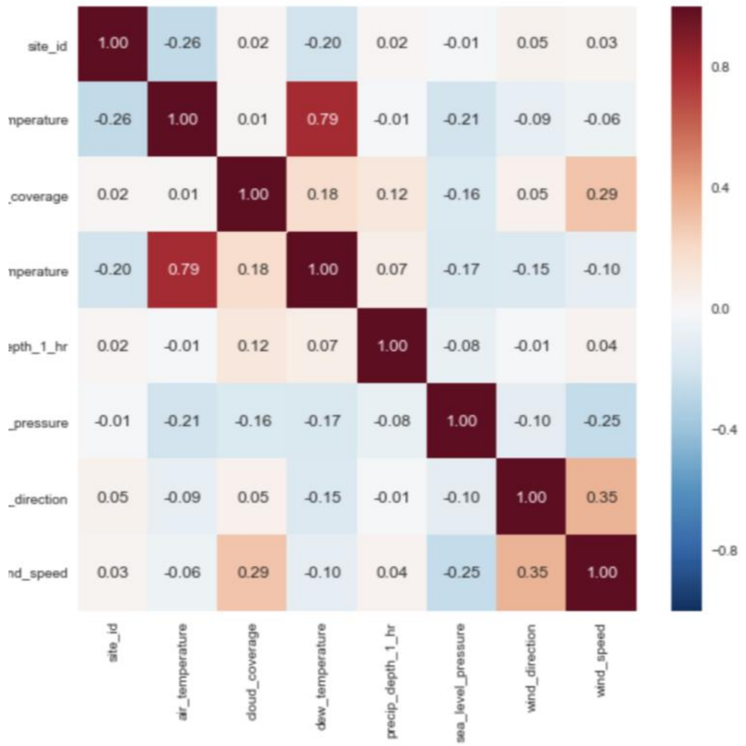
- building_fill & weather_fill

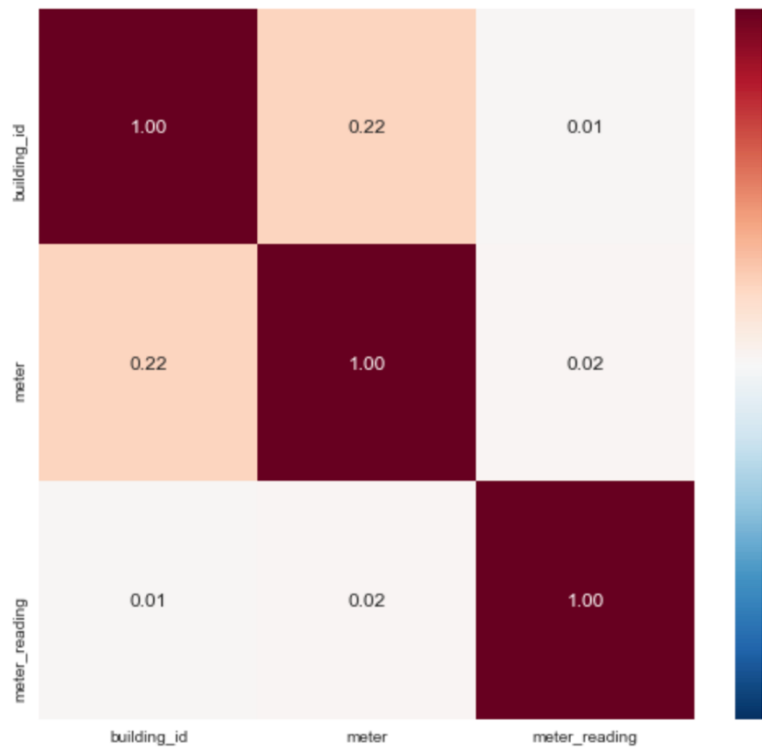


Feature Selection & Engineering

Correlation Matrix with Heatmap (For complete train)







For different datasets (correlation heatmap)

- From left to right:
  - building
  - weather
  - meter

PCA

1. Drop features:

   ('site_id', 'building_id', 'day', 'week', 'month', 'timestamp', 'primary_use', 'meter', 'meter_reading');

2. Use (StandardScaler) to standardize dataset;

3. **Cor Matrix: -->**

```
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
print('Covariance matrix \n%s' %cov_mat)
```

```
Covariance matrix
[[ 1.00000005  0.72749469 -0.01619819 -0.01698572 -0.02400392  0.02908052
  -0.02096596 -0.01810021]
 [ 0.72749469  1.00000005 -0.03009617 -0.05765468 -0.02717396  0.04390356
  -0.01960044 -0.00214412]
 [-0.01619819 -0.03009617  1.00000005  0.0236556   0.75151827  0.00404661
  -0.30835739 -0.1080514 ]
 [-0.01698572 -0.05765468  0.0236556   1.00000005  0.08655162  0.0160359
  -0.086134    0.03034822]
 [-0.02400392 -0.02717396  0.75151827  0.08655162  1.00000005  0.09193741
  -0.21615616 -0.18379703]
 [ 0.02908052  0.04390356  0.00404661  0.0160359   0.09193741  1.00000005
  -0.05431987  0.01077064]
 [-0.02096596 -0.01960044 -0.30835739 -0.086134   -0.21615616 -0.05431987
   1.00000005 -0.09421663]
 [-0.01810021 -0.00214412 -0.1080514   0.03034822 -0.18379703  0.01077064
  -0.09421663  1.00000005]]
```

```
print('NumPy covariance matrix: \n%s' %np.cov(X_std.T))
```

```
NumPy covariance matrix:
[[ 1.00000005  0.72749469 -0.01619819 -0.01698572 -0.02400392  0.02908052
  -0.02096596 -0.01810021]
 [ 0.72749469  1.00000005 -0.03009617 -0.05765468 -0.02717396  0.04390356
  -0.01960044 -0.00214412]
 [-0.01619819 -0.03009617  1.00000005  0.0236556   0.75151827  0.00404661
  -0.30835739 -0.1080514 ]
 [-0.01698572 -0.05765468  0.0236556   1.00000005  0.08655162  0.0160359
  -0.086134    0.03034822]
 [-0.02400392 -0.02717396  0.75151827  0.08655162  1.00000005  0.09193741
  -0.21615616 -0.18379703]
 [ 0.02908052  0.04390356  0.00404661  0.0160359   0.09193741  1.00000005
  -0.05431987  0.01077064]
 [-0.02096596 -0.01960044 -0.30835739 -0.086134   -0.21615616 -0.05431987
   1.00000005 -0.09421663]
 [-0.01810021 -0.00214412 -0.1080514   0.03034822 -0.18379703  0.01077064
  -0.09421663  1.00000005]]
```

PCA

4. Calculate Eigenvalues and Sort:

```
eig_vals, eig_vecs = np.linalg.eig(cov_mat)

print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[-1.00969717e-01 -6.94590867e-01  6.95139060e-01  1.17424381e-01
   3.27644379e-02  7.45460279e-02 -6.09173202e-02  5.54308309e-04]
 [-1.11214209e-01 -6.95242801e-01 -6.98037289e-01 -1.16869285e-01
   3.93112680e-02  1.48010754e-02 -3.98441507e-02 -1.53477344e-03]
 [ 6.40439345e-01 -8.53667421e-02  1.16621510e-01 -6.94153866e-01
   2.35014960e-01 -1.18441828e-01 -1.14576996e-01 -6.86397271e-02]
 [ 1.09973717e-01  4.71098447e-02 -2.79016677e-02 -7.26011738e-02
   6.70405379e-02  8.92501868e-01 -4.57158055e-02  4.20110768e-01]
 [ 6.36822063e-01 -8.23826679e-02 -1.18194396e-01  6.83057858e-01
   2.98190163e-01  1.80499599e-02  3.30341715e-02 -1.29608535e-01]
 [ 7.58351576e-02 -8.54766224e-02  3.02829673e-02 -8.14850844e-02
   7.07836184e-03 -6.63321709e-02  9.62074815e-01  2.22143849e-01]
 [-3.46319367e-01  9.43159996e-02  1.51872360e-02 -9.23939216e-02
   7.35999905e-01  2.18421514e-01  1.52415332e-01 -4.99765793e-01]
 [-1.54755705e-01  4.10547402e-02  5.37818809e-03  6.07294898e-02
   5.54067968e-01 -3.62218237e-01 -1.71971108e-01  7.09140859e-01]]

Eigenvalues
[1.95244956 1.73198718 0.27218663 0.22952678 0.73268605 0.96164828
 0.98772253 1.13179338]
```

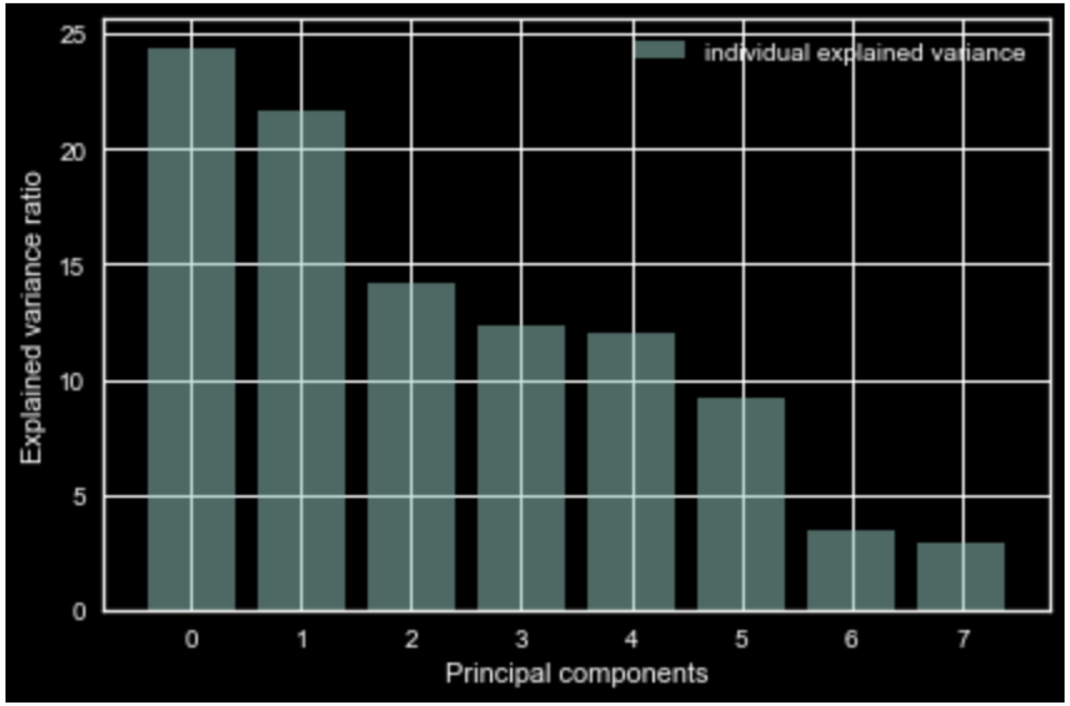
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenvalues
print('Eigenvalues in descending order:')
for i in eig_pairs:
    print(i[0])
```

```
Eigenvalues in descending order:
1.9524495621197497
1.731987182119231
1.1317933818078694
0.9877225278926159
0.9616482834175673
0.7326860481587607
0.2721866278444461
0.22952678236443003
```

PCA

5. Plot how much each Principle Component can explain

```
with plt.style.context('dark_background'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(8), var_exp, alpha=0.5, align='center',
            label='individual explained variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



PCA

6. Cumulative Explained Variance

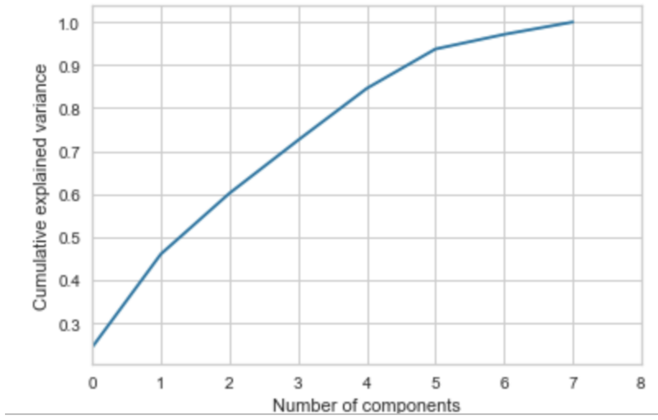
```
matrix_w = np.hstack((eig_pairs[0][1].reshape(8,1),
                      eig_pairs[1][1].reshape(8,1)
                    ))
print('Matrix W:\n', matrix_w)
```

```
Matrix W:
 [[-0.10096972 -0.69459087]
 [-0.11121421 -0.6952428 ]
 [ 0.64043934 -0.08536674]
 [ 0.10997372  0.04710984]
 [ 0.63682206 -0.08238267]
 [ 0.07583516 -0.08547662]
 [-0.34631937  0.094316  ]
 [-0.1547557   0.04105474]]
```

```
y = X_std.dot(matrix_w)
```

```
from sklearn.decomposition import PCA
pca = PCA().fit(X_std)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,8,1)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

```
Text(0, 0.5, 'Cumulative explained variance')
```



Training Methods and Models

- Regression Model **(1.87)**
- Random Forest Regressor **(1.53)**
- XGBoost (No result yet)
- **Lgbm (1.51)**