In [73]:

```
import pandas as pd
from datetime import datetime
import numpy as np
import datetime as dt
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, f1_score
import seaborn as sns
import plotly.express as px
```

Q1

In [74]:

```
weather_data = pd.read_csv('OneDrive\Desktop\weather_data.csv')
energy_data = pd.read_csv('OneDrive\Desktop\energy_data.csv',parse_dates=True)
```

In [75]:

```
display(weather_data)
```

	temperature	icon	humidity	visibility	summary	pressure	windSpeed	cloudCover	time
0	34.98	partly-cloudy-night	0.64	10.00	Partly Cloudy	1017.69	7.75	0.29	1388534400
1	16.49	clear-night	0.62	10.00	Clear	1022.76	2.71	0.06	1388538000
2	14.63	clear-night	0.68	10.00	Clear	1022.32	4.84	0.03	1388541600
3	13.31	clear-night	0.71	10.00	Clear	1021.64	4.00	0.14	1388545200
4	13.57	clear-night	0.71	9.93	Clear	1020.73	3.67	0.04	1388548800
...	...	...	...	...	...	...	...	...	...
8755	27.48	clear-day	0.35	10.00	Clear	1023.54	10.54	0.24	1420052400
8756	27.17	partly-cloudy-day	0.35	10.00	Partly Cloudy	1023.60	9.53	0.25	1420056000
8757	25.72	clear-day	0.37	10.00	Clear	1023.44	8.12	0.08	1420059600
8758	22.75	clear-night	0.42	10.00	Clear	1023.29	4.43	0.05	1420063200
8759	20.09	clear-night	0.51	10.00	Clear	1023.18	1.33	0.11	1420066800

8760 rows × 13 columns

In [76]:

```
display(energy_data)
```

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Utility Rm + Basement Bath [kW]	
0	2014-01-01 00:00:00	0.304439	0.0	0.304439	0.000058	0.009531	0.005336	0.000126	0.011175	0.003836	(
1	2014-01-01 00:30:00	0.656771	0.0	0.656771	0.001534	0.364338	0.005522	0.000043	0.003514	0.003512	(
2	2014-01-01 01:00:00	0.612895	0.0	0.612895	0.001847	0.417989	0.005504	0.000044	0.003528	0.003484	(
3	2014-01-01 01:30:00	0.683979	0.0	0.683979	0.001744	0.410653	0.005556	0.000059	0.003499	0.003476	(
4	2014-01-01 02:00:00	0.197809	0.0	0.197809	0.000030	0.017152	0.005302	0.000119	0.003694	0.003865	(
...	...	...	...	...	...	...	...	...	...	...	
17515	2014-12-31 21:30:00	1.560890	0.0	1.560890	0.003226	0.392996	0.006342	0.000872	0.030453	0.002248	(
17516	2014-12-31 22:00:00	0.958447	0.0	0.958447	0.000827	0.027369	0.006326	0.000811	0.030391	0.002543	(
17517	2014-12-31 22:30:00	0.834462	0.0	0.834462	0.001438	0.170561	0.020708	0.000636	0.012631	0.002372	(
17518	2014-12-31 23:00:00	0.543863	0.0	0.543863	0.001164	0.153533	0.008423	0.000553	0.003832	0.002353	(
17519	2014-12-31 23:30:00	0.414441	0.0	0.414441	0.000276	0.009223	0.006619	0.000526	0.003818	0.002424	(

17520 rows × 18 columns

In [77]:

```
weather_data['time'] = pd.to_datetime(weather_data['time'], unit='s', origin='unix')
```

In [78]:

```
display(weather_data)
```

	temperature	icon	humidity	visibility	summary	pressure	windSpeed	cloudCover	time	wi
0	34.98	partly-cloudy-night	0.64	10.00	Partly Cloudy	1017.69	7.75	0.29	2014-01-01 00:00:00	
1	16.49	clear-night	0.62	10.00	Clear	1022.76	2.71	0.06	2014-01-01 01:00:00	
2	14.63	clear-night	0.68	10.00	Clear	1022.32	4.84	0.03	2014-01-01 02:00:00	
3	13.31	clear-night	0.71	10.00	Clear	1021.64	4.00	0.14	2014-01-01 03:00:00	
4	13.57	clear-night	0.71	9.93	Clear	1020.73	3.67	0.04	2014-01-01 04:00:00	
...	...	...	...	...	...	...	...	...	...	
8755	27.48	clear-day	0.35	10.00	Clear	1023.54	10.54	0.24	2014-12-31 19:00:00	
8756	27.17	partly-cloudy-day	0.35	10.00	Partly Cloudy	1023.60	9.53	0.25	2014-12-31 20:00:00	
8757	25.72	clear-day	0.37	10.00	Clear	1023.44	8.12	0.08	2014-12-31 21:00:00	
8758	22.75	clear-night	0.42	10.00	Clear	1023.29	4.43	0.05	2014-12-31 22:00:00	
8759	20.09	clear-night	0.51	10.00	Clear	1023.18	1.33	0.11	2014-12-31 23:00:00	

8760 rows × 13 columns

In [79]:

```
display(energy_data)
```

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Utility Rm + Basement Bath [kW]	
0	2014-01-01 00:00:00	0.304439	0.0	0.304439	0.000058	0.009531	0.005336	0.000126	0.011175	0.003836	(
1	2014-01-01 00:30:00	0.656771	0.0	0.656771	0.001534	0.364338	0.005522	0.000043	0.003514	0.003512	(
2	2014-01-01 01:00:00	0.612895	0.0	0.612895	0.001847	0.417989	0.005504	0.000044	0.003528	0.003484	(
3	2014-01-01 01:30:00	0.683979	0.0	0.683979	0.001744	0.410653	0.005556	0.000059	0.003499	0.003476	(
4	2014-01-01 02:00:00	0.197809	0.0	0.197809	0.000030	0.017152	0.005302	0.000119	0.003694	0.003865	(
...	...	...	...	...	...	...	...	...	...	...	
17515	2014-12-31 21:30:00	1.560890	0.0	1.560890	0.003226	0.392996	0.006342	0.000872	0.030453	0.002248	(
17516	2014-12-31 22:00:00	0.958447	0.0	0.958447	0.000827	0.027369	0.006326	0.000811	0.030391	0.002543	(
17517	2014-12-31 22:30:00	0.834462	0.0	0.834462	0.001438	0.170561	0.020708	0.000636	0.012631	0.002372	(
17518	2014-12-31 23:00:00	0.543863	0.0	0.543863	0.001164	0.153533	0.008423	0.000553	0.003832	0.002353	(
17519	2014-12-31 23:30:00	0.414441	0.0	0.414441	0.000276	0.009223	0.006619	0.000526	0.003818	0.002424	(

17520 rows × 18 columns

In [80]:

```
energy_data['Date & Time'] = pd.to_datetime(energy_data['Date & Time'])
```

In [81]:

```
display(energy_data['Date & Time'])
```

```
0       2014-01-01 00:00:00
1       2014-01-01 00:30:00
2       2014-01-01 01:00:00
3       2014-01-01 01:30:00
4       2014-01-01 02:00:00
                ...
17515   2014-12-31 21:30:00
17516   2014-12-31 22:00:00
17517   2014-12-31 22:30:00
17518   2014-12-31 23:00:00
17519   2014-12-31 23:30:00
Name: Date & Time, Length: 17520, dtype: datetime64[ns]
```

In [82]:

```
#sum of energy per day
energy_data.groupby(energy_data['Date & Time'].dt.date)['use [kW]'].sum()
```

Out[82]:

```
Date & Time
2014-01-01    65.013592
2014-01-02    32.305336
2014-01-03    31.164468
2014-01-04    45.287782
2014-01-05    36.316643
                ...
2014-12-27    35.046127
2014-12-28    37.695824
2014-12-29    28.675929
2014-12-30    31.514313
2014-12-31    28.674498
Name: use [kW], Length: 365, dtype: float64
```

In [83]:

```
energy_sum_per_day = energy_data.groupby(energy_data['Date & Time'].dt.date)['use [kW]'].sum().rese
```

In [84]:

```
energy_sum_per_day['Date & Time'] = pd.to_datetime(energy_sum_per_day['Date & Time'])
```

In [85]:

```
#summary is just less detailed version of icon so we drop it
merged = pd.merge_asof(weather_data, energy_sum_per_day, left_on='time', right_on='Date & Time').d
```

In [86]:

```
merged = pd.concat([merged.drop(columns=['icon']),pd.get_dummies(merged.icon, drop_first=True)], a
```

In [87]:

```
merged
```

Out[87]:

	temperature	humidity	visibility	pressure	windSpeed	cloudCover	time	windBearing	precipIn
0	34.98	0.64	10.00	1017.69	7.75	0.29	2014-01-01 00:00:00	279	
1	16.49	0.62	10.00	1022.76	2.71	0.06	2014-01-01 01:00:00	195	
2	14.63	0.68	10.00	1022.32	4.84	0.03	2014-01-01 02:00:00	222	
3	13.31	0.71	10.00	1021.64	4.00	0.14	2014-01-01 03:00:00	209	
4	13.57	0.71	9.93	1020.73	3.67	0.04	2014-01-01 04:00:00	217	
...	...	...	...	...	...	...	...	...	
8755	27.48	0.35	10.00	1023.54	10.54	0.24	2014-12-31 19:00:00	311	
8756	27.17	0.35	10.00	1023.60	9.53	0.25	2014-12-31 20:00:00	297	
8757	25.72	0.37	10.00	1023.44	8.12	0.08	2014-12-31 21:00:00	292	
8758	22.75	0.42	10.00	1023.29	4.43	0.05	2014-12-31 22:00:00	299	
8759	20.09	0.51	10.00	1023.18	1.33	0.11	2014-12-31 23:00:00	275	

8760 rows × 20 columns

In [88]:

```
weather_data.index
```

Out[88]:

```
RangeIndex(start=0, stop=8760, step=1)
```

In [89]:

```
weather_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   temperature        8760 non-null   float64
 1   icon               8760 non-null   object
 2   humidity           8760 non-null   float64
 3   visibility         8760 non-null   float64
 4   summary            8760 non-null   object
 5   pressure           8760 non-null   float64
 6   windSpeed          8760 non-null   float64
 7   cloudCover         7290 non-null   float64
 8   time               8760 non-null   datetime64[ns]
 9   windBearing        8760 non-null   int64
 10  precipIntensity    8760 non-null   float64
 11  dewPoint           8760 non-null   float64
 12  precipProbability  8760 non-null   float64
dtypes: datetime64[ns](1), float64(9), int64(1), object(2)
memory usage: 889.8+ KB
```

In [90]:

```
energy_sum_per_day.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date & Time  365 non-null    datetime64[ns]
 1   use [kW]     365 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 5.8 KB
```

Q2

In [91]:

```
train = merged.query("time < '2014-12-01'" )
```

In [92]:

```
train
```

Out[92]:

	temperature	humidity	visibility	pressure	windSpeed	cloudCover	time	windBearing	precipIn
0	34.98	0.64	10.00	1017.69	7.75	0.29	2014-01-01 00:00:00	279	
1	16.49	0.62	10.00	1022.76	2.71	0.06	2014-01-01 01:00:00	195	
2	14.63	0.68	10.00	1022.32	4.84	0.03	2014-01-01 02:00:00	222	
3	13.31	0.71	10.00	1021.64	4.00	0.14	2014-01-01 03:00:00	209	
4	13.57	0.71	9.93	1020.73	3.67	0.04	2014-01-01 04:00:00	217	
...	...	...	...	...	...	...	...	...	
8011	46.43	0.65	10.00	1017.92	7.07	0.00	2014-11-30 19:00:00	191	
8012	46.10	0.66	10.00	1017.77	6.76	0.17	2014-11-30 20:00:00	199	
8013	44.75	0.71	10.00	1017.61	5.64	0.00	2014-11-30 21:00:00	193	
8014	44.71	0.71	10.00	1017.46	5.92	NaN	2014-11-30 22:00:00	186	
8015	44.53	0.71	9.96	1017.47	7.05	NaN	2014-11-30 23:00:00	189	

8016 rows × 20 columns

In [93]:

```
test = merged.query("time >= '2014-12-01'" )
```

In [94]:

```
test
```

Out[94]:

	temperature	humidity	visibility	pressure	windSpeed	cloudCover	time	windBearing	precipIn
8016	44.86	0.69	10.00	1017.71	5.52	1.00	2014-12-01 00:00:00	188	
8017	44.90	0.68	10.00	1017.82	6.96	NaN	2014-12-01 01:00:00	190	
8018	44.10	0.70	10.00	1017.81	5.29	NaN	2014-12-01 02:00:00	177	
8019	44.13	0.70	10.00	1017.55	5.83	NaN	2014-12-01 03:00:00	179	
8020	43.57	0.74	9.91	1017.43	6.35	NaN	2014-12-01 04:00:00	181	
...	...	...	...	...	...	...	...	...	
8755	27.48	0.35	10.00	1023.54	10.54	0.24	2014-12-31 19:00:00	311	
8756	27.17	0.35	10.00	1023.60	9.53	0.25	2014-12-31 20:00:00	297	
8757	25.72	0.37	10.00	1023.44	8.12	0.08	2014-12-31 21:00:00	292	
8758	22.75	0.42	10.00	1023.29	4.43	0.05	2014-12-31 22:00:00	299	
8759	20.09	0.51	10.00	1023.18	1.33	0.11	2014-12-31 23:00:00	275	

744 rows × 20 columns

Q3

In [95]:

```
#train the model
train = train.dropna()
test = test.dropna()

x_train = train.drop(columns=['time', 'use [kW]'])
y_train = train['use [kW]']

x_test = test.drop(columns=['time', 'use [kW]'])
y_test = test['use [kW]']

linear_regressor = LinearRegression()  # create object
linear_regressor.fit(x_train, y_train)  #linear regression

Y_pred = linear_regressor.predict(x_test)  #makes predictions
```

In [96]:

```
rmse = mean_squared_error(y_test, Y_pred)
```

In [97]:

```
rmse
```

Out[97]:

```
53.58733216875516
```

In [98]:

```
energy_sum_per_day['use [kW]'].mean()
```

Out[98]:

```
31.819442182739742
```

3) The model is quite bad. As one can see from the root mean squared error (rmse) value calculated above, the model doesn't work very well at all. I think that this makes some sense as the model uses data that works in a somewhat backwards day. It seems to use the daily values to estimate the hourly usage, which seems somewhat backwards. Due to this reverse nature of the model, it makes perfect sense for the root mean squared error to indicate a poor model.

In [99]:

```
prediction_df = pd.DataFrame({'date':test.time, 'prediction':Y_pred})
```

In [100]:

```
prediction_df
```

	date	prediction
8016	2014-12-01 00:00:00	26.984465
8022	2014-12-01 06:00:00	25.715309
8024	2014-12-01 08:00:00	29.144784
8025	2014-12-01 09:00:00	31.491754
8026	2014-12-01 10:00:00	29.187679
...	...	...
8755	2014-12-31 19:00:00	22.178062
8756	2014-12-31 20:00:00	23.271941
8757	2014-12-31 21:00:00	24.715877
8758	2014-12-31 22:00:00	25.947346
8759	2014-12-31 23:00:00	26.525612

In [101]:

```
prediction_df.to_csv("cse351_hw2_Shvartsman_Terrence_114311609_linear_regression.csv", index=False
```

In [ ]:

```

```

Q4

In [102]:

```
merged['high/low'] = np.where(merged.temperature >= 35, 1, 0)
```

In [103]:

```
merged
```

Out[103]:

	temperature	humidity	visibility	pressure	windSpeed	cloudCover	time	windBearing	precipIn
0	34.98	0.64	10.00	1017.69	7.75	0.29	2014-01-01 00:00:00	279	
1	16.49	0.62	10.00	1022.76	2.71	0.06	2014-01-01 01:00:00	195	
2	14.63	0.68	10.00	1022.32	4.84	0.03	2014-01-01 02:00:00	222	
3	13.31	0.71	10.00	1021.64	4.00	0.14	2014-01-01 03:00:00	209	
4	13.57	0.71	9.93	1020.73	3.67	0.04	2014-01-01 04:00:00	217	
...	...	...	...	...	...	...	...	...	
8755	27.48	0.35	10.00	1023.54	10.54	0.24	2014-12-31 19:00:00	311	
8756	27.17	0.35	10.00	1023.60	9.53	0.25	2014-12-31 20:00:00	297	
8757	25.72	0.37	10.00	1023.44	8.12	0.08	2014-12-31 21:00:00	292	
8758	22.75	0.42	10.00	1023.29	4.43	0.05	2014-12-31 22:00:00	299	
8759	20.09	0.51	10.00	1023.18	1.33	0.11	2014-12-31 23:00:00	275	

8760 rows × 21 columns

In [ ]:

In [104]:

```
#train the model
test = merged.query("time >= '2014-12-01'" )
train = merged.query("time < '2014-12-01'" )

train = train.dropna()
test = test.dropna()

x_train = train.drop(columns=['time', 'use [kW]', 'high/low'])
y_train = train['high/low']

x_test = test.drop(columns=['time', 'use [kW]','high/low'])
y_test = test['high/low']

logistic_regressor = LogisticRegression()  # create object for the class
logistic_regressor.fit(x_train, y_train)  # perform linear regression

Y_pred = logistic_regressor.predict(x_test)  # make predictions
```

```
C:\Users\ukolv\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Co
nvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-lear
n.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (h
ttps://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
```

In [105]:

```
Y_pred
```

Out[105]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [106]:

```
my_f1_score = f1_score(y_test, Y_pred)
```

In [107]:

```
my_f1_score
```

Out[107]:

```
0.9832935560859188
```

In [108]:

```
energy_data['Date & Time'].dt.hour #if hr >= 6 &&
```

Out[108]:

```
0         0
1         0
2         1
3         1
4         2
         ..
17515    21
17516    22
17517    22
17518    23
17519    23
Name: Date & Time, Length: 17520, dtype: int64
```

In [109]:

```
classification_df = pd.DataFrame({'date':test.time, 'prediction':Y_pred})
```

In [110]:

```
classification_df
```

Out[110]:

	date	prediction
8016	2014-12-01 00:00:00	1
8022	2014-12-01 06:00:00	1
8024	2014-12-01 08:00:00	1
8025	2014-12-01 09:00:00	1
8026	2014-12-01 10:00:00	1
...	...	...
8755	2014-12-31 19:00:00	0
8756	2014-12-31 20:00:00	0
8757	2014-12-31 21:00:00	0
8758	2014-12-31 22:00:00	0
8759	2014-12-31 23:00:00	0

500 rows × 2 columns

In [111]:

```
classification_df.to_csv("cse351_hw2_Shvartsman_Terrence_114311609_logistic_regression.csv", index
```

Q5

In [112]:

```
['time_of_day'] = np.where((energy_data['Date & Time'].dt.hour >= 6) & (energy_data['Date & Time'].
```

In [113]:

```
energy_data
```

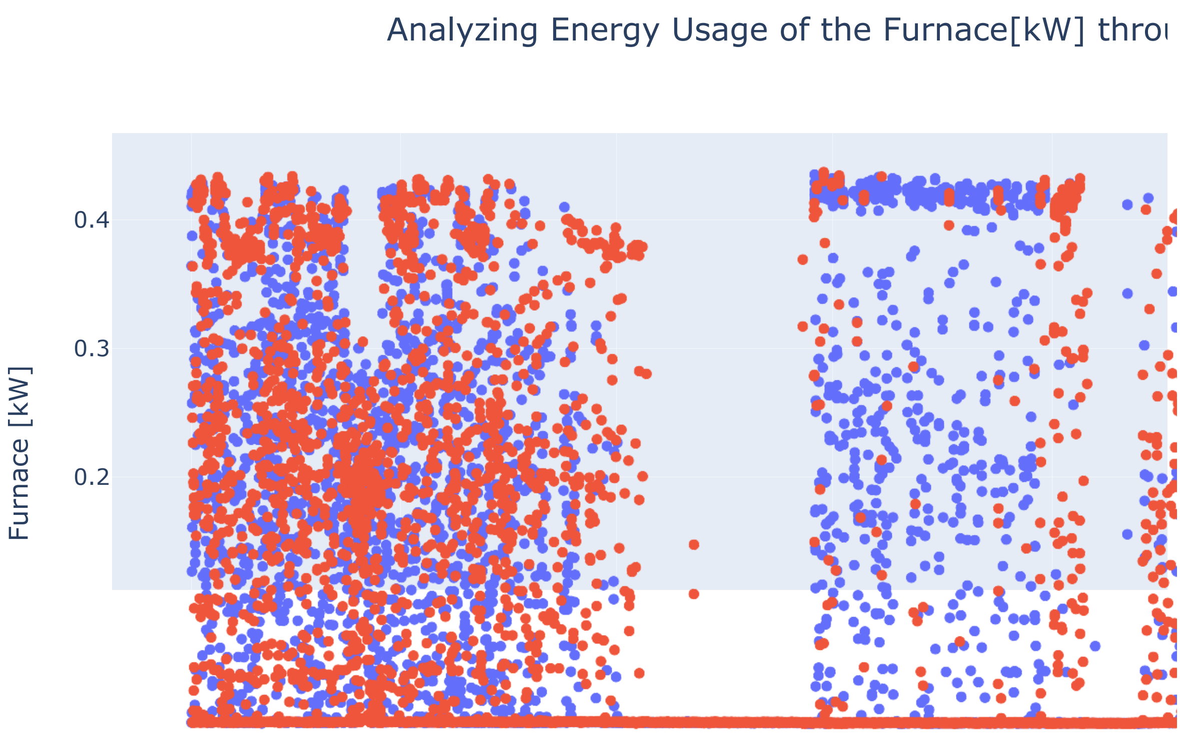
Out[113]:

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Utility Rm + Basement Bath [kW]
0	2014-01-01 00:00:00	0.304439	0.0	0.304439	0.000058	0.009531	0.005336	0.000126	0.011175	0.003836
1	2014-01-01 00:30:00	0.656771	0.0	0.656771	0.001534	0.364338	0.005522	0.000043	0.003514	0.003512
2	2014-01-01 01:00:00	0.612895	0.0	0.612895	0.001847	0.417989	0.005504	0.000044	0.003528	0.003484
3	2014-01-01 01:30:00	0.683979	0.0	0.683979	0.001744	0.410653	0.005556	0.000059	0.003499	0.003476
4	2014-01-01 02:00:00	0.197809	0.0	0.197809	0.000030	0.017152	0.005302	0.000119	0.003694	0.003865
...	...	...	...	...	...	...	...	...	...	...
17515	2014-12-31 21:30:00	1.560890	0.0	1.560890	0.003226	0.392996	0.006342	0.000872	0.030453	0.002248
17516	2014-12-31 22:00:00	0.958447	0.0	0.958447	0.000827	0.027369	0.006326	0.000811	0.030391	0.002543
17517	2014-12-31 22:30:00	0.834462	0.0	0.834462	0.001438	0.170561	0.020708	0.000636	0.012631	0.002372
17518	2014-12-31 23:00:00	0.543863	0.0	0.543863	0.001164	0.153533	0.008423	0.000553	0.003832	0.002353
17519	2014-12-31 23:30:00	0.414441	0.0	0.414441	0.000276	0.009223	0.006619	0.000526	0.003818	0.002424

17520 rows × 19 columns

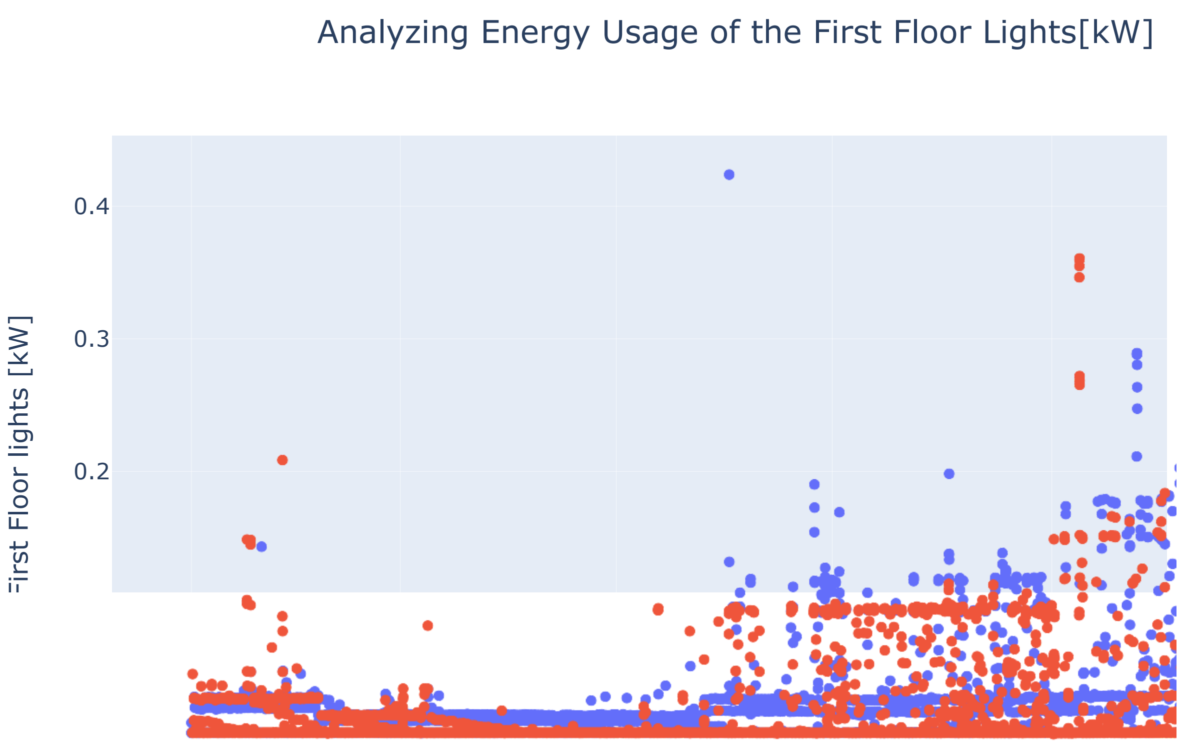
In [114]:

```
ergy_data, x='Date & Time', y='Furnace [kW]', color = 'time_of_day', title="Analyzing Energy Usage
axis_title='Time', title_x=0.5)
```



In [115]:

```
#select a device and then plot it
first_floor_plot = px.scatter(energy_data, x='Date & Time', y='First Floor lights [kW]', color = '
first_floor_plot.update_layout(xaxis_title='Time', title_x=0.5)
first_floor_plot.show()
```



I think that its very interesting that there is at large number of points in both plots that are near the bottom. Furthermore, it is interesting that all of these points are during the day. However, upon consideration, this makes sense since most people are out during the day so most appliances are either off or barely used during the day. Of course this isnt always true as we have certain points that are outliers in the y axis, meaning they use an abnormal amount of kW. However, this also makes sense since we define day to be 6am-7pm and most people are still home at 6am and get back home around 5pm, leaving 2 hours from 5-7 where a lot of energy would be used.

The most interesting thing about these two graphs, though, is how the first floor light usage vs time is a constant nearly bell plot shaped curve, while the furnace vs time graph has these rectangular clumps. This makes sense as during the cold months the furnace would be used throughout the entire day and month to keep the houses warm. As such, there is a large concentration of points ranging from 0kW to half a kW from about October to April. Then from May to June there is nearly no usage at all. I am not entirely sure what the reason behind the points in July-September is. However, due to it being almost entirely "night" points, it might be relatively cold at night, or at least cold enough to require the furnace to be used somewhat.

In [ ]:

```
```

In [ ]:

```
```