Getting Started

The goal of this assignment is to develop a method to predict the electricity load demand of 3 individual users. For each user, we are given following 2 datasets which cover 1 calendar year:

- Energy usage history (in kW) with 30-minute or 1-minute interval
- Weather history with 1-hour interval

Our objective is to create models that predict the future electricity consumption of these customers and measure the accuracy of your predictions by the Mean Absolute Error.

First and foremost, lets import the libraries and & datasets.

In [1]:
```python
import pandas as pd
import numpy as np
import datetime
from datetime import timedelta
from IPython.core.display import display
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.arima_model import ARIMA
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import ShuffleSplit
from sklearn import preprocessing


# Loading the data from csv files

energyB = pd.read_csv('./data/HomeB-meter1_2014.csv')
energyC = pd.read_csv('./data/HomeC-meter1_2016.csv')
energyF = pd.read_csv('./data/HomeF-meter3_2016.csv')
weatherB = pd.read_csv('./data/homeB2014.csv')
weatherC = pd.read_csv('./data/homeC2016.csv')
weatherF = pd.read_csv('./data/homeF2016.csv')
```

Data Exploration

In [2]:
```
display(weatherB.head())
display(energyB)
```

	temperature	icon	humidity	visibility	summary	apparentTemperature	pressure	windSpeed
0	34.98	partly-cloudy-night	0.64	10.00	Partly Cloudy	28.62	1017.69	7.75
1	16.49	clear-night	0.62	10.00	Clear	16.49	1022.76	2.71
2	14.63	clear-night	0.68	10.00	Clear	6.87	1022.32	4.84
3	13.31	clear-night	0.71	10.00	Clear	6.49	1021.64	4.00
4	13.57	clear-night	0.71	9.93	Clear	7.29	1020.73	3.67

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Uti Ba: Ba
0	2014-01-01 00:00:00	0.304439	0.0	0.304439	0.000058	0.009531	0.005336	0.000126	0.011175	0.
1	2014-01-01 00:30:00	0.656771	0.0	0.656771	0.001534	0.364338	0.005522	0.000043	0.003514	0.
2	2014-01-01 01:00:00	0.612895	0.0	0.612895	0.001847	0.417989	0.005504	0.000044	0.003528	0.
3	2014-01-01 01:30:00	0.683979	0.0	0.683979	0.001744	0.410653	0.005556	0.000059	0.003499	0.
4	2014-01-01 02:00:00	0.197809	0.0	0.197809	0.000030	0.017152	0.005302	0.000119	0.003694	0.
...	...	...	...	...	...	...	...	...	...	
17515	2014-12-31 21:30:00	1.560890	0.0	1.560890	0.003226	0.392996	0.006342	0.000872	0.030453	0.
17516	2014-12-31 22:00:00	0.958447	0.0	0.958447	0.000827	0.027369	0.006326	0.000811	0.030391	0.
17517	2014-12-31 22:30:00	0.834462	0.0	0.834462	0.001438	0.170561	0.020708	0.000636	0.012631	0.

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Uti Ba: Ba
17518	2014-12-31 23:00:00	0.543863	0.0	0.543863	0.001164	0.153533	0.008423	0.000553	0.003832	0.
17519	2014-12-31 23:30:00	0.414441	0.0	0.414441	0.000276	0.009223	0.006619	0.000526	0.003818	0.

17520 rows × 18 columns

In [3]:
```
energyB.describe()
```

Out[3]:

	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW
count	17520.000000	17520.0	17520.000000	17520.000000	17520.000000	17520.000000	17520.00000
mean	0.662905	0.0	0.662905	0.088999	0.085888	0.011036	0.00306
std	0.678399	0.0	0.678399	0.438887	0.129054	0.013123	0.02044
min	0.011083	0.0	0.011083	0.000000	0.000117	0.000083	0.00000
25%	0.314125	0.0	0.314125	0.000030	0.009340	0.005414	0.00009
50%	0.468725	0.0	0.468725	0.000069	0.009704	0.005881	0.00021
75%	0.700617	0.0	0.700617	0.000707	0.143531	0.007042	0.00033
max	6.833205	0.0	6.833205	3.687768	0.437212	0.146692	0.81916

In [4]:
```
weatherB.describe()
```

Out[4]:

	temperature	humidity	visibility	apparentTemperature	pressure	windSpeed	clo
count	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	729
mean	48.062076	0.682888	9.025791	45.289160	1016.450749	6.534568	
std	19.694743	0.188763	1.859263	22.860668	7.903670	3.884500	
min	-10.070000	0.140000	0.320000	-18.280000	979.980000	0.030000	
25%	33.165000	0.530000	9.040000	27.967500	1011.530000	3.630000	
50%	49.220000	0.710000	9.970000	47.360000	1016.430000	5.850000	
75%	63.832500	0.860000	10.000000	63.832500	1021.310000	8.692500	
max	89.460000	0.960000	10.000000	97.520000	1042.400000	24.750000	

We picked up House B data to analyse it further. The weather data set has different columns

specifiying the parameters and their corresponding values.
We are particularly interested in the count.

For house B we have **8760 data points** on weather which corresspond to hourly data for a year. **(365 days * 24)**. While, for energy usage , we have 17520 data points which correspond to bihourly data points for the year. (365 days * 24 hours * 2 (half hour).
We need to normalise the data to bring it in the same order and we shall address these in the next section.

In [5]:
```
energyC.tail()
```

Out[5]:

	Date & Time	use [kW]	gen [kW]	House overall [kW]	Dishwasher [kW]	Furnace 1 [kW]	Furnace 2 [kW]	Home office [kW]	Fridge [kW
503905	2016-12-15 22:25:00	1.601233	0.003183	1.601233	0.000050	0.085267	0.642417	0.041783	0.00526
503906	2016-12-15 22:26:00	1.599333	0.003233	1.599333	0.000050	0.104017	0.625033	0.041750	0.00523
503907	2016-12-15 22:27:00	1.924267	0.003217	1.924267	0.000033	0.422383	0.637733	0.042033	0.00498
503908	2016-12-15 22:28:00	1.978200	0.003217	1.978200	0.000050	0.495667	0.620367	0.042100	0.00533
503909	2016-12-15 22:29:00	1.990950	0.003233	1.990950	0.000050	0.494700	0.634133	0.042100	0.00491

In [6]:
```
energyF.tail()
```

Out[6]:

	Date & Time	Usage [kW]	Generation [kW]	Net_Meter [kW]	Volt [kW]	Garage_E [kW]	Garage_W [kW]	Phase_A [kW]	Phas [
503920	2016-12-15 22:40:00	0.643783	0.009100	0.652883	0.002200	0.000000	0.000350	0.490050	0.15
503921	2016-12-15 22:41:00	1.135383	0.009117	1.144500	0.002200	0.000000	0.000350	0.492050	0.64
503922	2016-12-15 22:42:00	1.395117	0.008150	1.403267	0.002200	0.000017	0.000350	0.427983	0.96
503923	2016-12-15 22:43:00	0.624050	0.007933	0.631983	0.002200	0.000000	0.000367	0.474950	0.14
503924	2016-12-15 22:44:00	0.597250	0.006600	0.603850	0.002167	0.000000	0.000350	0.454467	0.14

For House B we have energy usage data till 31st December, where as for House C & F, we have data points only till 15th Dec. Hence, as per assignment sepcification, as we are using data till 30th November as training set for House B and till 15th Nov for House C and F. This gives us effectively roughly 30days datapoints for testing dataset for the predictions.

Data Preprocessing

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as **preprocessing**. There are some qualities about certain features that must be adjusted before we go ahead. This preprocessing can help tremendously with the outcome and predictive power of nearly all learning algorithms.

For the purposes of this project, the following preprocessing steps have been made to the dataset:

1. The column 'Date & Time' needs to be converted to datetime data type. This would help in operations over the data.

In [7]:
```
# Convert to Date Time Format
energyB['Date & Time'] = pd.to_datetime(energyB['Date & Time'])
energyC['Date & Time'] = pd.to_datetime(energyC['Date & Time'])
energyF['Date & Time'] = pd.to_datetime(energyF['Date & Time'])
```

2. We need to change the column name of energyF data frame from 'Usage [kW]' to 'use [kW]' for consistency across different datasets. This would help in accessing the datasets in uniform manner for referring the columns.

In [8]:
```
# Changing column name
energyF = energyF.rename(columns={'Usage [kW]': 'use [kW]'})
```

3. Cardinality of Datasets

As observed above, we need to bring the features and target datasets to same order so as to map them. Lets check the shape of the data.

In [9]:
```
print("Shape of Data for House B (Energy Usage):", energyB.shape)
print("Shape of Data for House B (Weather):", weatherB.shape)
print("Shape of Data for House C (Energy Usage):", energyC.shape)
print("Shape of Data for House C (Weather):", weatherC.shape)
print("Shape of Data for House F (Energy Usage):", energyF.shape)
print("Shape of Data for House F (Weather):", weatherF.shape)
```

```
Shape of Data for House B (Energy Usage): (17520, 18)
Shape of Data for House B (Weather): (8760, 14)
Shape of Data for House C (Energy Usage): (503910, 19)
Shape of Data for House C (Weather): (8760, 14)
Shape of Data for House F (Energy Usage): (503925, 10)
Shape of Data for House F (Weather): (8760, 14)
```

Hence we see that the weather data available is hourly data,(i.e. number of rows = 365 * 24) but the energy data is bihourly for House C and Minute Level for House C & F.

Preparing Hourly and Daily timeline datasets

The count of the above datasets show. We need to convert the number of rows in Weather data corresponds to number of hours in a year (365*24 = 8760).

Hourly

To prepare Hously data on energy usage, we need to merge the bihourly (House B)or minute level (House C & F) datapoints.

Daily

To convert to daily data, we round the dates to their corresponding days and then group by on date.

For **Energy usage data**, we take the **sum** of each cell values, since energy usage for hour would be sum of usages of every minute in that hour.
For **weather data** values, we take the **median on group by instead of mean, because it isn't influenced by extremely large values or outliers** . With these basis we group the data and prepare the features and target datasets.

In [10]:
```
# Merge energy bihourly data into hourly data
energyB['Date & Time'] = energyB["Date & Time"].dt.floor('H')
energyC['Date & Time'] = energyC["Date & Time"].dt.floor('H')
energyF['Date & Time'] = energyF["Date & Time"].dt.floor('H')
energy_hourly_B = energyB.groupby('Date & Time').sum().reset_index()
energy_hourly_C = energyC.groupby('Date & Time').sum().reset_index()
energy_hourly_F = energyF.groupby('Date & Time').sum().reset_index()
```

In [11]:
```
energyB['Date & Time'] = energyB["Date & Time"].dt.floor('D')
energyC['Date & Time'] = energyC["Date & Time"].dt.floor('D')
energyF['Date & Time'] = energyF["Date & Time"].dt.floor('D')
energy_daily_B = energyB.groupby('Date & Time').sum().reset_index()
energy_daily_C = energyC.groupby('Date & Time').sum().reset_index()
energy_daily_F = energyF.groupby('Date & Time').sum().reset_index()
```

In [12]:
```
def getDateTime(inputYear, df):
    currenttime = datetime.datetime(inputYear,1,1,0,0,0)
    datearr = []
    for index, row in df.iterrows():
        datearr.append(currenttime)
        #add delta of one hour to each iteration
        currenttime = currenttime + timedelta(hours=1)
    return datearr

weatherB['Date & Time'] =  getDateTime(2014, weatherB)
weatherC['Date & Time'] =  getDateTime(2016, weatherC)
weatherF['Date & Time'] =  getDateTime(2016, weatherF)
weather_hourly_B = weatherB.copy()
weather_hourly_C = weatherC.copy()
weather_hourly_F = weatherF.copy()

weatherB['Date & Time'] = weatherB["Date & Time"].dt.floor('D')
weatherC['Date & Time'] = weatherC["Date & Time"].dt.floor('D')
weatherF['Date & Time'] = weatherF["Date & Time"].dt.floor('D')
weather_daily_B = weatherB.groupby('Date & Time').median().reset_index()
weather_daily_C = weatherC.groupby('Date & Time').median().reset_index()
weather_daily_F = weatherF.groupby('Date & Time').median().reset_index()
```

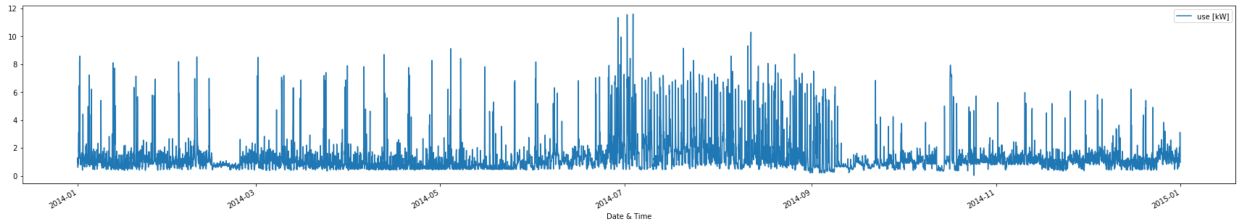
In [13]:
```
weatherData=[weather_hourly_B, weather_hourly_C, weather_hourly_F, weather_
energyData = [energy_hourly_B, energy_hourly_C, energy_hourly_F, energy_dai
tag = ['House B (Hourly)','House C (Hourly)','House F (Hourly)','House B (D

models=[]
maeBHourly=[]
maeCHourly=[]
maeFHourly=[]
maeBDaily=[]
maeCDaily=[]
maeFDaily=[]
aggregate=[maeBHourly,maeCHourly,maeFHourly,maeBDaily,maeCDaily,maeFDaily]
```

Data Visualisation

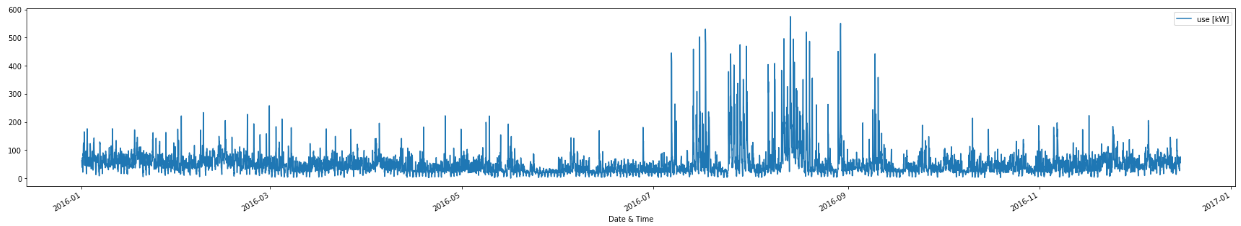
Let's visualise the different aggregates of the energy data and see what insights we can get from it.

Hourly Plots

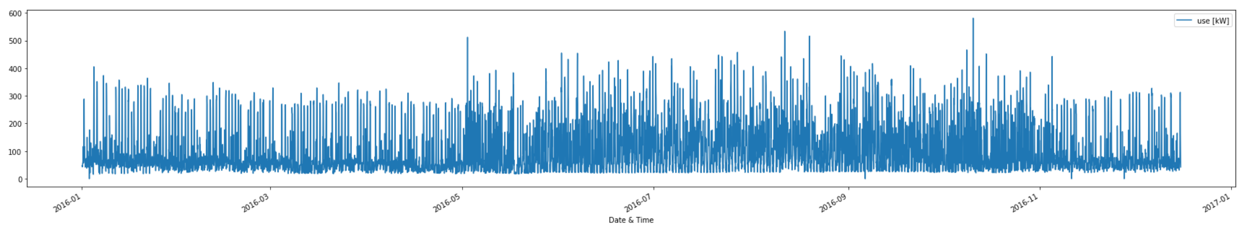
In [14]:
```
# Plot graphs for both the diagrams
energy_hourly_B.plot(kind='line', x='Date & Time', y='use [kW]', figsize=(3
plt.show()
```



In [15]:
```
energy_hourly_C.plot(kind='line', x='Date & Time', y='use [kW]', figsize=(3
plt.show()
```

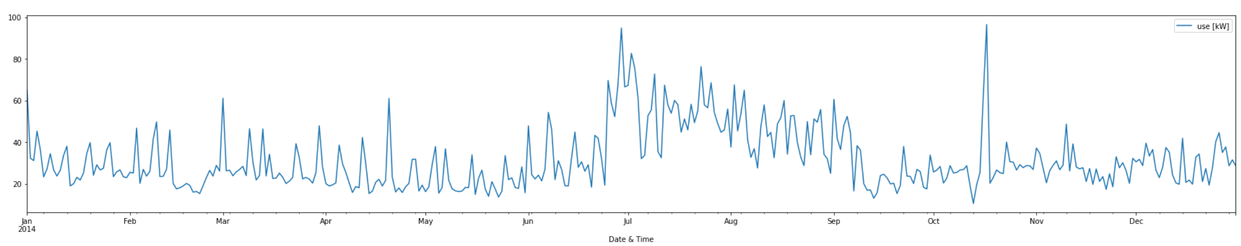


In [16]:
```
energy_hourly_F.plot(kind='line', x='Date & Time', y='use [kW]', figsize=(3
plt.show()
```

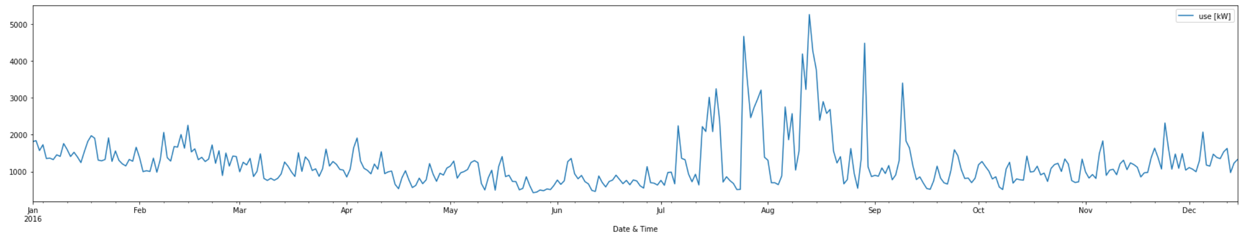


Daily Plots

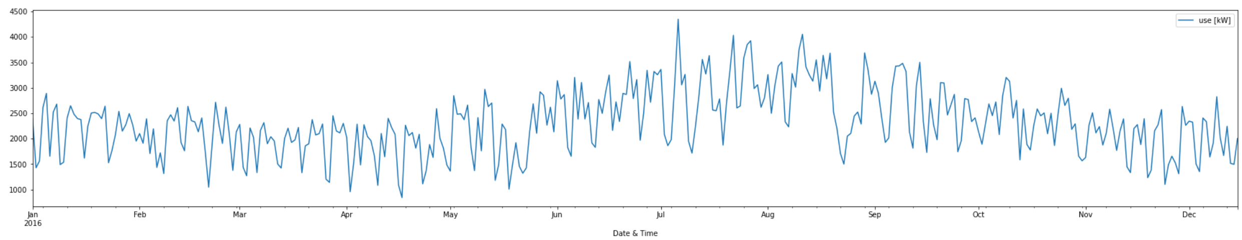
In [17]:
```
energy_daily_B.plot(kind='line', x='Date & Time', y='use [kW]', figsize=(30
plt.show()
```



In [18]:
```
energy_daily_C.plot(kind='line', x='Date & Time', y='use [kW]', figsize=(30
plt.show()
```



In [19]:
```
energy_daily_F.plot(kind='line', x='Date & Time', y='use [kW]', figsize=(30
plt.show()
```

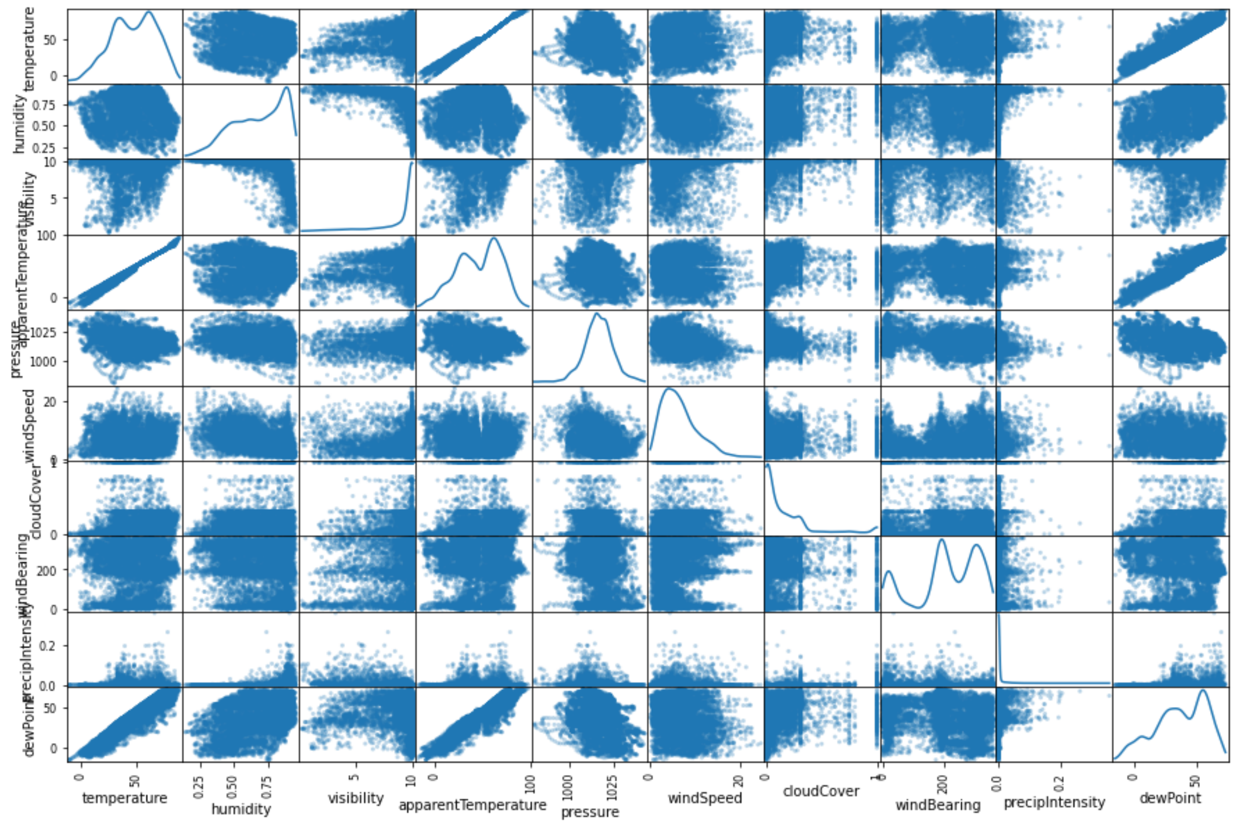


From the above visualisations we can clearly see, as for House B & C , we observe significant spike during the late summer months. This might me due to excessive usage of cooling appliances. House F on the other hand has high energy usage (High avg mean) all the year round. Probably House F would be a data of a bigger house or public place or any kind of hotels.

Visualizing Feature Distributions

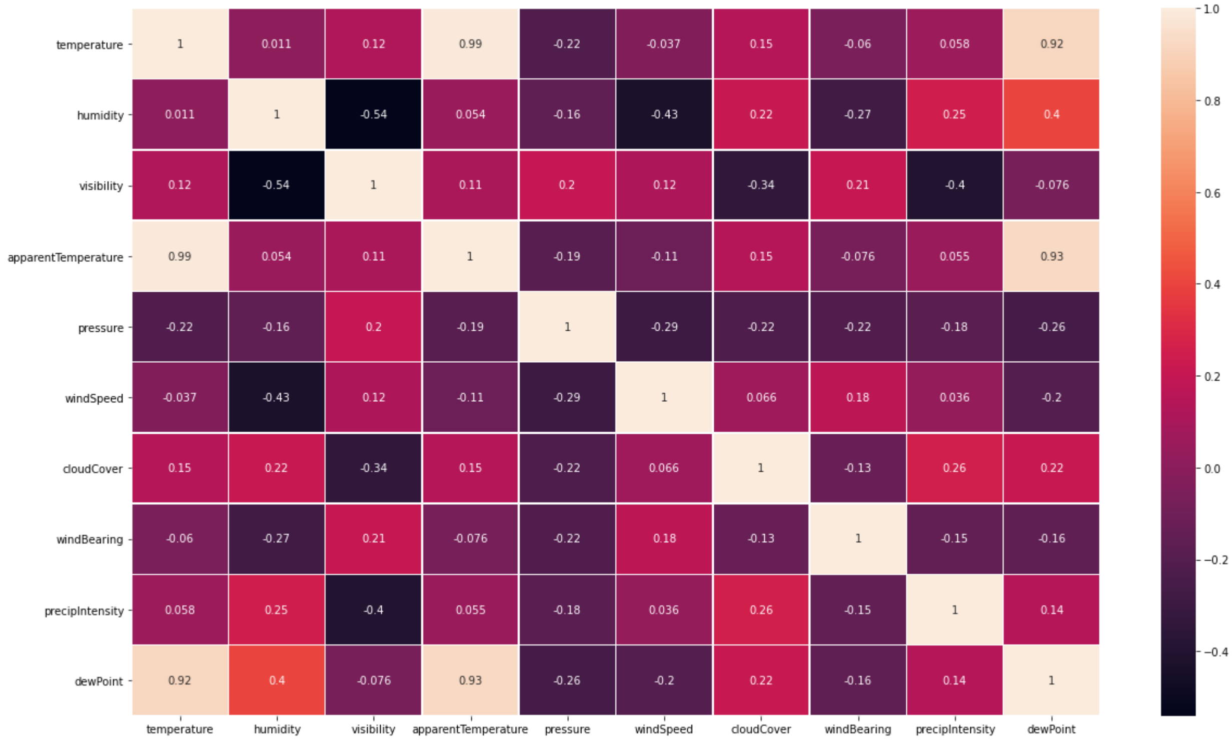
To get a better understanding of the dataset, we can construct a scatter matrix of each numerical features present in the weather dataset. If we believe that any feature is correlated with another features, it might be relevant for identifying the energy demand and the scatter matrix might show a correlation between that feature and another feature in the dataset.

In [20]:
```
from pandas.plotting import scatter_matrix

data = weather_hourly_B[['temperature','humidity','visibility','apparentTem
scatter_matrix(data, alpha = 0.3, figsize = (15,10), diagonal = 'kde');
```



From the above diagram, from the data points in the scatter plot we can clearly see that **Temperature, Apparent Temperature and Dew point** are the most tightly clustered along an imaginary line, hence they have some positive correlation with each other. To further accentuate out observation, lets compute pairwise correlation of columns and plot it on heat map.

In [21]:
```
feature_corr = data.corr()
fig, ax = plt.subplots(figsize=(20,12))
sns.heatmap(feature_corr, annot=True, linewidths=.5)
plt.show()
```



The high correlation values of **Temperature**, **Apparent Temperature** and **Dew point** shows that they are closely amongst themselves. Hence as part of training & prediction, we can use amongst these to train and test models on the dataset. Since apparent temperature has a close synergy with Temperature feture, we shall use **temperature and dewPoint** to train our models.

In [22]:
```
linkedFeatures = ['temperature', 'dewPoint']
```

Now lets try and create models to predict the future electricity consumption of these houses on hourly and daily data points.

Training & Prediction

Now that our data-preprocessing is done, we will use the datasets to train models, predict the energy usage and calculate the accuracy in terms of Mean Absolute Error.

We are using techniques like **Naive Method, Linear Regression, Decision tree Regression, Random Forest and ARIMA Model** to produce forecasts. For parameter tuning, I am using **Grid Search** Method to tune the hyperparameters.

As per the requirement specification, I am splitting the data set into two parts based on the date. I am using Jan to Nov data as the training set and Dec datapoints to test and calculate the MAE.

Naive Method Prediction

Naive Method Prediction

Naive method is an estimating technique in which the last period's actual values are used to predict the future forecast, without adjusting them or attempting to establish causal factors. It is here used here as a baseline to compare forecasts generated by the better (sophisticated) techniques.

As one can see below, we are preparing test and train dataset based on split on date. For House B, we use training data from 1st Jan to 2014-12-01 00:00:00. For House C & F, We use training data from 1st Jan to "2016-11-15 00:00:00", since this would gave 30days data to test in both the cases. Since we want to compare amongst similar data, we are storing the split points and using the same splits to test other algorithms so that we can compare them with Naive method.

In [23]:

```
# NAIVE Method Prediction
def naiveMethodPredictor(dataset, split):
    train_Y = dataset[:split]
    test_Y = dataset[split:]
    count = len(test_Y)
    predict_Y = pd.DataFrame(np.asarray(train_Y)[len(train_Y) - 1][0], inde
    mae = mean_absolute_error(test_Y, predict_Y)
    return mae
```

In [24]:
```
models.append("Naive Method")

splitHB = len(energy_hourly_B.loc[energy_hourly_B['Date & Time'] <= '2014-1
maeVal = naiveMethodPredictor(energy_hourly_B[['use [kW]']], splitHB)
maeBHourly.append(maeVal)
print("The Mean Absolute Error of Naive method for Hourly points on House B

splitHC = len(energy_hourly_C.loc[energy_hourly_C['Date & Time'] <= '2016-1
maeVal = naiveMethodPredictor(energy_hourly_C[['use [kW]']], splitHC)
maeCHourly.append(maeVal)
print("The Mean Absolute Error of Naive method for Hourly points on House C

splitHF = len(energy_hourly_F.loc[energy_hourly_F['Date & Time'] <= '2016-1
maeVal = naiveMethodPredictor(energy_hourly_F[['use [kW]']], splitHF)
maeFHourly.append(maeVal)
print("The Mean Absolute Error of Naive method for Hourly points on House F

splitDB = len(energy_daily_B.loc[energy_daily_B['Date & Time'] <= '2014-12-
maeVal = naiveMethodPredictor(energy_daily_B[['use [kW]']], splitDB)
maeBDaily.append(maeVal)
print("The Mean Absolute Error of Naive method for Daily points on House B

splitDC = len(energy_daily_C.loc[energy_daily_C['Date & Time'] <= '2016-11-
maeVal = naiveMethodPredictor(energy_daily_C[['use [kW]']], splitDC)
maeCDaily.append(maeVal)
print("The Mean Absolute Error of Naive method for Daily points on House C

splitDF = len(energy_daily_F.loc[energy_daily_F['Date & Time'] <= '2016-11-
maeVal = naiveMethodPredictor(energy_daily_F[['use [kW]']], splitDF)
maeFDaily.append(maeVal)
print("The Mean Absolute Error of Naive method for Daily points on House F
```

```
The Mean Absolute Error of Naive method for Hourly points on House B is :
0.4795294990740242
The Mean Absolute Error of Naive method for Hourly points on House C is :
25.24953158025876
The Mean Absolute Error of Naive method for Hourly points on House F is :
47.94122041828032
The Mean Absolute Error of Naive method for Daily points on House B is :
6.231544518466666
The Mean Absolute Error of Naive method for Daily points on House C is :
252.46967000100022
The Mean Absolute Error of Naive method for Daily points on House F is :
440.9711255574666
```

In [25]:
```
splitPoints=[]
splitPoints.append(splitHB)
splitPoints.append(splitHC)
splitPoints.append(splitHF)
splitPoints.append(splitDB)
splitPoints.append(splitDC)
splitPoints.append(splitDF)
```

In an attempt to compare apples to apples, we are memoizing the split points and use the same to test and train different models.

In [26]:
```
from matplotlib.pyplot import figure
figure(num=None, figsize=(10, 30), dpi=200, facecolor='w', edgecolor='k')

def plotvalues(x, y1, y2, house, model1, model2, interval):
    plt.title('Time Series plot at: ' + house + " " + interval)
    plt.xlabel('Date & Time')
    plt.ylabel('Use [kWh]')
    plt.plot(x, y1,'b', label=model1)
    plt.plot(x, y2,'r', label=model2)
    plt.legend(loc='upper left')
    plt.show()
```

```
<Figure size 2000x6000 with 0 Axes>
```

Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. We are using the the features which we obtained from feature distribution. I am using the split points I extracted above to prepare training and testing datasets.

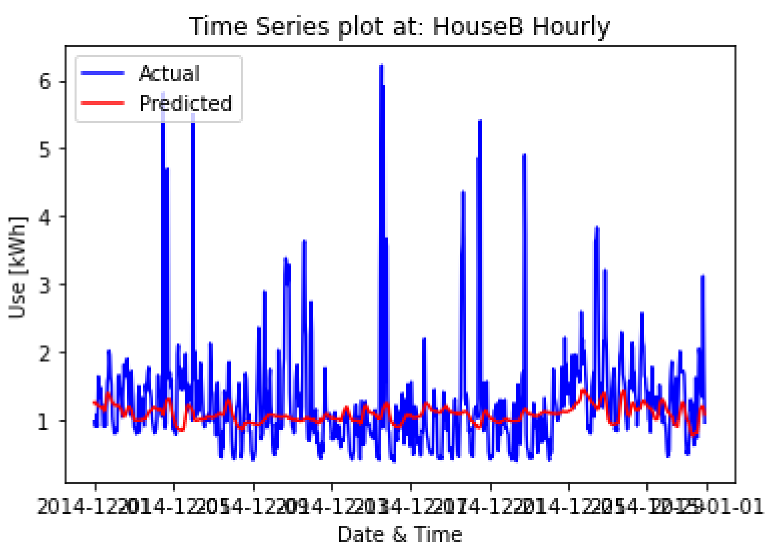
For training features, I am using the temperature and dewpoint for training and predict energy usages based on it. Then I have plots of graphs for the actual energy usage values vs the predicted values.

Normalising the features

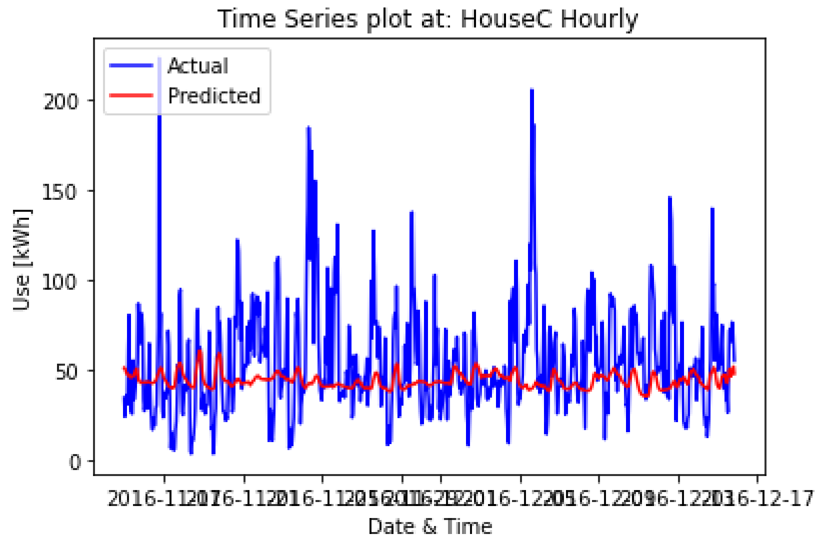
Since we will be using the features like Temperature and Dew Point, the values of these features are in different magnitude. For example for House B weather data, as seen above, (temperature has max value of 89 and min value of -10) and dew point varies from (-15 to 72). Hence we are normailising the features datatset to bring them into same order.

In [27]:
```
# Train Linear Regression model and predict

def LinearRegressionModel(features, target, split, house, interval):
    df = features[linkedFeatures].reset_index(drop=True)
    min_max_scaler = preprocessing.MinMaxScaler()
    scaled = min_max_scaler.fit_transform(df.values)
    scaled = pd.DataFrame(scaled, columns=df.columns, index=df.index)
    xtrain = scaled[:split].reset_index(drop=True)

    ytrain = target[:split][['use [kW]']].reset_index(drop=True)
    ytest = target[split:][['use [kW]']].reset_index(drop=True)
    xtest = scaled[split : (split + len(ytest))].reset_index(drop=True)
    linear_regressor = LinearRegression()
    model = linear_regressor.fit(xtrain, ytrain)
    ypredict = linear_regressor.predict(xtest)
    mae = mean_absolute_error(ytest, ypredict)
    plotvalues(features[split : (split + len(ytest))][["Date & Time"]], yte
    return mae
```

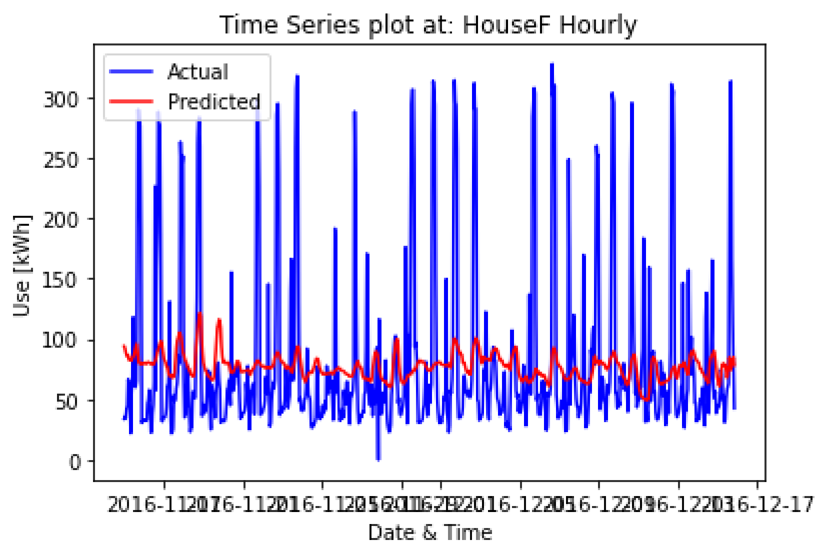
In [28]:
```
models.append("Linear Regression")

maeVal = LinearRegressionModel(weather_hourly_B, energy_hourly_B, splitHB,
maeBHourly.append(maeVal)
print("The Mean Absolute Error for Linear Regression for hourly points on H

maeVal = LinearRegressionModel(weather_hourly_C, energy_hourly_C, splitHC,
maeCHourly.append(maeVal)
print("The Mean Absolute Error for Linear Regression for hourly points on H

maeVal = LinearRegressionModel(weather_hourly_F, energy_hourly_F, splitHF,
maeFHourly.append(maeVal)
print("The Mean Absolute Error for Linear Regression for hourly points on H

maeVal = LinearRegressionModel(weather_daily_B, energy_daily_B, splitDB, "H
maeBDaily.append(maeVal)
print("The Mean Absolute Error for Linear Regression for daily points on Ho

maeVal = LinearRegressionModel(weather_daily_C, energy_daily_C, splitDC, "H
maeCDaily.append(maeVal)
print("The Mean Absolute Error for Linear Regression for daily points on Ho

maeVal = LinearRegressionModel(weather_daily_F, energy_daily_F, splitDF, "H
maeFDaily.append(maeVal)
print("The Mean Absolute Error for Linear Regression for daily points on Ho
```



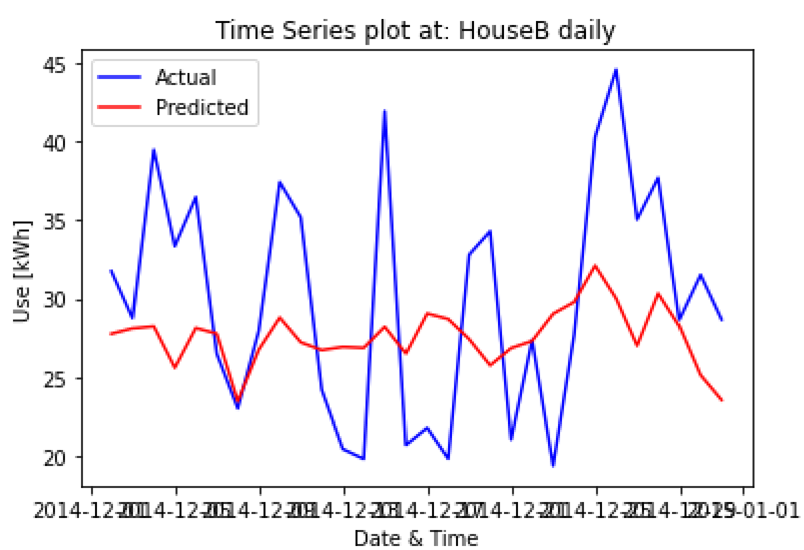

```
The Mean Absolute Error for Linear Regression for hourly points on House
B is 0.4637947126337114
```




```
The Mean Absolute Error for Linear Regression for hourly points on House
C is 21.191972179063775
```

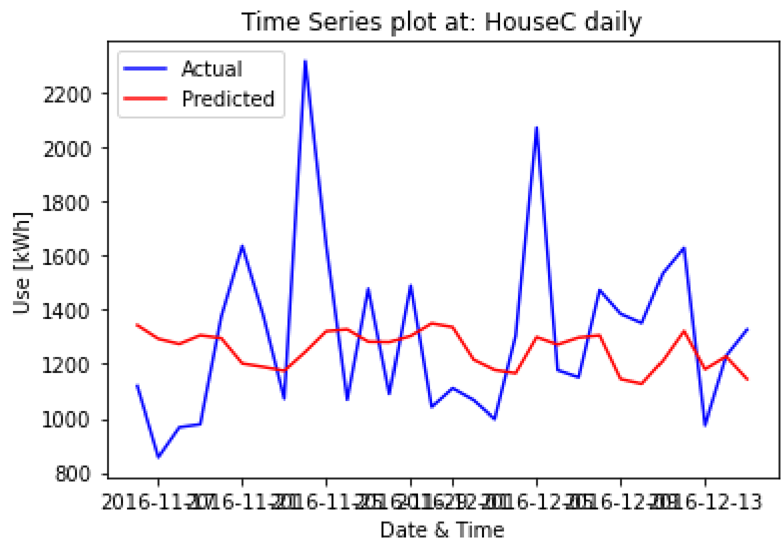



```
The Mean Absolute Error for Linear Regression for hourly points on House
F is 45.50591348229302
```

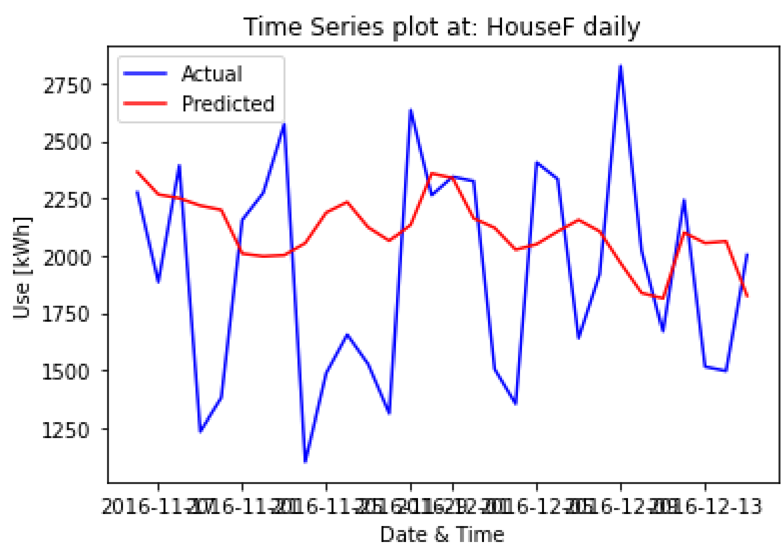



```
The Mean Absolute Error for Linear Regression for daily points on House B
```

```
is 6.157594756866734
```




```
The Mean Absolute Error for Linear Regression for daily points on House C
is 266.6723313494483
```




```
The Mean Absolute Error for Linear Regression for daily points on House F
is 431.7812314287284
```

Decision Tree Regressor

A decision tree is a supervised machine learning model used to predict a target by learning decision rules from features. As the name suggests, we can think of this model as breaking down our data by making a decision based on asking a series of questions. Initially we split the data based on the split points to prepare training and testing datasets and then we train models to get predicted values. We run it on default parameters in the first attemp. Following that , we try to improve the MAE by hyperparameter tuning using Grid Search as shown below.

In [29]:
```
def DecisionTree(features, target, split):
    xtrain = features[:split][linkedFeatures].reset_index(drop=True)
    ytrain = target[:split][['use [kW]']].reset_index(drop=True)
    ytest = target[split:][['use [kW]']].reset_index(drop=True)
    xtest = features[split : (split + len(ytest))][linkedFeatures].reset_in
    regressor = DecisionTreeRegressor(random_state=0)
    model = regressor.fit(xtrain, ytrain.values.ravel())
    mae = mean_absolute_error(ytest, model.predict(xtest))
    return mae
```

Hourly & Daily MAE for Houses B, C, & F

In [30]:
```
models.append("Decision Tree (Without Grid Search)")

maeVal = DecisionTree(weather_hourly_B, energy_hourly_B, splitHB)
maeBHourly.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for hourly points on

maeVal = DecisionTree(weather_hourly_C, energy_hourly_C, splitHC)
maeCHourly.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for hourly points on

maeVal = DecisionTree(weather_hourly_F, energy_hourly_F, splitHF)
maeFHourly.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for hourly points on

maeVal = DecisionTree(weather_daily_B, energy_daily_B, splitDB)
maeBDaily.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for daily points on H

maeVal = DecisionTree(weather_daily_C, energy_daily_C, splitDC)
maeCDaily.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for daily points on H

maeVal = DecisionTree(weather_daily_F, energy_daily_F, splitDF)
maeFDaily.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for daily points on H
```

```
The Mean Absolute Error for DecisionTree Model for hourly points on House
B is 0.6952422633539705
The Mean Absolute Error for DecisionTree Model for hourly points on House
C is 31.51969418227089
The Mean Absolute Error for DecisionTree Model for hourly points on House
F is 63.34120420022507
The Mean Absolute Error for DecisionTree Model for daily points on House
B is 9.442799666933333
The Mean Absolute Error for DecisionTree Model for daily points on House
C is 333.6515294442667
The Mean Absolute Error for DecisionTree Model for daily points on House
F is 430.68034778346725
```

Grid Search | Parameter Tuning & Optimising Model Performance

Grid search is an algorithm with which we tune hyperparameters (example max_depth in our case) of our model. The input to grid search is possible values of hyperparamters and possible tuning metric.

The algo tries all possible commbinations of of hyperparameters in a grid and evaluates the performance of each combo with some cross validation set. The output is the hyperparameter combo which produces the best result.

The performance metric is required to assess the which is the best performing hyperparamters to be used in learning algorithm. The grid signifies the exhaustive nature of approach to try out all possible combinations of the hyperparameters.

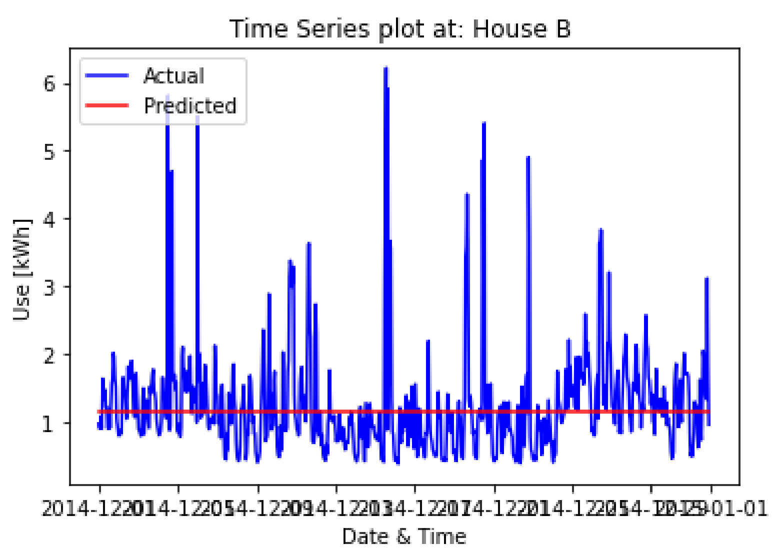
We are using **R2 Score**, to quantify your model's performance. The coefficient of determination for a model is a useful statistic in regression analysis, as it often describes how "good" that model is at making predictions.

In [31]:
```
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

def performance_metric(y_true, y_predict):
    """ Calculates and returns the performance score between
        true and predicted values based on the metric chosen."""
    score = r2_score(y_true, y_predict)
    return score

def fit_model(X, y):
    """ Performs grid search over the 'max_depth' parameter for a
        decision tree regressor trained on the input data [X, y]. """
    regressor = DecisionTreeRegressor()
    params = dict(max_depth=[1,2,3,4,5,6,7,8,9,10])
    scoring_fnc = make_scorer(performance_metric)
    grid = GridSearchCV(regressor,params,scoring=scoring_fnc)
    grid = grid.fit(X, y)
    return grid.best_estimator_

def DecisionTreeModel(features, target, split, house):
    xtrain = features[:split][linkedFeatures].reset_index(drop=True)
    ytrain = target[:split][['use [kW]']].reset_index(drop=True)
    ytest = target[split:][['use [kW]']].reset_index(drop=True)
    xtest = features[split : (split + len(ytest))][linkedFeatures].reset_in
    model = fit_model(xtrain, ytrain)
    print("Parameter 'max_depth' is {} for the optimal model.".format(model
    ypredict = model.predict(xtest)
    plotvalues(features[split : (split + len(ytest))][["Date & Time"]], yte
    mae = mean_absolute_error(ytest, ypredict)
    return mae
```

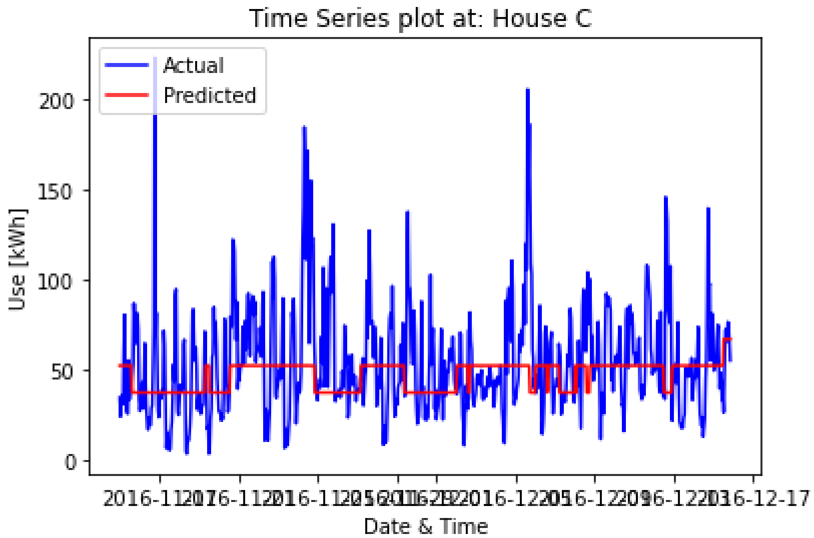
Hourly and Daily datapoints

In [32]:
```
models.append("Decision Tree (With Grid Search)")

maeVal = DecisionTreeModel(weather_hourly_B, energy_hourly_B, splitHB, "Hou
maeBHourly.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for hourly points on 

maeVal = DecisionTreeModel(weather_hourly_C, energy_hourly_C, splitHC, "Hou
maeCHourly.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for hourly points on 

maeVal = DecisionTreeModel(weather_hourly_F, energy_hourly_F, splitHF, "Hou
maeFHourly.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for hourly points on 

maeVal = DecisionTreeModel(weather_daily_B, energy_daily_B, splitDB, "House
maeBDaily.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for daily points on H

maeVal = DecisionTreeModel(weather_daily_C, energy_daily_C, splitDC, "House
maeCDaily.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for daily points on H

maeVal = DecisionTreeModel(weather_daily_F, energy_daily_F, splitDF, "House
maeFDaily.append(maeVal)
print("The Mean Absolute Error for DecisionTree Model for daily points on H
```

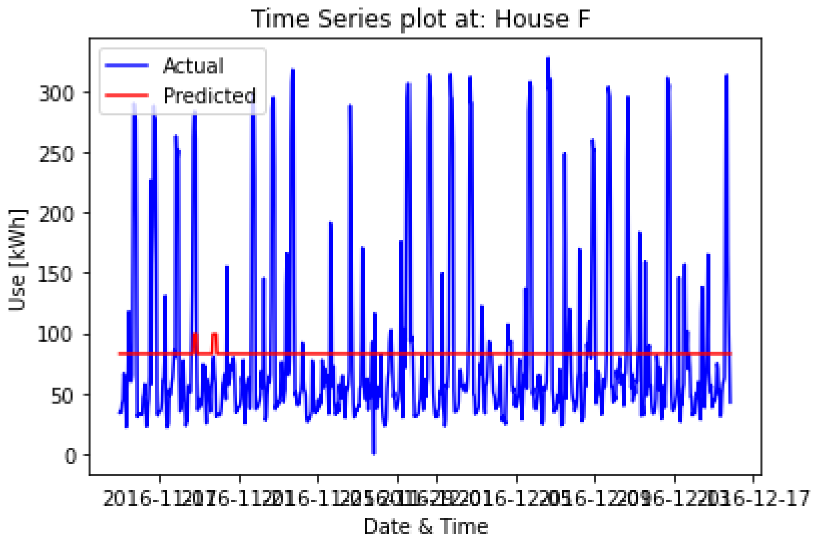
```
Parameter 'max_depth' is 2 for the optimal model.
```



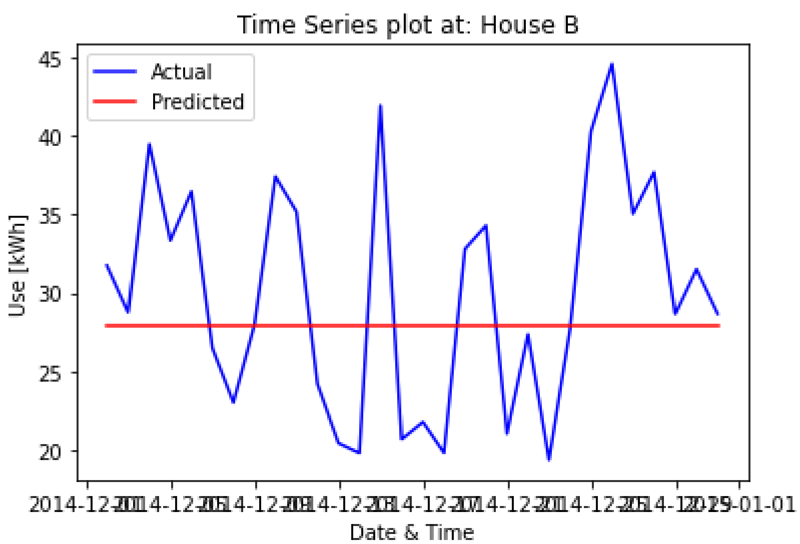
```
The Mean Absolute Error for DecisionTree Model for hourly points on House
B is 0.48084775861604884
Parameter 'max_depth' is 4 for the optimal model.
```



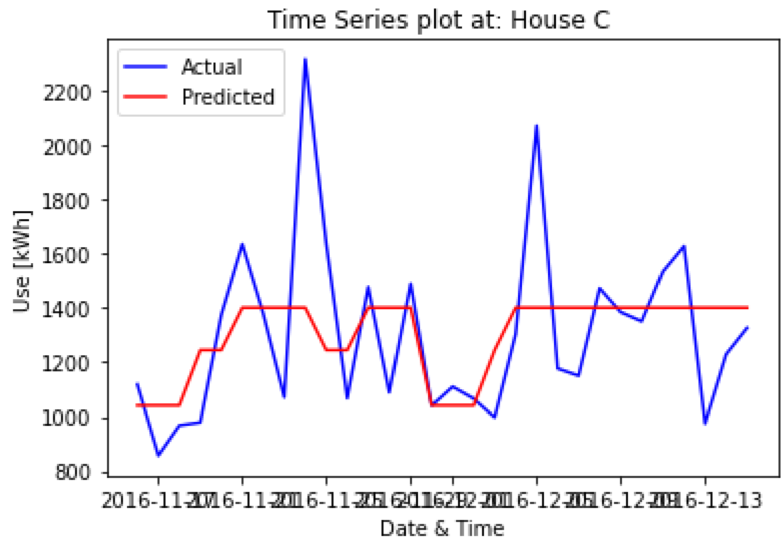

```
The Mean Absolute Error for DecisionTree Model for hourly points on House
C is 21.50087723226878
Parameter 'max_depth' is 2 for the optimal model.
```



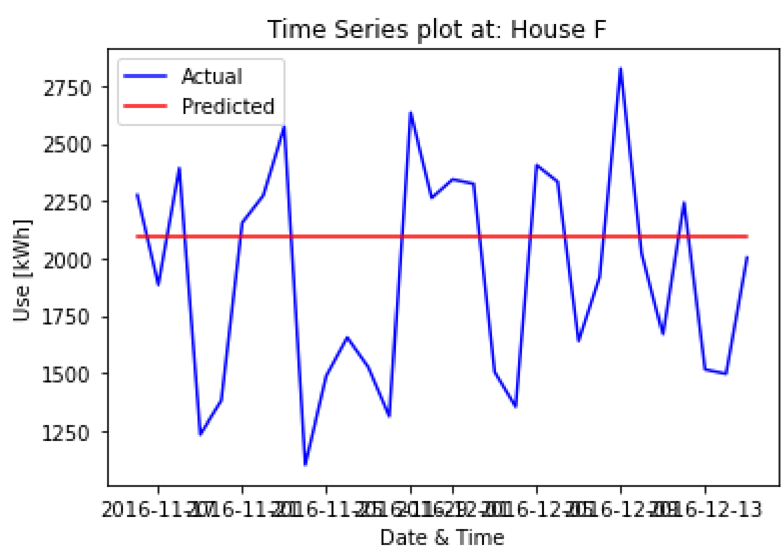

```
The Mean Absolute Error for DecisionTree Model for hourly points on House
F is 50.25079070120949
Parameter 'max_depth' is 1 for the optimal model.
```



```
The Mean Absolute Error for DecisionTree Model for daily points on Hous
e B is 6.228120276068112
Parameter 'max_depth' is 4 for the optimal model.
```




```
The Mean Absolute Error for DecisionTree Model for daily points on House
C is 201.46049513978446
Parameter 'max_depth' is 1 for the optimal model.
```



```
The Mean Absolute Error for DecisionTree Model for daily points on House
F is 424.30888697181285
```

In [33]:
```
decisionTree = pd.DataFrame({
    'Models': models,
    'House B (Hourly)': maeBHourly,
    'House C (Hourly)': maeCHourly,
    'House F (Hourly)': maeFHourly,
    'House B (Daily)': maeBDaily,
    'House C (Daily)': maeCDaily,
    'House F (Daily)': maeFDaily,
})
print("Differences between MAEs:")
display(decisionTree.loc[2:3])
```

```
Differences between MAEs:
```

	Models	House B (Hourly)	House C (Hourly)	House F (Hourly)	House B (Daily)	House C (Daily)	House F (Daily)
2	Decision Tree (Without Grid Search)	0.695242	31.519694	63.341204	9.44280	333.651529	430.680348
3	Decision Tree (With Grid Search)	0.480848	21.500877	50.250791	6.22812	201.460495	424.308887

Hence, its clear quantitatively, by how much **Grid Search parameter Tuning** improves the MAE of the predictions made by different datasets.

Random Forest

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. As we saw above the benefits of parameter tuning, we take multiple max depth values as input for tuning parameters. Then using the best estimator to calculate the MAE.

In [34]:
```
def RandomForestModel(features, target, split):
    xtrain = features[:split][linkedFeatures].reset_index(drop=True)
    ytrain = target[:split][['use [kW]']].reset_index(drop=True)
    ytest = target[split:][['use [kW]']].reset_index(drop=True)
    xtest = features[split : (split + len(ytest))][linkedFeatures].reset_in

    randomforest = RandomForestRegressor(random_state=3)
    params = dict(max_depth=[1,2,3,4,5,6,7,8,9,10])
    scoring_fnc = make_scorer(performance_metric)
    grid = GridSearchCV(randomforest,params,scoring=scoring_fnc)
    grid = grid.fit(xtrain, ytrain.values.ravel())
    model =  grid.best_estimator_
    mae = mean_absolute_error(ytest, model.predict(xtest))
    return mae
```

In [35]:
```
models.append("Random Forest")

maeVal = RandomForestModel(weather_hourly_B, energy_hourly_B, splitHB)
maeBHourly.append(maeVal)
print("The Mean Absolute Error for Random Forest for hourly points on House

maeVal = RandomForestModel(weather_hourly_C, energy_hourly_C, splitHC)
maeCHourly.append(maeVal)
print("The Mean Absolute Error for Random Forest for hourly points on House

maeVal = RandomForestModel(weather_hourly_F, energy_hourly_F, splitHF)
maeFHourly.append(maeVal)
print("The Mean Absolute Error for Random Forest for hourly points on House

maeVal = RandomForestModel(weather_daily_B, energy_daily_B, splitDB)
maeBDaily.append(maeVal)
print("The Mean Absolute Error for Random Forest for daily points on House

maeVal = RandomForestModel(weather_daily_C, energy_daily_C, splitDC)
maeCDaily.append(maeVal)
print("The Mean Absolute Error for Random Forest for daily points on House

maeVal = RandomForestModel(weather_daily_F, energy_daily_F, splitDF)
maeFDaily.append(maeVal)
print("The Mean Absolute Error for Random Forest for daily points on House
```

```
The Mean Absolute Error for Random Forest for hourly points on House B is
0.480918561313664
The Mean Absolute Error for Random Forest for hourly points on House C is
21.15884825500904
The Mean Absolute Error for Random Forest for hourly points on House F is
50.39096149110985
The Mean Absolute Error for Random Forest for daily points on House B is
7.6474998072644835
The Mean Absolute Error for Random Forest for daily points on House C is
209.23926077321673
The Mean Absolute Error for Random Forest for daily points on House F is
424.12097549005034
```

ARIMA

ARIMA, short for ‘Auto Regressive Integrated Moving Average’ is a class of models that ‘explains’ a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.

The parameters of the ARIMA model are defined as follows:

- p: The number of lag observations included in the model, also called the lag order.
- d: The number of times that the raw observations are differenced, also called the degree of differencing.
- q: The size of the moving average window, also called the order of moving average.

We will be iterating over the permissible values of p and d to get the best prediction in terms of minimum MAE.

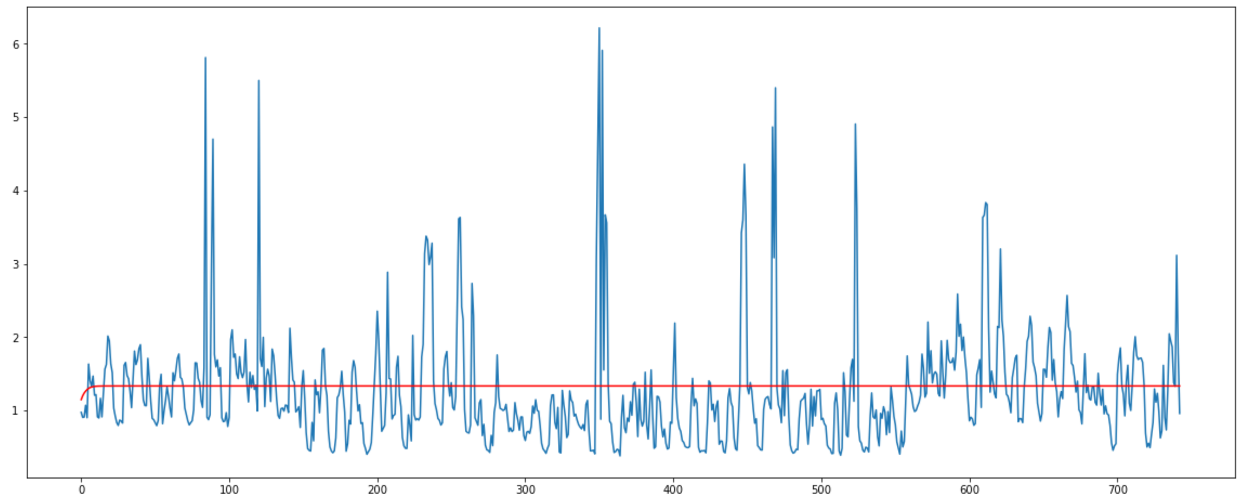
In [36]:
```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [37]:
```
def testForParameters(train, test, split):
    ypredicted=[]
    optimalp =0
    optimald = 0
    leasterror = float("inf")
    for p in range(0,10):
        for d in range(0,2):
            newmodel = ARIMA(train, order=(p,d,0))
            model_fit = newmodel.fit()
            ypredicted = model_fit.predict(start=split, end=(split+len(test
            error = mean_absolute_error(test, ypredicted)
            if(error<leasterror):
                leasterror = error
                optimalp = p
                optimald = d
    return optimalp,optimald

def ArimaModel(ytrain, ytest, split, p, d):
    model = ARIMA(ytrain, order=(p,d,0))
    model = model.fit()
    ypredict = model.predict(start=split, end=(split+len(ytest)-1))
    print(model.summary())
    mae = mean_absolute_error(ytest, ypredict)
    plt.figure(figsize=(20,8))
    plt.plot(ytest)
    plt.plot(ypredict, color='red')
    plt.show()
    return mae
```

In [38]:
```python
models.append("ARIMA")
for i in range(6):
    target = energyData[i][['use [kW]']].values
    split = splitPoints[i]
    train = target[:split]
    test = target[split:]
    optimalp,optimald = testForParameters(train, test, split)
    print("Best P & D Values for ", tag[i]," are :", optimalp, "  ", optima
    mae = ArimaModel(train, test, split, optimalp, optimald)
    print ("The Mean Absolute Error for ARIMA Model for ", tag[i], "is :",
    aggregate[i].append(mae)
```

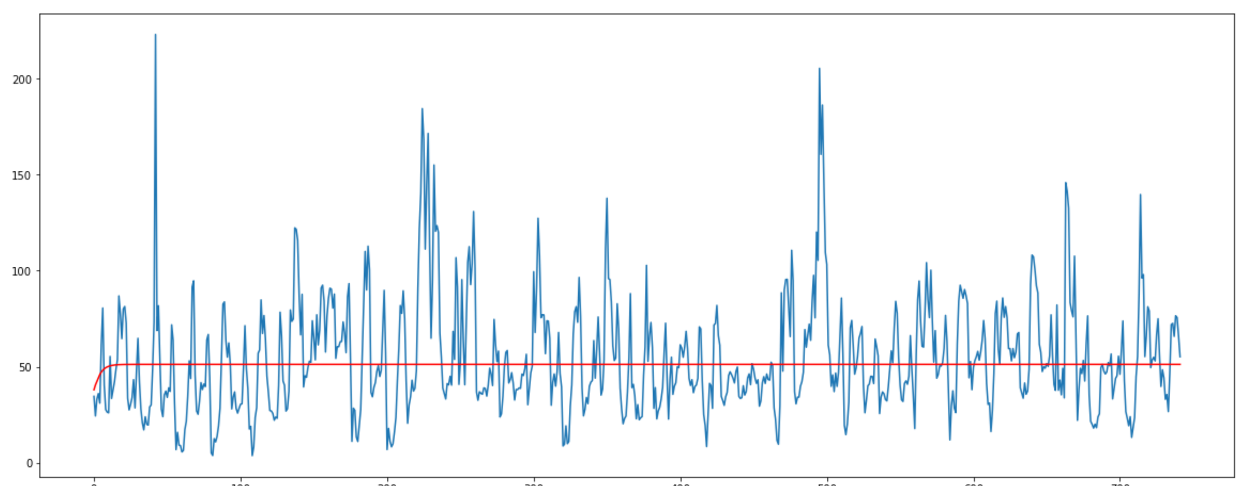
```
Best P & D Values for  House B (Hourly)  are : 6    0
                              ARMA Model Results
==========================================================================
=====
Dep. Variable:                      y   No. Observations:
8016
Model:                     ARMA(6, 0)   Log Likelihood              -1123
9.685
Method:                       css-mle   S.D. of innovations
0.983
Date:                Thu, 18 Mar 2021   AIC                          2249
5.370
Time:                        20:52:35   BIC                          2255
1.284
Sample:                             0   HQIC                         2251
4.507

==========================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const          1.3330      0.034     39.235      0.000       1.266
1.400
ar.L1.y        0.6481      0.011     58.033      0.000       0.626
0.670
ar.L2.y       -0.0008      0.013     -0.060      0.952      -0.027
0.025
ar.L3.y        0.0263      0.013      1.980      0.048       0.000
0.052
ar.L4.y        0.0290      0.013      2.184      0.029       0.003
0.055
ar.L5.y       -0.0119      0.013     -0.894      0.371      -0.038
0.014
ar.L6.y       -0.0139      0.011     -1.248      0.212      -0.036
0.008
                                    Roots
==========================================================================
====
                  Real          Imaginary           Modulus         Frequ
ency
--------------------------------------------------------------------------
```

```
----
AR.1            1.6300          -0.1340j            1.6355          -0.
0131
AR.2            1.6300          +0.1340j            1.6355           0.
0131
AR.3            0.2243          -1.9906j            2.0032          -0.
2321
AR.4            0.2243          +1.9906j            2.0032           0.
2321
AR.5           -2.2808          -1.2171j            2.5853          -0.
4220
AR.6           -2.2808          +1.2171j            2.5853           0.
4220
-----------------------------------------------------------------------
----
```



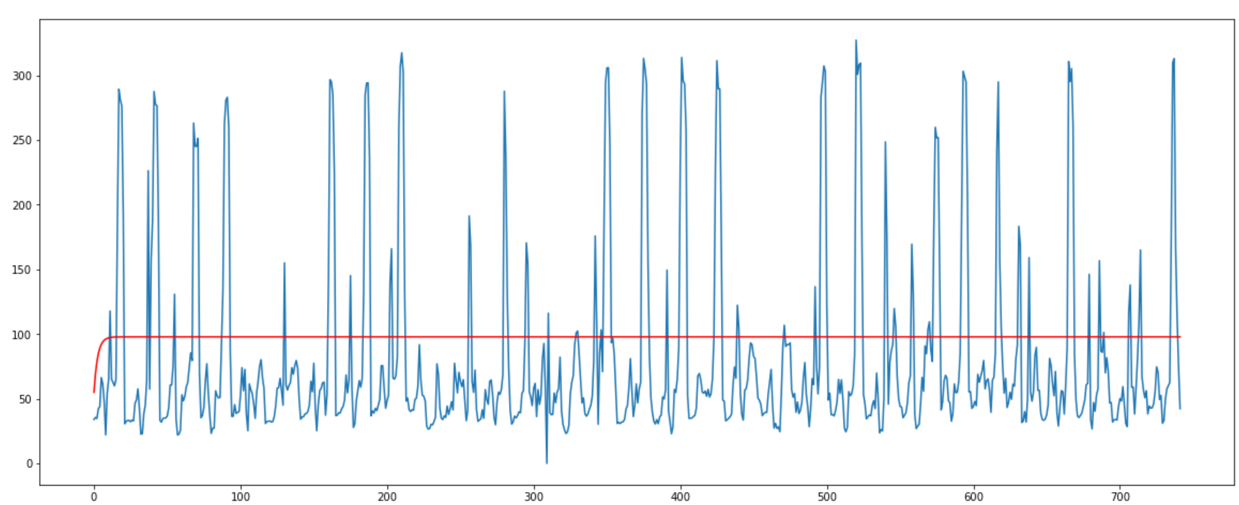

```
The Mean Absolute Error for ARIMA Model for  House B (Hourly) is : 0.52
15728809115998
Best P & D Values for  House C (Hourly)  are : 5    0
                              ARMA Model Results
======================================================================
=======
Dep. Variable:                      y   No. Observations:
7656
Model:                     ARMA(5, 0)   Log Likelihood               -37
190.857
Method:                       css-mle   S.D. of innovations
31.148
Date:                Thu, 18 Mar 2021   AIC                           74
395.714
Time:                        20:52:44   BIC                           74
444.317
Sample:                             0   HQIC                          74
412.386

======================================================================
=======
```

```
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------
-------
const         51.2534      1.582     32.393      0.000      48.152
54.355
ar.L1.y        0.7925      0.011     69.353      0.000       0.770
0.815
ar.L2.y       -0.0236      0.015     -1.621      0.105      -0.052
0.005
ar.L3.y        0.0195      0.015      1.338      0.181      -0.009
0.048
ar.L4.y       -0.0043      0.015     -0.295      0.768      -0.033
0.024
ar.L5.y       -0.0090      0.011     -0.786      0.432      -0.031
0.013
                                    Roots
========================================================================
======
                  Real          Imaginary           Modulus         Fre
quency
------------------------------------------------------------------------
------
AR.1            1.3188           -0.0000j            1.3188           -
0.0000
AR.2            2.5842           -0.0000j            2.5842           -
0.0000
AR.3           -0.3568           -2.9629j            2.9843           -
0.2691
AR.4           -0.3568           +2.9629j            2.9843
0.2691
AR.5           -3.6688           -0.0000j            3.6688           -
0.5000
------------------------------------------------------------------------
------
```



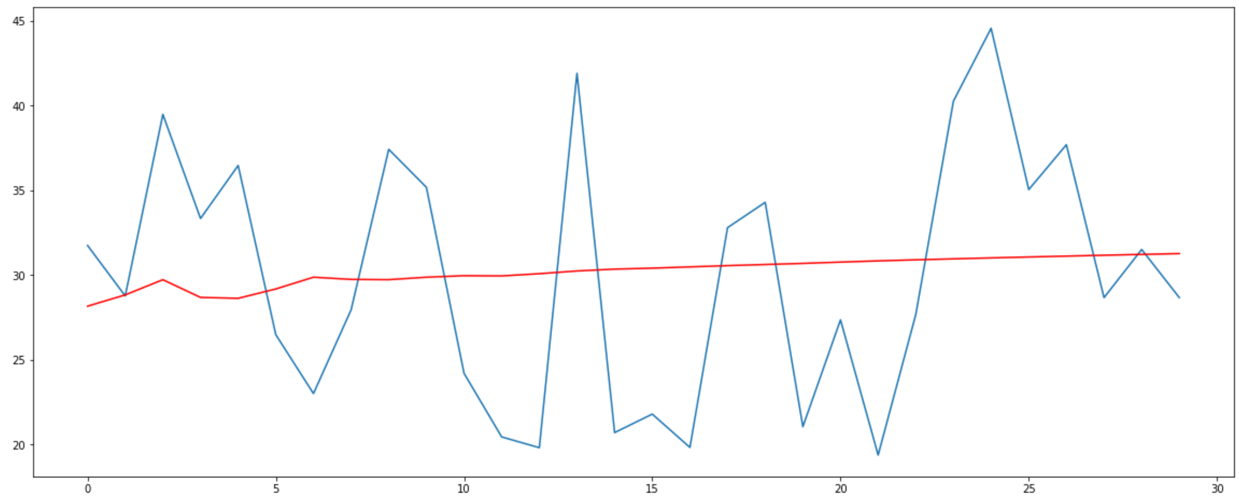

```
The Mean Absolute Error for ARIMA Model for  House C (Hourly) is : 21.5
34172498125194
```

```
Best P & D Values for  House F (Hourly)  are : 1    0
                              ARMA Model Results
==============================================================================
=======
Dep. Variable:                      y   No. Observations:
7656
Model:                     ARMA(1, 0)   Log Likelihood                   -42
787.887
Method:                       css-mle   S.D. of innovations
64.704
Date:                Thu, 18 Mar 2021   AIC                               85
581.773
Time:                        20:52:52   BIC                               85
602.603
Sample:                             0   HQIC                              85
588.919

==============================================================================
=======
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-------
const         97.8637      2.252     43.465      0.000      93.451
102.277
ar.L1.y        0.6717      0.008     79.338      0.000       0.655
0.688
                                    Roots
==============================================================================
======
                  Real          Imaginary           Modulus         Fre
quency
------------------------------------------------------------------------------
------
AR.1            1.4889           +0.0000j            1.4889
0.0000
------------------------------------------------------------------------------
------
```



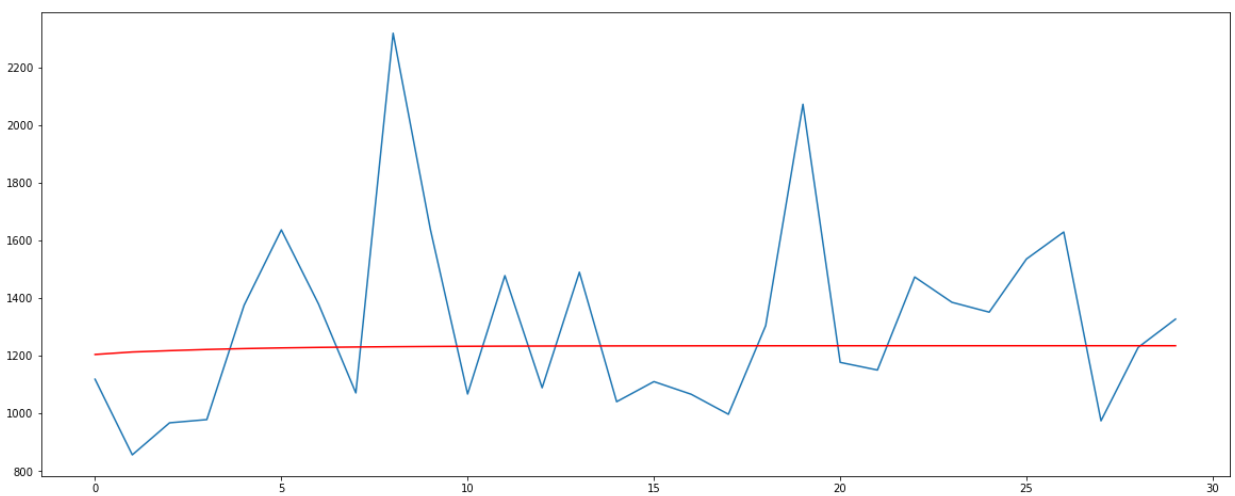
```
The Mean Absolute Error for ARIMA Model for  House F (Hourly) is : 59.175
42783489664
Best P & D Values for  House B (Daily)  are : 8    0
                              ARMA Model Results
==========================================================================
=====
Dep. Variable:                     y   No. Observations:
335
Model:                     ARMA(8, 0)   Log Likelihood                 -127
7.913
Method:                       css-mle   S.D. of innovations               1
0.958
Date:                Thu, 18 Mar 2021   AIC                             257
5.825
Time:                        20:52:58   BIC                             261
3.966
Sample:                             0   HQIC                            259
1.031

==========================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const         32.3668      4.045      8.001      0.000      24.438       4
0.296
ar.L1.y        0.3820      0.055      6.937      0.000       0.274
0.490
ar.L2.y       -0.0092      0.059     -0.157      0.875      -0.124
0.106
ar.L3.y        0.1094      0.058      1.876      0.061      -0.005
0.224
ar.L4.y        0.1143      0.059      1.949      0.051      -0.001
0.229
ar.L5.y       -0.0088      0.059     -0.151      0.880      -0.124
0.106
ar.L6.y        0.1120      0.058      1.922      0.055      -0.002
0.226
ar.L7.y        0.0850      0.059      1.449      0.147      -0.030
0.200
ar.L8.y        0.0759      0.055      1.383      0.167      -0.032
0.184
                                    Roots
==========================================================================
====
                  Real          Imaginary           Modulus         Frequ
ency
--------------------------------------------------------------------------
----
AR.1            1.0426           -0.0000j            1.0426           -0.
0000
AR.2            0.8617           -0.9593j            1.2895           -0.
1335
AR.3            0.8617           +0.9593j            1.2895            0.
```

```
1335
AR.4          -0.1296           -1.3691j              1.3753            -0.
2650
AR.5          -0.1296           +1.3691j              1.3753             0.
2650
AR.6          -1.4453           -0.0000j              1.4453            -0.
5000
AR.7          -1.0903           -1.2611j              1.6670            -0.
3635
AR.8          -1.0903           +1.2611j              1.6670             0.
3635
-------------------------------------------------------------------------
----
```



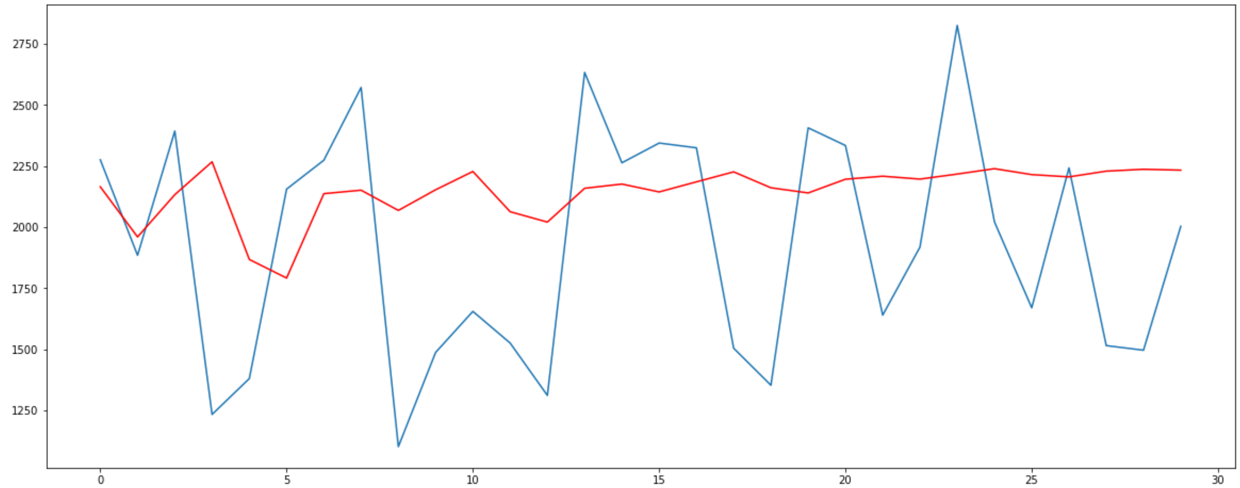

```
The Mean Absolute Error for ARIMA Model for  House B (Daily) is : 6.291
542500487939
Best P & D Values for  House C (Daily)  are : 3    0
                              ARMA Model Results
==============================================================================
=======
Dep. Variable:                      y   No. Observations:
320
Model:                     ARMA(3, 0)   Log Likelihood                   -2
464.075
Method:                       css-mle   S.D. of innovations
534.005
Date:                Thu, 18 Mar 2021   AIC                               4
938.150
Time:                        20:53:05   BIC                               4
956.991
Sample:                             0   HQIC                              4
945.674

==============================================================================
=======
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-------
const       1234.7748     98.865     12.490      0.000    1041.004      1
428.546
```

```
ar.L1.y        0.5462      0.056      9.799      0.000       0.437
0.655
ar.L2.y        0.1188      0.063      1.880      0.060      -0.005
0.243
ar.L3.y        0.0357      0.056      0.642      0.521      -0.073
0.145
                                    Roots
===========================================================================
======
                  Real          Imaginary           Modulus         Fre
quency
---------------------------------------------------------------------------
------
AR.1            1.3102           -0.0000j            1.3102            -
0.0000
AR.2           -2.3172           -3.9980j            4.6209            -
0.3336
AR.3           -2.3172           +3.9980j            4.6209
0.3336
---------------------------------------------------------------------------
------
```




```
The Mean Absolute Error for ARIMA Model for  House C (Daily) is : 247.890
7938571151
Best P & D Values for  House F (Daily)  are : 9    0
                              ARMA Model Results
============================================================================
=====
Dep. Variable:                      y   No. Observations:
320
Model:                     ARMA(9, 0)   Log Likelihood                  -242
5.504
Method:                       css-mle   S.D. of innovations               47
2.369
Date:                Thu, 18 Mar 2021   AIC                              487
3.008
Time:                        20:53:13   BIC                              491
```

```
4.460
Sample:                               0   HQIC                          488
9.561

=========================================================================
=====
                 coef    std err          z      P>|z|      [0.025      
0.975]
-------------------------------------------------------------------------
-----
const       2317.8725    147.265     15.739      0.000    2029.239     260
6.506
ar.L1.y        0.4337      0.056      7.771      0.000       0.324      
0.543
ar.L2.y       -0.0247      0.061     -0.408      0.683      -0.143      
0.094
ar.L3.y        0.0283      0.056      0.505      0.613      -0.081      
0.138
ar.L4.y        0.0571      0.056      1.015      0.310      -0.053      
0.167
ar.L5.y       -0.0348      0.056     -0.618      0.537      -0.145      
0.076
ar.L6.y        0.0415      0.057      0.734      0.463      -0.069      
0.152
ar.L7.y        0.4184      0.056      7.415      0.000       0.308      
0.529
ar.L8.y       -0.1181      0.061     -1.923      0.054      -0.238      
0.002
ar.L9.y        0.0278      0.057      0.491      0.623      -0.083      
0.139
                                    Roots                                
=========================================================================
====
                  Real          Imaginary           Modulus         Frequ
ency
-------------------------------------------------------------------------
----
AR.1           -1.0545           -0.4872j            1.1616           -0.
4311
AR.2           -1.0545           +0.4872j            1.1616            0.
4311
AR.3           -0.2404           -1.1059j            1.1317           -0.
2841
AR.4           -0.2404           +1.1059j            1.1317            0.
2841
AR.5            1.0503           -0.0000j            1.0503           -0.
0000
AR.6            0.7325           -0.8654j            1.1338           -0.
1382
AR.7            0.7325           +0.8654j            1.1338            0.
1382
AR.8            2.1620           -3.2782j            3.9270           -0.
1572
AR.9            2.1620           +3.2782j            3.9270            0.
1572
-------------------------------------------------------------------------
----
```




```
The Mean Absolute Error for ARIMA Model for  House F (Daily) is : 437.361
6662728573
```

Final Results

Below are the aggregated results of MAE achieved by different models we tested above.

In [39]:
```
results = pd.DataFrame({
    'Models': models,
    'House B (Hourly)': maeBHourly,
    'House C (Hourly)': maeCHourly,
    'House F (Hourly)': maeFHourly,
    'House B (Daily)': maeBDaily,
    'House C (Daily)': maeCDaily,
    'House F (Daily)': maeFDaily,
})
print("The final results on MAEs:")
display(results)
```

```
The final results on MAEs:
```

	Models	House B (Hourly)	House C (Hourly)	House F (Hourly)	House B (Daily)	House C (Daily)	House F (Daily)
0	Naive Method	0.479529	25.249532	47.941220	6.231545	252.469670	440.971126
1	Linear Regression	0.463795	21.191972	45.505913	6.157595	266.672331	431.781231
2	Decision Tree (Without Grid Search)	0.695242	31.519694	63.341204	9.442800	333.651529	430.680348
3	Decision Tree (With Grid Search)	0.480848	21.500877	50.250791	6.228120	201.460495	424.308887
4	Random Forest	0.480919	21.158848	50.390961	7.647500	209.239261	424.120975
5	ARIMA	0.521573	21.534172	59.175428	6.291543	247.890794	437.361666

Hence we find different models performing differently for different house data sets. In particular we find the goodness of HyperParameter tuning in Decision Tree Regressor (comparing both rows 2 & 3). We have models which perform better than Naive Method viz Linear Regression performing better for all instances. Decision Tree with Grid Search performing quite better for House C Daily. Thus using different techniques and machine learning algorithms, we are able to predict energy consumption for different users with performances better than out baseline Naive model.