In [1]:
```
# we will use the pandas library for data analysis and manipulation
import pandas as pd
# Often we need some functions from numpy for adding support for la
import numpy as np
# For data visualization, we import matplotlib
import matplotlib.pyplot as plt
```

In [2]:
```
Energy1 = pd.read_excel("Building energy consumption racord.xlsx")
Energy1
```

Out[2]:

| | Time | building 41 |
|---|---|---|
| **0** | 2016-01-01 01:00:00 | 23.783228 |
| **1** | 2016-01-01 02:00:00 | 23.783228 |
| **2** | 2016-01-01 03:00:00 | 23.783228 |
| **3** | 2016-01-01 04:00:00 | 23.783228 |
| **4** | 2016-01-01 05:00:00 | 23.783228 |
| **...** | ... | ... |
| **26298** | 2018-12-31 19:00:00 | 18.602723 |
| **26299** | 2018-12-31 20:00:00 | 18.838200 |
| **26300** | 2018-12-31 21:00:00 | 18.602723 |
| **26301** | 2018-12-31 22:00:00 | 18.131768 |
| **26302** | 2018-12-31 23:00:00 | 18.602723 |

26303 rows × 2 columns

In [3]:
```
# set time column as index
Energy = Energy1.set_index('Time')
Energy
```

Out[3]:

| Time | building 41 |
|---|---|
| 2016-01-01 01:00:00 | 23.783228 |
| 2016-01-01 02:00:00 | 23.783228 |
| 2016-01-01 03:00:00 | 23.783228 |
| 2016-01-01 04:00:00 | 23.783228 |
| 2016-01-01 05:00:00 | 23.783228 |
| ... | ... |
| 2018-12-31 19:00:00 | 18.602723 |
| 2018-12-31 20:00:00 | 18.838200 |
| 2018-12-31 21:00:00 | 18.602723 |
| 2018-12-31 22:00:00 | 18.131768 |
| 2018-12-31 23:00:00 | 18.602723 |

26303 rows × 1 columns

In [4]:
```
# Check the description of the data
Energy.describe()
```

Out[4]:

| | building 41 |
|---|---|
| count | 26303.000000 |
| mean | 25.694969 |
| std | 6.317738 |
| min | 15.541515 |
| 25% | 20.957498 |
| 50% | 23.783228 |
| 75% | 28.728255 |
| max | 59.340330 |

In [9]:
```
#Load the Weather data from the excel file
knmi= pd.read_excel("WeatherData.xlsx ")
knmi
```

Out[9]:

| | Time | month | HH | TD | U | Temp | RH | Q | DR | FF | FX | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2016-01-01 01:00:00 | 1 | 1 | 38 | 82 | 6.6 | 0.82 | 0 | 0 | 30 | 70 | 10224 |
| **1** | 2016-01-01 02:00:00 | 1 | 2 | 43 | 83 | 7.0 | 0.83 | 0 | 0 | 40 | 80 | 10228 |
| **2** | 2016-01-01 03:00:00 | 1 | 3 | 46 | 91 | 5.9 | 0.91 | 0 | 0 | 30 | 80 | 10232 |
| **3** | 2016-01-01 04:00:00 | 1 | 4 | 36 | 96 | 4.2 | 0.96 | 0 | 0 | 20 | 40 | 10237 |
| **4** | 2016-01-01 05:00:00 | 1 | 5 | 37 | 98 | 4.0 | 0.98 | 0 | 0 | 20 | 30 | 10240 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **26298** | 2018-12-31 19:00:00 | 12 | 19 | 78 | 93 | 8.7 | 0.93 | 0 | 0 | 30 | 60 | 10341 |
| **26299** | 2018-12-31 20:00:00 | 12 | 20 | 74 | 92 | 8.5 | 0.92 | 0 | 0 | 30 | 50 | 10338 |
| **26300** | 2018-12-31 21:00:00 | 12 | 21 | 66 | 89 | 8.2 | 0.89 | 0 | 0 | 40 | 60 | 10336 |
| **26301** | 2018-12-31 22:00:00 | 12 | 22 | 68 | 94 | 7.6 | 0.94 | 0 | 0 | 40 | 70 | 10332 |
| **26302** | 2018-12-31 23:00:00 | 12 | 23 | 67 | 94 | 7.6 | 0.94 | 0 | 7 | 40 | 60 | 10333 |

26303 rows × 12 columns

In [10]:
```
#Set the Time column as index
knmi = knmi.set_index("Time")
knmi
```

Out[10]:

| Time | month | HH | TD | U | Temp | RH | Q | DR | FF | FX | P |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **2016-01-01 01:00:00** | 1 | 1 | 38 | 82 | 6.6 | 0.82 | 0 | 0 | 30 | 70 | 10224 |
| **2016-01-01 02:00:00** | 1 | 2 | 43 | 83 | 7.0 | 0.83 | 0 | 0 | 40 | 80 | 10228 |
| **2016-01-01 03:00:00** | 1 | 3 | 46 | 91 | 5.9 | 0.91 | 0 | 0 | 30 | 80 | 10232 |
| **2016-01-01 04:00:00** | 1 | 4 | 36 | 96 | 4.2 | 0.96 | 0 | 0 | 20 | 40 | 10237 |
| **2016-01-01 05:00:00** | 1 | 5 | 37 | 98 | 4.0 | 0.98 | 0 | 0 | 20 | 30 | 10240 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2018-12-31 19:00:00** | 12 | 19 | 78 | 93 | 8.7 | 0.93 | 0 | 0 | 30 | 60 | 10341 |
| **2018-12-31 20:00:00** | 12 | 20 | 74 | 92 | 8.5 | 0.92 | 0 | 0 | 30 | 50 | 10338 |
| **2018-12-31 21:00:00** | 12 | 21 | 66 | 89 | 8.2 | 0.89 | 0 | 0 | 40 | 60 | 10336 |
| **2018-12-31 22:00:00** | 12 | 22 | 68 | 94 | 7.6 | 0.94 | 0 | 0 | 40 | 70 | 10332 |
| **2018-12-31 23:00:00** | 12 | 23 | 67 | 94 | 7.6 | 0.94 | 0 | 7 | 40 | 60 | 10333 |

26303 rows × 11 columns

In [11]:
```
#concatenating the datasets of weather data and electricity consump
df = pd.concat([knmi, Energy], axis=1) #axis =1 for considering the
df
```

Out[11]:

| Time | month | HH | TD | U | Temp | RH | Q | DR | FF | FX | P | building 41 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **2016-01-01 01:00:00** | 1 | 1 | 38 | 82 | 6.6 | 0.82 | 0 | 0 | 30 | 70 | 10224 | 23.783228 |
| **2016-01-01 02:00:00** | 1 | 2 | 43 | 83 | 7.0 | 0.83 | 0 | 0 | 40 | 80 | 10228 | 23.783228 |
| **2016-01-01 03:00:00** | 1 | 3 | 46 | 91 | 5.9 | 0.91 | 0 | 0 | 30 | 80 | 10232 | 23.783228 |
| **2016-01-01 04:00:00** | 1 | 4 | 36 | 96 | 4.2 | 0.96 | 0 | 0 | 20 | 40 | 10237 | 23.783228 |
| **2016-01-01 05:00:00** | 1 | 5 | 37 | 98 | 4.0 | 0.98 | 0 | 0 | 20 | 30 | 10240 | 23.783228 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2018-12-31 19:00:00** | 12 | 19 | 78 | 93 | 8.7 | 0.93 | 0 | 0 | 30 | 60 | 10341 | 18.602723 |
| **2018-12-31 20:00:00** | 12 | 20 | 74 | 92 | 8.5 | 0.92 | 0 | 0 | 30 | 50 | 10338 | 18.838200 |
| **2018-12-31 21:00:00** | 12 | 21 | 66 | 89 | 8.2 | 0.89 | 0 | 0 | 40 | 60 | 10336 | 18.602723 |
| **2018-12-31 22:00:00** | 12 | 22 | 68 | 94 | 7.6 | 0.94 | 0 | 0 | 40 | 70 | 10332 | 18.131768 |
| **2018-12-31 23:00:00** | 12 | 23 | 67 | 94 | 7.6 | 0.94 | 0 | 7 | 40 | 60 | 10333 | 18.602723 |

26303 rows × 12 columns

In [12]:
```
# check missing data status
df.isna().sum()
```

Out[12]:
```
month          0
HH             0
TD             0
U              0
Temp           0
RH             0
Q              0
DR             0
FF             0
FX             0
P              0
building 41    0
dtype: int64
```

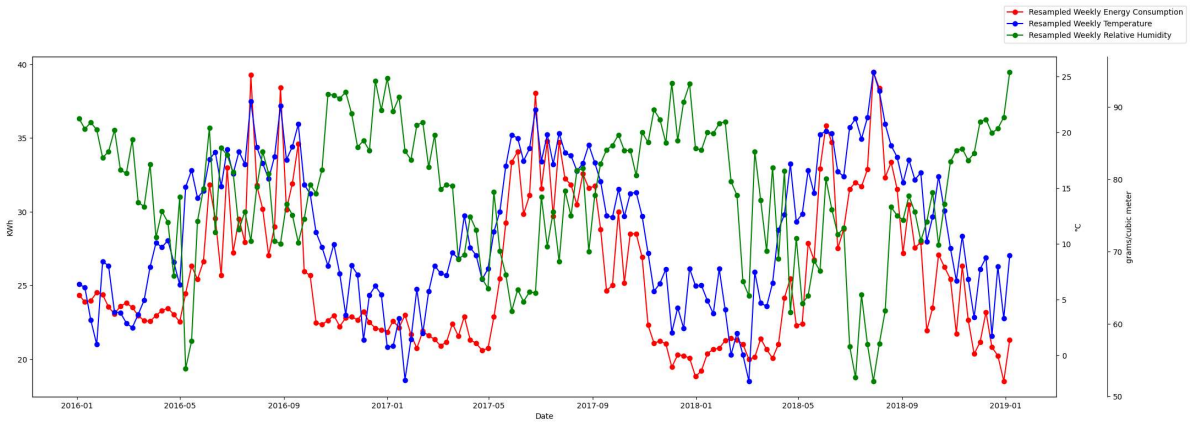
# Energy Against Humidity And temperature

In [57]:
```
df_sum_weekly = df['building 41'].resample('W').mean()
# Resample the temperature over a week.
df_feature1= df["Temp"].resample("W").mean()
# Resample the relative humidity over a week.
df_feature2 = df["U"].resample("W").mean()
```

fig,ax = plt.subplots(figsize=(24,8)) # Create matplotlib figure ax.plot(df_sum_weekly.index, df_sum_weekly, color="red",marker="o") ax.set_ylabel("KWh") ax.set_xlabel('Date') ax2 = ax.twinx() #Create a new Axes with an invisible x-axis and an ax3 = ax.twinx() ax2.plot(df_sum_weekly.index, df_feature1, color="blue", marker="o") ax2.set_ylabel("°C") ax3.plot(df_sum_weekly.index, df_feature2, color="green", marker="o") ax3.set_ylabel("grams/cubic meter") ax3.spines["right"].set_position(("axes",1.05)) fig.legend(["Resampled Weekly Energy Consumption","Resampled Weekly"]) fig.show()

In [58]:
```
ult
ubplots(figsize=(24,8))  # Create matplotlib figure
_weekly.index, df_sum_weekly, color="red",marker="o")
"KWh")
'Date')
() #Create a new Axes with an invisible x-axis and an independent y-axis positi
()
m_weekly.index, df_feature1, color="blue", marker="o")
("°C")
m_weekly.index, df_feature2, color="green", marker="o")
("grams/cubic meter")
ght"].set_position(("axes", 1.05))
esampled Weekly Energy Consumption","Resampled Weekly Temperature","Resampled W
```

```
C:\Users\ritik\AppData\Local\Temp\ipykernel_16664\4074322926.py:14: UserWarni
ng: Matplotlib is currently using module://matplotlib_inline.backend_inline,
which is a non-GUI backend, so cannot show the figure.
  fig.show()
```




# Feature selection using non-linear correlation

In [59]:
```
# calculate the spearmans's correlation between two variables
from scipy.stats import spearmanr

#filter columns from dataframe
energy = np.array(df["building 41"])
hour = np.array(df["HH"])
month= np.array(df["month"])

# calculate spearman's correlation
corr1, _ = spearmanr(energy, hour)
corr2,_ = spearmanr(energy, month)
print('Spearmans correlation between Energy and hour feature: %.3f' % corr1)
print('Spearmans correlation between Energy and month feature: %.3f' % corr2)
```

```
Spearmans correlation between Energy and hour feature: 0.068
Spearmans correlation between Energy and month feature: 0.077
```

In [60]:
```
#Reduce number of features with lower correlation values or it has an inverse
knmi_updated= knmi.loc[:, ~knmi.columns.isin(["TD","U","DR","FX"])] # ~ sign d
knmi_updated
```

Out[60]:

| Time | month | HH | Temp | RH | Q | FF | P |
|---|---|---|---|---|---|---|---|
| **2016-01-01 01:00:00** | 1 | 1 | 6.6 | 0.82 | 0 | 30 | 10224 |
| **2016-01-01 02:00:00** | 1 | 2 | 7.0 | 0.83 | 0 | 40 | 10228 |
| **2016-01-01 03:00:00** | 1 | 3 | 5.9 | 0.91 | 0 | 30 | 10232 |
| **2016-01-01 04:00:00** | 1 | 4 | 4.2 | 0.96 | 0 | 20 | 10237 |
| **2016-01-01 05:00:00** | 1 | 5 | 4.0 | 0.98 | 0 | 20 | 10240 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **2018-12-31 19:00:00** | 12 | 19 | 8.7 | 0.93 | 0 | 30 | 10341 |
| **2018-12-31 20:00:00** | 12 | 20 | 8.5 | 0.92 | 0 | 30 | 10338 |
| **2018-12-31 21:00:00** | 12 | 21 | 8.2 | 0.89 | 0 | 40 | 10336 |
| **2018-12-31 22:00:00** | 12 | 22 | 7.6 | 0.94 | 0 | 40 | 10332 |
| **2018-12-31 23:00:00** | 12 | 23 | 7.6 | 0.94 | 0 | 40 | 10333 |

26303 rows × 7 columns

# Develop ML regression model based on the weather parameters to predict the energy consumption of the building.

In [61]:
```
#Splitting the data into training (80%) and testing (20%) set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(knmi_updated, Energy, test
y_train
```

Out[61]:

| Time | building 41 |
|---|---|
| 2016-11-29 03:00:00 | 22.370362 |
| 2018-01-29 11:00:00 | 24.725137 |
| 2018-07-03 20:00:00 | 36.028058 |
| 2017-09-05 16:00:00 | 44.034292 |
| 2017-03-08 06:00:00 | 19.309155 |
| ... | ... |
| 2017-06-30 20:00:00 | 29.434687 |
| 2018-03-29 17:00:00 | 21.899408 |
| 2017-02-14 06:00:00 | 21.192975 |
| 2017-03-26 00:00:00 | 19.544633 |
| 2016-04-23 21:00:00 | 20.722020 |

21042 rows × 1 columns

In [62]:
```
y_train = y_train.values.ravel() #ravel is a numpy function to change a 2-dime
y_train
```

Out[62]:
```
array([22.3703625, 24.7251375, 36.0280575, ..., 21.192975 , 19.5446325,
       20.72202  ])
```

In [63]:
```
y_test = y_test.values.ravel()
y_test
```

Out[63]:
```
array([19.5446325, 32.2604175, 20.0155875, ..., 20.72202  , 28.4927775,
       25.1960925])
```

In [64]:
```
# importing regression model
from sklearn.svm import SVR

#Creating an instance or object of the support vector machine regressor class
SVReg = SVR(kernel= 'rbf') # It must be one of 'linear', 'poly', 'rbf', 'sigmo

# fitting the regression model to the training dataset
SVReg.fit(X_train, y_train) #Fit the SVM model according to the given training
```

Out[64]:
```
SVR()
```

In [65]:
```
# predicting on the training data
Predicted_Train= SVReg.predict(X_train)
Predicted_Train
```

Out[65]:
```
array([23.33747534, 23.52037698, 23.57941113, ..., 23.3715236 ,
       23.38476265, 23.42220531])
```

In [66]:
```
# To evaluate the performance of the model, importing error metrics function
from sklearn.metrics import r2_score #(coefficient of determination) regressio
from sklearn.metrics import mean_squared_error #The MSE indicates the average

print(r2_score(y_train,Predicted_Train))
print(mean_squared_error(y_train,Predicted_Train))
```

```
0.01964854734223298
39.16973198670731
```

In [67]:
```
# Import the required packages
from sklearn.preprocessing import StandardScaler #standardizes the data to a r
from sklearn.preprocessing import MinMaxScaler #normalizes the data and brings
from sklearn.preprocessing import RobustScaler #standardizes the data. But is

#Generate the scaler
sc1= StandardScaler()
sc2= MinMaxScaler()
sc3= RobustScaler()
```

In [68]:
```
#Scaling the input data
X1 = sc1.fit_transform(knmi_updated)
X2 = sc2.fit_transform(knmi_updated)
X3 = sc3.fit_transform(knmi_updated)
```

In [69]:
```

#Split your data set into training (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X1, Energy, test_size=0.2,
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()
```

In [70]:
```
#building the regressor and fit the training data to the regressor
regr = SVR(kernel='rbf')
regr= regr.fit(X_train, y_train)
regr
```

Out[70]:
```
SVR()
```

In [73]:
```
# fitting the regression model to the training data
regr.fit(X_train, y_train) #Fit the SVM model according to the given training
# predicting on the training data
predict_train= regr.predict(X_train)
```

In [74]:
```
#testing the model training accuracy
print(r2_score(y_train, predict_train))
print(mean_squared_error(y_train, predict_train))
```

```
0.8676607662388549
5.287585695616702
```

In [75]:
```
#Predicting on the test data
pred= regr.predict(X_test)
##testing the models accuracy on the test data
print(r2_score(y_test, pred))
print(mean_squared_error(y_test, pred))
```

```
0.865011398619708
5.364047670294078
```

In [76]:
```
#Split your data set into training (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X2, Energy, test_size=0.2,
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

#building the regressor and fit the training data to the regressor
regr = SVR(kernel='rbf')
regr= regr.fit(X_train, y_train)

# fitting the regression model to the training data
regr.fit(X_train, y_train) #Fit the SVM model according to the given training
# predicting on the training data
predict_train= regr.predict(X_train)

#testing the model training accuracy
print(r2_score(y_train, predict_train))
print(mean_squared_error(y_train, predict_train))
```

```
0.8543589078950264
5.819058592240027
```

In [77]:
```
#Predicting on the test data
pred= regr.predict(X_test)
##testing the models accuracy on the test data
print(r2_score(y_test, pred))
print(mean_squared_error(y_test, pred))
```

```
0.8514063183174979
5.904673312407653
```

In [78]:
```
#Split your data set into training (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X3, Energy, test_size=0.2,
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

#building the regressor and fit the training data to the regressor
regr = SVR(kernel='rbf')
regr= regr.fit(X_train, y_train)

# fitting the regression model to the training data
regr.fit(X_train, y_train) #Fit the SVM model according to the given training
# predicting on the training data
predict_train= regr.predict(X_train)

#testing the model training accuracy
print(r2_score(y_train, predict_train))
print(mean_squared_error(y_train, predict_train))
```

```
0.861376711378752
5.538663759501532
```

In [79]:
```
#Predicting on the test data
pred= regr.predict(X_test)
##testing the models accuracy on the test data
print(r2_score(y_test, pred))
print(mean_squared_error(y_test, pred))
```

```
0.8581322813412404
5.637403439847456
```

# Same way we can compare between different karnels

In [80]:
```
#Split your data set into training (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X1, Energy, test_size=0.2,
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

#building the regressor and fit the training data to the regressor
regr = SVR(kernel='poly', degree=5) # y = ax5 + bx4 + cx3 + dx2 + ex + f

# fitting the regression model to the training data
regr.fit(X_train, y_train) #Fit the SVM model according to the given training
# predicting on the training data
predict_train= regr.predict(X_train)

#testing the model training accuracy
print(r2_score(y_train, predict_train))
print(mean_squared_error(y_train, predict_train))
```

```
0.6243431637453879
15.009288306939679
```

In [81]:
```
X4 = sc1.fit_transform(knmi.loc[:, ~knmi.columns.isin(["U"])])
```

In [82]:
```
#We redefine the data for standard scaling and split into training (80%) and t
X_train, X_test, y_train, y_test = train_test_split(X4, Energy, test_size=0.2,
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

#building the regressor and fit the training data to the regressor
regr = SVR(kernel='rbf')
regr= regr.fit(X_train, y_train)

# fitting the regression model to the training data
regr.fit(X_train, y_train) #Fit the SVM model according to the given training
# predicting on the training data
predict_train= regr.predict(X_train)

#testing the model training accuracy
print(r2_score(y_train, predict_train))
print(mean_squared_error(y_train, predict_train))
```

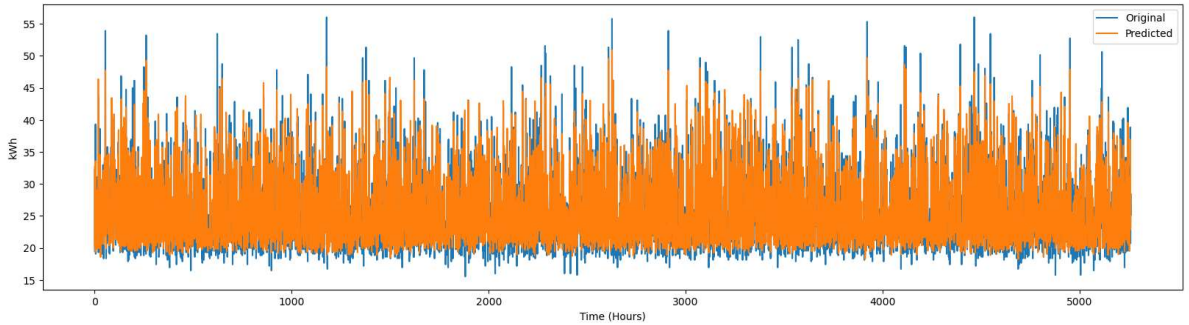
```
0.8692363907335192
5.2246319569341315
```

In [83]:
```
#Predicting on the test data
pred= regr.predict(X_test)
##testing the models accuracy on the test data
print(r2_score(y_test, pred))
print(mean_squared_error(y_test, pred))
```

```
0.865643553596894
5.338927701659936
```

In [85]:
```
plt.figure(figsize = (20,5))
plt.plot(y_test, label="Original")
plt.plot(pred, label="Predicted")
plt.legend(loc='best')
plt.xlabel('Time (Hours)')
plt.ylabel('kWh')
```

Out[85]: Text(0, 0.5, 'kWh')




In [92]:
```
from sklearn.model_selection import GridSearchCV
#settings for hyperparameters
check_parameters = {'C':[10,20,30], 'epsilon':[0.03, 0.5, 1], 'gamma':[5,6,7]}

gridsearchcv = GridSearchCV(regr, check_parameters, n_jobs=-1, cv=3)
gridsearchcv.fit(X_train, y_train)

print('Best parameters found:\n', gridsearchcv.best_params_)
```

```
Best parameters found:
 {'C': 30, 'epsilon': 0.03, 'gamma': 5}
```

In [87]:
```
# We find best_svr result: C=30, epsilon=0.03, gamma=5. Considering these para
Regr = SVR(kernel= 'rbf', C=30, epsilon = 0.03, gamma = 5)

# fitting the regression model to the training data
regr.fit(X_train, y_train) #Fit the SVM model according to the given training
# predicting on the training data
predict_train= regr.predict(X_train)

#testing the model training accuracy
print(r2_score(y_train, predict_train))
print(mean_squared_error(y_train, predict_train))
```

```
0.8692363907335192
5.2246319569341315
```

In [88]:
```
Regr = SVR(kernel= 'rbf', C=40, epsilon = 0.03, gamma = 5)

# fitting the regression model to the training data
regr.fit(X_train, y_train) #Fit the SVM model according to the given training
# predicting on the training data
predict_train= regr.predict(X_train)

#testing the model training accuracy
print(r2_score(y_train, predict_train))
print(mean_squared_error(y_train, predict_train))
```

```
0.8692363907335192
5.2246319569341315
```

# Check the RF regressor model performance

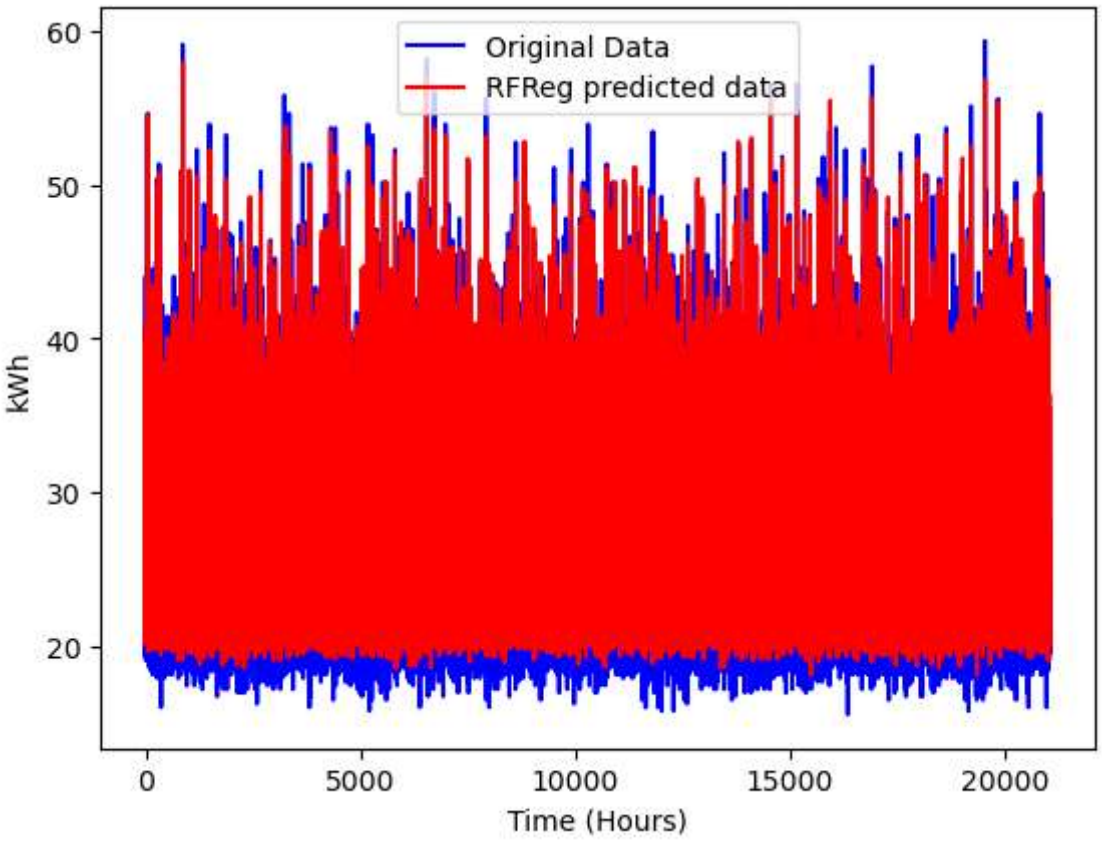
In [89]:
```
#importing the ensemble module for the random forest regressor from sklearn li
from sklearn.ensemble import RandomForestRegressor

# Creating an instance of the random forest regressor
RFReg = RandomForestRegressor(max_depth=10, random_state=0)

# fitting the regression model to the training data
X_train2, X_test2, y_train2, y_test2 = train_test_split(X1, Energy, test_size=
y_train2 = y_train2.values.ravel()
y_test2 = y_test2.values.ravel()
RFReg.fit(X_train2, y_train2)

#Predicting on the training data
Predicted_Train2= RFReg.predict(X_train2)

#Caculating R2 score and Root mean square error
print(r2_score(y_train2, Predicted_Train2))
print(mean_squared_error(y_train2, Predicted_Train2))
```

```
0.9160860073645251
3.3527655745857405
```

In [90]:
```
#importing the ensemble module for the random forest regressor from sklearn li
from sklearn.ensemble import RandomForestRegressor

# Creating an instance of the random forest regressor
RFReg = RandomForestRegressor(max_depth=10, random_state=0)

# fitting the regression model to the training data
X_train2, X_test2, y_train2, y_test2 = train_test_split(X4, Energy, test_size=
y_train2 = y_train2.values.ravel()
y_test2 = y_test2.values.ravel()
RFReg.fit(X_train2, y_train2)

#Predicting on the training data
Predicted_Train2= RFReg.predict(X_train2)

#Caculating R2 score and Root mean square error
print(r2_score(y_train2, Predicted_Train2))
print(mean_squared_error(y_train2, Predicted_Train2))
```

```
0.916772327031522
3.325343819519643
```

In [91]:
```
# Lets visualise our fit to the training data.
plt.plot(y_train2, color="b",label= 'Original Data')
plt.plot(Predicted_Train2, color ="red", label="RFReg predicted data")
plt.xlabel('Time (Hours)')
plt.ylabel('kWh')
plt.legend(loc='best')
plt.show()
```

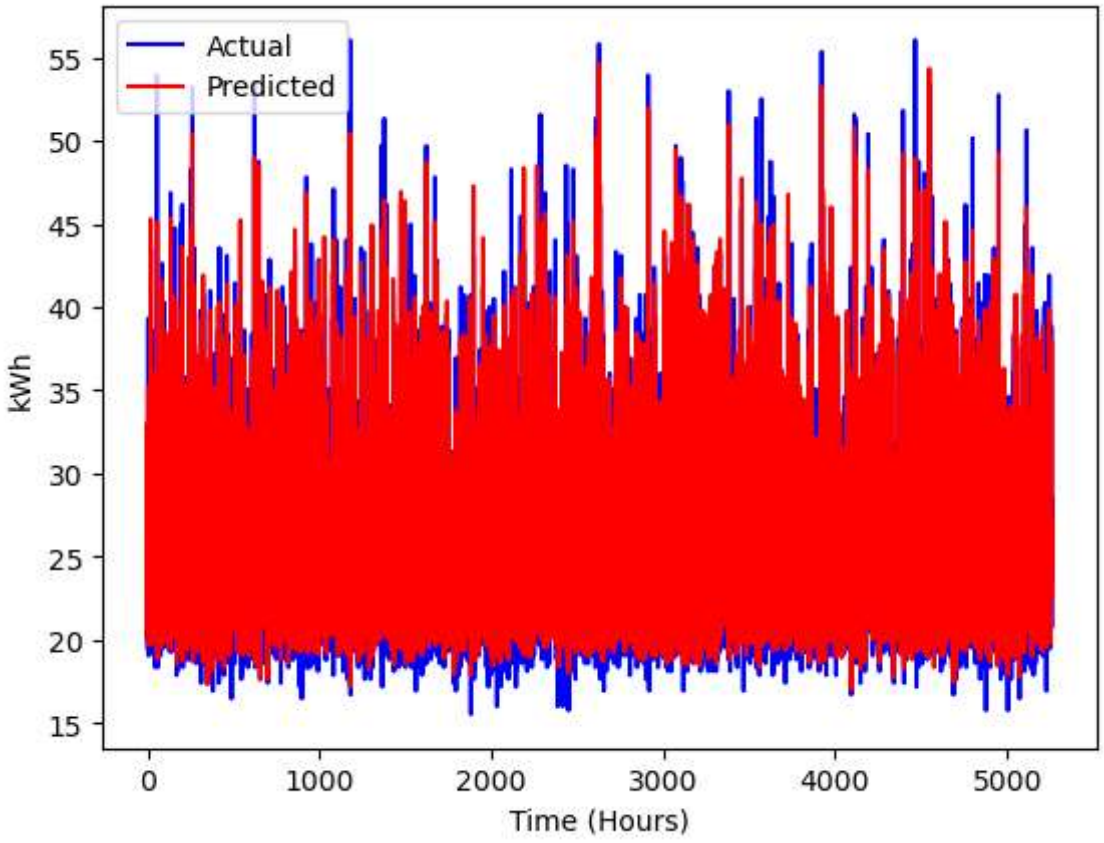


In [111]:
```
#Predicting on the test set (X_test)
Predicted_Test2 = RFReg.predict(X_test2)

#Caculating R2 score and Root mean square error
print(r2_score(y_test2,Predicted_Test2))
print(mean_squared_error(y_test2,Predicted_Test2))
```

```
0.9059580709782538
3.736948046879361
```

In [112]:
```
# Lets visualise our fit to the test data.
plt.plot(y_test2, color='blue', label="Actual")
plt.plot(Predicted_Test2, color='Red', label="Predicted")
plt.xlabel('Time (Hours)')
plt.ylabel('kWh')
plt.legend(loc='best')
```

Out[112]: <matplotlib.legend.Legend at 0x1adc79ecdc0>




In [113]:
```
from sklearn.model_selection import GridSearchCV
```

In [114]:
```
#settings for hyperparameters
check_parameters = {'max_depth':[8,9,11,12]}

gridsearchcv = GridSearchCV(RFReg, check_parameters, n_jobs=-1, cv=3)
gridsearchcv.fit(X_train, y_train)

print('Best parameters found:\n', gridsearchcv.best_params_)
```

```
Best parameters found:
 {'max_depth': 12}
```

In [115]:
```
#importing the ensemble module for the random forest regressor from sklearn li
from sklearn.ensemble import RandomForestRegressor

# Creating an instance of the random forest regressor
RFReg = RandomForestRegressor(max_depth=12, random_state=0)

# fitting the regression model to the training data
X_train2, X_test2, y_train2, y_test2 = train_test_split(X4, Energy, test_size=
y_train2 = y_train2.values.ravel()
y_test2 = y_test2.values.ravel()
RFReg.fit(X_train2, y_train2)

#Predicting on the training data
Predicted_Train2= RFReg.predict(X_train2)

#Caculating R2 score and Root mean square error
print(r2_score(y_train2, Predicted_Train2))
print(mean_squared_error(y_train2, Predicted_Train2))
```

```
0.9413680428300156
2.3426272710447176
```

In [116]:
```
#Predicting on the test set (X_test)
Predicted_Test2 = RFReg.predict(X_test2)

#Caculating R2 score and Root mean square error
print(r2_score(y_test2,Predicted_Test2))
print(mean_squared_error(y_test2,Predicted_Test2))
```

```
0.8914761615776285
4.312416283363921
```

In [117]:
```
#settings for hyperparameters
check_parameters = {'max_depth':[15,20,30]}

gridsearchcv = GridSearchCV(RFReg, check_parameters, n_jobs=-1, cv=10)
gridsearchcv.fit(X_train, y_train)

print('Best parameters found:\n', gridsearchcv.best_params_)
```

```
Best parameters found:
 {'max_depth': 30}
```

In [118]:
```
#importing the ensemble module for the random forest regressor from sklearn li
from sklearn.ensemble import RandomForestRegressor

# Creating an instance of the random forest regressor
RFReg = RandomForestRegressor(max_depth=30, random_state=0)

# fitting the regression model to the training data
X_train2, X_test2, y_train2, y_test2 = train_test_split(X4, Energy, test_size=
y_train2 = y_train2.values.ravel()
y_test2 = y_test2.values.ravel()
RFReg.fit(X_train2, y_train2)

#Predicting on the training data
Predicted_Train2= RFReg.predict(X_train2)

#Caculating R2 score and Root mean square error
print(r2_score(y_train2, Predicted_Train2))
print(mean_squared_error(y_train2, Predicted_Train2))
```

```
0.9871735597631518
0.5124776681452499
```

In [119]:
```
#Predicting on the test set (X_test)
Predicted_Test2 = RFReg.predict(X_test2)

#Caculating R2 score and Root mean square error
print(r2_score(y_test2,Predicted_Test2))
print(mean_squared_error(y_test2,Predicted_Test2))
```

```
0.9059580709782538
3.736948046879361
```

# Allocate budget using predictive modeling

In [120]:
```
# Import the weather cost file
weather_cost = pd.read_excel('Weather_Cost.xlsx')
weather_cost
```

Out[120]:

| | Time | month | HH | TD | U | Temp | RH | Q | DR | FF | FX | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2019-01-01 00:00:00 | 1 | 1 | 68 | 96 | 73 | 1 | 0 | 6 | 40 | 90 | 10323 |
| **1** | 2019-01-01 01:00:00 | 1 | 2 | 65 | 94 | 74 | -1 | 0 | 0 | 40 | 70 | 10320 |
| **2** | 2019-01-01 02:00:00 | 1 | 3 | 63 | 93 | 73 | 0 | 0 | 0 | 40 | 70 | 10314 |
| **3** | 2019-01-01 03:00:00 | 1 | 4 | 61 | 92 | 73 | 0 | 0 | 0 | 50 | 60 | 10308 |
| **4** | 2019-01-01 04:00:00 | 1 | 5 | 58 | 92 | 69 | 0 | 0 | 0 | 50 | 70 | 10299 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **739** | 2019-01-31 19:00:00 | 1 | 20 | -24 | 93 | -15 | 0 | 0 | 0 | 30 | 60 | 9929 |
| **740** | 2019-01-31 20:00:00 | 1 | 21 | -22 | 95 | -15 | 0 | 0 | 0 | 30 | 60 | 9920 |
| **741** | 2019-01-31 21:00:00 | 1 | 22 | -24 | 91 | -11 | 0 | 0 | 0 | 40 | 70 | 9911 |
| **742** | 2019-01-31 22:00:00 | 1 | 23 | -25 | 87 | -6 | 0 | 0 | 0 | 50 | 80 | 9900 |
| **743** | 2019-01-31 23:00:00 | 1 | 24 | -25 | 86 | -4 | 0 | 0 | 0 | 50 | 90 | 9893 |

744 rows × 12 columns

In [121]:
```
# Make time column as index
weather_cost = weather_cost.set_index('Time')
```

In [122]:
```
#check missing value
weather_cost.isna().sum()
```

Out[122]:
```
month    0
HH       0
TD       0
U        0
Temp     0
RH       0
Q        0
DR       0
FF       0
FX       0
P        0
dtype: int64
```

In [123]:
```
#remove relative humidity column from the data set
weather_cost_updated= weather_cost.loc[:, ~weather_cost.columns.isin(['U'])]
```

In [124]:
```
#scale the input data
X5 = sc1.transform(weather_cost_updated)
```

In [125]:
```
#predict the consumption
predicted = RFReg.predict(X5)
predicted.shape
```

Out[125]: (744,)

In [126]:
```
#Converting the predicted array into a dataframe so it is easier when plotting
predicted= pd.DataFrame(predicted, columns=['kWh'])
predicted
```

Out[126]:

| | kWh |
|---|---|
| **0** | 45.623766 |
| **1** | 45.343547 |
| **2** | 45.374159 |
| **3** | 45.506027 |
| **4** | 45.400062 |
| **...** | ... |
| **739** | 22.660000 |
| **740** | 22.629388 |
| **741** | 22.605840 |
| **742** | 22.563454 |
| **743** | 22.309138 |

744 rows × 1 columns

In [127]:
```
#Import the index from the weather cost file
predicted['Time']= weather_cost.index
predicted
```

Out[127]:

| | kWh | Time |
|---|---|---|
| **0** | 45.623766 | 2019-01-01 00:00:00 |
| **1** | 45.343547 | 2019-01-01 01:00:00 |
| **2** | 45.374159 | 2019-01-01 02:00:00 |
| **3** | 45.506027 | 2019-01-01 03:00:00 |
| **4** | 45.400062 | 2019-01-01 04:00:00 |
| **...** | ... | ... |
| **739** | 22.660000 | 2019-01-31 19:00:00 |
| **740** | 22.629388 | 2019-01-31 20:00:00 |
| **741** | 22.605840 | 2019-01-31 21:00:00 |
| **742** | 22.563454 | 2019-01-31 22:00:00 |
| **743** | 22.309138 | 2019-01-31 23:00:00 |

744 rows × 2 columns

In [128]:
```
#Set the time column as index
predicted= predicted.set_index('Time')
predicted
```

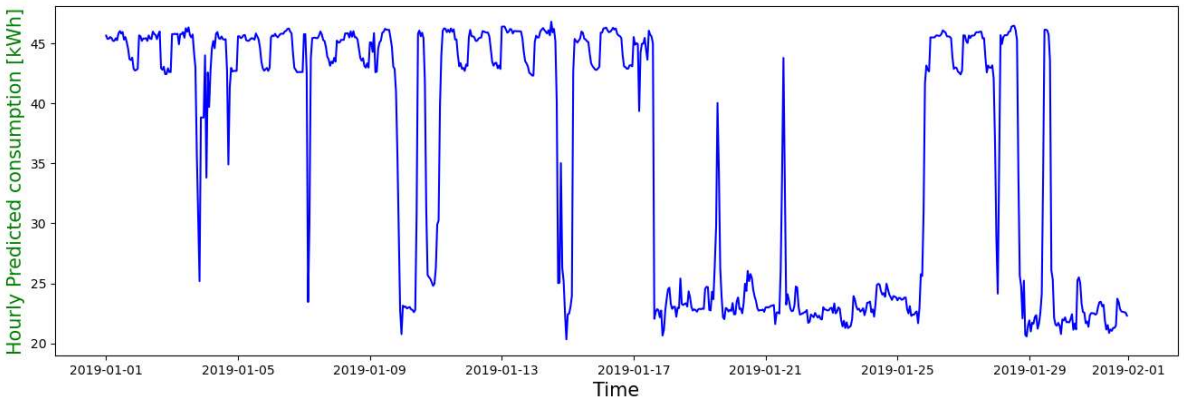
Out[128]:

| | kWh |
|---|---|
| **Time** | |
| **2019-01-01 00:00:00** | 45.623766 |
| **2019-01-01 01:00:00** | 45.343547 |
| **2019-01-01 02:00:00** | 45.374159 |
| **2019-01-01 03:00:00** | 45.506027 |
| **2019-01-01 04:00:00** | 45.400062 |
| **...** | ... |
| **2019-01-31 19:00:00** | 22.660000 |
| **2019-01-31 20:00:00** | 22.629388 |
| **2019-01-31 21:00:00** | 22.605840 |
| **2019-01-31 22:00:00** | 22.563454 |
| **2019-01-31 23:00:00** | 22.309138 |

744 rows × 1 columns

In [129]:
```
#Plot the hourly forecast consumption in kWh
fig, ax = plt.subplots(figsize = (16,5))
ax.plot(predicted,  label='Hourly Predicted consumption',color = 'blue')
ax.set_ylabel('Hourly Predicted consumption [kWh]',size=15, color='green')
ax.set_xlabel('Time',size=15)
```

Out[129]:
```
Text(0.5, 0, 'Time')
```




In [130]:
```
#Calculating the hourly consumtion cost.
Hourly_Cost= predicted*0.23
Hourly_Cost
```

Out[130]:

| Time | kWh |
|---|---|
| **2019-01-01 00:00:00** | 10.493466 |
| **2019-01-01 01:00:00** | 10.429016 |
| **2019-01-01 02:00:00** | 10.436057 |
| **2019-01-01 03:00:00** | 10.466386 |
| **2019-01-01 04:00:00** | 10.442014 |
| **...** | ... |
| **2019-01-31 19:00:00** | 5.211800 |
| **2019-01-31 20:00:00** | 5.204759 |
| **2019-01-31 21:00:00** | 5.199343 |
| **2019-01-31 22:00:00** | 5.189594 |
| **2019-01-31 23:00:00** | 5.131102 |

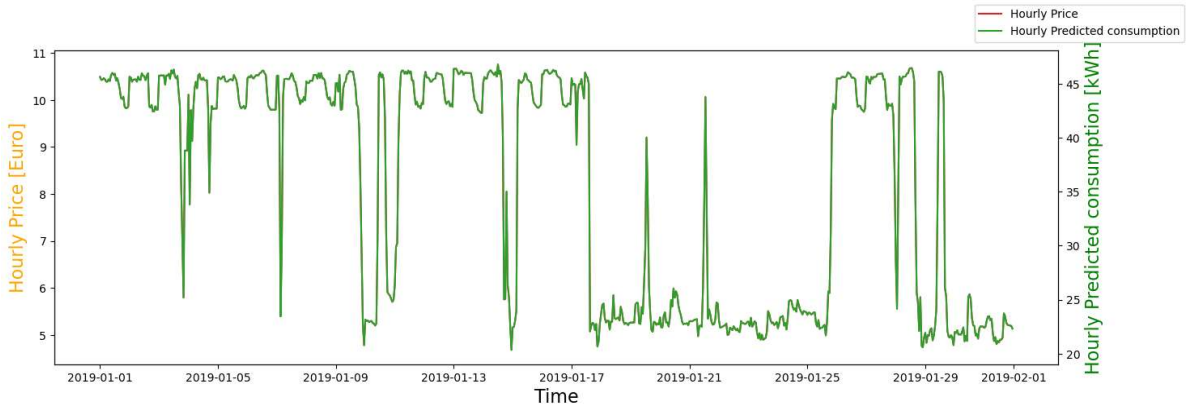
744 rows × 1 columns

In [131]:
```
#Resampling the hourly consumption charges into daily by using the resample fu
Daily_Cost = Hourly_Cost.resample("D").sum()

print("total cost", Daily_Cost.sum())
```

```
total cost kWh    6120.192351
dtype: float64
```

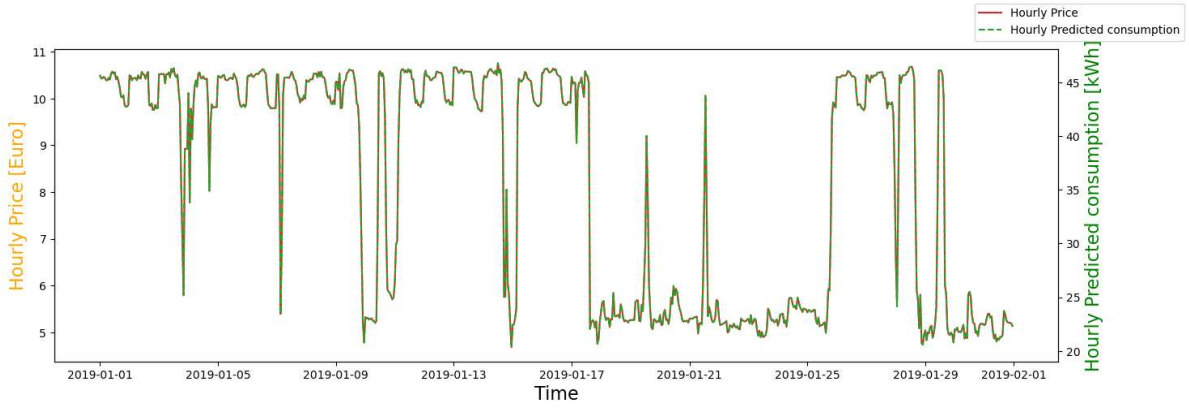
In [132]:
```
fig, ax = plt.subplots(figsize = (16,5))
ax2 = ax.twinx() # Create another axes that shares the same x-axis as ax.
ax.plot(Hourly_Cost,  label='Hourly Price',color = 'tab:red')
ax2.plot(predicted,  label='Hourly Predicted consumption',color = 'tab:green')
ax.set_ylabel('Hourly Price [Euro]', size=16, color='orange')
ax2.set_ylabel('Hourly Predicted consumption [kWh]',size=16, color='green')
ax.set_xlabel('Time',size=16,)
fig.legend()
```

Out[132]:
```
<matplotlib.legend.Legend at 0x1adb681ceb0>
```



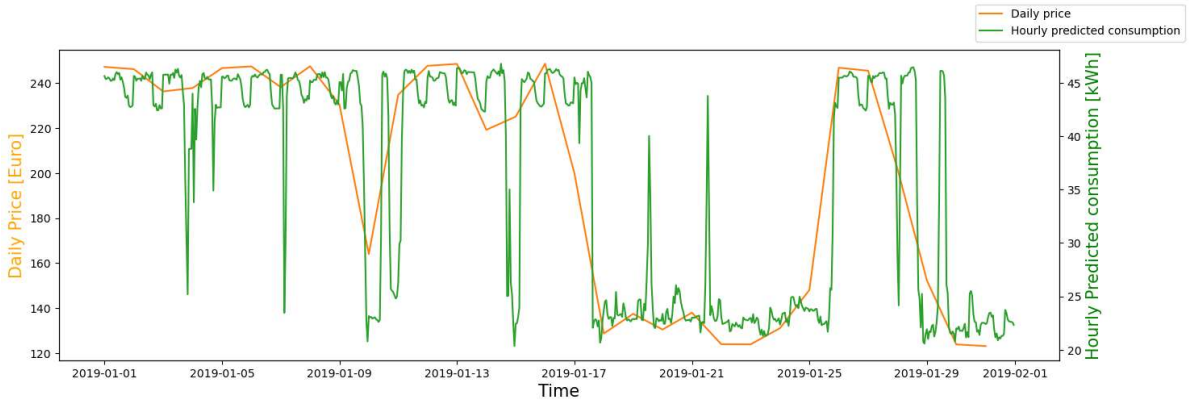
In [133]:
```
fig, ax = plt.subplots(figsize = (16,5))
ax2 = ax.twinx() # Create another axes that shares the same x-axis as ax.
ax.plot(Hourly_Cost,  label='Hourly Price',color = 'tab:red')
ax2.plot(predicted,  label='Hourly Predicted consumption',color = 'tab:green',
ax.set_ylabel('Hourly Price [Euro]', size=16, color='orange')
ax2.set_ylabel('Hourly Predicted consumption [kWh]',size=16, color='green')
ax.set_xlabel('Time',size=16)
fig.legend()
```

Out[133]:
```
<matplotlib.legend.Legend at 0x1adb6280a30>
```




In [134]:
```
fig, ax = plt.subplots(figsize=(16,5))
ax2 = ax.twinx() # Create another axes that shares the same x-axis as ax.
ax.plot(Daily_Cost, label= 'Daily price', color = 'tab:orange')
ax2.plot(predicted, label='Hourly predicted consumption', color = 'tab:green')
ax.set_ylabel('Daily Price [Euro]', size=15, color='orange')
ax2.set_ylabel('Hourly Predicted consumption [kWh]',size=15, color='green')
ax.set_xlabel('Time',size=15)
fig.legend()
```

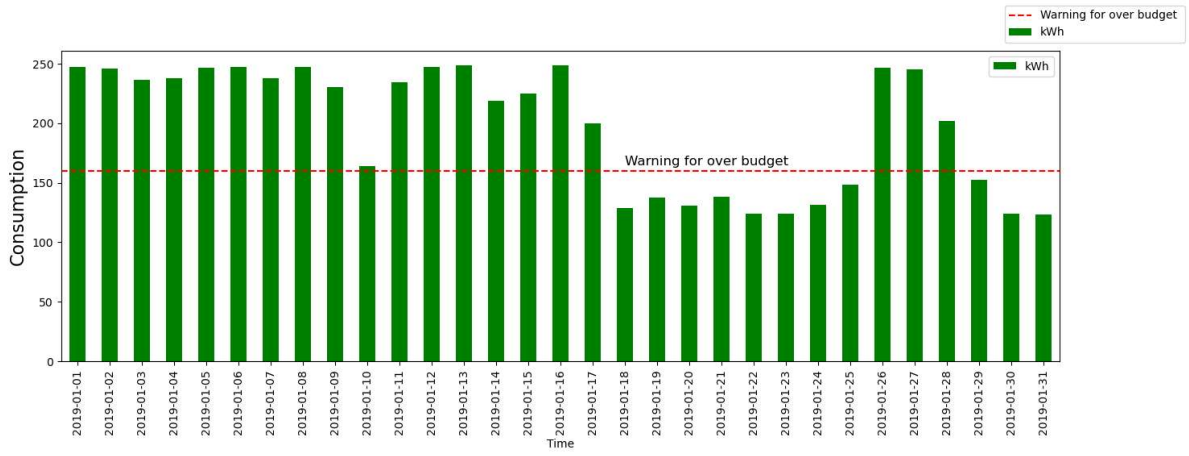
Out[134]:
```
<matplotlib.legend.Legend at 0x1adc77da490>
```




# Set a visual threshold in the forecast when the model predicts higher than a budget limit

In [135]:
```
fig = plt.figure(figsize = (16,5)) # Create matplotlib figure
ax = fig.add_subplot(111) # Create matplotlib axes
Daily_Cost.plot(kind='bar', ax=ax, rot=0,color='green')
ax.axhline(y=160, color='red', linestyle='--', label="Warning for over budget
plt.text(17, 165, 'Warning for over budget', fontsize=12)

ax.set_ylabel('Consumption', size=16, color='black')
plt.xticks(rotation='vertical')
ax.set_xticklabels([dt.strftime('%Y-%m-%d') for dt in Daily_Cost.index])
fig.legend()
```

Out[135]: <matplotlib.legend.Legend at 0x1adc7c3cca0>




The maximum daily allocated budget for the building is 160 euros. A visual threshold is set for when the model predicts a cost which is higher than the maximum budget. A bar graph is used to identify if the daily consumption exceeds the budget. As can be seen, the daily cost of energy continuously exceeds the set budget of 160 euros.

In [ ]:
```
```