Analysis of New-York’s Energy and Water Data, for Years 2017, 2018 and 2019

Introduction

This project aims at understanding the relationship that exist between New York City (NYC) buildings, and their respective Greenhouse gas emissions. The Energy and Water disclosure data used for this project was obtained from NYC OpenData. The analysis was carried out using Amazon Sagemaker (Linear learner Algorithm).

Problem Statement

To identify the key parameters/features that affect the amount of GHG emissions from buildings, and develop a predictive model based.

Data

The data used for this project were obtained from NYC OpenData (https://data.cityofnewyork.us/browse?q=Energy%20and%20Water%20Data%20Disclosure&sortBy=relevance).

Data Cleaning and Extraction

The 2018 data was cleaned, and important features were obtained using XGBRegressor, these top features were then checked for multicollinearity using Variance Inflation Factor. Using the 2018 data, a preprocessing script was created which was used to preprocess the 2017 and 2019 data. The preprocessing script worked perfectly on the 2019 dataset, but it required little modification for the 2017 dataset.

Methodology/Data Analysis

The following steps were carried-out in this analysis.

1. Data set up
2. Data Preparation/Preprocessing
3. Training the model
4. Batch Transformation for Offline Inferences
5. Data-Capture Configuration, and Model deployment for Real-time inferences
6. Model Monitoring
7. Clean up

- Required libraries were imported.

```
# Data processing

import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Visualization

%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt

# Machine Learning

from sklearn.model_selection import train_test_split
from sklearn import metrics
import sklearn.feature_selection as fs
from statsmodels.stats.outliers_influence import variance_inflation_factor
import sklearn.metrics as metrics
from sklearn.datasets import *
import sklearn.model_selection
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.inspection import permutation_importance
import xgboost
from xgboost import XGBRegressor

# SageMaker

from time import sleep
import boto3
import sagemaker
import s3fs
import os
from time import sleep, gmtime, strftime
import json
import sys
import subprocess
import pkg_resources
import io
from urllib.parse import urlparse

# SageMaker Experiment

import time
from time import strftime

#!pip install sagemaker-experiments --> if it isn't installed.
from sagemaker.analytics import ExperimentAnalytics
from smexperiments.experiment import Experiment
from smexperiments.trial import Trial
from smexperiments.trial_component import TrialComponent
from smexperiments.tracker import Tracker
```

-

- Bucket name and dataset were defined and the datafile was read and renamed.

```
[3]: data_bucket = 'just-abdul-aws' #Defining bucket name
     subfolder = 'NewYork' #Defining sub-folder name
     dataset = '2018.csv'

[4]: role = sagemaker.get_execution_role()
     sess = sagemaker.Session()
     s3 = s3fs.S3FileSystem(anon=False)
     sm = boto3.Session().client('sagemaker')

[5]: df = pd.read_csv(f's3://{data_bucket}/{subfolder}/{dataset}')
```

Understanding the data

```
[6]: df.head(5)
```

[6]:

	Order	Property Id	Property Name	Parent Property Id	Parent Property Name	City Building	BBL - 10 digits	NYC Borough, Block and Lot (BBL) self - reported	NYC Building Identification Number (BIN)	Address 1 (self - reported)	...
0	1	7365	1155	Not Applicable: Standalone Property	Not Applicable: Standalone Property	No	1009970029	1009970029	1022631	1155 Avenue of the Americas	...
1	2	8139	200	Not Applicable: Standalone Property	Not Applicable: Standalone Property	No	1013150001	1013150001	1037545	200 East 42nd St.	...
2	3	8604	114	Not Applicable: Standalone Property	Not Applicable: Standalone Property	No	1009990019	1009990019	1022667	114 West 47th st	...
3	4	8841	733	Not Applicable: Standalone Property	Not Applicable: Standalone Property	No	1013190047	1013190047	1037596	733 Third Avenue	...
4	5	11809	Conde Nast Building	Not Applicable: Standalone Property	Not Applicable: Standalone Property	No	1009950005	1009950005	1085682	4 Times Square	...

5 rows × 67 columns

- Cleaning the data to deal with Missing Values and Duplicates.

Data Cleaning/Preparation

```
[10]: df = df.replace('Not Available', None)

[11]: # Removing the character in the data.

      df[['DOF Gross Floor Area  (ft²)', 'Year Built']] = df[['DOF Gross Floor Area  (ft²)', 'Year Built']].replace(',','', regex=True)

[12]: # Selecting Obj type which are actually numbers

      cols = ['NYC Building Identification Number (BIN)','Postcode','DOF Gross Floor Area  (ft²)','Year Built','ENERGY STAR Score','Sourc

[13]: # Selecting Obj type which are actually numbers

      cols2 = ['BBL - 10 digits','Street Number', 'Self-Reported Gross Floor Area (ft²)','Largest Property Use Type - Gross Floor Area (f

[14]: # Converting Obj type to Num

      col = cols + cols2
      df[col] = df[col].apply(pd.to_numeric, errors='coerce', axis=1)

[15]: Sum = df.isnull().sum()
      Percentage = ( df.isnull().sum()/df.isnull().count())

      nulls = pd.concat([Sum,Percentage], axis =1, keys= ['Sum', 'Percentage']).sort_values(by='Sum', ascending=False).head(30)
      nulls

INFO:numexpr.utils:NumExpr defaulting to 2 threads.
```

[15]:

	Sum	Percentage
Largest Property Use Type - Gross Floor Area (ft²)	25936	0.999884
Self-Reported Gross Floor Area (ft²)	25934	0.999807
Propane Use (kBtu)	15939	0.614480
Longitude	9105	0.351016
Latitude	9105	0.351016
NTA	9105	0.351016
Community Board	9105	0.351016
Council District	9105	0.351016
Census Tract	9105	0.351016

- Columns/Features with more than 30% missing values were dropped

A 30% missing value threshold is selected. columns with more than 30% missing values will be dropped.

[16]:
```python
#Selecting the columns with more than 30% missing values

remove = nulls[nulls['Percentage']>=0.30]
remove.index.name = 'Features' # Renames the index to 'Features'
remove.reset_index(level=0, inplace=True) # Re-index's the dataframe, turning 'Features' into a column
remove
```

[16]:

	Features	Sum	Percentage
0	Largest Property Use Type - Gross Floor Area (...	25936	0.999884
1	Self-Reported Gross Floor Area (ft²)	25934	0.999807
2	Propane Use (kBtu)	15939	0.614480
3	Longitude	9105	0.351016
4	Latitude	9105	0.351016
5	NTA	9105	0.351016
6	Community Board	9105	0.351016
7	Council District	9105	0.351016
8	Census Tract	9105	0.351016

[17]:
```python
remove['Features'].tolist()
```

[17]:
```
['Largest Property Use Type - Gross Floor Area (ft²)',
 'Self-Reported Gross Floor Area (ft²)',
 'Propane Use (kBtu)',
 'Longitude',
 'Latitude',
 'NTA',
 'Community Board',
 'Council District',
 'Census Tract']
```

- Categorical data types were converted to numerical values using label encoder.

Dealing with Cat datatype

[26]:
```python
le=LabelEncoder()

for col in df_obj.columns.values:
    # Encoding only categorical variables
    if df_obj[col].dtypes=='object':
        # Using whole data to form an exhaustive list of levels
        data=df_obj[col]
        le.fit(data.values)
        df_obj[col]=le.transform(df_obj[col])
```

[27]:
```python
df_obj
```

[27]:

	City Building	Borough	Primary Property Type - Self Selected	List of All Property Use Types at Property
0	0	2	33	86
1	0	2	33	97
2	0	2	13	325
3	0	2	33	889
4	0	2	33	973
...	...	...	...	...
25934	0	0	30	679
25935	0	0	30	679
25936	0	1	19	478
25937	0	2	30	679
25938	0	0	28	987

25939 rows × 9 columns

[28]:
```python
df_cleaned = pd.concat([df_pipe,df_obj], axis =1)
```

- Feature selection was carried out using XGBRegressor.

Feature Selection

[30]:
```python
# Selecting features and the target variable

Y_selection = df_cleaned['Total GHG Emissions (Metric Tons CO2e)']  #Target Feature
X_selection = df_cleaned.drop("Total GHG Emissions (Metric Tons CO2e)", 1)
```

[31]:
```python
# Partitioning the dataset into 2/3 training and 1/3 test set.
X_train_selection, X_test_selection, Y_train_selection, Y_test_selection = sklearn.model_selection.train_test_split(X_selection, Y_selection, test_size=0.33)

# Further splitting the training set into a validation set; 2/3 training set, and 1/3 validation set
X_train_selection, X_val_selection, Y_train_selection, Y_val_selection = sklearn.model_selection.train_test_split(X_train_selection, Y_train_selection, test_size=0.33)
```

[32]:
```python
# Feature Selection using XGBRegressor

X_data, y_label = make_regression(
    n_samples=X_train_selection.shape[0], n_features=X_train_selection.shape[1], n_informative=10, random_state=1
)
xgboost_model = XGBRegressor()
xgboost_model.fit(X_data, y_label)

feature_scores = []
feature_importances_xgboost = xgboost_model.feature_importances_
for index, importance_score in enumerate(feature_importances_xgboost):
    feature_scores.append([X_train_selection.columns[index], importance_score])
```

[33]:
```python
# List of features and their respective scores, depicting their importance

sorted_scores = sorted(np.array(feature_scores), key=lambda s: float(s[1]), reverse=True)
print(np.array(sorted_scores))
```

```
[['Weather Normalized Site Electricity (kWh)' '0.17540875']
 ['Site EUI (kBtu/ft²)' '0.1368753']
 ['3rd Largest Property Use Type' '0.1244295']
 ['Indirect GHG Emissions (Metric Tons CO2e)' '0.12201577']
 ['Largest Property Use Type' '0.12166264']
 ['Fuel Oil #5 & 6 Use (kBtu)' '0.09345414']
 ['DOF Gross Floor Area  (ft²)' '0.06590423']
 ['3rd Largest Property Use Type - Gross Floor Area (ft²)' '0.059950735']
 ['List of All Property Use Types at Property' '0.057724368']
 ['Diesel #2 Use (kBtu)' '0.0021826986']
 ['Direct GHG Emissions (Metric Tons CO2e)' '0.0020963778']
 ['Electricity Use - Grid Purchase (kBtu)' '0.0019719838']
 ['2nd Largest Property Use Type' '0.0018877045']
```

- Checking if there's multicollinearity among the top features, using Variance inflation factor

```python
Top_features['Features'].tolist()
```

[36]:
```
['DOF Gross Floor Area  (ft²)',
 '3rd Largest Property Use Type - Gross Floor Area (ft²)',
 'Site EUI (kBtu/ft²)',
 'Fuel Oil #5 & 6 Use (kBtu)',
 'Diesel #2 Use (kBtu)',
 'Weather Normalized Site Electricity (kWh)',
 'Direct GHG Emissions (Metric Tons CO2e)',
 'Indirect GHG Emissions (Metric Tons CO2e)',
 'List of All Property Use Types at Property',
 'Largest Property Use Type',
 '3rd Largest Property Use Type']
```

Checking for Multi-Collinearity

[37]:
```python
# VIF = 1, indicates no correlation between the independent variable and the other variables
# VIF exceeding 5 or 10 indicates high multicollinearity between the independent variable and the others

def calc_vif(C):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = C.columns
    vif["VIF"] = [variance_inflation_factor(C.values, i) for i in range(C.shape[1])]

    return(vif)
```

- High Correlation found among; “List of All Property Use Types at Property” , “Largest Property Use Type” and “3rd Largest Property Use Type”.

[39]:
```
C = X_selection[Top_features['Features'].tolist()].iloc[:,:]
calc_vif(C)
```

[39]:

	variables	VIF
0	DOF Gross Floor Area (ft²)	1.432119
1	3rd Largest Property Use Type - Gross Floor Ar...	1.219515
2	Site EUI (kBtu/ft²)	5.524459
3	Fuel Oil #5 & 6 Use (kBtu)	1.477446
4	Diesel #2 Use (kBtu)	1.239906
5	Weather Normalized Site Electricity (kWh)	1.483562
6	Direct GHG Emissions (Metric Tons CO2e)	1.193485
7	Indirect GHG Emissions (Metric Tons CO2e)	5.494043
8	List of All Property Use Types at Property	33.208199
9	Largest Property Use Type	29.140909
10	3rd Largest Property Use Type	6.306261

-

- Preparing data for Amazon Sagemaker

Converting Data to csv

```
[53]: train_file = train.to_csv(None, header=False, index=False).encode() # Doesn't include column header
      train_file_header = train.to_csv(None, index=False).encode()
      val_file = val.to_csv(None, header=False, index=False).encode() #Encode is to ensure text in csv is save
      test_file = test.to_csv(None, header=False, index=False).encode()
      test_2017 = X_test_2017.to_csv(None, header=False, index=False).encode()
      test_2019 = X_test_2019.to_csv(None, header=False, index=False).encode()
```

Saving the CSV files to S3

```
[54]: with s3.open(f'{data_bucket}/{subfolder}/processed/train.csv', 'wb') as f:
          f.write(train_file)
      with s3.open(f'{data_bucket}/{subfolder}/train_headers/train_data_with_headers.csv', 'wb') as f:
          f.write(train_file_header)
      with s3.open(f'{data_bucket}/{subfolder}/processed/val.csv', 'wb') as f:
          f.write(val_file)
      with s3.open(f'{data_bucket}/{subfolder}/processed/test.csv', 'wb') as f:
          f.write(test_file)
      with s3.open(f'{data_bucket}/{subfolder}/processed/test_2017.csv', 'wb') as f:
          f.write(test_2017)
      with s3.open(f'{data_bucket}/{subfolder}/processed/test_2019.csv', 'wb') as f:
          f.write(test_2019)
```

File Location

```
[55]: raw_data_location = f's3://{data_bucket}/{subfolder}/{dataset}'
      train_location = f's3://{data_bucket}/{subfolder}/processed/train.csv'
      train_header_location = f's3://{data_bucket}/{subfolder}/train_headers/train_data_with_headers.csv'
      val_location = f's3://{data_bucket}/{subfolder}/processed/val.csv'
      test_location = f's3://{data_bucket}/{subfolder}/processed/test.csv'
      test_2017_location = f's3://{data_bucket}/{subfolder}/processed/test_2017.csv'
      test_2019_location = f's3://{data_bucket}/{subfolder}/processed/test_2019.csv'
```

Preparing the CSV data for SageMaker

```
[56]: input_train = sagemaker.TrainingInput(s3_data=train_location, content_type="text/csv")
      input_validation = sagemaker.TrainingInput(s3_data=val_location, content_type="text/csv")
```

- Creating SageMaker Experiment

Creating SageMaker Experiment

Sagemaker experiment helps track preprocessing and training changes

[58]:
```
# Create a SageMaker Experiment

create_date = strftime("%Y-%m-%d-%H-%M-%S")
LL_experiment = Experiment.create(experiment_name = "LL-Build-train-deploy-{}".format(create_date),
                                   description = "Predict GHG Emission from Energy and Water data",
                                  sagemaker_boto_client=sm,
                                   tags = [{'Key': 'LL-experiments', 'Value': 'NYC'}])
print(LL_experiment)
```

```
Experiment(sagemaker_boto_client=<botocore.client.SageMaker object at 0x7f347d53c0d0>,experiment_name='LL-Build-t
d Water data',tags=[{'Key': 'LL-experiments', 'Value': 'NYC'}],experiment_arn='arn:aws:sagemaker:us-east-2:077107
{'RequestId': '37c28d59-51c6-488d-9e96-0a517e2b9c93', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid':
1.1', 'content-length': '113', 'date': 'Sat, 26 Jun 2021 07:32:28 GMT'}, 'RetryAttempts': 0})
```

[59]:
```
# Start Tracking parameters used in the Pre-processing pipeline.

with Tracker.create(display_name="Preprocessing", sagemaker_boto_client=sm) as tracker:
    tracker.log_parameters({"train_test_split_ratio": 0.333, "random_state":0})
    # we can log the s3 uri to the dataset we just uploaded
    tracker.log_input(name="ccdefault-raw-dataset", media_type="s3/uri", value=raw_data_location)
    tracker.log_input(name="ccdefault-train-header-dataset", media_type="s3/uri", value=train_header_location)
    tracker.log_input(name="ccdefault-train-dataset", media_type="s3/uri", value=train_location)
    tracker.log_input(name="ccdefault-val-dataset", media_type="s3/uri", value=val_location)
    tracker.log_input(name="ccdefault-test-dataset", media_type="s3/uri", value=test_location)
    tracker.log_input(name="ccdefault-test2017-dataset", media_type="s3/uri", value=test_2017_location)
    tracker.log_input(name="ccdefault-test2019-dataset", media_type="s3/uri", value=test_2019_location)
```

- Training the model

Training the Model

[60]:
```
sess = sagemaker.Session()
from sagemaker.amazon.amazon_estimator import get_image_uri
container = get_image_uri(boto3.Session().region_name, 'linear-learner', 'latest')

# Tracking/Monitoring
preprocessing_trial_component = tracker.trial_component

trial_name = "NewYork-Training-job-{}".format(create_date)
Linear_trial = Trial.create(trial_name=trial_name, experiment_name=LL_experiment.experiment_name)

Linear_trial.add_trial_component(preprocessing_trial_component)
Linear_training_job_name = "NewYork-Training-job-{}".format(create_date)



Linear_Model = sagemaker.estimator.Estimator(
    container,
    role,
    instance_count=1,
    instance_type="ml.m5.large",
    output_path= f's3://{data_bucket}/{subfolder}/newmodel',
    sagemaker_session=sess)

# Setting-up Hyperparameter

Linear_Model.set_hyperparameters(
    feature_dim=X_train.shape[1],
    predictor_type="regressor",
    mini_batch_size=100
    )

Linear_Model.fit({"train": input_train, "validation": input_validation}, wait=True, job_name=Linear_training_job_name,
         experiment_config={"TrialName": Linear_trial.trial_name, #log training job in Trials for lineage
            "TrialComponentDisplayName": "Training"})
```

...

- Batch Transform to be able to access inference in offline mode without invoking the endpoint.

Batch Transform for offline inference

[61]:
```
%%time

Linear_transformer = Linear_Model.transformer(instance_count = 1, instance_type = 'ml.m5.large', accept = 'text/csv')
```

```
INFO:sagemaker:Creating model with name: linear-learner-2021-06-26-07-38-12-600
CPU times: user 21.7 ms, sys: 0 ns, total: 21.7 ms
Wall time: 394 ms
```

[62]:
```
%%time

Linear_transformer.transform(test_location, split_type='Line', content_type='text/csv')
```

•••

[63]:
```
Linear_transformer.wait()
```

•••

[64]:
```
# Specify's the batch output location

data_dir =  f's3://{data_bucket}/{subfolder}/processed/'

!aws s3 cp --recursive $Linear_transformer.output_path $data_dir
```

```
copy: s3://sagemaker-us-east-2-077107849065/linear-learner-2021-06-26-07-38-13-001/test.csv.out to s3://just-abdul-aws/NewYork/processed/test.csv.out
```

Reviewing Batch Transform Output

[65]:
```
# Function to access the batch transform result

def get_csv_output_from_s3(s3uri, file_name):
    parsed_url = urlparse(s3uri)
    bucket_name = parsed_url.netloc
    prefix = parsed_url.path[1:]
    s3 = boto3.resource('s3')
    obj = s3.Object(bucket_name, '{}/{}'.format(prefix, file_name))
    return obj.get()["Body"].read().decode('utf-8')
pred = get_csv_output_from_s3(Linear_transformer.output_path, 'test.csv.out')
pred = pd.read_csv(io.StringIO(pred), sep=",", header=None)
pred
```

- Offline inference using Batch transform.

[69]:
```
test_mae_linear = np.mean(np.abs(Y_test - pred['Prediction'].tolist()))
test_mae_baseline = np.mean(np.abs(Y_test - np.median(Y_train)))  ## training median as baseline predictor

print("Test MAE Baseline :", round(test_mae_baseline, 3))
print("Test MAE Linear:", round(test_mae_linear, 3))
```

```
Test MAE Baseline : 1461.564
Test MAE Linear: 102.206
```

[70]:
```
actual = Y_test
actual = actual.reset_index(drop=True)
```

[71]:
```
model_result = pd.concat([pred,actual], axis=1)
model_result
```

[71]:

	Prediction	Total GHG Emissions (Metric Tons CO2e)
0	569.815125	557.8
1	2471.048096	2542.9
2	508.671387	504.9
3	167.366013	198.3
4	261.943848	271.6
...	...	...
8555	63.372627	74.2
8556	935.438354	933.8
8557	6.237604	27.2
8558	389.995972	433.9
8559	271.183960	270.9

8560 rows × 2 columns

- Deploying model and configuring DataCapture to retrieve real time inference upon invoking the endpoint.

```
[72]: from sagemaker.model_monitor import DataCaptureConfig

s3_capture_upload_path = 's3://{}/{}/monitoring/newDC/datacapture'.format(data_bucket, subfolder)

endpoint_name = "LL-model-monitor-" + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("EndpointName={}".format(endpoint_name))

data_capture_config = DataCaptureConfig(
    enable_capture=True,
    sampling_percentage=100,
    destination_s3_uri=s3_capture_upload_path,
    capture_options = ["REQUEST"]

)

# Deploying the Model

predictor = Linear_Model.deploy(
    initial_instance_count=1,
    instance_type="ml.m4.xlarge",
    endpoint_name=endpoint_name,
    data_capture_config=data_capture_config,
)
```

```
INFO:sagemaker:Creating model with name: linear-learner-2021-06-26-07-49-39-583
EndpointName=LL-model-monitor-2021-06-26-07-49-39
INFO:sagemaker:Creating endpoint with name LL-model-monitor-2021-06-26-07-49-39
---------------!
```

- Testing the model on the 2019 data.

Testing the Model on the 2019 data

```
[77]: %%time
from itertools import islice
import math
import struct
import boto3

runtime_client = boto3.client('sagemaker-runtime')


with s3.open(f'{data_bucket}/{subfolder}/processed/test_2019.csv') as f:
    payload = f.read().strip()
response = runtime_client.invoke_endpoint(
    EndpointName=predictor.endpoint, ContentType="text/csv", Body=payload
)
```

```
WARNING:sagemaker.deprecations:The endpoint attribute has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
CPU times: user 40.5 ms, sys: 7.38 ms, total: 47.9 ms
Wall time: 477 ms
```

```
[78]: result_2019 = json.loads(response["Body"].read().decode())
pred_2019 = np.array([r["score"] for r in result_2019["predictions"]])

pred_2019 = pd.DataFrame(pred_2019)
pred_2019.columns=['Prediction']

#Evaluating the model on the 2019 data
test_mae_linear = np.mean(np.abs(Y_test_2019 - pred_2019['Prediction'].tolist()))
test_mae_baseline = np.mean(np.abs(Y_test_2019 - np.median(Y_train_2019)))  ## training median as baseline predictor

print("Test MAE Baseline :", round(test_mae_baseline, 3))
print("Test MAE Linear:", round(test_mae_linear, 3))
```

```
Test MAE Baseline : 586.107
Test MAE Linear: 182.055
```

- 2019 predictions against the actual GHG emission.

[80]:

	Prediction	Total GHG Emissions (Metric Tons CO2e)
0	487.402191	481.4
1	355.717224	335.7
2	1151.969971	1126.0
3	448.382568	453.7
4	43.560131	0.0
...	...	...
20830	1807.272461	1185.5
20831	146.661255	153.8
20832	365.344299	289.0
20833	171.194366	186.9
20834	447.833649	406.6

20835 rows × 2 columns

- Testing the model on the 2017 data

Testing the Model on the 2017 data

[81]:
```
%%time
from itertools import islice
import math
import struct
import boto3

runtime_client = boto3.client('sagemaker-runtime')


with s3.open(f'{data_bucket}/{subfolder}/processed/test_2017.csv') as f:
    payload = f.read().strip()
response = runtime_client.invoke_endpoint(
    EndpointName=predictor.endpoint_name, ContentType="text/csv", Body=payload
)
```

```
CPU times: user 42.2 ms, sys: 4.48 ms, total: 46.7 ms
Wall time: 547 ms
```

[82]:
```
result_2017 = json.loads(response["Body"].read().decode())
pred_2017 = np.array([r["score"] for r in result_2017["predictions"]])

pred_2017 = pd.DataFrame(pred_2017)
pred_2017.columns=['Prediction']

# Evaluating the model on the 2017 data
test_mae_linear = np.mean(np.abs(Y_test_2017 - pred_2017['Prediction'].tolist()))
test_mae_baseline = np.mean(np.abs(Y_test_2017 - np.median(Y_train_2017)))  ## training median as baseline predictor

print("Test MAE Baseline :", round(test_mae_baseline, 3))
print("Test MAE Linear:", round(test_mae_linear, 3))
```

```
Test MAE Baseline : 146.415
Test MAE Linear: 23971018.269
```

- 2017 predictions against actual GHG emissions

[84]:

	Prediction	Total GHG Emissions (Metric Tons CO2e)
0	4.485686e+05	506.8
1	4.053576e+06	280.3
2	8.780939e+05	230.9
3	3.448236e+06	311.8
4	4.845730e+06	798.5
...	...	...
26010	2.956354e+06	52.6
26011	1.166531e+07	706.6
26012	5.052904e+06	402.7
26013	4.856892e+06	446.5
26014	1.063050e+07	578.6

26015 rows × 2 columns

- Monitoring setup and verifying its correctly capturing the incoming data.

Monitoring

Verify that Model Monitor is correctly capturing the incoming data.

[85]:

```
# Extract the captured json files.

data_capture_prefix = '{}/monitoring'.format(subfolder)
s3_client = boto3.Session().client('s3')
current_endpoint_capture_prefix = '{}/newDC/datacapture/{}/AllTraffic'.format(data_capture_prefix, endpoint_name)
print(current_endpoint_capture_prefix)
result = s3_client.list_objects(Bucket=data_bucket, Prefix=current_endpoint_capture_prefix)
capture_files = [capture_file.get("Key") for capture_file in result.get('Contents')]
```

```
NewYork/monitoring/newDC/datacapture/LL-model-monitor-2021-06-26-07-49-39/AllTraffic
```

[86]:

```
def get_obj_body(obj_key):
    return s3_client.get_object(Bucket=data_bucket, Key=obj_key).get("Body").read().decode("utf-8")


capture_file = get_obj_body(capture_files[-1])
print(capture_file[:2000])
```


```
{"captureData":{"endpointInput":{"observedContentType":"text/csv","mode":"INPUT","data":"73313.0,3000.0,127.7,0.0,0.0,735124.4,
0.0,2542.9,32\n69590.0,13857.0,95.1,6875249.5,602797.8,288079.3,418.0,86.9,29\n36200.0,0.0,67.5,522165.0,20492310.3,335426.6,10
5,32\n40916.0,23200.0,17.4,0.0,0.0,163325.3,7.7,48.2,29\n50170.0,5343.0,79.6,5086026.6,0.0,237512.7,167.7,71.0,29\n39600.0,6170
1,34074.0,14322605.8,709304.3,725.8,205.8,29\n56986.0,6200.0,64.7,0.0,602797.8,390923.5,493.3,117.0,29\n56986.0,30000.0,88.9,28
0436.2,20492310.3,963999.8,6.7,278.6,50\n26225.0,9561.0,366.2,2739775.5,20492310.3,613946.6,397.6,179.4,29\n114003.0,0.0,84.5,3
2,67204.2,777382.4,248.5,233.0,29\n135000.0,16000.0,96.0,26012250.6,69000.0,2482010.4,229.0,731.9,59\n47062.0,25806.0,106.6,0.0
0.1,2517504.2,336.5,2642.7,29\n130000.0,16418.0,103.0,8434640.4,0.0,734072.6,574.4,217.6,29\n119985.0,8000.0,77.2,8342250.2,234
5.8,98408.0,119.2,28.9,29\n77502.0,2220.0,97.8,5690550.1,69000.0,1203450.8,181.0,347.8,18\n30400.0,9318.0,71.2,4950600.1,143226
2600.5,531.4,200.1,29\n56986.0,3400.0,89.8,5250000.1,2718599.9,105920.1,187.3,31.2,29\n26403.0,4282.0,37.3,2379513.0,0.0,59999.
8.4,113.6,29\n62345.0,400.0,127.0,4225499.6,14322605.8,286929.3,391.6,86.4,29\n62612.0,3400.0,85.5,351678.0,2718599.9,854972.9,
13.0,442.9,21\n93480.0,2777.0,76.8,8342250.2,67204.2,453424.1,2
```

- Creating a baseline with which realtime traffic can be compared and setting up a schedule to continuously evaluate and compare against the baseline after it has been created.

```
[92]: constraints_df = pd.io.json.json_normalize(
          baseline_job.suggested_constraints().body_dict["features"]
      )
      constraints_df.head(10)
```

[92]:

	name	inferred_type	completeness	num_constraints.is_non_negative
0	Total GHG Emissions (Metric Tons CO2e)	Fractional	1.0	True
1	DOF Gross Floor Area (ft?)	Fractional	1.0	True
2	3rd Largest Property Use Type - Gross Floor Ar...	Fractional	1.0	True
3	Site EUI (kBtu/ft?)	Fractional	1.0	True
4	Fuel Oil #5 & 6 Use (kBtu)	Fractional	1.0	True
5	Diesel #2 Use (kBtu)	Fractional	1.0	True
6	Weather Normalized Site Electricity (kWh)	Fractional	1.0	True
7	Direct GHG Emissions (Metric Tons CO2e)	Fractional	1.0	True
8	Indirect GHG Emissions (Metric Tons CO2e)	Fractional	1.0	False
9	Largest Property Use Type	Integral	1.0	True

- Setting up Monitoring Frequency

Set up endpoint monitoring frequency/timing

```
[94]: from sagemaker.model_monitor import CronExpressionGenerator
      from time import gmtime, strftime

      mon_schedule_name = "Linear-Learner-model-monitor-schedule-" + strftime("%Y-%m-%d-%H-%M-%S", gmtime())

      my_default_monitor.create_monitoring_schedule(
          monitor_schedule_name=mon_schedule_name,
          endpoint_input=predictor.endpoint,
          statistics=my_default_monitor.baseline_statistics(),
          constraints=my_default_monitor.suggested_constraints(),
          schedule_cron_expression=CronExpressionGenerator.hourly(),
          enable_cloudwatch_metrics=True,
      )
```

```
WARNING:sagemaker.deprecations:The endpoint attribute has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
INFO:sagemaker.model_monitor.model_monitoring:Creating Monitoring Schedule with name: Linear-Learner-model-monitor-schedule-2021-06-26-08-09-58
```

- Generating artificial traffic to test the model monitoring

Start generating some artificial data traffic with the 2019 data

The cell below starts a thread to send some traffic to the endpoint. If there is no traffic, the monitoring jobs will be marked as Failed since there is no data to process.

[95]:
```python
from threading import Thread
from time import sleep
import time

endpoint_name = predictor.endpoint
runtime_client = boto3.client("runtime.sagemaker")

# (just repeating code from above for convenience/ able to run this section independently)
def invoke_endpoint(ep_name, runtime_client):
    with s3.open(f'{data_bucket}/{subfolder}/processed/test_2019.csv') as f:
        for row in f:
            payload = row.rstrip()
            response = runtime_client.invoke_endpoint(
                EndpointName=ep_name, ContentType="text/csv", Body=payload
            )
            response["Body"].read()
            time.sleep(1)


def invoke_endpoint_forever():
    while True:
        invoke_endpoint(endpoint_name, runtime_client)


thread = Thread(target=invoke_endpoint_forever)
thread.start()
```

```
WARNING:sagemaker.deprecations:The endpoint attribute has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

- Checking Monitoring Status. Monitoring status indicates "**Completed job with one violation found**". This tells us that the 2019 dataset violates one of **baseline/constraint** set by the 2018 dataset.

Checking Monitoring Status

[96]:
```python
desc_schedule_result = my_default_monitor.describe_schedule()
print('Schedule status: {}'.format(desc_schedule_result['MonitoringScheduleStatus']))
```

```
Schedule status: Scheduled
```

[98]:
```python
# View Monitoring Schedule

executions = my_default_monitor.list_executions()
executions
```

[98]:
```
[<sagemaker.model_monitor.model_monitoring.MonitoringExecution at 0x7f345c1a3450>]
```

[99]:
```python
latest_execution = executions[-1]

latest_execution.describe()['ProcessingJobStatus']
latest_execution.describe()['ExitMessage']
```

[99]:
```
'CompletedWithViolations: Job completed successfully with 1 violations.'
```

- Clean Up. Deleting the monitoring schedule and the endpoint.

Clean Up (Delete Monitoring Schedule and End Point)

[119]:
```
# Deleting the Monitoring Schedule

my_default_monitor.delete_monitoring_schedule()
time.sleep(10)
```
...

[ ]:
```
# Deleting the endpoint

sm.delete_endpoint(EndpointName = endpoint_name)
```

[ ]:

Result and findings

Some of the findings from the project include:

1. **Top 5 property use in NYC**: Multifamily Housing, Offices, Hotels, Non-refrigerated Warehouses, and K-12 Schools.

2. **Top 5 GHG emitters**: Energy and Power Stations, Laboratories, Data Centers, Stadiums, and Offices.

3. **Top 5 Use of New York's Oldest Buildings**: Bar/Night Clubs, Convention Centers, Vocational Schools, Convenience stores without gas station, and Performing Arts

4. **Top 5 Use of New York's Newest Building**: Courthouse, Senior care community, Lifestyle Center, Restaurants, and Offices

5. **Top factors that affect the GHG emissions of NYC buildings**: Diesel Use, Electricity use, Gross floor area, property use type etc.

6. The model performed well against the 2018 dataset (with baseline MAE = **1461.564**, while the model's MAE = **102.206**).

7. The model also performed well against the 2019 dataset (with baseline MAE = **586.107**, while the model's MAE = **182.055**). The 2019 and the 2018 datasets had the same features fed into the Linear learner model.

8. The model performed very poorly against the 2017 dataset (Baseline Mean Absolute Error – MAE was **146.415**, while the model's MAE was **23971018.269)**, this could be due to the differences in the features selected, some features in the 2017 dataset were not present in 2019 dataset and viceversa.

Implementation issues

Some of the top features selected based on the 2018 dataset to train the model, had up to 90% missing values in the 2017 dataset, thus they were removed and new feature selection process (XGBRegressor) was applied to the 2017 dataset to select the top features, leading to different features between the 2018 and 2017 datasets. For the 2019 dataset, it shared the same top features with the 2018 data.

Summary and Conclusion

The NYC Water and Energy data contains metrics on water and energy consumption of NYC buildings over 25,000 ft2. The data was cleaned to address duplicates, null values, etc. Tableau was used to visualize and to understand relationships between variables in the dataset. Three (3) datasets were used 2017, 2018 and 2019. The 2018 dataset was used to train the model and the trained model was tested on the 2017 and 2019 datasets. The 2018 data set was split into test, train, and validation sets. The model performed well against the 2019 and 2018 dataset and performed poorly on the 2017 dataset.

Recommendation

- I would recommend retaining the linear model with some of the 2017 dataset.
- Alternatively, I would recommend trying to use other methods to handle the 2017 missing values. Rather than removing the columns with excess missing values, the columns could be kept or filled up with the average column value. Keeping the column would significantly reduce

the available 2017 data to test the model, but this way all three datasets (2017, 2018 and 2019) will have similar features to test the model with and the models true performance against the 2017 dataset could be measured.