## Work Breakdown

**Vasi Reiva:** Worked on incorporating and analyzing new data (building size vs. building consumption), redeveloping solar production vs. building consumption based on more contemporary data (building location and weather), and conducting cost optimization.

**Carter Baller:** Worked on data visualization, as well as the SARIMA modelling process, including many trial and error methods, and final convincing results.

## Abstract

For fifteen buildings in Colorado, the Colorado Energy Office (CEO) provided energy consumption, building size, and solar irradiance data, taken every fifteen minutes for around three years. Though the data provided is similar in quantity and quality, variation among buildings includes missing data, slightly different time frames, and the different provenances of weather data. This project analyzes the patterns of energy consumption – comparing consumption to time, building size, and approximate solar panel production, providing several different visualizations to identify such patterns and seasonalities. The project also uses the collected data to create ARIMA models, accounting for three relevant seasonalities: daily, weekly, and annually.

## Introduction

The Colorado Energy Office (CEO) has a long-standing commitment to reducing greenhouse gas emissions and consumer energy costs by advancing clean energy, energy efficiency, and zero-emission vehicles to benefit all Coloradans. According to the CEO's recent statistical survey, buildings account for about 40% of energy consumed in the United States. While traditional energy production provides economic benefits to Colorado, greenhouse gas emissions from fossil fuel combustion have a negative impact on the overall environment. Thus, institutional electricity consumption habits and the encouragement of clean energy use are important to consider when attempting to reduce the negative environmental impacts of energy consumption by government agencies.

In the spring 2021 semester, Math Clinic students Xiang Li, Yushan Xu, Yuxuan Han, and Zixu Wang constructed a project that focused mainly on forecasted electricity consumption of government buildings in order to better understand their usage patterns. When concluding their project, they acknowledged the fact that there are limitations and possible inaccuracies in their study, mostly stemming from inadequate information. They found that the original data, with building energy

consumption data spanning over a period of about three years, were insufficient when building their seasonal ARIMA model.

This project will further analyze the building energy consumption of the same government buildings in order to provide our sponsor and/or other stakeholders with updated recommendations for optimization of building energy usage. On top of the original data used by the previous group, we will use building size, location and weather data to provide further evaluation and explanation of each building's energy consumption. Using the new data, this project will provide suggestions for solar energy investment and utilization through more recent data and more accurate calculations of solar production. While using building energy consumption data spanning over a period of only three years remains insufficient for producing accurate forecasting, this project develops a seasonal ARIMA model with all seasons accounted for and parameters that have shown to be chosen with accuracy. We hope that our sponsor and/or other stakeholders will be able to use our improvements to better understand the electricity consumption characteristics of each agency.

# Methods

**Reading Data Sets and Data Cleaning**

- Loads excel files:
    - Master Interval Data_public.xlsb (Buildings 1-9)
    - Master Interval Data_public2.xlsb (Buildings 10-15)
        - Original Building Data
            - DATE: numerical variable representing the date (mmddyy)
            - HOUR: numerical variable representing the minute of day
            - kW: numerical variable representing the rate of consumption of electricity (kW)
            - BUILDING: categorical variable representing each building
    - Facility Size_Location_temp.xlsx
        - Original Building Size Data
            - Floor Area: numerical variable representing the square footage of the building ($ft^2$)
            - Building ID: numerical variable representing the building number
        - Original Building Location Data
            - Name/Location: categorical variable representing the city in which the building resides
        - Original NREL Data
            - Year: numerical variable representing the year (yyyy)

        - Month: numerical variable representing the month (mm)
        - Day: numerical variable representing the day (dd)
        - Hour: numerical variable representing the hour of day
        - Minute: numerical variable representing the minute of hour
        - DHI: numerical variable representing solar radiation that does not arrive on a direct path from the sun, but has been scattered by clouds and particles in the atmosphere and comes equally from all directions ($W/m^2$)
        - DNI: numerical variable representing the amount of light that is coming perpendicular to the surface ($W/m^2$)
        - Temperature: numerical variable representing the temperature in celsius
        - City: categorical variable representing the city for which the NREL data represents
    - Original Electricity Rate Data
        - Rate: categorical variable representing the electricity service provider
        - Summer Season kW Demand: numerical variable representing the additional cost of summer energy, based on the demand of summer energy
        - Winter Season kW Demand: numerical variable representing the additional cost of winter energy, based on the demand of winter energy
        - Summer Season kWh: numerical variable representing the base cost of summer energy
        - Winter Season kWh: numerical variable representing the base cost of winter energy
- Reads in data ensuring that the original file remains unchanged.
- Cleans and defines new variables in preparation for the subsequent modelling and analysis work on data.

**Data Visualization**

- Analyzes data through time series, density plots, box plots, and scatter matrix plots.
- Explores different seasonalities in order to gain more knowledge on the behavior of data.

**Building Size vs. Building Consumption**

- Compares building size to respective building consumption for buildings in which building size data are available.

**Solar Production vs. Building Consumption**

- Calculates Solar PV (photovoltaic).
- Uses Solar PV and three variations of solar panel amounts in order to provide reference for solar panel installation decisions.
- **Solar PV Calculation Explanation:** Begin by calculating the solar zenith angle: a function of time, day, and latitude. The Solar Zenith Angle ($\theta$) can be calculated using the equation $\cos(\theta) = \sin(\delta)\sin(\ell) + \cos(\delta)\cos(\ell)\cos(\omega)$, where $\delta$ is the declination of the sun, $\ell$ is

latitude (defined as a positive constant in the northern hemisphere), and $\omega$ is the hour angle. The Declination of the Sun ($\delta$) can be calculated using the equation $\delta = \Phi \cos(C(d - dr)/dy)$, where $\Phi$ is the tilt angle of 23.5 degrees, $C$ is a full 360 degrees, $d$ is the Julian day, $dr$ is the Julian day for summer solstice on June 21 (172), and $dy$ is the number of days per year (365). The Hour Angle ($\omega$) can be calculated using the equation $\omega = 15(h - 12)$, where $h$ is the military hour and $\omega$ increases by 15 degrees for every hour before or after noon.

Once the solar zenith angle has been calculated, it is used to calculate Global Horizontal Irradiance (GHI). GHI can be calculated using the equation $GHI = (DNI \times \cos(\theta)) + DHI$, where DNI is Direct Normal Irradiance and DHI is Diffused Horizontal Irradiance.

Once GHI has been calculated, it is used to calculate Solar PV. The Solar PV ($E$) can be calculated using the equation $E = A \times r \times GHI \times PR$, where $A$ is the total area of the panel ($m^2$), $r$ is the solar panel yield ($\%$), and PR is the performance ratio with a default value of 0.75. Without knowledge of the PV systems that would be used to provide energy to each of the fifteen buildings, a few assumptions must be made. In the calculations, total area is estimated using the dimensions of a 78in $\times$ 39in commercial solar panel, solar panel yield is estimated using a value of 15% (with expectations for improvement), and the performance ratio is estimated using the default value of 0.75.

- **Visualization Explanation:** Using the buildings for which location – and thus DHI and DNI data – has been provided, a comparison among Solar PV, building consumption, and residual consumption can be made. The first column in the Solar Production vs. Consumption visualization uses one solar panel to calculate solar production. The last column in or Solar Production vs. Consumption visualization uses $n$ solar panels to calculate solar production, where $n$ is the optimized number of solar panels where the mean value of the residual curve is just below the minimum value of the building consumption curve. The middle column in the Solar Production vs. Consumption visualization uses $\sqrt{n}$, rounded down to the nearest integer, to represent a geometric medium that favors the smaller value in order to keep the amount of solar panels to a minimum.

**Optimize Spending on Existing Data Consumption**

- Calculates and compares the average cost of electricity usage for each building based on three different service providers.

**SARIMA Modelling**

- Creates ARIMA models for energy consumption for each of the buildings, taking into account daily, weekly, and yearly seasonalities.
- **Process of Creating a SARIMA Model:** Given the series of all energy consumption data from one building, the Fast Fourier Transform is calculated and plotted to determine relevant seasonality frequencies. It is important to discover what seasonalities the data exhibit so that those seasonalities can be removed to create an accurate model, as Autoregressive Integrated Moving Average (ARIMA) models cannot support seasonality. The reason ARIMA models are being used

instead of SARIMA (Seasonal ARIMA) is because a SARIMA model can only handle one type of seasonality, where most of the buildings exhibit multiple relevant seasonalities. As such, removing the relevant seasonalities and creating an ARIMA model off that can provide more accuracy when accounting for the multiple seasonalities.

Once the seasonalities are discovered, the function `seasonal_decompose`, which has a seasonal input, is used to break the series into three different parts: trend, residuals, and seasonality. The series is broken down such that if the three parts are added together, the series reverts back to normal. The reason the series is broken down is to remove the seasonality, in other words, add only the trend and residual components together, excluding the seasonality. This is repeated for each of the relevant seasonalities discovered.

After the series has been seasonally differenced, the autocorrelation function (ACF) and partial autocorrelation function (PACF) of the series is plotted. The ACF and PACF are essential to determining what parameters to plug into the ARIMA model. Once the ACF and PACF are plotted, for these data, the series is not initially stationary, which the series needs to be for accurate ACF and PACF analysis. To become stationary, the series is differenced, or each point is subtracted by the next point in the series. However many times the series is differenced to become stationary is the $d$ parameter in the ARIMA $(p, d, q)$ model.

Once the series is stationary, both the ACF and PACF are analyzed to determine at which lag ($p$ or $q$ respectively) do all the autocorrelation or partial autocorrelation values after $p$ or $q$ resemble white noise, or, reside in the confidence interval.

Once all the parameters are determined, the ARIMA model is calculated along with test statistics showing reliability of the model.

## Dependencies

## Installs

```
# Install library to support .xlsb files
!pip install pyxlsb
# Install library to support statsmodels and ARIMA
!pip install pmdarima
```

```
Collecting pyxlsb
  Downloading pyxlsb-1.0.9-py2.py3-none-any.whl (23 kB)
Installing collected packages: pyxlsb
Successfully installed pyxlsb-1.0.9
Collecting pmdarima
  Downloading pmdarima-1.8.4-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.ma
     |████████████████████████████████| 1.4 MB 4.8 MB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: Cython!=0.29.18,>=0.29 in /usr/local/lib/python3.7/dis
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.
```

```
Collecting statsmodels!=0.12.0,>=0.11
  Downloading statsmodels-0.13.1-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_6
     |████████████████████████████████| 9.8 MB 16.8 MB/s
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages
Installing collected packages: statsmodels, pmdarima
  Attempting uninstall: statsmodels
    Found existing installation: statsmodels 0.10.2
    Uninstalling statsmodels-0.10.2:
      Successfully uninstalled statsmodels-0.10.2
Successfully installed pmdarima-1.8.4 statsmodels-0.13.1
```

## Imports

```
from google.colab import drive
import warnings
import os                                                    # Loading Excel Files - I
import pandas as pd                                          # Loading Excel Files - M
from tqdm import tqdm                                        # Building Data - Monthy
from datetime import datetime                                # Building Data - Solar I
import matplotlib.pyplot as plt                              # Time Series, ACF, & PAC
from statsmodels.graphics.tsaplots import plot_acf           # Time Series, ACF, & PAC
from statsmodels.graphics.tsaplots import plot_pacf          # Time Series, ACF, & PAC
import seaborn as sns                                        # Data Visualization - Sc
import statistics                                            # Solar Production vs. Co
import numpy as np                                           # Building Consumption vs
from scipy.fft import fft                                    # SARIMA Modelling - SARI
from scipy import signal                                     # SARIMA Modelling - SARI
from statsmodels.tsa.seasonal import seasonal_decompose      # SARIMA Modelling - SARI
from statsmodels.tsa.arima.model import ARIMA                # SARIMA Modelling - SARI
from statsmodels.tsa.statespace.sarimax import SARIMAXResults # SARIMA Modelling - SARI
```

```
# Allows access to Google Drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
# Many of the SARIMA modelling processes produce pages upon pages of warnings, this com
warnings.filterwarnings('ignore')
```

```
# Global variables
Δ = 0.25      # Sampling interval in hours
```

```
hrpd = 24      # Hours per day
dpwk = 7       # Days per week
```

## Functions

## Data Visualization Functions

```
def density_plot(plot_type,axs,n,build_name):
  if plot_type == 'Annual':
    sns.histplot(df[(df["BUILDING"]==build_name)]['kW'],
                 label='Annual',bins=100,color='b',alpha=0.3,stat='density',
                 ax=axs[n,0],kde=True)
    axs[n,0].set_title(build_name+' Annual')
    axs[n,0].legend(loc='upper left')
    axs[n,0].set_xlabel('Electricity Consumption (kWh)')
  elif plot_type == 'Seasonal':
    sns.histplot(df[(df["BUILDING"]==build_name)&(df["SEASON"]=="Summer")]["kW"],
                 label='Summer',bins=100,color='r',alpha=0.3,stat='density',
                 ax=axs[n,1],kde=True)
    sns.histplot(df[(df["BUILDING"]==build_name)&(df["SEASON"]=="Winter")]["kW"],
                 label='Winter',bins=100,color='b',alpha=0.3,stat='density',
                 ax=axs[n,1],kde=True)
    axs[n,1].set_title(build_name+' Summer & Winter')
    axs[n,1].legend(loc='upper left')
    axs[n,1].set_xlabel('Electricity Consumption (kWh)')
    axs[n,1].set_ylabel(' ')
  elif plot_type == 'Days':
    sns.histplot(df[(df["BUILDING"]==build_name)&(df["DAY_OF_WEEK"].isin([0,1,2,3,4]))]
                 label='Weekday',bins=100,color='r',alpha=0.3,stat='density',
                 ax=axs[n,2],kde=True)
    sns.histplot(df[(df["BUILDING"]==build_name)&(df["DAY_OF_WEEK"].isin([5,6]))]["kW"]
                 label='Weekend',bins=100,color='b',alpha=0.3,stat='density',
                 ax=axs[n,2],kde=True)
    axs[n,2].set_title(build_name+' Weekday & Weekend')
    axs[n,2].legend(loc='upper left')
    axs[n,2].set_xlabel('Electricity Consumption (kWh)')
    axs[n,2].set_ylabel(' ')
```

```
def semilogydensity_plot(plot_type,axs,m,build_name):
  if plot_type == 'Annual':
    sns.histplot(df[(df["BUILDING"]==build_name)]['kW'],
                 label='Annual',bins=100,color='b',alpha=0.3,stat='density',
                 ax=axs[m,0],kde=True, log_scale=(False, True))
    axs[m,0].set_title(build_name+' Annual')
    axs[m,0].legend(loc='upper left')
    axs[m,0].set_xlabel('Electricity Consumption (kWh)')
  elif plot_type == 'Seasonal':
    sns.histplot(df[(df["BUILDING"]==build_name)&(df["SEASON"]=="Summer")]["kW"],
                 label='Summer',bins=100,color='r',alpha=0.3,stat='density',
                 ax=axs[m,1],kde=True, log_scale=(False, True))
```

```
    sns.histplot(df[(df["BUILDING"]==build_name)&(df["SEASON"]=="Winter")]["kW"],
                 label='Winter',bins=100,color='b',alpha=0.3,stat='density',
                 ax=axs[m,1],kde=True, log_scale=(False, True))
    axs[m,1].set_title(build_name+' Summer & Winter')
    axs[m,1].legend(loc='upper left')
    axs[m,1].set_xlabel('Electricity Consumption (kWh)')
    axs[m,1].set_ylabel(' ')
  elif plot_type == 'Days':
    sns.histplot(df[(df["BUILDING"]==build_name)&(df["DAY_OF_WEEK"].isin([0,1,2,3,4]))]
                 label='Weekday',bins=100,color='r',alpha=0.3,stat='density',
                 ax=axs[m,2],kde=True, log_scale=(False, True))
    sns.histplot(df[(df["BUILDING"]==build_name)&(df["DAY_OF_WEEK"].isin([5,6]))]["kW"]
                 label='Weekend',bins=100,color='b',alpha=0.3,stat='density',
                 ax=axs[m,2],kde=True, log_scale=(False, True))
    axs[m,2].set_title(build_name+' Weekday & Weekend')
    axs[m,2].legend(loc='upper left')
    axs[m,2].set_xlabel('Electricity Consumption (kWh)')
    axs[m,1].set_ylabel(' ')
```

```
def box_plot(plot_type,axs,index,build_name):
  labels={'Night':'royalblue','Day':'darkorange'}
  sns.boxplot(x="HOUR",y="kW",data=df[(df["BUILDING"]==build_name)],
              ax=axs[index,0],flierprops=dict(markersize=2),hue="DAYTIME",palette=label
  axs[index,0].set_title(build_name+' Consumption')
  axs[index,0].legend(loc="upper left")
  axs[index,0].set_xlabel('Hours of the Day')
  axs[index,0].set_ylabel('Electricity Consumption (kWh)')
  if plot_type == 'Seasonal':
    sns.boxplot(x="HOUR",y="kW",data=df[(df["BUILDING"]==build_name)&(df["SEASON"]=="Su
                ax=axs[index,1],flierprops=dict(markersize=2),hue="DAYTIME",palette=lab
    axs[index,1].set_title(build_name+' Summer Consumption')
    axs[index,1].legend(loc="upper left")
    axs[index,1].set_xlabel('Hours of the Day')
    axs[index,1].set_ylabel(' ')
    sns.boxplot(x="HOUR",y="kW",data=df[(df["BUILDING"]==build_name)&(df["SEASON"]=="Wi
                ax=axs[index,2],flierprops=dict(markersize=2),hue="DAYTIME",palette=lab
    axs[index,2].set_title(build_name+' Winter Consumption')
    axs[index,2].legend(loc="upper left")
    axs[index,2].set_xlabel('Hours of the Day')
    axs[index,2].set_ylabel(' ')
  elif plot_type == 'Days':
    sns.boxplot(x="HOUR",y="kW",data=df[(df["BUILDING"]==build_name)&(df["DAY_OF_WEEK"]
                ax=axs[index,1],flierprops=dict(markersize=2),hue="DAYTIME",palette=lab
    axs[index,1].set_title(build_name+' Weekday Consumption')
    axs[index,1].legend(loc="upper left")
    axs[index,1].set_xlabel('Hours of the Day')
    axs[index,1].set_ylabel(' ')
    sns.boxplot(x="HOUR",y="kW",data=df[(df["BUILDING"]==build_name)&(df["DAY_OF_WEEK"]
                ax=axs[index,2],flierprops=dict(markersize=2),hue="DAYTIME",palette=lab
    axs[index,2].set_title(build_name+' Weekend Consumption')
    axs[index,2].legend(loc="upper left")
    axs[index,2].set_xlabel('Hours of the Day')
    axs[index,2].set_ylabel(' ')
```

## Solar Production vs. Consumption Functions

```
def solar_plot(plot_type,axs,index):
  if plot_type == 1:
    sns.lineplot(data=sub, x="HOUR", y="kW", label="Building kW", ax=axs[index,0])
    sns.lineplot(data=sub, x="HOUR", y="Solar PV ({})".format(plot_type), label="Solar
    sns.lineplot(data=sub, x="HOUR", y="Residual ({})".format(plot_type), label="Residu
    axs[index,0].axvline(x=0, c="Black")
    axs[index,0].axhline(y=0, c="Black")
    axs[index,0].set_title(f"Building {x} Solar Production vs. Consumption\n for 1 Sola
    axs[index,0].legend(loc='lower right')

  elif plot_type == 2:
    sns.lineplot(data=sub, x="HOUR", y="kW", label="Building kW", ax=axs[index,1])
    sns.lineplot(data=sub, x="HOUR", y="Solar PV ({})".format(plot_type), label="Solar
    sns.lineplot(data=sub, x="HOUR", y="Residual ({})".format(plot_type), label="Residu
    axs[index,1].axvline(x=0, c="Black")
    axs[index,1].axhline(y=0, c="Black")
    axs[index,1].set_title(f"Building {x} Solar Production vs. Consumption\n for {int(r
    axs[index,1].legend(loc='lower right')
    axs[index,1].set_ylabel(' ')

  elif plot_type == 3:
    sns.lineplot(data=sub, x="HOUR", y="kW", label="Building kW", ax=axs[index,2])
    sns.lineplot(data=sub, x="HOUR", y="Solar PV ({})".format(plot_type), label="Solar
    sns.lineplot(data=sub, x="HOUR", y="Residual ({})".format(plot_type), label="Residu
    axs[index,2].axvline(x=0, c="Black")
    axs[index,2].axhline(y=0, c="Black")
    axs[index,2].set_title(f"Building {x} Solar Production vs. Consumption\n for {n} So
    axs[index,2].legend(loc='lower right')
    axs[index,2].set_ylabel(' ')
```

```
def calculation_demand(sub):
  n = 1
  minkW = min(sub["kW"])
  res = statistics.mean(sub["kW"]) - statistics.mean(sub["Solar PV (1)"])
  while res > minkW:
    n = n + 1
    res = statistics.mean(sub["kW"]) - (statistics.mean(sub["Solar PV (1)"]) * n)
  return n
```

## SARIMA Modelling Functions

```
# Coded primarily by Professor Fournier
def fft_AF(building,dfkw):
  n = len(dfkw)
  r = np.arange(n)
  x = np.abs(fft(dfkw))[:n//2]
```

```
  f, axs = plt.subplots(1,1,figsize=(30,10))
  axs.loglog(r[1:n//2]/(n*Δ*(1/hrpd)*(1/dpwk))    # Frequency in 1/week
                 ,x[1:],'.-')
  labels={1/52.:'1/year',1/26.:'2/year',3/52.:'3/year',1/13.:'4/year',
          .25:'1/month',.5:'2/month',
          1:'1/week',2:'2/week',5:'5/week',
          7:'1/day',14:'2/day',21:'3/day',28:'4/day',35:'5/day',42:'6/day',
          49:'7/day',168:'1/hour',336:'2/hour'}  # Mark seasonalities for easy reading
  for f in [1/52.,1/26.,3/52.,1/13.,.25,.5,1,2,5,7,14,21,28,35,42,49,24*7,48*7]:
    axs.text(f, x.min(), labels[f], rotation='vertical', ha='center')
  plt.title('FFT Series of Building '+str(building)+' with Logarithmic y-scale',fontsiz
  plt.xlabel('Frequency (1/week)')
  plt.ylabel('|FFT Value|')
  plt.show()
  # Plots FFT with linear y-scale
  f, axs = plt.subplots(1,1,figsize=(30,10))
  axs.semilogx(r[1:n//2]/(n*Δ*(1/hrpd)*(1/dpwk))
                 ,x[1:]**2,'.-')
  for f in [1/52.,1/26.,3/52.,1/13.,.25,.5,1,2,5,7,14,21,28,35,42,49,24*7,48*7]:
    axs.text(f, x.min(), labels[f], rotation='vertical', ha='center', position=(f,max(x
  plt.title('FFT Series of Building '+str(building)+' with Linear y-scale',fontsize=30)
  plt.xlabel('Frequency (1/week)')
  plt.ylabel('|FFT Value|^2')
  plt.show()
  return x
```

```
def rseas(x,dfkw):
  prominence = max(x[1:]**2)/100    # After trial and error, this prominence detects ev
                                    # while leaving out non-relevant seasonalities.
  fftpeaks = signal.find_peaks(x**2,height=prominence)
  n = len(dfkw)
  fftmul = []
  for i in np.arange(len(fftpeaks[0])):
    fftmul.append(1/(fftpeaks[0][i]/(n*Δ*(1/hrpd))))     # Convert spike indices to un
  fftseas = []
  for i in fftmul:    # Evaluates every FFT spike
    bo = (i >= 0.5 and i <= 1.5) or (i >= 6 and i <= 8) or (i >= 350 and i <= 380)
    if bo == True:
      fftseas.append(i)     # Keeps only daily, weekly, or yearly
  rseas = ['n','n','n']     # Creates a list of seasonalities, used for each building
                            # rseas[0] is daily, rseas[1] is weekly, rseas[2] is yearly
  for i in fftseas:
    if i >= 0.5 and i <= 1.5:
      rseas[0] = 'y'
    if i >= 6 and i <= 8:
      rseas[1] = 'y'
    if i >= 350 and i <= 380:
      rseas[2] = 'y'
  return(rseas)
```

```
# Print statement of seasonalities
def rstate(rseas):   # rseas is the set of seasonalities, y/n for each
```

```
  if rseas[0] == 'y':
    if rseas[1] == 'y':
      if rseas[2] == 'y':
        print('\nThere is evidence of a daily, weekly, and yearly seasonality.\n')
      elif rseas[2] == 'n':
        print('\nThere is evidence of a daily and weekly seasonality, but not much evic
    elif rseas[1] == 'n':
      if rseas[2] == 'y':
        print('\nThere is evidence of a daily and yearly seasonality, but not much evic
      elif rseas[2] == 'n':
        print('\nThere is evidence of a daily seasonality, but not much evidence of a v
  elif rseas[0] == 'n':
    if rseas[1] == 'y':
      if rseas[2] == 'y':
        print('\nThere is evidence of a weekly and yearly seasonality, but not much evi
      elif rseas[2] == 'n':
        print('\nThere is evidence of a weekly seasonality, but not much evidence of a
    elif rseas[1] == 'n':
      if rseas[2] == 'y':
        print('\nThere is evidence of a yearly seasonality, but not much evidence of a
      elif rseas[2] == 'n':
        print('\nThere is not much evidence for seasonality.\n')
```

```
def daily(building):
  # Creates a set of all timestamps and corresponding kWh values for one building
  df_build = df[df["BUILDING"]=="Building "+str(building)].sort_values("DATETIME")[["ye
  df_build.index = df_build['year_month_day']
  df_build = df_build.drop(columns='year_month_day')
  print("Consumption Data for all measurements for Building "+str(building)+':\n',df_bu
  # Removes daily seasonality
  df_sd  = seasonal_decompose(df_build['kWh'],period=96,extrapolate_trend='freq')
  df_sd = df_sd.trend + df_sd.resid
  df_sd = df_sd.groupby('year_month_day').sum().reset_index()
  df_sd.index = df_sd['year_month_day']
  df_sd = df_sd.drop(columns='year_month_day')
  df_sd['kWh'] = df_sd[0]
  df_sd = df_sd.drop(columns=0)
  return df_sd
```

```
# Removes seasonalities
def removal(rseasin,df_build):    # rseasin is the set of seasonalities, y/n for each
                                  # rseasin[1] is weekly, rseasin[2] is yearly
                                  # df_build is the set of dates and corresponding kWh
  if rseasin[0] == 'y' and rseasin[1] == 'n' and rseasin[2] == 'n':
    sd = df_build
  elif rseasin[1] == 'y':
    # Breaks the series into seasonal, trend, and residual components
    # Adding the trend and residual components removes the relevant seasonality
    sd = seasonal_decompose(df_build['kWh'],period=7,extrapolate_trend='freq')
    sd = sd.trend + sd.resid
    if rseasin[2] == 'y':
      sd = seasonal_decompose(sd,period=365,extrapolate_trend='freq')
```

```
      sd = sd.trend + sd.resid
  elif rseasin[1] == 'n':
    if rseasin[2] == 'y':
      sd = seasonal_decompose(df_build['kWh'],period=365,extrapolate_trend='freq')
      sd = sd.trend + sd.resid
  return sd   # sd is the seasonally-differenced series
```

```
# Plots time series, ACF, and PACF
def rplots(sd,building,alpha):    # sd is the seasonally-differenced series
                                  # building is the building number
                                  # alpha is the confidence interval (typically 0.05)
  # Plots time series
  plt.figure(figsize=(30, 10))
  plt.plot(sd,'.',color='k')
  plt.title('Plot of One-Day Building '+str(building)+' Data After Removing Seasonality
  plt.xlabel('Date (YYYY-MM)')
  plt.ylabel('Electricity Consumption (kWh)')
  # Plots ACF
  fig, ax = plt.subplots(figsize=(30,10))
  plot_acf(sd,lags=30,alpha=alpha,ax=ax)     # 30 lags shows enough to determine model
  plt.title('ACF of One-Day Building '+str(building)+' Data After Removing Seasonality'
  plt.xlabel('Lags (1-Day Intervals)')
  plt.ylabel('Autocorrelation')
  # Plots PACF
  fig, ax = plt.subplots(figsize=(30,10))
  plot_pacf(sd,lags=30,alpha=alpha,ax=ax)    # 30 lags shows enough to determine model
  plt.title('PACF of One-Day Building '+str(building)+' Data After Removing Seasonality
  plt.xlabel('Lags (1-Day Intervals)')
  plt.ylabel('Partial Autocorrelation')
  plt.show()
```

```
# Takes the first-order difference of the series
def diff(sd,building,d):    # sd is the seasonally-differenced series
                            # building is the building number
                            # d is how many times to difference the series
  if d == 0:
    print('The series is stationary without differencing.')
  else:
    nd = sd.diff().dropna()   # Takes the first difference
    d = d - 1
    while d > 0:              # Takes any additional differences
      nd = nd.diff().dropna()
      d = d - 1
      if d == 0:
        break
  rplots(nd,building,0.00005)   # Use a lower alpha value to better see the parameters
```

```
# Plots the created model against original data
def tracing(best_model,sd,df_build):    # best_model is the ARIMA model built
                                        # sd is the seasonally-differenced series
                                        # df_build is the set of dates and correspondir
```

```
                                        #  electrical energy consumption in kWh for one
  # Creates a data set based off the model for all the dates
  modelling = best_model.predict(start=str(sd.index[0]),end=str(sd.index[len(sd)-1]))
  plt.figure(figsize=(30,10))
  plt.plot(df_build.index,df_build['kWh'],'.',color='b',label='Original Data')
  plt.plot(df_build.index[1:len(modelling)],modelling[1:len(modelling)],'.',color='r',l
  plt.title('Plot of Created Model Compared to Original Data',fontsize=30)
  plt.xlabel('Date (YYYY-MM)')
  plt.ylabel('Electrical Energy Consumption (kWh)')
  plt.legend()
  cor = df_build['kWh'].corr(modelling[1:len(modelling)])
  print('\nCorrelation:\n',cor)
  return cor
```

```
def stattest(best_model,cor):
  stattest = []
  lj = best_model.test_serial_correlation(method='ljungbox')[0][1][0] # Ljung-Box test
  het = best_model.test_heteroskedasticity(method='breakvar')[0][1]   # Heteroscedastic
  stattest.append(lj)
  stattest.append(het)
  if lj > 0.05 and het > 0.05:    # If neither tests are statistically significant
    stattest.append('n')          # Evidence to support the model, not necessary to ree
  else:
    stattest.append('y')          # Evidence to refute the model, should be reevaluatec
  stattest.append(cor)
  return stattest
```

```
def sarima_groundwork(building):    # building is the building number
  # Creates a set of all timestamps and corresponding kWh values for one building
  df_build = df[df["BUILDING"]=="Building "+str(building)].sort_values("DATETIME")[["ye
  df_build.index = df_build['year_month_day']
  df_build = df_build.drop(columns='year_month_day')
  print("Consumption Data for all measurements for Building "+str(building)+':\n',df_bu
  # Calculates and plots the FFT of a data set
  dfkw = df_build['kWh'].values
  print('\nPlotting FFT of series:\n')
  x = fft_AF(building,dfkw)
  # Determines relevant seasonalities
  rseasin = rseas(x,dfkw)
  return(rseasin)   # rseasin is a set of seasonalities, y/n for each
```

```
def sarima_analysis(building,rseasin,sparam):   # building is building number
                                                # rseasin is the set of seasonalities,
                                                # sparam is a manually determined set o
                                                # each building, based off the differer
  rstate(rseasin)
  if rseasin[0] == 'y':
    df_sd = daily(building)
    # Removes other seasonalities
    sd = removal(rseasin,df_sd)
  else:
  # Creates a set of daily timestamps and corresponding kWh-sum values for one buildinc
```

```
    df_build = df[df["BUILDING"]=="Building "+str(building)].sort_values("DATETIME")[['
    df_build.index = df_build['year_month_day']
    df_build = df_build.drop(columns='year_month_day')
    print("Consumption Data for daily measurements for Building "+str(building)+':\n',d
    # Removes seasonalities
    sd = removal(rseasin,df_build)
  print('\nPlotting series after removing seasonalities:\n')
  # Plots time series, ACF, and PACF
  rplots(sd,building,0.05)
  print('\nPlotting series after removing seasonalities and differencing:\n')
  # Takes the first-order difference of the series
  diff(sd,building,int(sparam[1]))    # sparam[1] is the d parameter of the ARIMA model
                                      # analyzing the differenced ACF and PACF
  # Creates an ARIMA model based off the manually determined parameters
  best_model = ARIMA(sd, order=(int(sparam[0]),int(sparam[1]),int(sparam[2]))).fit()
  print('\nCreating ARIMA model for the series:\n',best_model.summary())
  # Plots the created model against original data
  if rseasin[0] == 'y':
    cor = tracing(best_model,sd,df_sd)
  else:
    cor = tracing(best_model,sd,df_build)
  stats = stattest(best_model,cor)
  return stats
```

## Reading Data Sets and Data Cleaning

### Loading Excel Files

```
# Creates path to data
cwd = os.getcwd()
pathCEO = cwd + '/drive/My Drive/'
# Path for Professor Fournier
pathProfessor = 'Colab Notebooks/Math Clinic/2021fa/CEO/'
if os.path.exists(pathCEO + pathProfessor):
   pathCEO += pathProfessor
pathCEO += 'CEO_data/'
os.listdir(pathCEO)
```

```
['Master Interval Data_public.xlsb',
 'Facility Size_Location_temp.xlsx',
 'DEN Solar Data Clean.csv',
 'Master Interval Data_public2.xlsb']
```

```
# Loading data files into variables
  # Import data for Buildings 1-9
data_file = []
efile = pd.ExcelFile(pathCEO + 'Master Interval Data_public.xlsb', engine='pyxlsb')
data_file.append(efile)
  # Import data for Buildings 10-15
efile = pd.ExcelFile(pathCEO + 'Master Interval Data_public2.xlsb', engine='pyxlsb')
```

```
data_file.append(efile)
  # Import size, location, and weather data
efile = pd.ExcelFile(pathCEO + 'Facility Size_Location_temp.xlsx')
data_file.append(efile)
```

```
# Delete unwanted variables and installs
del(efile)
```

## Reading Sheets from Excel

## Building Data

```
# Define a list to hold the DataFrame for each Building
dfs = []
```

```
# Run Time: Approximately 2min
for sheet_names, excel_file in [(data_file[0].sheet_names[1:-1],data_file[0]),
                                (data_file[1].sheet_names[1:],data_file[1])]:
  for sheet_name in tqdm(sheet_names):
    if sheet_name == "Building 1":
      df_build = pd.read_excel(excel_file,sheet_name=sheet_name,header=10,usecols="B:C,
      df_build["BUILDING"] = sheet_name
      df_build.columns = df_build.columns.str.strip()
      dfs.append(df_build)
    elif sheet_name == "Building 15":
      df_build = pd.read_excel(excel_file,sheet_name=sheet_name,header=9,usecols="B,F")
      df_build["BUILDING"] = sheet_name
      df_build["DATE"] = df_build[" DATE & Time"].map(lambda x: datetime.fromtimestamp(
      df_build["HOUR"] = df_build[" DATE & Time"].map(lambda x: datetime.fromtimestamp(
      df_build = df_build.drop([" DATE & Time"],axis=1)
      df_build.columns = df_build.columns.str.strip()
      dfs.append(df_build)
    else:
      df_build = pd.read_excel(excel_file,sheet_name=sheet_name,header=1,usecols="B:C,F
      df_build["BUILDING"] = sheet_name
      df_build.columns = df_build.columns.str.strip()
      dfs.append(df_build)
```

```
100%|██████████| 9/9 [01:00<00:00,  6.77s/it]
100%|██████████| 6/6 [00:43<00:00,  7.22s/it]
```

**`datetime` Function Explanation:** Building 15 is different from the other buildings in that the date is in the form of datetime (mm/dd/yy hh:mm); to correspond with the other buildings, we want date in the form of date (mmddyy) and hour (hhmm). Excel's date system starts from 01/01/1900. In order to convert from Excel's decimal-date-value to Python's, which starts 01/01/1970, we must subtract the number of days between those two dates (25569 days) and then multiply the days since 01/01/1970 by the number of

seconds in a day (86400 sec/day) to get seconds since 01/01/1970. Once converted to seconds, `strftime` converts the decimal value to date (mmddyy) and hour (hhmm).

```
# Merge all data from 15 buildings
df = pd.concat(dfs)
print(df)
```

```
             DATE   HOUR      kW     BUILDING
0         10118.0   15.0  15.360   Building 1
1         10118.0   30.0  17.920   Building 1
2         10118.0   45.0  17.280   Building 1
3         10118.0  100.0  16.000   Building 1
4         10118.0  115.0  18.560   Building 1
...           ...    ...     ...          ...
108924    21721.0  130.0  29.852  Building 15
108925    21721.0  145.0  30.180  Building 15
108926    21721.0  200.0  31.292  Building 15
108927    21721.0  215.0  28.620  Building 15
108928    21721.0  230.0  31.740  Building 15

[1587418 rows x 4 columns]
```

```
# Delete possible missing values
df = df.dropna()
df = df.reset_index(drop=True)
```

```
# Column modification
  # Change DATE and HOUR type to string
df["DATE"] = df["DATE"].astype(int).astype(str)
df["HOUR"] = df["HOUR"].astype(int).astype(str)
  # Convert DATE to Year-Month-Day
df["year_month_day"] = pd.to_datetime(df["DATE"].str.zfill(6), format="%m%d%y")
  # Modify data with HOUR equal to 2400
df.loc[df[df["HOUR"] == "2400"].index, "year_month_day"] = df.loc[df[df["HOUR"] == "240
df.loc[df[df["HOUR"] == "2400"].index, "HOUR"] = "0"
  # Add new datetime column
df["DATETIME"] = pd.to_datetime(df["DATE"].str.zfill(6)+df["HOUR"].str.zfill(4), format
#df = df.drop(["year_month_day"], axis=1)
  # Add year, month, day, hour columns
df["YEAR"] = df["DATETIME"].dt.year
df["MONTH"] = df["DATETIME"].dt.month
df["DAY"] = df["DATETIME"].dt.day
df["HOUR"] = df["DATETIME"].dt.hour
# Add daytime column (0900-1700 = Day; 1701-0859 = Night)
df["DAYTIME"] = df["HOUR"].astype(int).map(lambda x: "Day" if 9 <= x <= 17 else "Night"
  # Add day_of_week column (Monday = 0; Sunday = 6)
df["DAY_OF_WEEK"] = df["DATETIME"].dt.dayofweek
  # Add season column (June-September = Summer; October-May = Winter)
df["SEASON"] = df["MONTH"].map(lambda x: "Summer" if x in [6, 7, 8, 9] else "Winter")
  # Add kWh column
```

```
df["kWh"] = df["kW"] * Δ
print(df)
```

```
             DATE  HOUR      kW     BUILDING  ... DAYTIME DAY_OF_WEEK  SEASON    kWh
0        10118     0  15.360   Building 1  ...   Night           0  Winter  3.840
1        10118     0  17.920   Building 1  ...   Night           0  Winter  4.480
2        10118     0  17.280   Building 1  ...   Night           0  Winter  4.320
3        10118     1  16.000   Building 1  ...   Night           0  Winter  4.000
4        10118     1  18.560   Building 1  ...   Night           0  Winter  4.640
...        ...   ...     ...          ...  ...     ...         ...     ...    ...
1587411  21721     1  29.852  Building 15  ...   Night           2  Winter  7.463
1587412  21721     1  30.180  Building 15  ...   Night           2  Winter  7.545
1587413  21721     2  31.292  Building 15  ...   Night           2  Winter  7.823
1587414  21721     2  28.620  Building 15  ...   Night           2  Winter  7.155
1587415  21721     2  31.740  Building 15  ...   Night           2  Winter  7.935
```

```
del(dfs, sheet_names, excel_file, sheet_name, df_build)
```

## Building Size Data

```
sdf = pd.read_excel(data_file[2], sheet_name="Facility information", header=2, usecols=
sdf.columns = sdf.columns.str.strip()
print(sdf)
```

```
    Building ID  Floor Area
0             1     17888.0
1             2         NaN
2             3         NaN
3             4         NaN
4             5         NaN
5             6         NaN
6             7         NaN
7             8         NaN
8             9         NaN
9            10    338777.0
10           11         NaN
11           12    174908.0
12           13    131990.0
13           14    164553.0
14           15         NaN
```

## Building Location Data

```
ldf = pd.read_excel(data_file[2], sheet_name="Facility information", header=2, usecols=
ldf.columns = ldf.columns.str.strip()
ldf.dropna(subset = ["Name/ Location"], inplace=True)
ldf["Name/ Location"].replace({"Fort Lupton": "Ft Lupton"}, inplace=True)
ldf.reset_index(inplace=True)
print(ldf)
```

```
    index  Building ID  Name/ Location
0       1            2          Denver
```

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | Denver |
| 2 | 3 | 4 | Golden |
| 3 | 4 | 5 | Rifle |
| 4 | 5 | 6 | Sterling |
| 5 | 9 | 10 | Denver |
| 6 | 10 | 11 | Denver |
| 7 | 11 | 12 | Grand Junction |
| 8 | 12 | 13 | Aurora |
| 9 | 13 | 14 | Golden |
| 10 | 14 | 15 | Ft Lupton |

## NREL Data

```
# Define a list to hold the DataFrame for each Building
wdfs = []
```

```
for sheet_names, excel_file in [(data_file[2].sheet_names[1:],data_file[2])]:
  for sheet_name in tqdm(sheet_names):
    df_build = pd.read_excel(excel_file,sheet_name=sheet_name,header=2,usecols="A:G,K")
    df_head = pd.read_excel(excel_file,sheet_name=sheet_name,usecols="F")
    df_build["Latitude"] = df_head["Latitude"][0]
    df_build["City"] = sheet_name
    df_build.columns = df_build.columns.str.strip()
    wdfs.append(df_build)
```

```
100%|██████████| 7/7 [00:01<00:00,  3.78it/s]
```

```
# Merge all data from 15 buildings
wdf = pd.concat(wdfs)
print(wdf)
```

```
      Year  Month  Day  Hour  Minute  DHI  DNI  Temperature  Latitude      City
0     2017      1    1     0      30    0    0         -2.0     39.73    Aurora
1     2017      1    1     1      30    0    0         -2.0     39.73    Aurora
2     2017      1    1     2      30    0    0         -2.0     39.73    Aurora
3     2017      1    1     3      30    0    0         -3.0     39.73    Aurora
4     2017      1    1     4      30    0    0         -3.0     39.73    Aurora
...    ...    ...  ...   ...     ...  ...  ...          ...       ...       ...
8755  2015     12   31    19      30    0    0        -11.0     40.61  Sterling
8756  2015     12   31    20      30    0    0        -10.0     40.61  Sterling
8757  2015     12   31    21      30    0    0        -10.0     40.61  Sterling
8758  2015     12   31    22      30    0    0        -10.0     40.61  Sterling
8759  2015     12   31    23      30    0    0        -10.0     40.61  Sterling

[61320 rows x 10 columns]
```

```
del(wdfs, sheet_names, excel_file, sheet_name, df_build, df_head)
```

## Electricity Rate Data

```
df_info = pd.read_excel(data_file[0],sheet_name="Rates & Info",header=1,usecols="E, G:J
# Change "Rate" to index
df_info = df_info.set_index(["Rate"])
print(df_info)
```

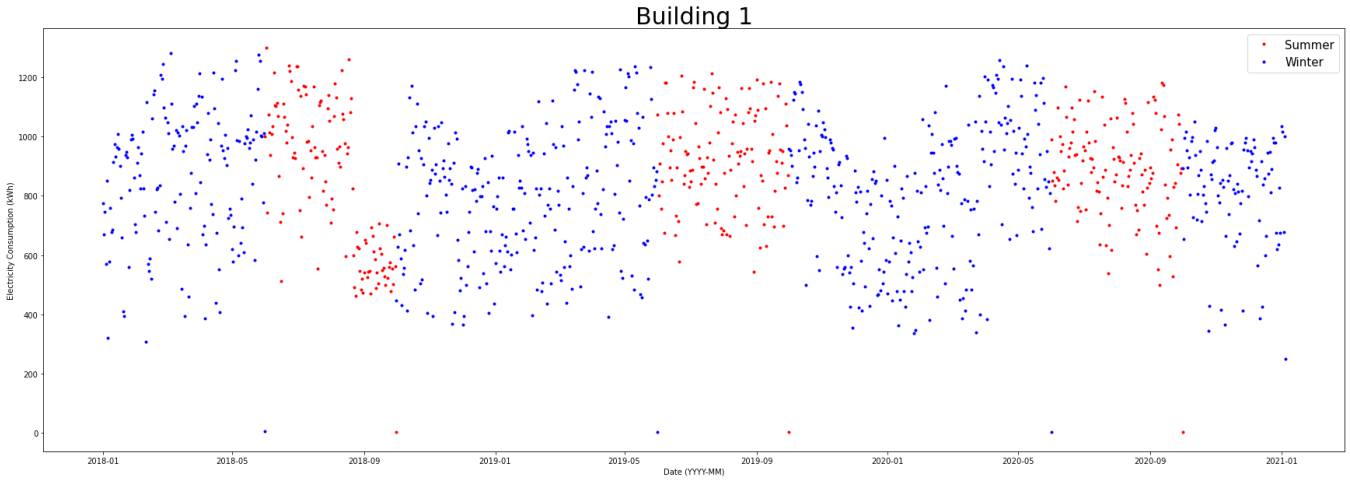
```
                                    Summer Season kW Demand  ...  Winter Season
Rate                                                         ...
Secondary General (SG)                                19.65  ...           0.00
Secondary General Low-Load Factor (SGL)                5.63  ...           0.12
Primary General (PG)                                  18.12  ...           0.00

[3 rows x 4 columns]
```

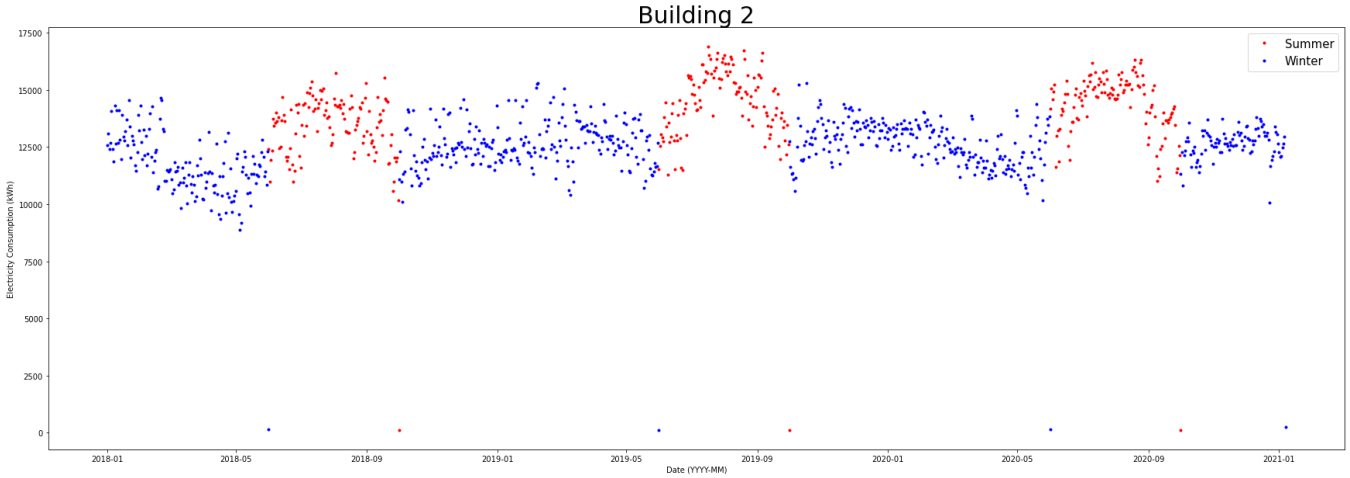
# Results/Discussion

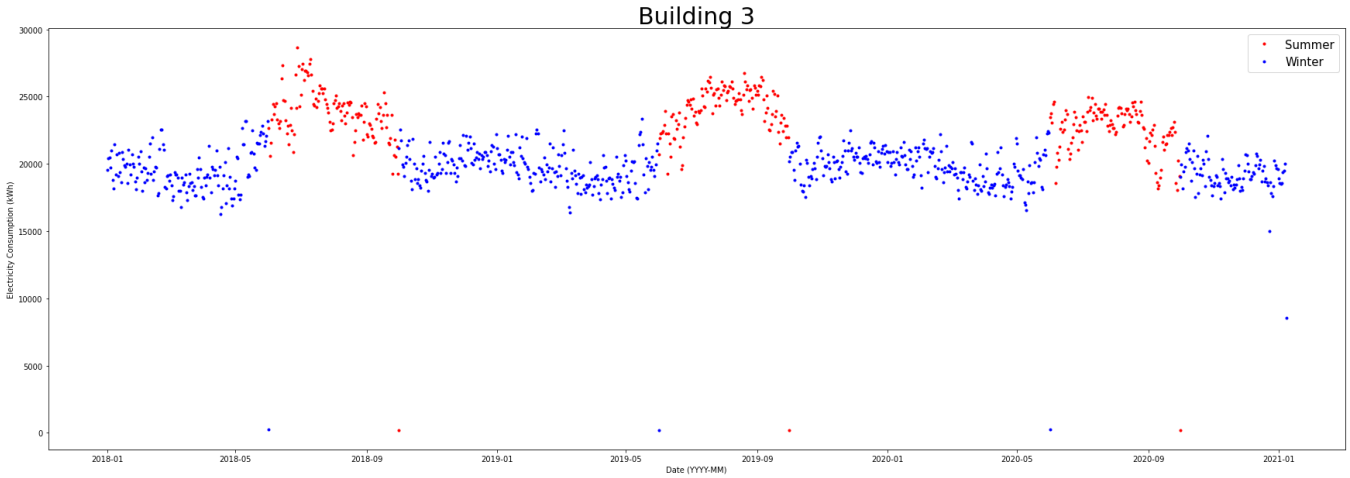
## Data Visualization

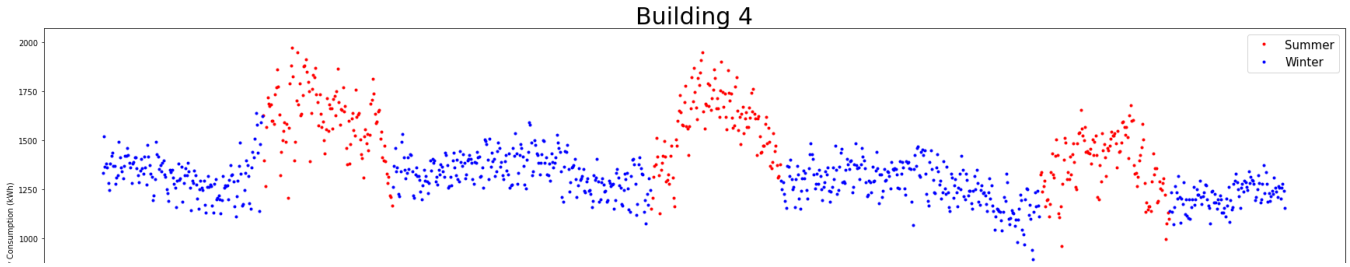
### Time Series by Season with One-Day Interval

```
for i in range(15):
  # Creates set of all timestamps, and their corresponding kWh values and season catego
  df_build = df[df["BUILDING"]=="Building "+str(i+1)].sort_values("DATETIME")[["year_mo
  # Sorts set by season, producing summer values first, then winter
  df_sum = df_build.sort_values('SEASON').reset_index()
  # The amount of days in the summer months (June, July, August, September) is always 1
  # Multiply by 96 to turn the measurements into units of days, and multiply by 3 to ac
  df_summer = df_sum[:122*96*3]
  df_summer = df_summer.drop(columns='index').drop(columns='SEASON').sort_values('year_
  df_summer = df_summer.groupby('year_month_day').sum()   # Converts to daily, summer-o
  df_winter = df_sum[122*96*3:]
  df_winter = df_winter.drop(columns='index').drop(columns='SEASON').sort_values('year_
  df_winter = df_winter.groupby('year_month_day').sum()   # Converts to daily, winter-o
  plt.figure(figsize=(30,10))
  plt.plot(df_summer,'.',color='r',label='Summer')
  plt.plot(df_winter,'.',color='b',label='Winter')
  plt.title("Building "+str(i+1),fontsize=30)
  plt.xlabel('Date (YYYY-MM)')
  plt.ylabel('Electricity Consumption (kWh)')
  plt.legend(fontsize=15)
  plt.show()
del(i,df_build,df_sum,df_summer,df_winter)
```



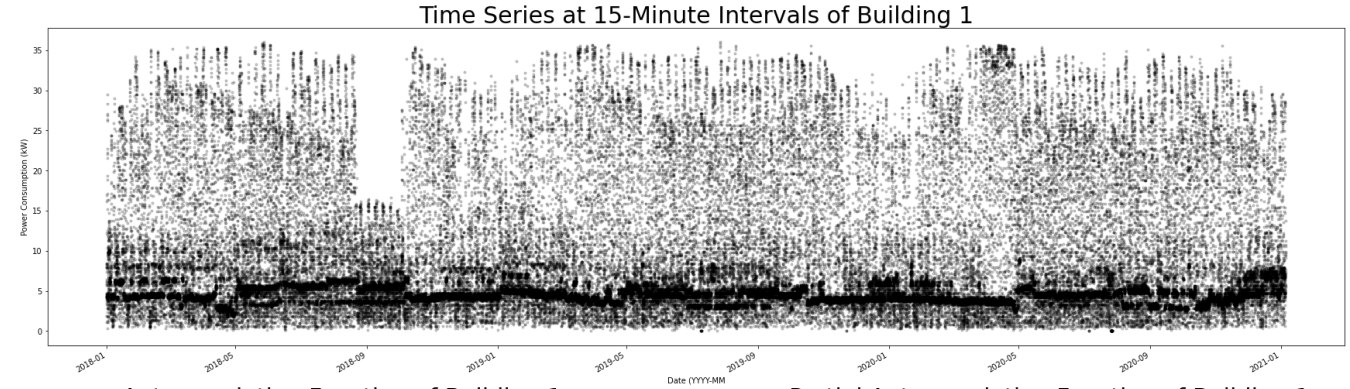



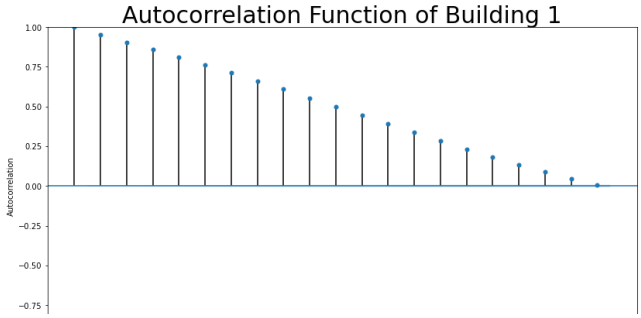

## Time Series, ACF, & PACF with 15-Minute Interval

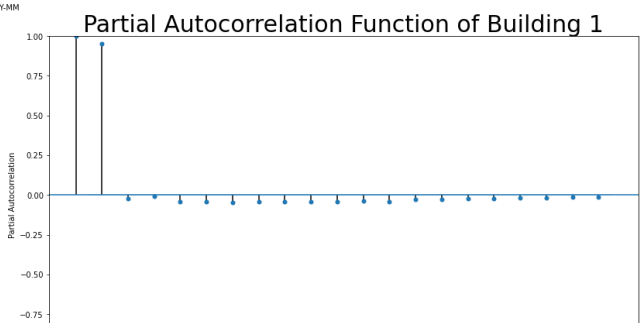
```
1 # Run Time: Approximately 1.5min
2 for index, build_name in enumerate(tqdm(df["BUILDING"].unique())):
```

```
  date_kWh = pd.DataFrame(df[(df["BUILDING"]==build_name)],columns=["DATETIME","kWh"])
  date_kWh.index = date_kWh['DATETIME']
  layout = (30,2)
  date_kWh['kWh'].plot(figsize=(30,300),title=build_name,
                       ax=plt.subplot2grid(layout,(index*2,0),colspan=2),
                       style='k.',alpha=0.2)
  plt.title('Time Series at 15-Minute Intervals of '+str(build_name),fontsize=30)
  plt.xlabel('Date (YYYY-MM')
  plt.ylabel('Power Consumption (kW)')
  plot_acf(date_kWh['kWh'],lags=20,
           ax=plt.subplot2grid(layout,(2*index+1,0)),alpha=0.5)
  plt.title('Autocorrelation Function of '+str(build_name),fontsize=30)
  plt.xlabel('Lags (15-minute Intervals)')
  plt.ylabel('Autocorrelation')
  plot_pacf(date_kWh['kWh'],lags=20,
            ax=plt.subplot2grid(layout,(2*index+1,1)),alpha=0.5)
  plt.title('Partial Autocorrelation Function of '+str(build_name),fontsize=30)
  plt.xlabel('Lags (15-minute Intervals)')
  plt.ylabel('Partial Autocorrelation')
plt.show()
del(index,build_name,date_kWh,layout)
```

```
100%|██████████| 15/15 [00:52<00:00,  3.50s/it]
```



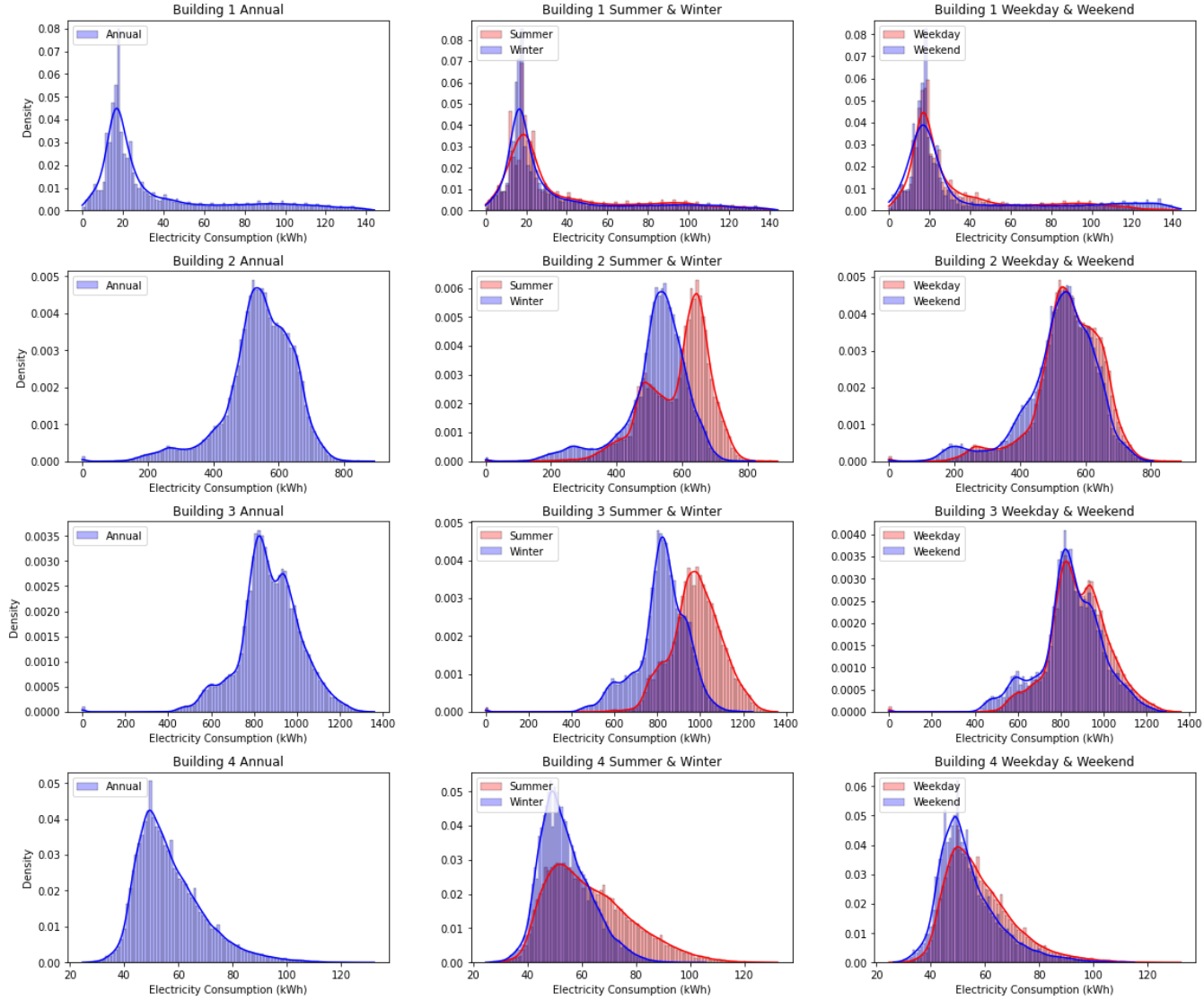




## Density Plots

```
# Run Time: Approximately 2min
fig, axs = plt.subplots(nrows=15,ncols=3,figsize=(16,50),constrained_layout=True)
for index, build_name in enumerate(tqdm(df["BUILDING"].unique())):
  density_plot('Annual',axs,index,build_name)
  density_plot('Seasonal',axs,index,build_name)
  density_plot('Days',axs,index,build_name)
plt.show()
del(fig, axs, index, build_name)
```

```
100%|██████████| 15/15 [00:51<00:00,  3.44s/it]
```



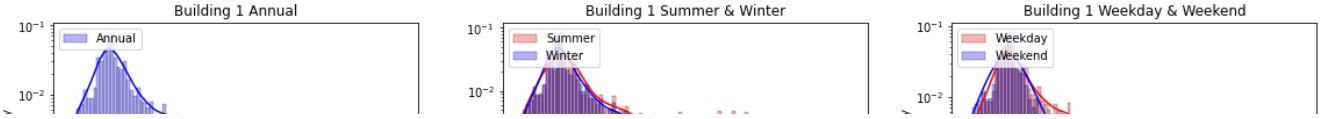

**Results/Discussion:** The above density plots illustrate energy consumption at 15-minute intervals annually, summer/winter, and weekday/weekend. Building 1 and Building 15's density plots are unimodal and relatively similar between annual, summer/winter, and weekday/weekend plots suggesting that consumption does not change through summer/winter or weekday/weekend. The lack of data illustrated in the density plots for Building 8 and Building 9 is a result of periods of zero consumption. Data from Building 8 shows no energy consumption from March 6th, 2019 to June 12th, 2019 and the data from Building 9 shows no energy consumption from February 28th, 2019 to March 2nd, 2019. A possible explanation for these gaps in energy consumption could be due to building closure, or it could be error. While data illustrated in Building 7's density plots looks minimal, data from Building 7 shows no lengthy periods in which energy consumption is zero.

## Semilogy Density Plots

```
1 # Run Time: Approximately 2min
2 fig, axs = plt.subplots(nrows=15,ncols=3,figsize=(16,50),constrained_layout=True)
3 for index, build_name in enumerate(tqdm(df["BUILDING"].unique())):
```

```
  semilogydensity_plot('Annual',axs,index,build_name)
  semilogydensity_plot('Seasonal',axs,index,build_name)
  semilogydensity_plot('Days',axs,index,build_name)
plt.show()
del(fig, axs, index, build_name)
```

```
100%|██████████| 15/15 [00:51<00:00,  3.41s/it]
```

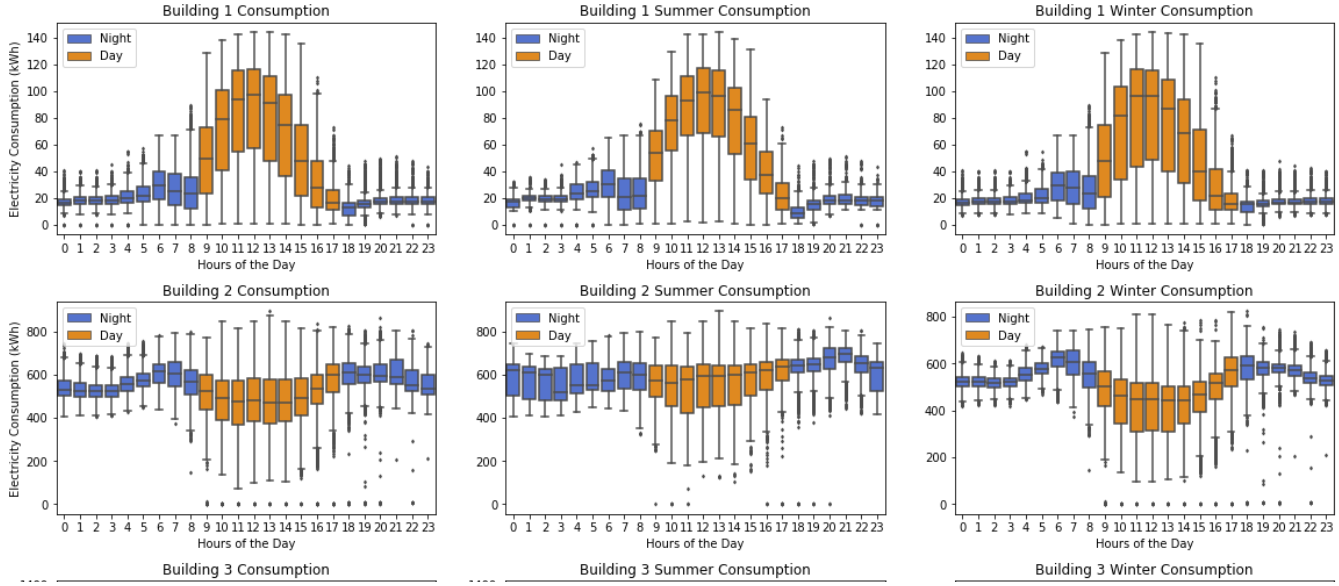



## Boxplots

## Boxplots by Season

```
# Run Time: Approximately 1.5min
fig, axs = plt.subplots(nrows=15,ncols=3,figsize=(16,50),constrained_layout=True)
for index, build_name in enumerate(tqdm(df["BUILDING"].unique())):
  box_plot('Seasonal',axs,index,build_name)
plt.show()
del(fig,axs,index,build_name)
```

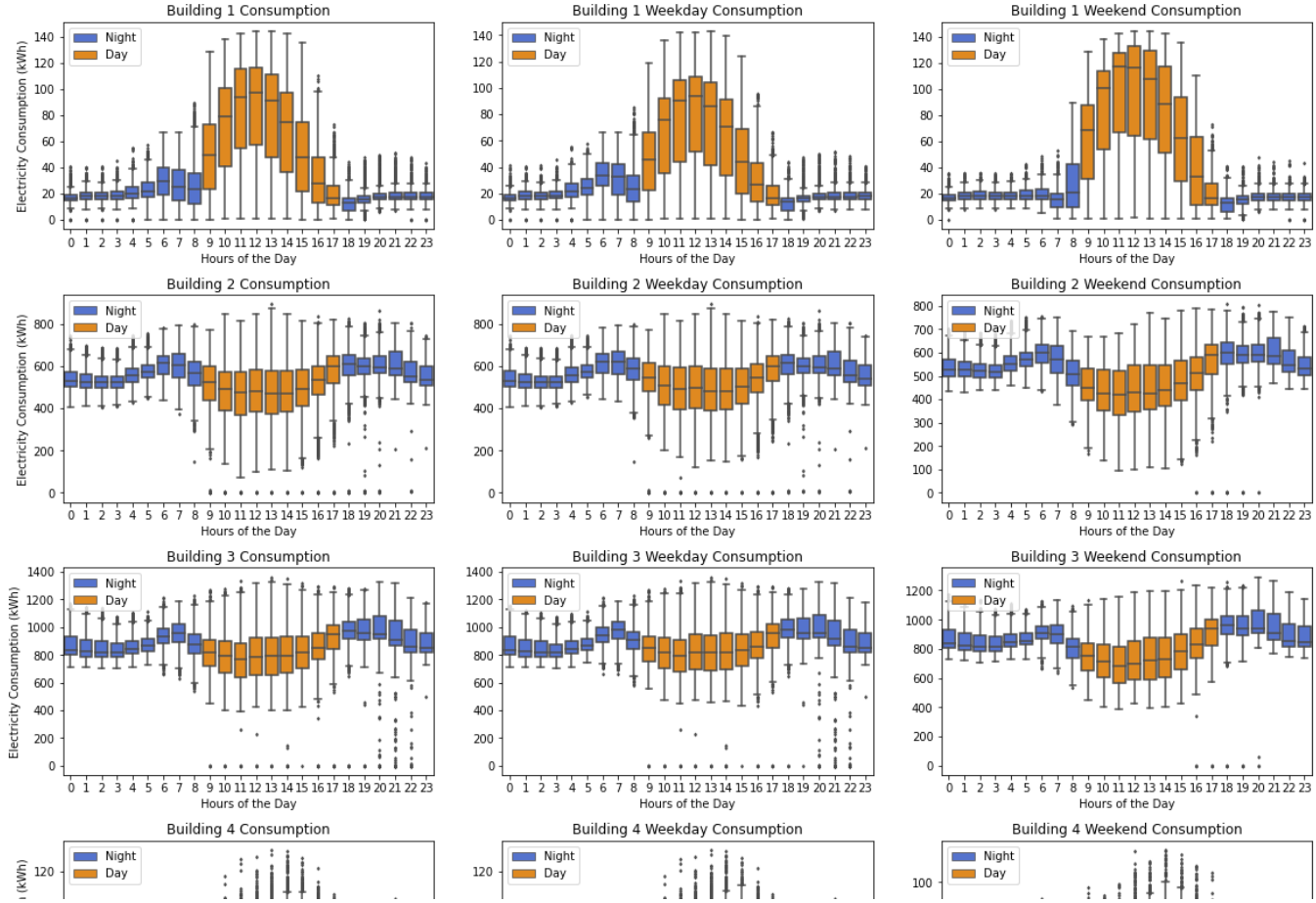
```
100%|██████████| 15/15 [00:20<00:00,  1.35s/it]
```




## Boxplots by Day of Week

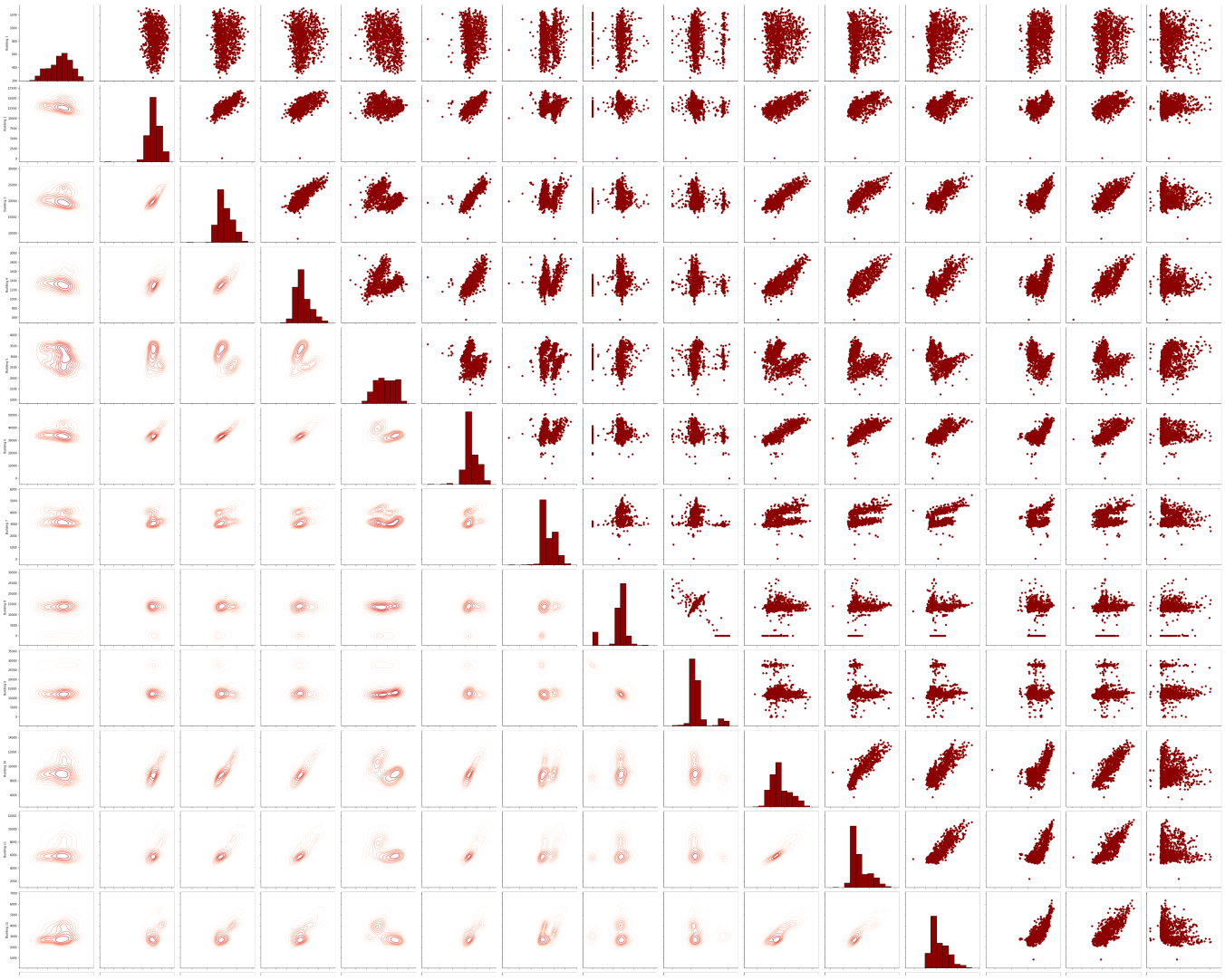
```
# Run Time: Approximately 1.5min
fig, axs = plt.subplots(nrows=15,ncols=3,figsize=(16,50),constrained_layout=True)
for index, build_name in enumerate(tqdm(df["BUILDING"].unique())):
  box_plot('Days',axs,index,build_name)
plt.show()
del(fig,axs,index,build_name)
```

```
100%|██████████| 15/15 [00:17<00:00,  1.16s/it]
```




## Scatter Matrix Plot

```
# Run Time: Approximately 4min
dfpp = df[["BUILDING","DATE","kWh"]].groupby(["DATE","BUILDING"]).sum()
dates = sorted(set([d for d, b in dfpp.index]))
items = []
for date in dates:
  items.append(dfpp.loc[date].to_dict()["kWh"])
dfpp = pd.DataFrame(items)
dfpp = dfpp[[f"Building {x}" for x in range(1,16)]]
grid = sns.PairGrid(dfpp,height=4)
grid = grid.map_upper(plt.scatter,color='darkred')
grid = grid.map_lower(sns.kdeplot,cmap='Reds')
grid = grid.map_diag(plt.hist,bins=10,color='darkred',edgecolor='k');
plt.show()
del(dfpp,dates,items,date,grid)
```



The upper triangle of the scatter matrix plot is a scatter plot, showing correlation or lack thereof between power consumption of any combination of two buildings. The lower triangle is a kernel density estimate (KDE) plot, which "represents the data using a continuous probability density curve," (seaborn) essentially showing the same correlations in a way that is less cluttered than the scatter plot.

## Building Size vs. Building Consumption

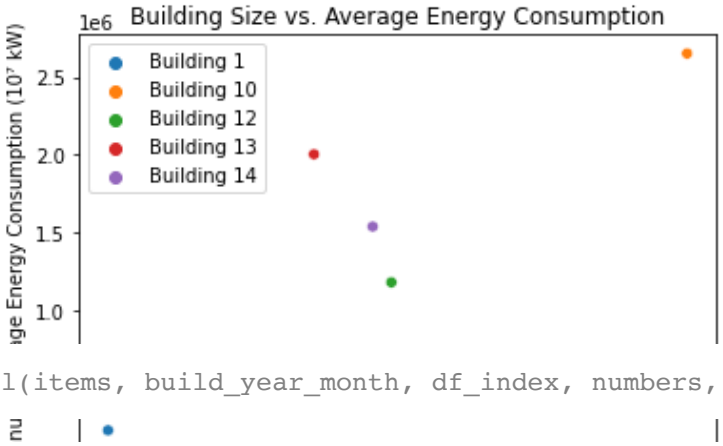

```
# Calculate Yearly Average Energy Consumption
items = {}
[items.setdefault(x, []) for x in ("Building Number","Year","Month","kWsum")]

build_year_month = list(df[['BUILDING','YEAR','MONTH']].drop_duplicates().to_records(ir
df_index = df.set_index(["BUILDING", "YEAR", "MONTH"]).sort_index()

numbers = lambda x:[int(word) for word in x.split() if word.isdigit()][0]

for BUILDING, YEAR, MONTH in tqdm(build_year_month):

  sub_df = df_index.loc[BUILDING, YEAR, MONTH]
```

```
  # Change in Time (Every Quarter Hour)
  sum_kW = sub_df.kW.sum() * Δ

  items['Building Number'].append(numbers(BUILDING))
  items["Year"].append(YEAR)
  items["Month"].append(MONTH)
  items["kWsum"].append(sum_kW)

  df_cost = pd.DataFrame(items)
df_cost = df_cost[["Building Number", "Year", "Month", "kWsum"]].groupby(["Building Num
df_cost = df_cost[[ "kWsum"]].groupby(["Building Number", "Year"]).sum()
df_cost = np.round(df_cost[['kWsum']], 2)
df_cost_average = df_cost[['kWsum']]
df_cost_average = df_cost_average.reset_index()
df_cost_average = df_cost_average[df_cost_average["Year"].isin([2018, 2019, 2020, 2021]
df_cost_average = np.round(df_cost_average[['kWsum']], 2)

dfc = df_cost_average.copy(deep=True)
dfc.reset_index(inplace=True)
```

```
100%|██████████| 554/554 [00:06<00:00, 87.31it/s]
```

```
# Add average energy consumption to building size data
sdf["Average Energy Consumption"] = dfc["kWsum"]
sdf.dropna(subset = ["Floor Area"], inplace=True)
sdf['Building ID'] = ['Building 1','Building 10','Building 12','Building 13','Building
```

```
# Create graph to show relationship between building energy consumption and building si
sns.scatterplot(
    x="Floor Area",
    y="Average Energy Consumption",
    data=sdf,
    hue = "Building ID"
)

plt.legend()
plt.title("Building Size vs. Average Energy Consumption")
plt.ylabel('Annual Average Energy Consumption ($10^7$ kW)')
plt.xlabel('Building Size ($ft^2$)')
plt.show()
print(sdf)
```



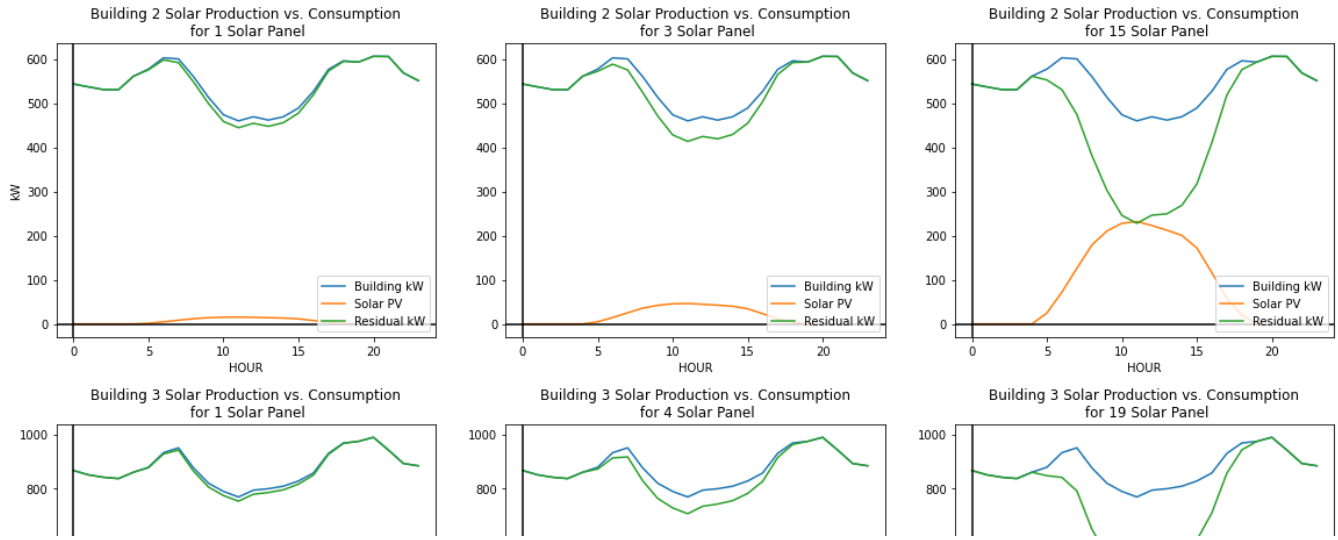

```
del(items, build_year_month, df_index, numbers, sub_df, sum_kW, df_cost, df_cost_averag
```

**Results/Discussion:** The building consumption vs. building size visualization illustrates a positive relationship between a building's size and its respective energy consumption, implying that, in general, building size increases with building energy consumption. This confirms the previous groups speculations that building size affects building consumption.

## Solar Production vs. Building Consumption

```
# Solar PV Calculations
# Tilt Angle (Constant)
f = 23.5
# Full 360 Degrees (Constant)
C = 360
# Julian Day
wdf["datetime"] = pd.to_datetime(wdf[['Year','Month','Day']])
wdf["d"] = ((wdf["datetime"].map(lambda x: (x-x.replace(month = 1, day = 1)).days)) + 1
# Julian day for Summer Solstice (Constant; Use 173 on a Leap Year)
dr = 172
# 365 days (Constant; Use 366 on a Leap Year)
dy = 365
# Declination of the Sun
wdf["del"] = f*(np.cos((C*(wdf["d"]-dr))/dy))
# Hour Angle
wdf["w"] = 15*(wdf["Hour"]-12)
# Solar Zenith Angle
wdf["sza"] = (np.sin(wdf["del"]) * np.sin(wdf["Latitude"])) + (np.cos(wdf["del"]) * np.
# GHI (kW)
wdf["GHI"] = ((wdf["DNI"] * np.cos(wdf["sza"])) + wdf["DHI"]) / 1000
# Area of Solar Panel (1in. = .0254m)
dim = [78,39]
A1 = (dim[0] * .0254) * (dim[1] * .0254)
# Solar Panel Yield (Estimated Constant)
y = 15
# Solar Panel Performance Ratio (Constant; Default Value)
pr = .75
  # Solar PV (Power)
wdf["Solar PV (1)"] = A1 * y * (wdf["GHI"]) * pr
```

```
fig, axs = plt.subplots(nrows=11,ncols=3,figsize=(16,50),constrained_layout=True)
for index, x in enumerate(ldf["Building ID"].unique()):
  df_sub = df[df["BUILDING"] == "Building {}".format(x) ][["HOUR", "kW"]].groupby("HOUR
  wdf_sub = wdf[wdf["City"] == ldf["Name/ Location"][index]][["Hour", "Solar PV (1)"]].
  wdf_sub = wdf_sub.rename({'Hour': 'HOUR'}, axis=1)
  sub = pd.merge(df_sub, wdf_sub, on="HOUR")
  n = calculation_demand(sub.copy())
  sub["Solar PV (2)"] = int(np.sqrt(n)) * sub["Solar PV (1)"]
  sub["Solar PV (3)"] = n * sub["Solar PV (1)"]
  sub["Residual (1)"] = sub["kW"] - sub["Solar PV (1)"]
  sub["Residual (2)"] = sub["kW"] - sub["Solar PV (2)"]
  sub["Residual (3)"] = sub["kW"] - sub["Solar PV (3)"]
  solar_plot(1,axs,index)
  solar_plot(2,axs,index)
  solar_plot(3,axs,index)
```

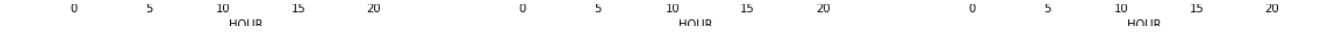



```
# wdf = wdf.drop(["datetime", "d", "del", "w", "sza", "GHI"], axis=1)
# del(f, C, dr, dy, dim, A1)
# Passes: df, sdf, ldf, wdf, df_info, dfc
```

**Solar Production vs. Consumption Results/Disscusiion:** Although a larger depression in the middle of the net demand curve can reduce the total electricity consumption of the building, it is inefficient to use solar energy in this way. As summarized in the line graphs in the data visualization section, a building's electricity consumption curve does not necessarily follow the same pattern as the solar production curve, which achieves a single peak maximum at noon. Thus, in order to keep a building's load at zero during the day, a large number of solar systems would need to be installed. An agency could certainly choose to make this investment, but such an investment would be inefficient. This project is devoted to exploring the most efficient case, i.e., installing the fewest solar systems possible, to reduce a building's electrical load.

When calculating cost of electricity consumption for each building, maximum kW of electricity used is an important parameter. By keeping this maximum value as low as possible, money can be saved on electricity consumption. With that being said, we can make an educated decision on wheather or not each building is suitable for installation of solar panels based on how many solar panels it takes to produce a residual curve that is relatively horizontal. The middle column of the Solar Production vs. Building Consumption visualization above, representing the happy medium number of solar panels, suggests that Building 5 and Building 10 produce relatively horizontal residual curves with installation of 2 and 3 commercial sized solar panels and, thus, are potential candidates for solar panel installation. Looking closer at the kW values of each buildings residual, Building 4, 12, and 15 show less than 50 kW between the residual minimum and the residual maximum suggesting they are also potential candidates for solar panel installation.

## Optimize Spending on Existing Data Consumption



## Calculate Cost


## Monthly Utility Cost


Summer: Monthly Utility Cost = Summer season kW Demand × MAX(monthly kW) + Summer season kWh × SUM(Monthly kWh)

Winter: Monthly Utility Cost = Winter season kW Demand × MAX(monthly kW) + Winter season kWh × SUM(Monthly kWh))


```
# Define a list to hold monthly utility cost
items = {}
[items.setdefault(x, []) for x in ("Building Number","Year","Month","kWsum","SG Cost","

build_year_month = list(df[['BUILDING','YEAR','MONTH']].drop_duplicates().to_records(ir
df_index = df.set_index(["BUILDING", "YEAR", "MONTH"]).sort_index()

numbers = lambda x:[int(word) for word in x.split() if word.isdigit()][0]

for BUILDING, YEAR, MONTH in tqdm(build_year_month):

  sub_df = df_index.loc[BUILDING, YEAR, MONTH]
  sum_kW = sub_df.kW.sum()

  max_kw = sub_df.kW.max()
  sum_kwh = sub_df.kW.sum() * Δ
  Season = "Summer" if MONTH in {6, 7, 8, 9} else "Winter"

  SG_kw = max_kw * df_info.loc["Secondary General (SG)"][f"{Season} Season kW Demand"]
  SG_kwh = sum_kwh * df_info.loc["Secondary General (SG)"][f"{Season} Season kWh"]

  SGL_kw = max_kw * df_info.loc["Secondary General Low-Load Factor (SGL)"][f"{Season} S
  SGL_kwh = sum_kwh * df_info.loc["Secondary General Low-Load Factor (SGL)"][f"{Season}

  PG_kw = max_kw * df_info.loc["Primary General (PG)"][f"{Season} Season kW Demand"]
  PG_kwh = sum_kwh * df_info.loc["Primary General (PG)"][f"{Season} Season kWh"]

  items['Building Number'].append(numbers(BUILDING))
  items["Year"].append(YEAR)
  items["Month"].append(MONTH)
  items["kWsum"].append(sum_kW)
  items["SG Cost"].append(SG_kw + SG_kwh)
  items["SGL Cost"].append(SGL_kw + SGL_kwh)
  items["PG Cost"].append(PG_kw + PG_kwh)
```

```
100%|██████████| 554/554 [00:05<00:00, 95.40it/s]
```


```
# Merge monthly utility cost data
df_cost = pd.DataFrame(items)
```

```
df_cost = df_cost[["Building Number", "Year", "Month", "kWsum", "SG Cost", "SGL Cost",
```

## Annual Utility Cost

Annual Utility Cost = Sum(Monthly Utility Cost)



```
df_cost = df_cost[[ "kWsum", "SG Cost", "SGL Cost", "PG Cost"]].groupby(["Building Numb
df_cost = np.round(df_cost[['kWsum', 'SG Cost', 'SGL Cost','PG Cost']], 2)
df_cost
```

| | | kWsum | SG Cost | SGL Cost | PG Cost |
|---|---|---|---|---|---|
| Building Number | Year | | | | |
| 1 | 2018 | 1217073.28 | 28116.12 | 51877.36 | 25097.58 |
| | 2019 | 1241296.00 | 29509.35 | 53462.84 | 26390.75 |
| | 2020 | 1239062.40 | 29266.87 | 53117.36 | 26171.79 |
| | 2021 | 15885.44 | 1847.59 | 1154.79 | 1605.93 |
| 2 | 2018 | 18222269.52 | 173677.07 | 697064.64 | 156854.01 |

## Calculate Cost Savings

| | | | | | |
|---|---|---|---|---|---|
| | 2020 | 19238786.70 | 172763.05 | 733699.63 | 156087.02 |

```
# Average cost savings for each building
df_cost_average = df_cost[['kWsum', 'SG Cost', 'SGL Cost','PG Cost']]
df_cost_average = df_cost_average.reset_index()
# Takes mean of three years of data for buildings with years 2018-2020, takes mean of f
df_cost_average = df_cost_average[df_cost_average["Year"].isin([2018, 2019, 2020, 2021]
df_cost_average = np.round(df_cost_average[['kWsum', 'SG Cost', 'SGL Cost','PG Cost']],

df_cost_average["save_SG_SGL"] = np.round(df_cost_average["SG Cost"] - df_cost_average[
df_cost_average["percent_SG_SGL"] = np.round((df_cost_average["save_SG_SGL"]/df_cost_av

df_cost_average["save_SG_PG"] = np.round(df_cost_average["SG Cost"] - df_cost_average['
df_cost_average["percent_SG_PG"] = np.round((df_cost_average["save_SG_PG"]/df_cost_aver

df_cost_average
```

| Building Number | kWsum | SG Cost | SGL Cost | PG Cost | save_SG_SGL | percent_SG_SGL | save |
|---|---|---|---|---|---|---|---|
| 1 | 928329.28 | 22184.98 | 39903.09 | 19816.51 | -17718.11 | -79.87 | |
| 2 | 14342575.49 | 134234.96 | 547976.64 | 121177.34 | -413741.68 | -308.22 | 1 |
| 3 | 23119065.90 | 206979.43 | 883691.66 | 187039.08 | -676712.23 | -326.95 | 1 |

```
1 del(items, build_year_month, df_index, numbers, sub_df, sum_kW, max_kw, sum_kwh, Seasor
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 4225774.91 | 39653.73 | 158036.84 | 35728.97 | -118383.11 | -298.54 | |

**Results/Discussion:** There are three electricity companies offering services to our buildings: Secondary General (SG), Secondary General Low-Load Factor (SGL), and Primary General (PG); all of which have different service charges. Primary General service requires the utility customer to own and operate their own transformer, whereas, the other service providers do not.

Data shows SGL was the most expensive service, followed by SG, leaving us with PG as the cheapest service. The SGL service is optimal in the winter when electricity consumption peaks are low. However, overall, it is not cost effective to use it as a year-round service.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 12 | 4730269.31 | 46285.27 | 185093.15 | 41948.04 | -138807.88 | -299.90 | |

## SARIMA Modelling

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 14 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | |

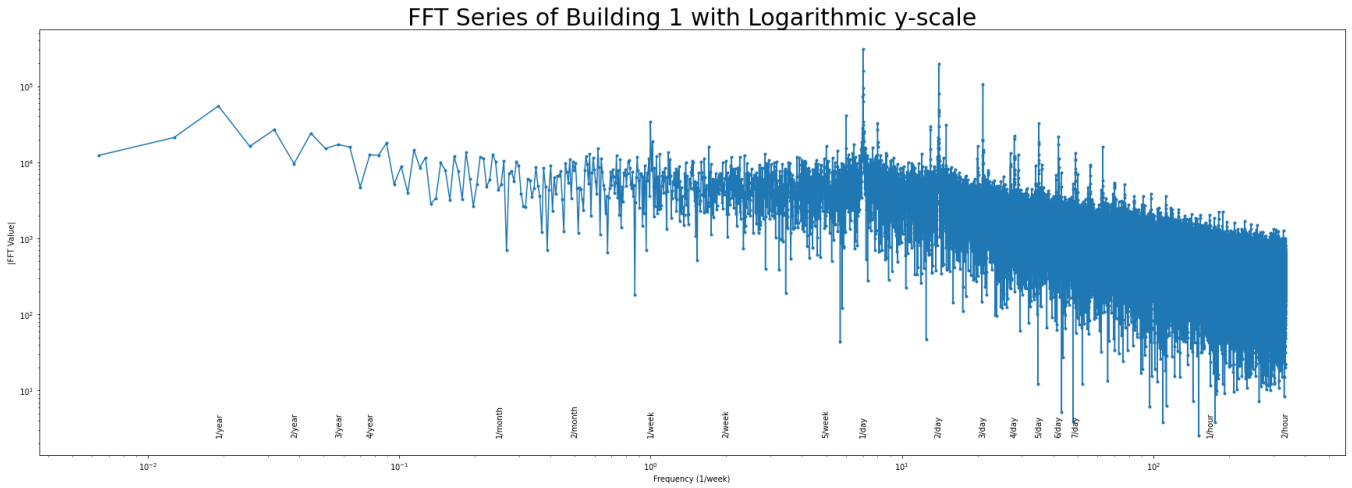
```
1 # Creates a blank table to fill out with each model created
2 sarima_summary = [[' ']*16]
3 sarima_summary = pd.DataFrame(sarima_summary,columns=['Building',1,2,3,4,5,6,7,8,9,10,1
4 sarima_summary['Building'] = ['Daily','Weekly','Yearly','p','d','q','Correlation','Ljur
5 sarima_summary.index = sarima_summary['Building']
6 sarima_summary = sarima_summary.drop(columns='Building')
7 sarima_summary
```

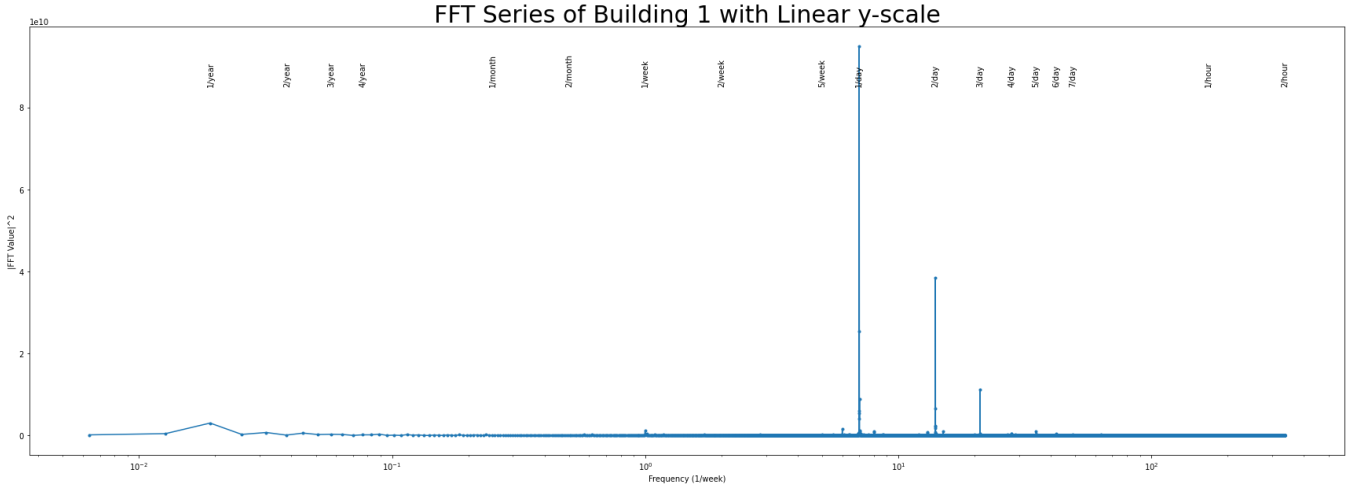
## Building 1

Weekly

```
1 rseasin = sarima_groundwork(1)
```

```
Consumption Data for all measurements for Building 1:
                    kWh
year_month_day
2018-01-01         3.84
2018-01-01         4.48
2018-01-01         4.32
2018-01-01         4.00
2018-01-01         4.64
...                 ...
2021-01-05        10.72
2021-01-05         9.60
2021-01-05         9.76
2021-01-05         9.12
2021-01-05         9.44

[105634 rows x 1 columns]

Plotting FFT of series:
```



```
sparam = [1,1,9]    # Chosen from analysis of ACF and PACF below
stat = sarima_analysis(1,rseasin,sparam)
```

There is evidence of a daily, weekly, and yearly seasonality.

```
1 sarima_summary[1] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

kWh

## Building 2

2018-01-01 4.32

```
1 rseasin = sarima_groundwork(2)
```

Consumption Data for all measurements for Building 2:

kWh

```
1 sparam = [2,1,7]
2 stat = sarima_analysis(2,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 2:
```

```
1 sarima_summary[2] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

```
2018-01-01      143.430
```

## Building 3

```
2018-01-01      142.380
```

```
1 rseasin = sarima_groundwork(3)
```

```
   Consumption Data for all measurements for Building 3:
                        kWh
   year_month_day
   2018-01-01      222.885
```

```
1 sparam = [2,1,6]
2 stat = sarima_analysis(3,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 3:
                     kWh
year_month_day
```

```
1 sarima_summary[3] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

```
2018-01-01      222.885
```

## Building 4

```
2021-01-08      176.784
```

```
1 rseasin = sarima_groundwork(4)
```

```
Consumption Data for all measurements for Building 4:
                     kWh
year_month_day
2018-01-01      13.7550
2018-01-01      13.2300
2018-01-01      13.6500
```

```
1 sparam = [5,1,12]
2 stat = sarima_analysis(4,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 4:
                     kWh
year_month_day
2018-01-01      13.7550
2018-01-01      13.2300
```

```
1 sarima_summary[4] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

```
2018-01-01      13.3875
```

## Building 5

```
2021-01-18      14.2275
```

```
1 rseasin = sarima_groundwork(5)
```

```
Consumption Data for all measurements for Building 5:
                     kWh
year_month_day
2018-01-01      32.400
2018-01-01      30.816
2018-01-01      32.112
2018-01-01      32.256
2018-01-01      30.816
```

```
1 sparam = [1,1,3]
2 stat = sarima_analysis(5,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 5:
                    kWh
year_month_day
2018-01-01      32.400
2018-01-01      30.816
2018-01-01      32.112
2018-01-01      32.256
```

```
1 sarima_summary[5] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

## Building 6

```
1 rseasin = sarima_groundwork(6)
```

```
Consumption Data for all measurements for Building 6:
                     kWh
year_month_day
2018-01-01      374.904
2018-01-01      377.571
2018-01-01      364.998
2018-01-01      369.189
2018-01-01      372.999
...                 ...
2021-01-25      354.711
```

```
1 sparam = [3,2,8]
2 stat = sarima_analysis(6,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 6:
                     kWh
year_month_day
2018-01-01       374.904
2018-01-01       377.571
2018-01-01       364.998
2018-01-01       369.189
2018-01-01       372.999
...                  ...
2021-01-25       354.711
```

```
1 sarima_summary[6] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

```
2021-01-25       353.187
```

## Building 7

```
[redacted]
```

```
1 rseasin = sarima_groundwork(7)
```

```
Consumption Data for all measurements for Building 7:
                  kWh
year_month_day
2018-01-19      44.1
2018-01-19      44.1
2018-01-19      42.0
2018-01-19      37.8
2018-01-19      37.8
...              ...
2021-01-13      48.3
2021-01-13      46.2
2021-01-13      46.2
```

```
1 sparam = [3,1,3]
2 stat = sarima_analysis(7,rseasin,sparam)
```

```
There is evidence of a daily seasonality, but not much evidence of a weekly or yearly

Consumption Data for all measurements for Building 7:
                  kWh
year_month_day
2018-01-19      44.1
2018-01-19      44.1
2018-01-19      42.0
2018-01-19      37.8
2018-01-19      37.8
...              ...
2021-01-13      48.3
2021-01-13      46.2
2021-01-13      46.2
```

```
1 sarima_summary[7] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

## Building 8

```
1 rseasin = sarima_groundwork(8)
```

```
Consumption Data for all measurements for Building 8:
                     kWh
year_month_day
2018-04-27      112.86
2018-04-27      112.32
2018-04-27      112.05
2018-04-27      111.78
2018-04-27      112.32
...                ...
2021-02-04      110.70
2021-02-04      109.35
2021-02-04      109.89
2021-02-04      110.97
2021-02-04      109.08
```

```
1 sparam = [2,1,5]
2 stat = sarima_analysis(8,rseasin,sparam)
```

```
There is evidence of a daily and weekly seasonality, but not much evidence of a yearl

Consumption Data for all measurements for Building 8:
                     kWh
year_month_day
2018-04-27      112.86
2018-04-27      112.32
2018-04-27      112.05
2018-04-27      111.78
2018-04-27      112.32
...                ...
2021-02-04      110.70
2021-02-04      109.35
2021-02-04      109.89
2021-02-04      110.97
2021-02-04      109.08
```

```
1 sarima_summary[8] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

## Building 9

```
1 rseasin = sarima_groundwork(9)
```

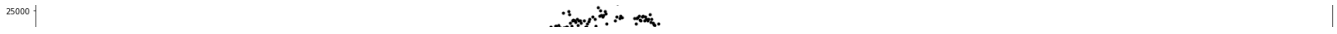
```
Consumption Data for all measurements for Building 9:
                     kWh
year_month_day
2018-01-01      137.52
2018-01-01      138.24
2018-01-01      151.92
2018-01-01      163.44
2018-01-01      159.12
...                ...
2021-01-07       96.00
2021-01-07       96.00
2021-01-07       96.00
2021-01-07       96.00
2021-01-07       96.00

[105673 rows x 1 columns]
```

```
1 sparam = [5,1,5]
2 stat = sarima_analysis(9,rseasin,sparam)
```

```
There is evidence of a daily and yearly seasonality, but not much evidence of a week

Consumption Data for all measurements for Building 9:
                     kWh
year_month_day
2018-01-01      137.52
2018-01-01      138.24
2018-01-01      151.92
2018-01-01      163.44
2018-01-01      159.12
...                ...
2021-01-07       96.00
2021-01-07       96.00
2021-01-07       96.00
2021-01-07       96.00
2021-01-07       96.00

[105673 rows x 1 columns]
```

```
1 sarima_summary[9] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],sta
```

## Building 10



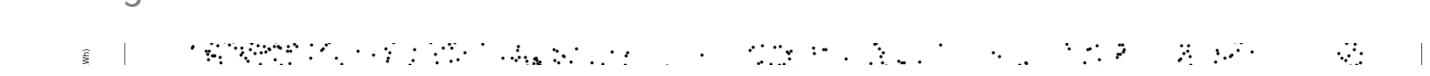

```
1 rseasin = sarima_groundwork(10)
```

```
Consumption Data for all measurements for Building 10:
                         kWh
year_month_day
2018-01-01       89.725500
2018-01-01       89.344500
2018-01-01       88.344375
2018-01-01       90.916125
2018-01-01       89.582625
...                    ...
2021-02-08      114.109500
2021-02-08      114.633375
2021-02-08      116.347875
2021-02-08      110.918625
2021-02-08      111.061500

[108917 rows x 1 columns]

Plotting FFT of series:
```

```
1 sparam = [3,1,13]
2 stat = sarima_analysis(10,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 10:
                        kWh
year_month_day
2018-01-01       89.725500
2018-01-01       89.344500
2018-01-01       88.344375
2018-01-01       90.916125
2018-01-01       89.582625
...                    ...
2021-02-08      114.109500
2021-02-08      114.633375
2021-02-08      116.347875
2021-02-08      110.918625
2021-02-08      111.061500

[108917 rows x 1 columns]

Plotting series after removing seasonalities:
```

```
1 sarima_summary[10] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],st
```

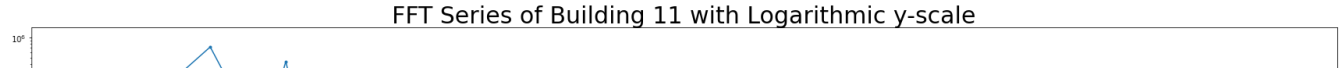


## Building 11



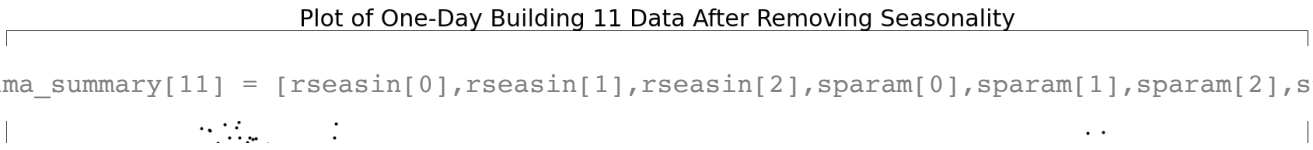
```
1 rseasin = sarima_groundwork(11)
```

```
Consumption Data for all measurements for Building 11:
                      kWh
year_month_day
2018-01-01      65.05575
2018-01-01      67.05600
2018-01-01      69.43725
2018-01-01      65.43675
2018-01-01      64.10325
...                  ...
2021-01-20      61.15050
2021-01-20      58.38825
2021-01-20      60.10275
2021-01-20      61.91250
2021-01-20      62.86500

[107079 rows x 1 columns]

Plotting FFT of series:
```



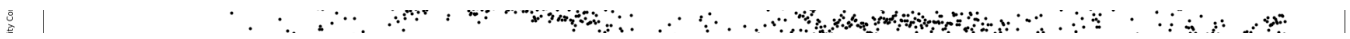

```
1 sparam = [3,1,11]
2 stat = sarima_analysis(11,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 11:
                      kWh
year_month_day
2018-01-01      65.05575
2018-01-01      67.05600
2018-01-01      69.43725
2018-01-01      65.43675
2018-01-01      64.10325
...                  ...
2021-01-20      61.15050
2021-01-20      58.38825
2021-01-20      60.10275
2021-01-20      61.91250
2021-01-20      62.86500

[107079 rows x 1 columns]

Plotting series after removing seasonalities:
```



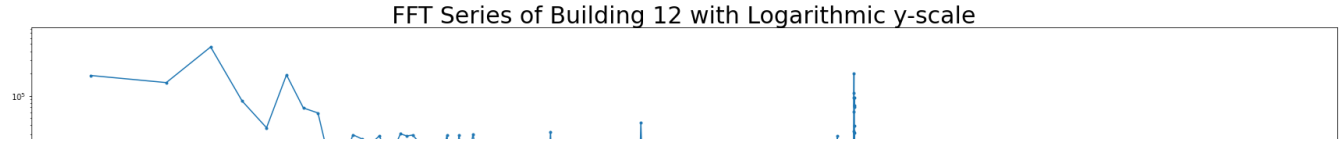

```
1 sarima_summary[11] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],st
```

## Building 12



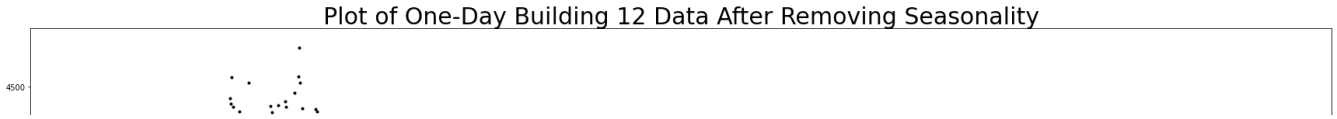
```
1 rseasin = sarima_groundwork(12)
```

```
Consumption Data for all measurements for Building 12:
                     kWh
year_month_day
2018-01-01       29.505
2018-01-01       29.820
2018-01-01       29.820
2018-01-01       30.555
2018-01-01       28.140
...                 ...
2020-12-09       25.200
2020-12-09       25.200
2020-12-09       24.500
2020-12-09       25.200
2020-12-09       25.200

[103044 rows x 1 columns]

Plotting FFT of series:
```




```
1 sparam = [2,2,7]
2 stat = sarima_analysis(12,rseasin,sparam)
```

```
There is evidence of a daily and yearly seasonality, but not much evidence of a week

Consumption Data for all measurements for Building 12:
                     kWh
year_month_day
2018-01-01      29.505
2018-01-01      29.820
2018-01-01      29.820
2018-01-01      30.555
2018-01-01      28.140
...                ...
2020-12-09      25.200
2020-12-09      25.200
2020-12-09      24.500
2020-12-09      25.200
2020-12-09      25.200

[103044 rows x 1 columns]

Plotting series after removing seasonalities:
```

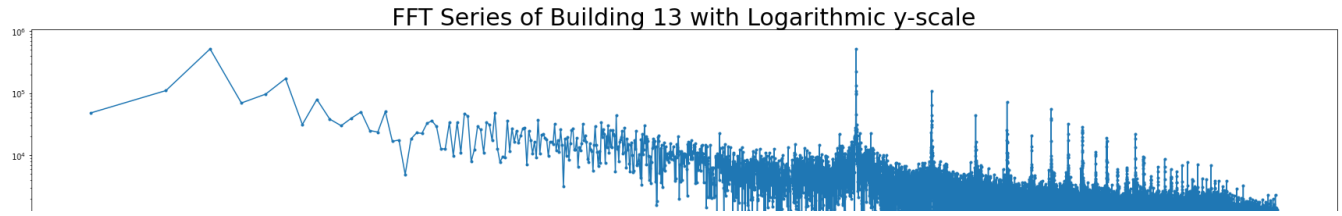



```
1 sarima_summary[12] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],st
```

## Building 13

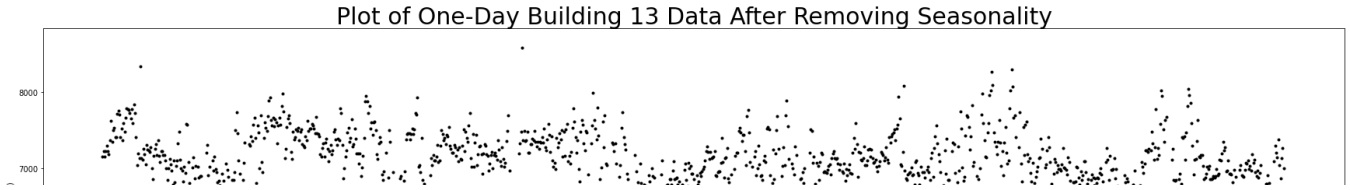
```
1 rseasin = sarima_groundwork(13)
```

```
Consumption Data for all measurements for Building 13:
                     kWh
year_month_day
2018-01-01       57.384
2018-01-01       57.096
2018-01-01       56.304
2018-01-01       56.304
2018-01-01       55.872
...                 ...
2021-02-05       69.408
2021-02-05       71.280
2021-02-05       69.120
2021-02-05       68.688
2021-02-05       68.184

[108605 rows x 1 columns]

Plotting FFT of series:
```




```
1 sparam = [3,1,4]
2 stat = sarima_analysis(13,rseasin,sparam)
```

```
There is evidence of a daily and yearly seasonality, but not much evidence of a weekl

Consumption Data for all measurements for Building 13:
                     kWh
year_month_day
2018-01-01      57.384
2018-01-01      57.096
2018-01-01      56.304
2018-01-01      56.304
2018-01-01      55.872
...                ...
2021-02-05      69.408
2021-02-05      71.280
2021-02-05      69.120
2021-02-05      68.688
2021-02-05      68.184

[108605 rows x 1 columns]

Plotting series after removing seasonalities:
```

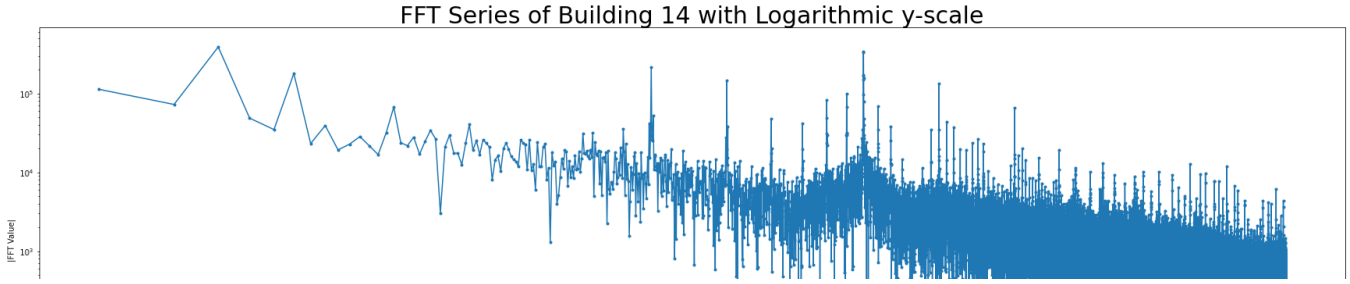



```
1 sarima_summary[13] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],st
```

## Building 14

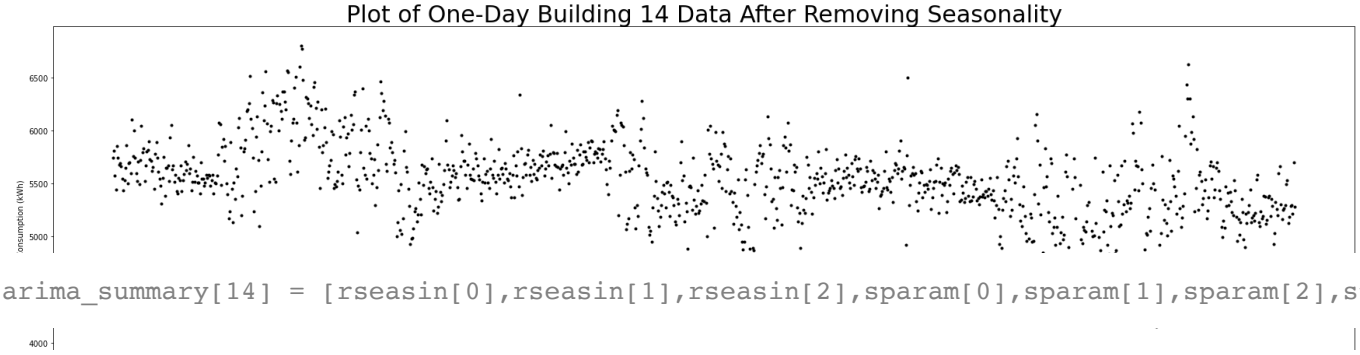

```
1 rseasin = sarima_groundwork(14)
```

```
Consumption Data for all measurements for Building 14:
                        kWh
year_month_day
2018-01-01       49.72050
2018-01-01       51.14925
2018-01-01       53.34000
2018-01-01       48.67275
2018-01-01       50.67300
...                   ...
2021-01-18       54.00675
2021-01-18       54.38775
2021-01-18       56.10225
2021-01-18       52.00650
2021-01-18       52.67325

[106892 rows x 1 columns]

Plotting FFT of series:
```



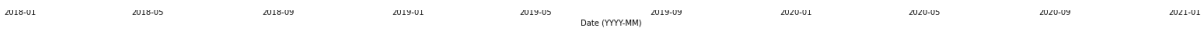

```
1 sparam = [8,3,7]
2 stat = sarima_analysis(14,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 14:
                        kWh
year_month_day
2018-01-01      49.72050
2018-01-01      51.14925
2018-01-01      53.34000
2018-01-01      48.67275
2018-01-01      50.67300
...                  ...
2021-01-18      54.00675
2021-01-18      54.38775
2021-01-18      56.10225
2021-01-18      52.00650
2021-01-18      52.67325

[106892 rows x 1 columns]

Plotting series after removing seasonalities:
```



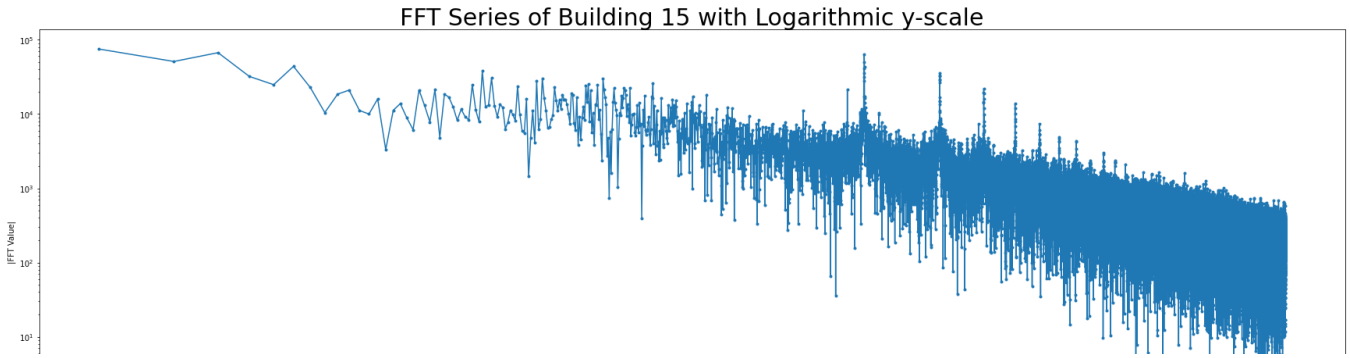

```
1 sarima_summary[14] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],st
```

## Building 15



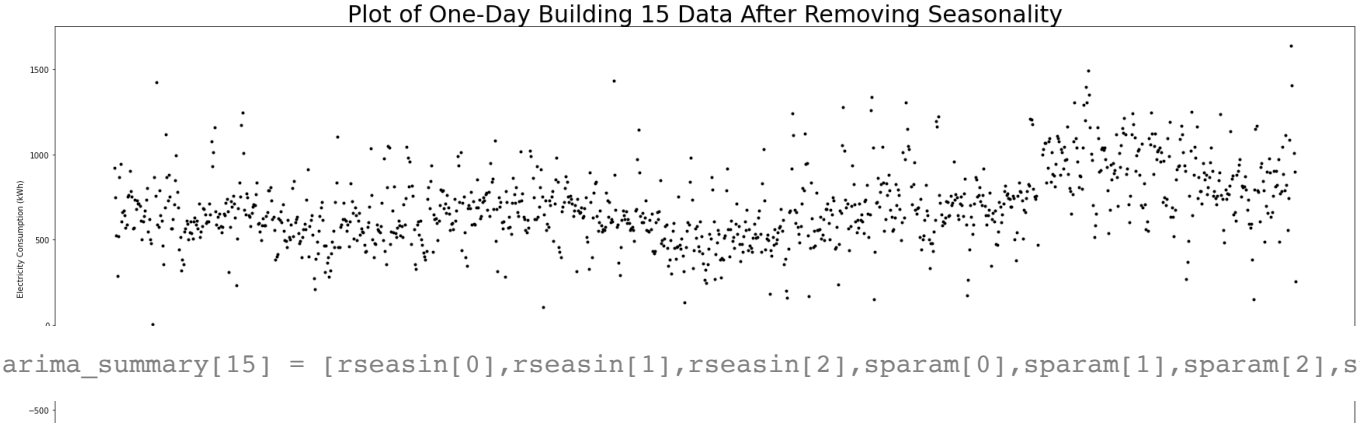

```
1 rseasin = sarima_groundwork(15)
```

```
Consumption Data for all measurements for Building 15:
                     kWh
year_month_day
2018-01-01       10.553
2018-01-01       12.293
2018-01-01       13.553
2018-01-01       10.410
2018-01-01       10.688
...                 ...
2021-02-17        7.463
2021-02-17        7.545
2021-02-17        7.823
2021-02-17        7.155
2021-02-17        7.935

[108929 rows x 1 columns]

Plotting FFT of series:
```




```
1 sparam = [2,1,8]
2 stat = sarima_analysis(15,rseasin,sparam)
```

```
There is evidence of a daily, weekly, and yearly seasonality.

Consumption Data for all measurements for Building 15:
                     kWh
year_month_day
2018-01-01      10.553
2018-01-01      12.293
2018-01-01      13.553
2018-01-01      10.410
2018-01-01      10.688
...                ...
2021-02-17       7.463
2021-02-17       7.545
2021-02-17       7.823
2021-02-17       7.155
2021-02-17       7.935

[108929 rows x 1 columns]

Plotting series after removing seasonalities:
```




```
1 sarima_summary[15] = [rseasin[0],rseasin[1],rseasin[2],sparam[0],sparam[1],sparam[2],st
```

## SARIMA Summary

```
1 sarima_summary
```

| Building | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Daily | y | y | y | y | y | y | y |
| Weekly | y | y | y | y | y | y | n |
| Yearly | y | y | y | y | y | y | n |

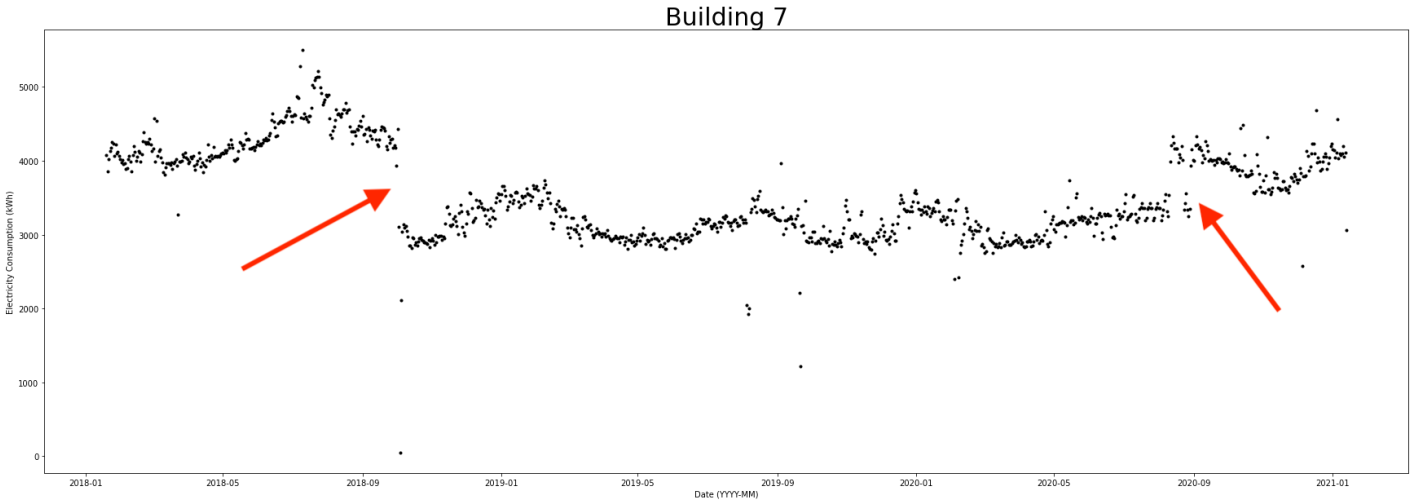
**Explanation of Statistical Tests:** The Ljung-Box test assumes a null hypothesis that data are independently distributed, and an alternative hypothesis that data are not independently distributed. Independent distribution of data indicates a valid forecasting model. According to the p-values of the Ljung-Box test of all the models, all models have independent distribution of their data after removing seasonalities.

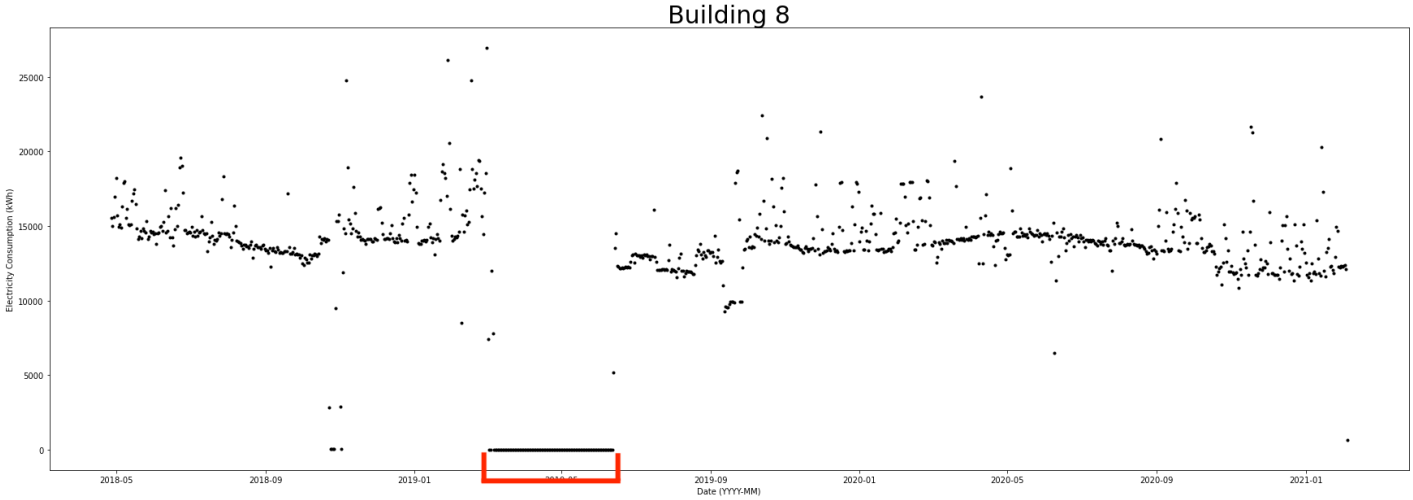
Homoscedasticity is a condition of models stating that all modelling errors have approximately the same variance, and heteroscedasticity states that modelling errors do not have the same variance. The statistical test here assumes a null hypothesis that the model has homoscedasticity, and an alternative hypothesis that the model has heteroscedasticity. Homoscedasticity is indicative of a valid forecasting model. According to the p-values of the heteroscedasticity test of all the models, Buildings 7, 8, 10, 13, 14, and 15 have heteroscedasticity, which could bring up forecasting errors for those buildings.

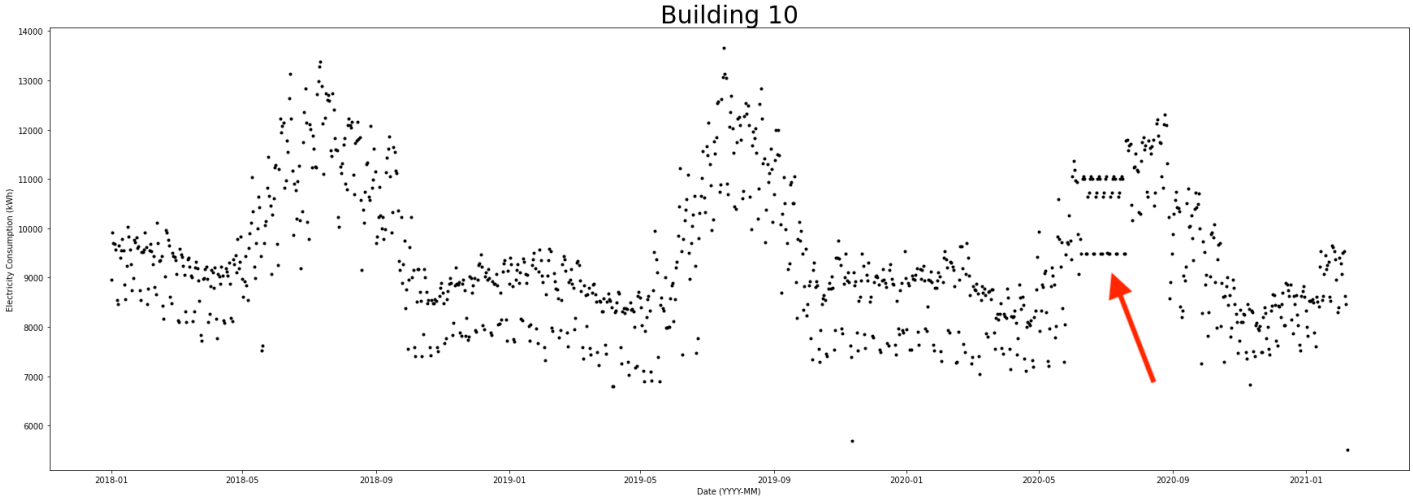
**Explanation of Correlation Coefficients:** Assuming the statistical tests are indicative of the validity of the models, the correlation coefficients of the buildings who pass the statistical tests are quite difficult to interpret as "good" or "bad" values. Ideally, a perfectly valid model would not have a correlation coefficient of 1, simply because models are not perfect representations of data. One way to interpret the correlation coefficients is relative comparison to other models who have passed the statistical tests. For example, Buildings 1 and 2 are both determined as valid by the Ljung-Box test and the heteroscedasticity test, however the correlation coefficient of Building 2's model is greater than that of Building 1, so the model created for Building 2 can be interpreted as more accurate than the model created for Building 1.

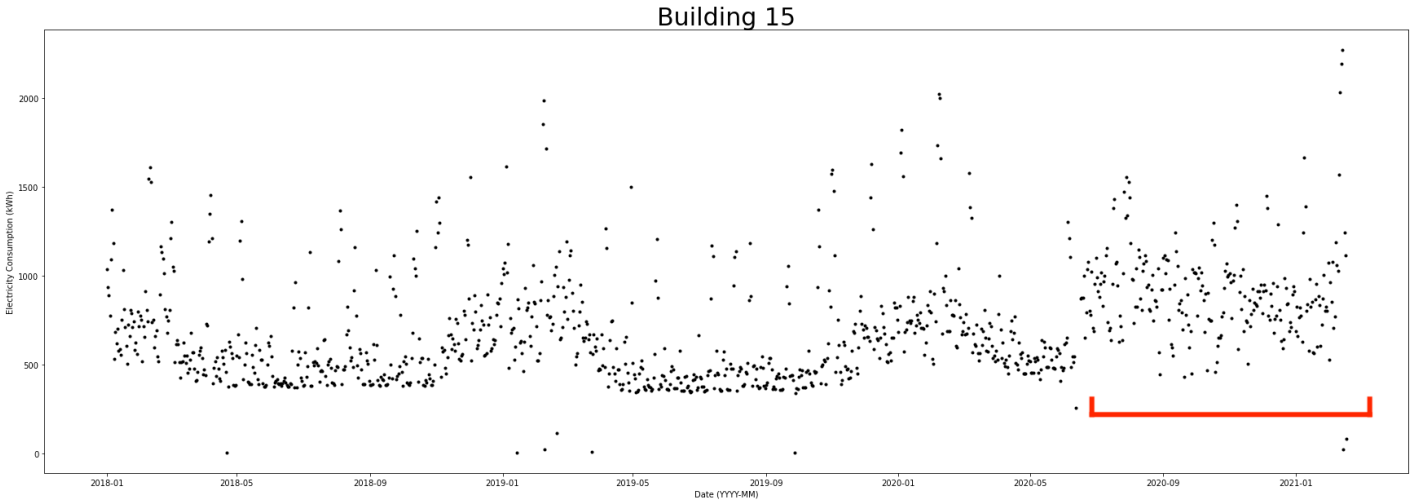
For the buildings who do not pass the statistical tests, the correlation coefficients should not carry much weight in their validity, as heteroscedasticity already invalidates the model.


**Individual Building Observations:** Observing the time series of Buildings 7, 8, 10, and 15, there seems to be visual errors or odd occurrences in data, which are outlined below. Whether or not these data errors are the causes to the modelling errors is unknown, however another possibility is that there are other relevant seasonalities that were not accounted for.


Building 7
Electricity Consumption (kWh)
Date (YYYY-MM)

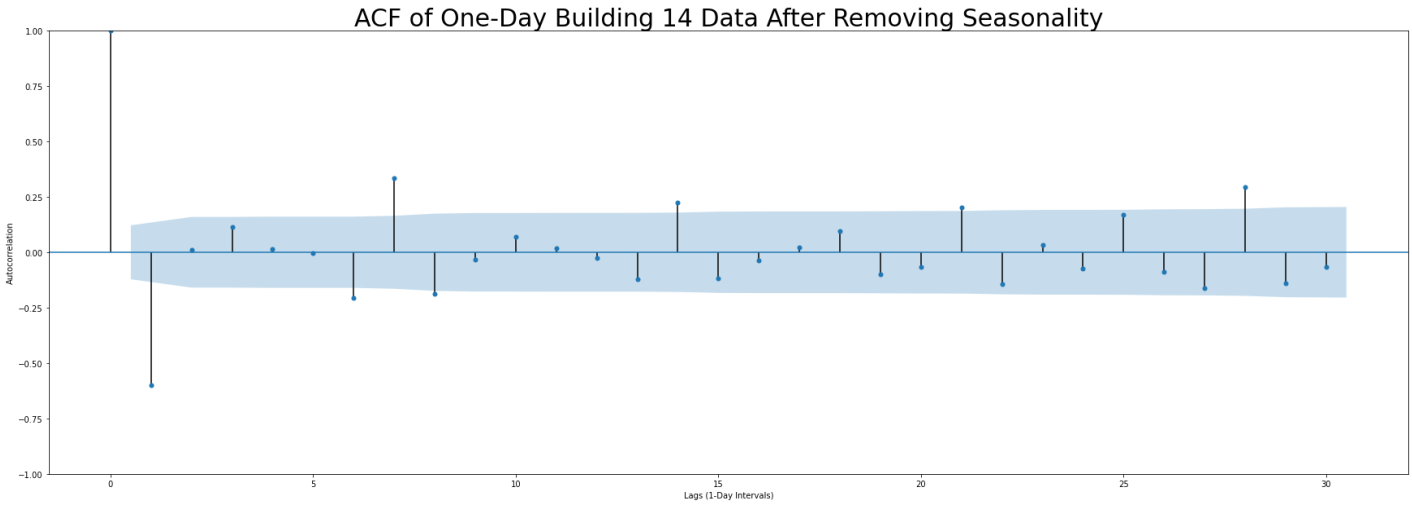

Building 8
Electricity Consumption (kWh)
Date (YYYY-MM)


Building 10
Electricity Consumption (kWh)
Date (YYYY-MM)



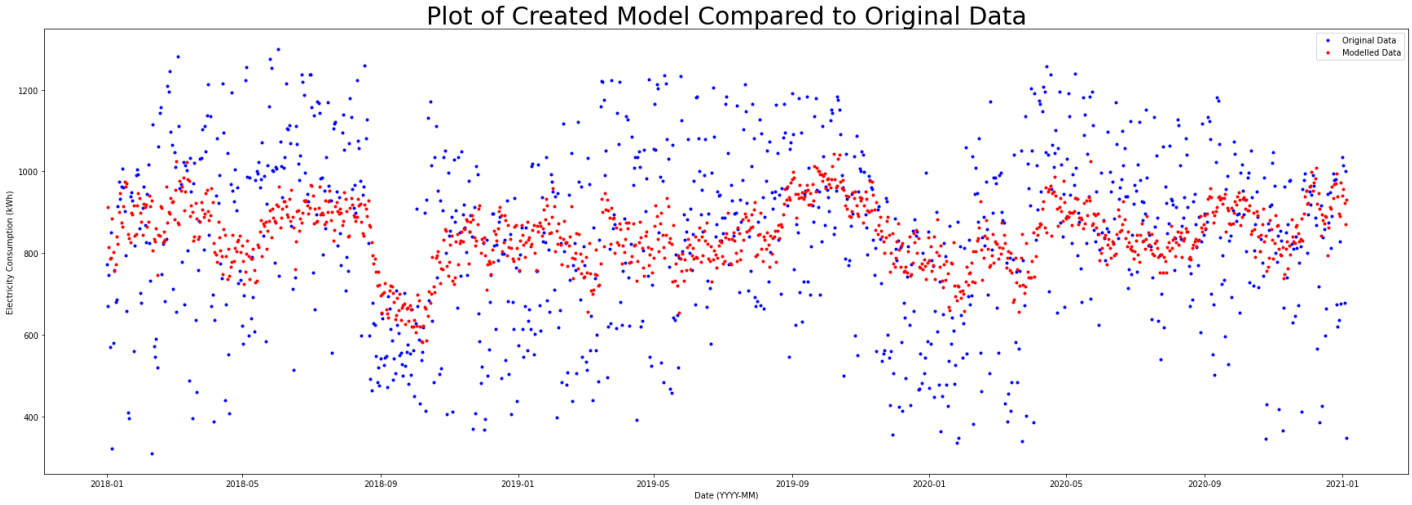

Observing Building 13, there do not seem to be any data errors, nor any errors in the ACF or PACF with differencing. So there is no indication as to why the model is heteroscedastic, so there could be a relevant seasonality that is not daily, weekly, or yearly that could create this modelling error.

When creating the model for Building 14, determining the differencing parameter was different compared to the other buildings in that the series had to be differenced three times, and differencing a fourth makes the ACF and PACF unreadable for the modelling process. However even when differenced three times, the ACF is still not ideal. This could be due to some error in data that cannot be seen visually, or it could have some additional seasonality that is not yearly, weekly, or daily.




Plotted below is the Modelled Data vs. Original Data Time Series for Building 1.

Building 1 is unique compared to the rest of the buildings, where the data shows a greater spread. After observing the effect of building size on energy consumption, it is possible that, because of the small size of Building 1, the spread is greater because there is less energy being consumed overall. Due to this spread, it seems to be more difficult to model the data with precision, and so the modelled data appears as more concise than the original data.




**Trial and Error Description:** The process for creating a convincing and accurate SARIMA model required a great amount of trial and error to arrive at a method that works well.

Trying to interpret the work done by the previous group was quite difficult, many of the decisions they made were unclear and unsupported. The models created by the previous group were SARIMA models with a weekly seasonality, for the purpose of more accuracy, ARIMA models were created instead, while taking into account different seasonalities, essentially creating seasonal models, while bypassing the limitation of a singular seasonality. In addition, the models created in this project are supported with tests of statistical significance and clear visual tracking.

*Outliers:* At one point, for instances like Building 8, where many data were missing, outliers (or values of zero for any day) were removed, and models created from that. However, this posed seasonality differencing errors. For example, if a data set had a month of its data missing, but outside of that it had a yearly seasonality. When the data are differenced by 365 days, ideally what should happen is the data point from 03-03-2018 is subtracted from the data point from 03-03-2019, however if data points are removed, what happens is that the differencing is offset, and a yearly seasonal difference is not an accurate yearly seasonal difference.

*Seasonal Differencing Method:* Currently, for first-degree differencing (after seasonal differencing), the function `diff()` is used – this function takes the value of one data point and subtracts the value of the preceding data point from it. This was also used to for seasonal differencing at one point, using `diff(periods=365)` to account for a yearly seasonality. Currently, the function `seasonal_decompose` is used as explained in Methods section. The difference between the two methods is not drastically evident, however comparing Akaike Information Criterion (AIC) from the first method to the current, the current produces a lesser AIC, which indicates a better model. Though there is no evidence to suggest that the first method is invalid by any means.

*ARIMA Parameter Grid Search:* Both with the previous group and at one point in this process, an ARIMA parameter gridsearch was conducted to come up with the best possible ARIMA parameters. However, the time commitment of this search is about 5-10 minutes per search, to repeat this for every building

would make the notebook timeout. In addition, choosing $p = 8$ over $p = 9$ would only change the AIC of the model by such a miniscule amount (usually $< 1\%$), that deciding to leave the search out did not sacrifice much.

# Conclusions

**Building Size vs. Building Consumption:** After confirming the previous group's speculations that building size affects building consumption, it can be further speculated as to why building consumption increases with building size. Our Building Size vs. Building Consumption results show that smaller sized buildings consume less energy. The most common speculation is that smaller sized buildings use less energy to power, heat, and air condition. However, another possibility is that smaller buildings may not have as much commercial use, and thus do not have as high of an electricity demand.

**Solar Production vs. Building Consumption:** The use of solar systems, as well as the lowest and most efficient solar system installation for each building was determined by each building's electricity consumption habits. From the analysis conducted, the building was suitable for solar system installation only when the building's electricity load pattern and solar production pattern were similar, i.e., the maximum values were both obtained around noon. This project found that Buildings 4, 5, 10, 12, and 15 were all potential candidates for solar panel installation.

**Optimize Spending on Existing Data Consumption:** This project confirmed that the existing data are not sufficient enough to forecast electricity consumption farther than the three to four years provided. Therefore, this project does not forecast electricity consumption for each building, and thus does not predict future annual electricity expenditures.

After calculating three to four years of electricity consumption expenditures for each building under different commercial electricity services, this project found SGL to be the most expensive service, followed by SG. Our calculations found PG to be the cheapest service for all buildings. Dependent on the cost of installing a transformer, the use of PG services could be debated.

**SARIMA Modelling:** To add to the previous group's work, the process of creating ARIMA models was made more clear such that the process can be easily manipulated to make improvements for future groups. In addition, with the use of Fast Fourier Transform analysis of the buildings, it can be said with accuracy that, if any of the buildings had daily, weekly, or yearly seasonalities, they were correctly accounted for and differenced properly. Models created for Buildings 1, 2, 3, 4, 5, 6, 9, 11, and 12 have statistical evidence to suggest they are valid forecasting models, and can be ranked according to their correlation coefficients. Although some errors or anomalies in the raw data may be the cause of the heteroscedasticity present in the models of Buildings 7, 8, 10, 13, 14, and 15, it may also be due to a relevant seasonality that was not daily, weekly, or yearly – a possibility that seems quite evident with the

abnormality of Building 14, as explained in the "SARIMA Summary" section. A clear observation of the correlation coefficients shows that Buildings 7, 8, and 9 have abnormally high coefficients – these should not be taken into consideration when determining if these models are valid, the presence of heteroscedasticity overrides and invalidates the correlation coefficient value. Should the heteroscedasticity be a result of errors in data, it is likely that accurate forecasting models for those buildings cannot be created from these data with confidence.

One possibility to create valid models for the buildings whose data show abnormalities is to collect more data – an increased data set size decreases the effect of the abnormalities. An extension of this project should be to understand if there are other relevant seasonalities present with the consumption of all buildings, and if so, create ARIMA models with those seasonalities in mind and test the validity of the models, which should ideally improve.

**Reader Takeaways**

This project studied and observed that building size is positively correlated with building electricity consumption. With the studies of solar production, it was found that Buildings 4, 5, 10, 12, and 15 are potential candidates that would benefit from solar panel installation, whereas Buildings 1, 2, 3, 6, 7, 8, 9, 11, 13, and 14 do not have much evidence to suggest a benefit. Of the three commercial electricity services, SGL was found to be the most expensive, followed by SG, followed by PG as the least expensive. In addition, SARIMA models created for Buildings 1, 2, 3, 4, 5, 6, 9, 11, and 12 were found to have statistical evidence to suggest they are valid forecasting models, whereas models created for Buildings 7, 8, 10, 13, 14, and 15 have statistical evidence to refute their accuracy.