Projeto_08_v2

October 30, 2019

1 Formação Cientista de Dados - DSA

1.0.1 Machine Learning

1.1 Projeto com Feedback 8 - Modelagem Preditiva em IoT - Previsão de Uso de Energia

1.1.1 Leonardo Molero

2 Análise Exploratória

[1]:
```
# Importação dos pacotes
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn import svm
from xgboost import XGBRegressor
from sklearn.model_selection import RandomizedSearchCV

# Faz ajustes para não exibir warnings
warnings.filterwarnings("ignore")

# Parametriza impressão dos gráficos dentro do notebook
%matplotlib inline

# Configura fundo azul com barras brancas para os gráficos
sns.set(color_codes=True)

# Configura a exibição da borda das barras nos gráficos
plt.rcParams["patch.force_edgecolor"] = True
```

[2]:

```
## Descrição das variáveis
### date: Data da coleta dos dados pelos sensores (datetime)
### Appliances: Uso de energia (em W)
### lights: Potência de energia de eletrodomesticos na casa (em W)
### TXX: Temperatura em um lugar da casa (em Celsius)
### RH_XX: Umidade em um lugar da casa (em %)
### T_out:Temperatura externa (em Celsius)
### Press_mm_hg: Pressão externa (em mmHg)
### RH_out: Umidade externa (em %)
### Windspeed: Velocidade do vento (em m/s)
### Visibility: Visibilidade (em Km)
### Tdewpoint: Temperatura do Ponto de Orvalho (em Celsius)
### rv1: Variável aleatória 01
### rv2: Variável aleatória 02
### WeekStatus: Dia útil ou final de semana (weekend ou weekday)
### Day_of_week: Dia da semana
### NSM: Medida do tempo (em segundos)
```

[3]:

```
# Carrega o dados de treino e de teste usando a coluna 'date' com index
df = pd.read_csv('dados\projeto8-training.csv', index_col='date')
dft = pd.read_csv('dados\projeto8-testing.csv', index_col='date')
```

[4]:

```
# Checa o tamanho do dataframe de treino
print(df.shape)
```

```
(14803, 31)
```

[5]:

```
# Checa o tamanho do dataframe de teste
print(dft.shape)
```

```
(4932, 31)
```

[6]:

```
# Visualiza os dados treino
df.head(10)
```

[6]:

```
                     Appliances  lights         T1       RH_1         T2  \
date
2016-01-11 17:00:00          60      30  19.890000  47.596667  19.200000
2016-01-11 17:10:00          60      30  19.890000  46.693333  19.200000
2016-01-11 17:20:00          50      30  19.890000  46.300000  19.200000
2016-01-11 17:40:00          60      40  19.890000  46.333333  19.200000
2016-01-11 17:50:00          50      40  19.890000  46.026667  19.200000
2016-01-11 18:10:00          60      50  19.856667  45.560000  19.200000
2016-01-11 18:20:00          60      40  19.790000  45.597500  19.200000
2016-01-11 18:30:00          70      40  19.856667  46.090000  19.230000
2016-01-11 19:00:00         430      50  20.133333  48.000000  19.566667
2016-01-11 19:10:00         250      40  20.260000  52.726667  19.730000
```

```
                         RH_2     T3       RH_3         T4       RH_4  …  \
date                                                                   …
2016-01-11 17:00:00  44.790000  19.79  44.730000  19.000000  45.566667  …
2016-01-11 17:10:00  44.722500  19.79  44.790000  19.000000  45.992500  …
2016-01-11 17:20:00  44.626667  19.79  44.933333  18.926667  45.890000  …
2016-01-11 17:40:00  44.530000  19.79  45.000000  18.890000  45.530000  …
2016-01-11 17:50:00  44.500000  19.79  44.933333  18.890000  45.730000  …
2016-01-11 18:10:00  44.500000  19.73  44.900000  18.890000  45.863333  …
2016-01-11 18:20:00  44.433333  19.73  44.790000  18.890000  45.790000  …
2016-01-11 18:30:00  44.400000  19.79  44.863333  18.890000  46.096667  …
2016-01-11 19:00:00  44.400000  19.89  44.900000  19.000000  46.363333  …
2016-01-11 19:10:00  45.100000  19.89  45.493333  19.000000  47.223333  …

                     Press_mm_hg     RH_out  Windspeed  Visibility  Tdewpoint  \
date
2016-01-11 17:00:00   733.500000  92.000000   7.000000   63.000000   5.300000
2016-01-11 17:10:00   733.600000  92.000000   6.666667   59.166667   5.200000
2016-01-11 17:20:00   733.700000  92.000000   6.333333   55.333333   5.100000
2016-01-11 17:40:00   733.900000  92.000000   5.666667   47.666667   4.900000
2016-01-11 17:50:00   734.000000  92.000000   5.333333   43.833333   4.800000
2016-01-11 18:10:00   734.166667  91.833333   5.166667   40.000000   4.683333
2016-01-11 18:20:00   734.233333  91.666667   5.333333   40.000000   4.666667
2016-01-11 18:30:00   734.300000  91.500000   5.500000   40.000000   4.650000
2016-01-11 19:00:00   734.500000  91.000000   6.000000   40.000000   4.600000
2016-01-11 19:10:00   734.616667  90.500000   6.000000   40.000000   4.516667

                           rv1        rv2    NSM  WeekStatus  Day_of_week
date
2016-01-11 17:00:00  13.275433  13.275433  61200     Weekday       Monday
2016-01-11 17:10:00  18.606195  18.606195  61800     Weekday       Monday
2016-01-11 17:20:00  28.642668  28.642668  62400     Weekday       Monday
2016-01-11 17:40:00  10.084097  10.084097  63600     Weekday       Monday
2016-01-11 17:50:00  44.919484  44.919484  64200     Weekday       Monday
2016-01-11 18:10:00  33.039890  33.039890  65400     Weekday       Monday
2016-01-11 18:20:00  31.455702  31.455702  66000     Weekday       Monday
2016-01-11 18:30:00   3.089314   3.089314  66600     Weekday       Monday
2016-01-11 19:00:00  34.351142  34.351142  68400     Weekday       Monday
2016-01-11 19:10:00  19.205186  19.205186  69000     Weekday       Monday

[10 rows x 31 columns]
```

[7]:

```python
# Verifica os tipos das colunas
df.dtypes
```

[7]:

```
Appliances          int64
lights              int64
```

```
T1               float64
RH_1             float64
T2               float64
RH_2             float64
T3               float64
RH_3             float64
T4               float64
RH_4             float64
T5               float64
RH_5             float64
T6               float64
RH_6             float64
T7               float64
RH_7             float64
T8               float64
RH_8             float64
T9               float64
RH_9             float64
T_out            float64
Press_mm_hg      float64
RH_out           float64
Windspeed        float64
Visibility       float64
Tdewpoint        float64
rv1              float64
rv2              float64
NSM                int64
WeekStatus        object
Day_of_week       object
dtype: object
```

[8]:
```
# Verifica os tipos de colunas agrupados (devido a quantidade de colunas)
df.dtypes.value_counts()
```

[8]:
```
float64    26
int64       3
object      2
dtype: int64
```

[9]:
```
# Verifica estatísticas dos dados
df.describe()
```

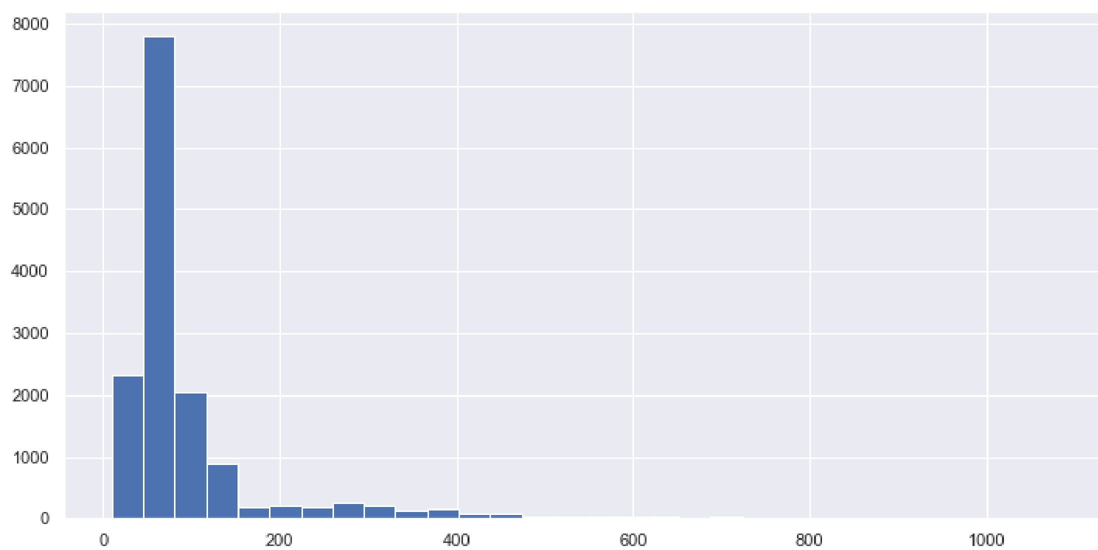
[9]:
```
         Appliances        lights            T1          RH_1            T2  \
count  14803.000000  14803.000000  14803.000000  14803.000000  14803.000000
mean      98.011214      3.802608     21.684167     40.271439     20.342506
std      102.828019      7.940340      1.607780      3.981741      2.191842
min       10.000000      0.000000     16.790000     27.023333     16.100000
```

```
25%      50.000000      0.000000     20.730000     37.361667     18.823333
50%      60.000000      0.000000     21.600000     39.656667     20.000000
75%     100.000000      0.000000     22.600000     43.090000     21.500000
max    1080.000000     50.000000     26.260000     63.360000     29.856667

               RH_2            T3          RH_3            T4          RH_4  \
count  14803.000000  14803.000000  14803.000000  14803.000000  14803.000000
mean      40.418056     22.262628     39.249149     20.855433     39.030359
std        4.066223      2.013785      3.252755      2.044786      4.339783
min       20.893333     17.200000     28.766667     15.100000     27.660000
25%       37.900000     20.790000     36.900000     19.500000     35.530000
50%       40.500000     22.100000     38.530000     20.666667     38.400000
75%       43.290000     23.290000     41.761667     22.100000     42.130000
max       56.026667     29.236000     50.163333     26.200000     51.063333

       …          RH_9         T_out   Press_mm_hg        RH_out  \
count  …  14803.000000  14803.000000  14803.000000  14803.000000
mean   …     41.542065      7.413018    755.502983     79.734122
std    …      4.150839      5.323843      7.427684     14.955609
min    …     29.166667     -5.000000    729.300000     24.000000
25%    …     38.500000      3.666667    750.866667     70.000000
50%    …     40.863333      6.900000    756.100000     83.666667
75%    …     44.363333     10.400000    760.933333     91.666667
max    …     53.326667     25.966667    772.300000    100.000000

          Windspeed    Visibility     Tdewpoint           rv1           rv2  \
count  14803.000000  14803.000000  14803.000000  14803.000000  14803.000000
mean       4.034470     38.330141      3.756859     25.078087     25.078087
std        2.436870     11.812780      4.200297     14.481537     14.481537
min        0.000000      1.000000     -6.600000      0.005322      0.005322
25%        2.000000     29.000000      0.900000     12.580425     12.580425
50%        3.666667     40.000000      3.450000     25.043993     25.043993
75%        5.500000     40.000000      6.533333     37.665907     37.665907
max       13.500000     66.000000     15.500000     49.996530     49.996530

                NSM
count  14803.000000
mean   42985.989326
std    24968.649028
min        0.000000
25%    21600.000000
50%    43200.000000
75%    64800.000000
max    85800.000000

[8 rows x 29 columns]
```

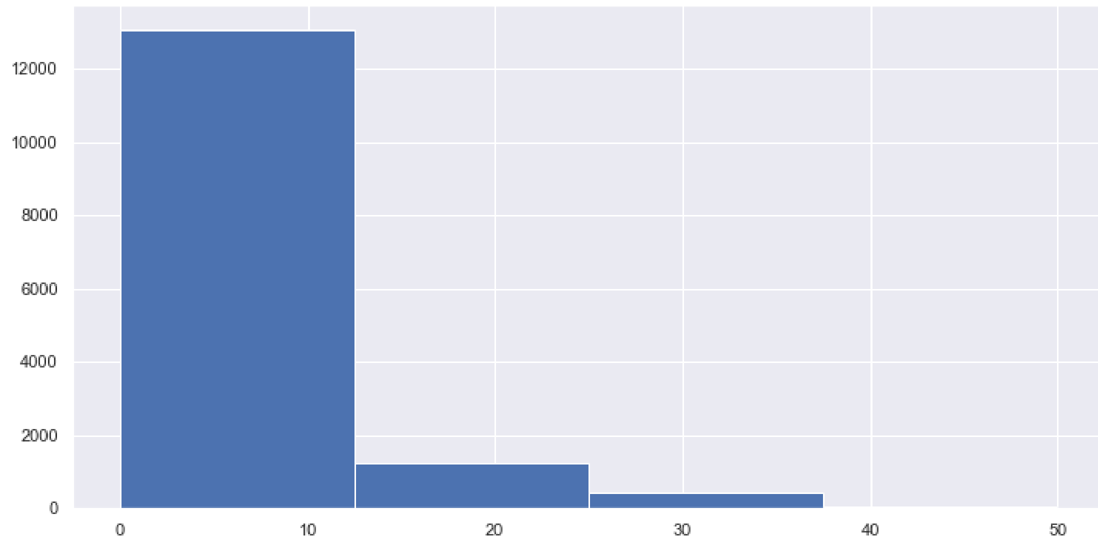
[10]:
```python
# Verifica a distribuição da variável alvo
df['Appliances'].value_counts()
```

[10]:
```
50      3275
60      2462
40      1488
70      1156
80       912
        …
1080       1
780        1
860        1
900        1
910        1
Name: Appliances, Length: 88, dtype: int64
```

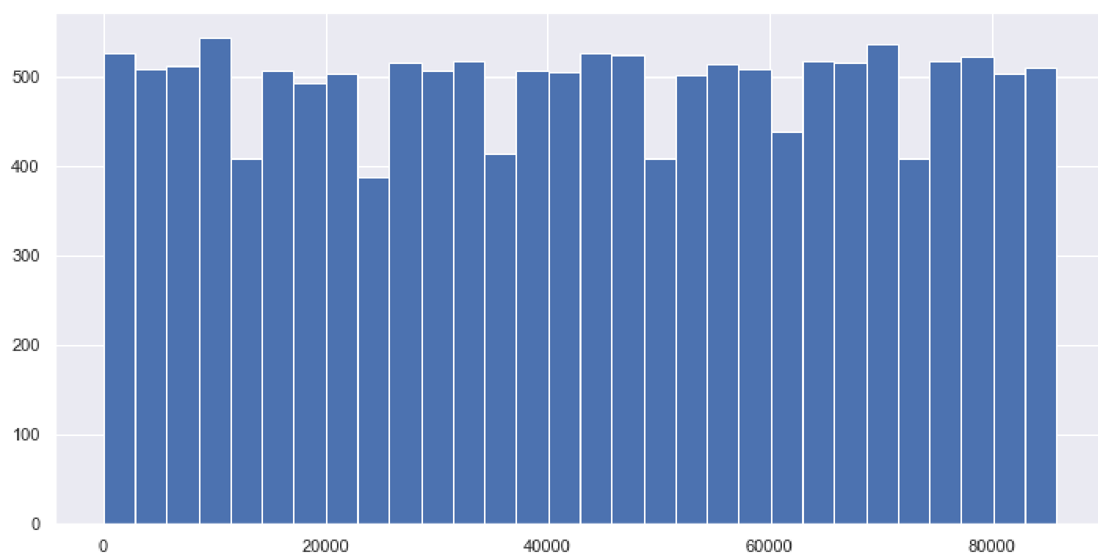
[11]:
```python
# Histograma com a distribuição da variável alvo
fig,ax = plt.subplots(figsize=(12,6))
plt.hist('Appliances',data=df,bins=30)
plt.show()
```



[12]:
```python
# Histograma com a distribuição da variável 'lights'
fig,ax = plt.subplots(figsize=(12,6))
plt.hist('lights',data=df, bins=4)
plt.show()
```



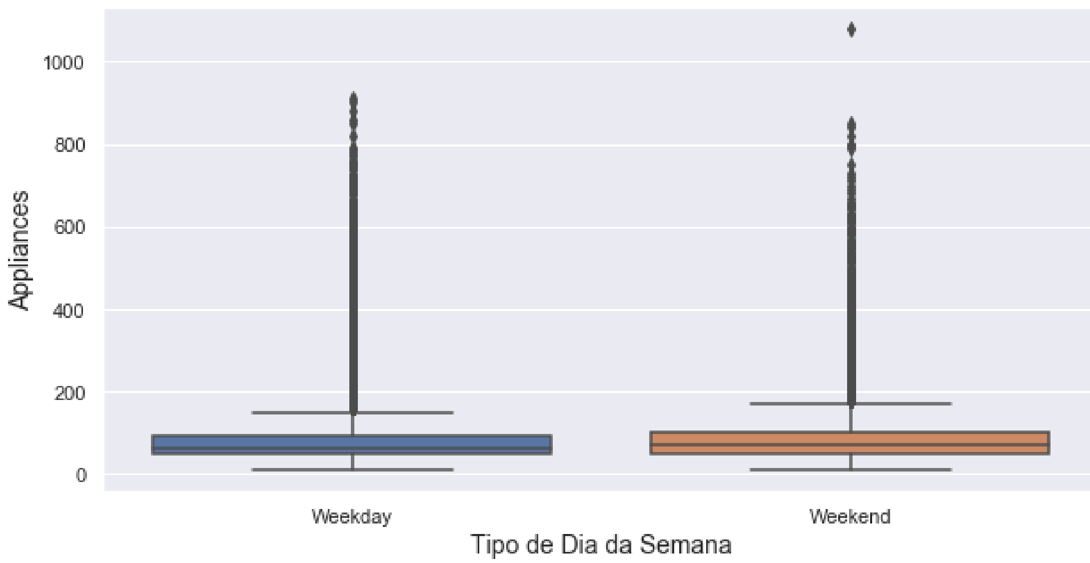
[13]:
```python
# Histograma com a distribuição da variável 'NSM'
fig,ax = plt.subplots(figsize=(12,6))
plt.hist('NSM',data=df, bins=30)
plt.show()
```



[14]:
```python
# Distribuição dos do tipo de dia da semana
df['WeekStatus'].value_counts()
```

[14]:
```
Weekday     10720
Weekend      4083
Name: WeekStatus, dtype: int64
```

[15]:
```
# Verifica a variável alvo por tipo de dia da semana
fig, ax = plt.subplots(figsize=(10,5))
sns.boxplot(x='WeekStatus', y='Appliances',data=df)
ax.xaxis.set_label_text("Tipo de Dia da Semana",fontdict= {'size':14})
ax.yaxis.set_label_text("Appliances",fontdict= {'size':14})
plt.show()
```

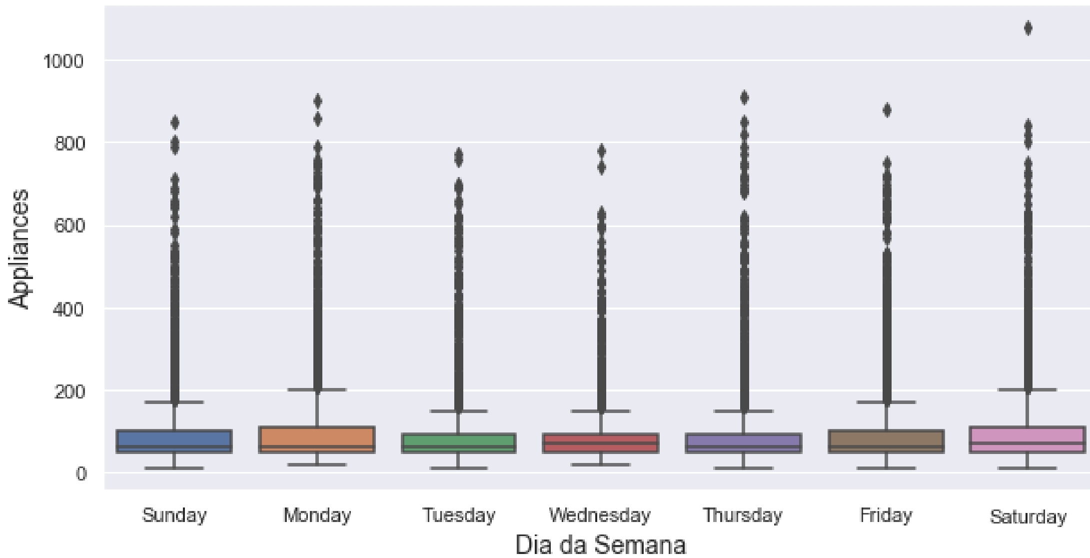


[16]:
```
# Distribuição dos dias da semana
df['Day_of_week'].value_counts()
```

[16]:
```
Wednesday    2170
Tuesday      2161
Friday       2157
Thursday     2131
Monday       2101
Sunday       2061
Saturday     2022
Name: Day_of_week, dtype: int64
```

[17]:
```
# Verifica a variável alvo por dia da semana
fig, ax = plt.subplots(figsize=(10,5))
sns.boxplot(x='Day_of_week',␣
 ↪y='Appliances',data=df,order=['Sunday','Monday','Tuesday',
```

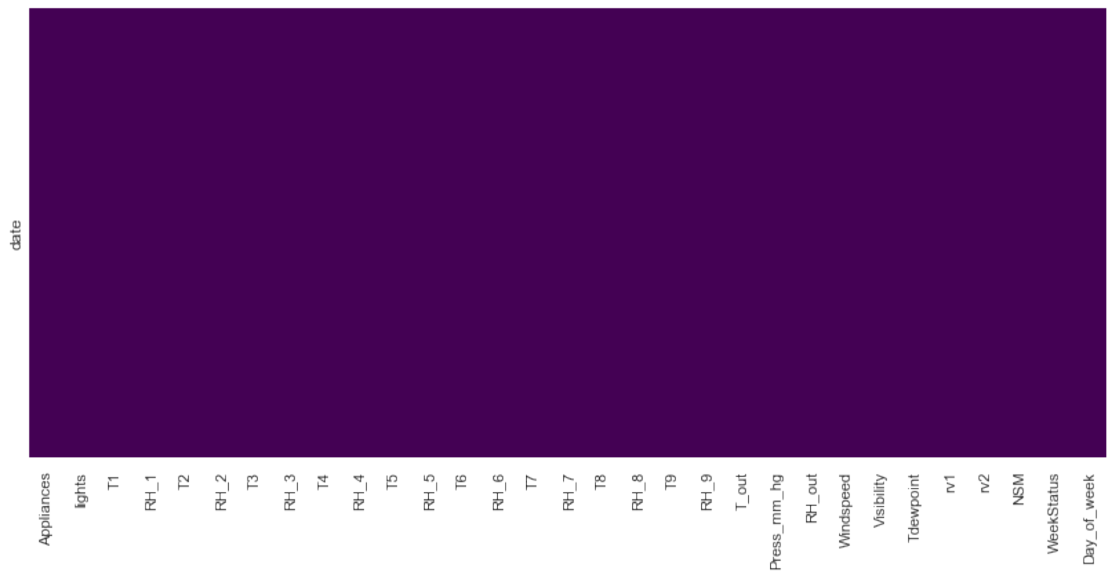
```
↪'Wednesday','Thursday','Friday','Saturday'])
ax.xaxis.set_label_text("Dia da Semana",fontdict= {'size':14})
ax.yaxis.set_label_text("Appliances",fontdict= {'size':14})
plt.show()
```



[18]:
```
# Procura e localiza valores nulos
print(df.isnull().values.any())
fig, ax = plt.subplots(figsize=(14,6))
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
False
```

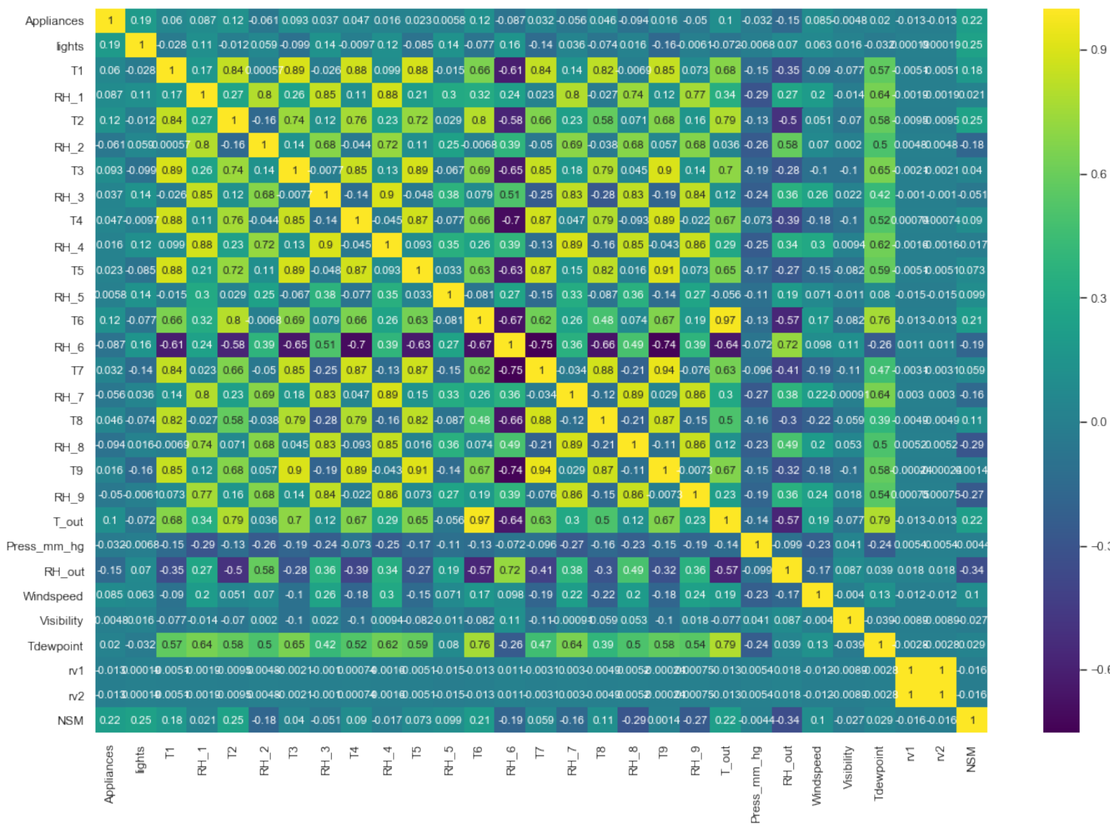
[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23b3478aef0>
```



3 Pré-processamento

[19]:
```
# Verifica a correlação das variáveis numéricas
fig,ax = plt.subplots(figsize=(18,12))
sns.heatmap(df.corr(),annot=True,cmap='viridis')
```

[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23b3438d400>
```



[20]:
```
# Transforma as variáveis texto em marcações numéricas
le = LabelEncoder()
le.fit(df['WeekStatus'])
df['WeekStatus'] = le.transform(df['WeekStatus'])
le.fit(df['Day_of_week'])
df['Day_of_week'] = le.transform(df['Day_of_week'])
```

[21]:
```
# Separa as variáveis preditoras da variável alvo
x = df.drop(['Appliances'],axis=1)
y = df['Appliances']
```

[22]:
```
# Cria um modelo de RandomFlorest para verificar a importância das variáveis␣
↪preditoras
rf = RandomForestRegressor(n_estimators=100)
rf.fit(x,y)
```

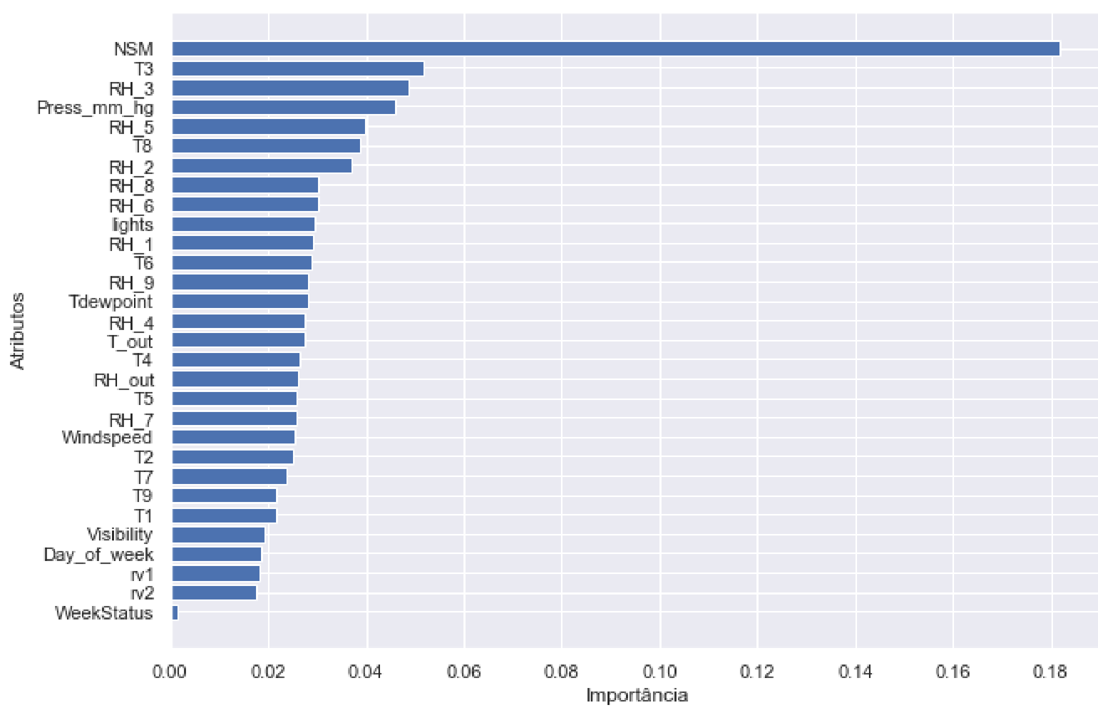
[22]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100,
```

```
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)
```

[23]:
```python
# Extraindo a importância do modelo Random Florest
importances = rf.feature_importances_
indices = np.argsort(importances)
```

[24]:
```python
# Obtém os índices dos modelo
ind=[]
for i in indices:
    ind.append(x.columns[i])
```

[25]:
```python
# Plot da Importância dos Atributos
fig,ax = plt.subplots(figsize=(10,7))
plt.barh(range(len(indices)), importances[indices])
plt.xlabel('Importância')
plt.ylabel('Atributos')
plt.xticks(np.arange(0,max(importances[indices]), step=0.02))
plt.yticks(range(len(indices)),ind)
plt.show()
```



[26]:
```python
# Seleciona as variáveis de treino com a importâcia igual ou superior a 0.02␣
 ↪para compor os modelos
col_temp = pd.DataFrame({'coluna':ind,'indice':importances[indices]})
```

```
cols = np.array(col_temp.coluna[col_temp['indice']>=0.02])
X = df[cols]
X.head()
```

[26]:
```
                        T1         T9         T7    T2  Windspeed       RH_7  \
date
2016-01-11 17:00:00  19.89  17.033333  17.200000  19.2   7.000000  41.626667
2016-01-11 17:10:00  19.89  17.066667  17.200000  19.2   6.666667  41.560000
2016-01-11 17:20:00  19.89  17.000000  17.200000  19.2   6.333333  41.433333
2016-01-11 17:40:00  19.89  17.000000  17.200000  19.2   5.666667  41.230000
2016-01-11 17:50:00  19.89  17.000000  17.133333  19.2   5.333333  41.260000

                            T5  RH_out         T4     T_out  …  lights  \
date                                                         …
2016-01-11 17:00:00  17.166667    92.0  19.000000  6.600000  …      30
2016-01-11 17:10:00  17.166667    92.0  19.000000  6.483333  …      30
2016-01-11 17:20:00  17.166667    92.0  18.926667  6.366667  …      30
2016-01-11 17:40:00  17.200000    92.0  18.890000  6.133333  …      40
2016-01-11 17:50:00  17.133333    92.0  18.890000  6.016667  …      40

                          RH_6       RH_8       RH_2    T8   RH_5  \
date
2016-01-11 17:00:00  84.256667  48.900000  44.790000  18.2  55.20
2016-01-11 17:10:00  84.063333  48.863333  44.722500  18.2  55.20
2016-01-11 17:20:00  83.156667  48.730000  44.626667  18.2  55.09
2016-01-11 17:40:00  84.893333  48.590000  44.530000  18.1  55.09
2016-01-11 17:50:00  85.766667  48.590000  44.500000  18.1  55.03

                     Press_mm_hg       RH_3     T3    NSM
date
2016-01-11 17:00:00        733.5  44.730000  19.79  61200
2016-01-11 17:10:00        733.6  44.790000  19.79  61800
2016-01-11 17:20:00        733.7  44.933333  19.79  62400
2016-01-11 17:40:00        733.9  45.000000  19.79  63600
2016-01-11 17:50:00        734.0  44.933333  19.79  64200

[5 rows x 25 columns]
```

[27]:
```
# Seleciona as variáveis de teste com a importâcia igual ou superior a 0.02␣
 ↪para compor os modelos
Xt = dft[cols]
yt = dft['Appliances']
Xt.head()
```

[27]:
```
                            T1     T9         T7         T2  Windspeed  \
date
2016-01-11 17:30:00  19.890000  17.00  17.133333  19.200000   6.000000
```

```
2016-01-11 18:00:00  19.890000  17.00  17.133333  19.200000   5.000000
2016-01-11 18:40:00  19.926667  17.00  17.247500  19.356667   5.666667
2016-01-11 18:50:00  20.066667  16.89  17.530000  19.426667   5.833333
2016-01-11 19:30:00  20.566667  16.89  17.890000  20.033333   6.000000

                          RH_7         T5     RH_out     T4     T_out  …  \
date                                                                   …
2016-01-11 17:30:00  41.290000  17.166667  92.000000  18.89  6.250000  …
2016-01-11 18:00:00  41.200000  17.100000  92.000000  18.89  5.900000  …
2016-01-11 18:40:00  42.717500  17.100000  91.333333  18.89  5.966667  …
2016-01-11 18:50:00  44.263333  17.100000  91.166667  19.00  5.983333  …
2016-01-11 19:30:00  44.926667  17.150000  89.500000  19.00  6.000000  …

                     lights       RH_6       RH_8       RH_2         T8  \
date
2016-01-11 17:30:00      40  83.423333  48.590000  44.590000  18.100000
2016-01-11 18:00:00      50  86.090000  48.590000  44.500000  18.100000
2016-01-11 18:40:00      70  87.866667  48.590000  44.400000  18.100000
2016-01-11 18:50:00      60  87.993333  48.633333  44.400000  18.066667
2016-01-11 19:30:00      10  88.366667  49.200000  46.756667  18.150000

                          RH_5  Press_mm_hg       RH_3     T3    NSM
date
2016-01-11 17:30:00  55.090000   733.800000  45.000000  19.79  63000
2016-01-11 18:00:00  54.966667   734.100000  44.900000  19.79  64800
2016-01-11 18:40:00  55.000000   734.366667  44.900000  19.79  67200
2016-01-11 18:50:00  55.000000   734.433333  44.826667  19.79  67800
2016-01-11 19:30:00  56.042500   734.850000  48.466667  20.10  70200

[5 rows x 25 columns]
```

[28]:
```python
# Treina Modelo 01 Regressão Linear Múltipla
modelo_1 = LinearRegression()
modelo_1.fit(X,y)
y_pred = modelo_1.predict(X)
previsao = modelo_1.predict(Xt)
print("R^2  dados treino:", r2_score(y,y_pred))
print("RMSE dados treino:", np.sqrt(mean_squared_error(y,y_pred)))
print('\n')
print("R^2  dados teste:", r2_score(yt,previsao))
print("RMSE dados teste:", np.sqrt(mean_squared_error(yt,previsao)))
```

```
R^2  dados treino: 0.17160327471761294
RMSE dados treino: 93.58709945590935


R^2  dados teste: 0.1515311515316855
```

```
RMSE dados teste: 93.58966225468515
```

[29]:
```python
# Treina Modelo 02 com SVM
modelo_2 = svm.SVR()
modelo_2.fit(X,y)
y_pred = modelo_2.predict(X)
previsao = modelo_2.predict(Xt)
print("R^2  dados treino:", r2_score(y,y_pred))
print("RMSE dados treino:", np.sqrt(mean_squared_error(y,y_pred)))
print('\n')
print("R^2  dados teste:", r2_score(yt,previsao))
print("RMSE dados teste:", np.sqrt(mean_squared_error(yt,previsao)))
```

```
R^2  dados treino: -0.12302620699597089
RMSE dados treino: 108.96618502439901


R^2  dados teste: -0.12662594791454618
RMSE dados teste: 107.84494761571585
```

[30]:
```python
# Treina Modelo 03 com XGBoost
modelo_3 = XGBRegressor(objective='reg:squarederror')
modelo_3.fit(X,y)
y_pred = modelo_3.predict(X)
previsao = modelo_3.predict(Xt)
print("R^2  dados treino:", r2_score(y,y_pred))
print("RMSE dados treino:", np.sqrt(mean_squared_error(y,y_pred)))
print('\n')
print("R^2  dados teste:", r2_score(yt,previsao))
print("RMSE dados teste:", np.sqrt(mean_squared_error(yt,previsao)))
```

```
R^2  dados treino: 0.36301356985775024
RMSE dados treino: 82.06574019211119


R^2  dados teste: 0.2981707159633773
RMSE dados teste: 85.11883516439435
```

4 Modelo Preditivo

4.0.1 Otimização do modelo com XGBoost que apresentou o melhor RMSE

[31]:
```python
# Normaliza as variáveis preditoras para tentar melhorar a acurácia do modelo
scaler = Normalizer().fit(X)
xn = scaler.transform(X)
X = pd.DataFrame(xn,columns=[cols])
X.head()
```

[31]:
```
         T1        T9        T7        T2  Windspeed     RH_7        T5  \
0  0.000325  0.000278  0.000281  0.000314  0.000114  0.000680  0.000280
1  0.000322  0.000276  0.000278  0.000311  0.000108  0.000672  0.000278
2  0.000319  0.000272  0.000276  0.000308  0.000101  0.000664  0.000275
3  0.000313  0.000267  0.000270  0.000302  0.000089  0.000648  0.000270
4  0.000310  0.000265  0.000267  0.000299  0.000083  0.000643  0.000267

     RH_out        T4     T_out  …    lights      RH_6      RH_8      RH_2  \
0  0.001503  0.000310  0.000108  …  0.000490  0.001377  0.000799  0.000732
1  0.001489  0.000307  0.000105  …  0.000485  0.001360  0.000791  0.000724
2  0.001474  0.000303  0.000102  …  0.000481  0.001333  0.000781  0.000715
3  0.001446  0.000297  0.000096  …  0.000629  0.001335  0.000764  0.000700
4  0.001433  0.000294  0.000094  …  0.000623  0.001336  0.000757  0.000693

         T8      RH_5 Press_mm_hg      RH_3        T3       NSM
0  0.000297  0.000902    0.011984  0.000731  0.000323  0.999923
1  0.000294  0.000893    0.011870  0.000725  0.000320  0.999925
2  0.000292  0.000883    0.011757  0.000720  0.000317  0.999926
3  0.000285  0.000866    0.011538  0.000707  0.000311  0.999929
4  0.000282  0.000857    0.011432  0.000700  0.000308  0.999930

[5 rows x 25 columns]
```

[32]:
```python
scaler = Normalizer().fit(Xt)
xnt = scaler.transform(Xt)
Xt = pd.DataFrame(xnt,columns=[cols])
Xt.head()
```

[32]:
```
         T1        T9        T7        T2  Windspeed     RH_7        T5  \
0  0.000316  0.000270  0.000272  0.000305  0.000095  0.000655  0.000272
1  0.000307  0.000262  0.000264  0.000296  0.000077  0.000636  0.000264
2  0.000297  0.000253  0.000257  0.000288  0.000084  0.000636  0.000254
3  0.000296  0.000249  0.000259  0.000287  0.000086  0.000653  0.000252
4  0.000293  0.000241  0.000255  0.000285  0.000085  0.000640  0.000244

     RH_out        T4     T_out  …    lights      RH_6      RH_8      RH_2  \
0  0.001460  0.000300  0.000099  …  0.000635  0.001324  0.000771  0.000708
1  0.001420  0.000291  0.000091  …  0.000772  0.001328  0.000750  0.000687
2  0.001359  0.000281  0.000089  …  0.001042  0.001307  0.000723  0.000661
3  0.001345  0.000280  0.000088  …  0.000885  0.001298  0.000717  0.000655
4  0.001275  0.000271  0.000085  …  0.000142  0.001259  0.000701  0.000666

         T8      RH_5 Press_mm_hg      RH_3        T3       NSM
0  0.000287  0.000874    0.011647  0.000714  0.000314  0.999927
1  0.000279  0.000848    0.011328  0.000693  0.000305  0.999931
2  0.000269  0.000818    0.010927  0.000668  0.000294  0.999936
3  0.000266  0.000811    0.010832  0.000661  0.000292  0.999937
```

```
4  0.000259  0.000798   0.010467  0.000690  0.000286  0.999941

[5 rows x 25 columns]
```

[33]:
```
# Treina Modelo 04 com XGBoost e dados normalizados
modelo_4 = XGBRegressor(objective='reg:squarederror', n_jobs=-1)
modelo_4.fit(X,y)
y_pred = modelo_4.predict(X)
previsao = modelo_4.predict(Xt)
print("R^2  dados treino:", r2_score(y,y_pred))
print("MSE  dados treino:", np.sqrt(mean_squared_error(y,y_pred)))
print('\n')
print("R^2  dados teste:", r2_score(yt,previsao))
print("MSE  dados teste:", np.sqrt(mean_squared_error(yt,previsao)))
```

```
R^2  dados treino: 0.3364301184245656
MSE  dados treino: 83.76066826569253


R^2  dados teste: 0.2769649361567972
MSE  dados teste: 86.39519886356211
```

[34]:
```
# Tenta melhorar o modelo com o RandomizedSearchCV
modelo_5 = XGBRegressor(objective='reg:squarederror', n_jobs=-1)
params = {'min_child_weight':[4,5], 'gamma':[i/10.0 for i in␣
 ↪range(3,6)],'subsample':[i/10.0 for i in range(6,11)],
'colsample_bytree':[i/10.0 for i in range(6,11)], 'max_depth': [2,3,4]}
n_iter_search = 20
random_search = RandomizedSearchCV(modelo_5,␣
 ↪param_distributions=params,n_iter=n_iter_search)
random_search.fit(X,y)
```

[34]:
```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:squarederro…
                                          reg_lambda=1, scale_pos_weight=1,
                                          seed=None, silent=None, subsample=1,
                                          verbosity=1),
                   iid='warn', n_iter=20, n_jobs=None,
```

```
                   param_distributions={'colsample_bytree': [0.6, 0.7, 0.8, 0.9,
                                                              1.0],
                                        'gamma': [0.3, 0.4, 0.5],
                                        'max_depth': [2, 3, 4],
                                        'min_child_weight': [4, 5],
                                        'subsample': [0.6, 0.7, 0.8, 0.9, 1.0]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

[35]:

```
# Cria função para exibir relatório com os três melhores resultados do␣
 ↪RandomizedSearchCV
def relatorio(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

[36]:

```
# Exibe o relatório com o resultado do RandomizedSearchCV
relatorio(random_search.cv_results_)
```

```
Model with rank: 1
Mean validation score: 0.105 (std: 0.058)
Parameters: {'subsample': 1.0, 'min_child_weight': 5, 'max_depth': 2, 'gamma':
0.4, 'colsample_bytree': 0.7}

Model with rank: 2
Mean validation score: 0.102 (std: 0.069)
Parameters: {'subsample': 0.6, 'min_child_weight': 5, 'max_depth': 2, 'gamma':
0.5, 'colsample_bytree': 0.7}

Model with rank: 3
Mean validation score: 0.100 (std: 0.063)
Parameters: {'subsample': 0.7, 'min_child_weight': 5, 'max_depth': 2, 'gamma':
0.3, 'colsample_bytree': 0.9}
```

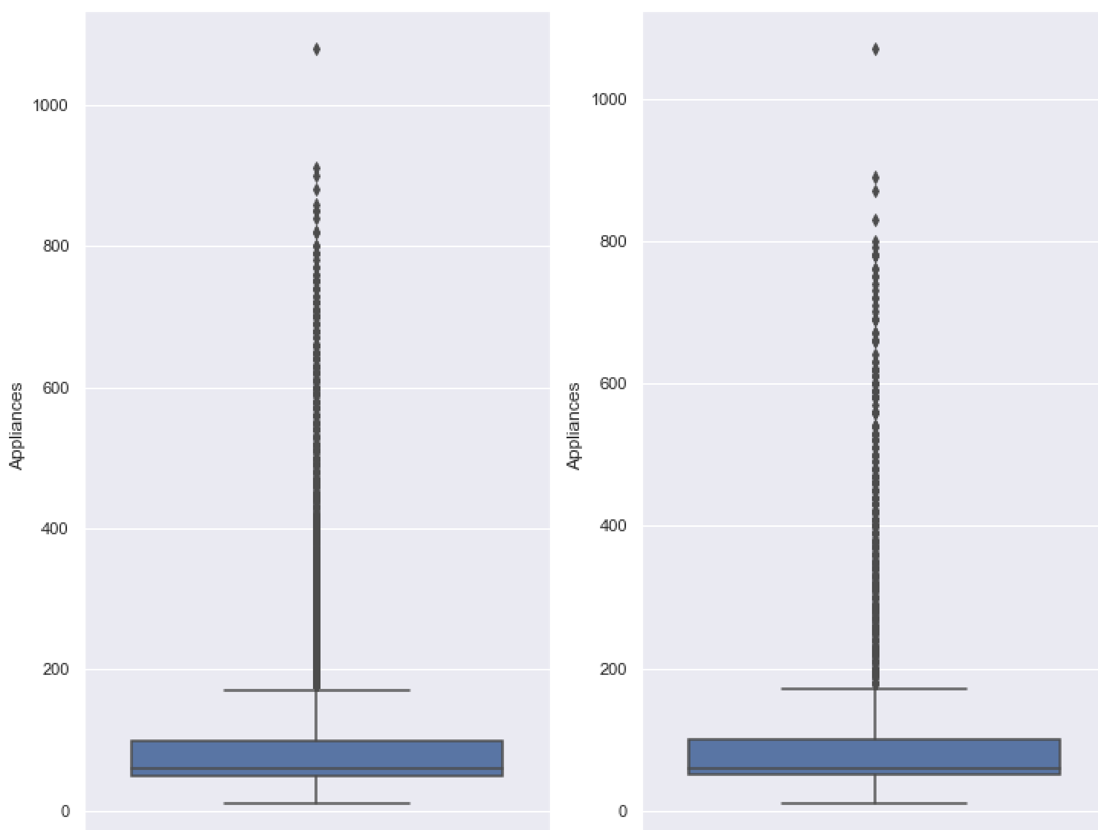
[37]:

```
# Faz previsões com o melhor modelo
y_pred = random_search.best_estimator_.predict(X)
previsao = random_search.best_estimator_.predict(Xt)
print("R^2  dados treino:", r2_score(y,y_pred))
print("MSE  dados treino:", np.sqrt(mean_squared_error(y,y_pred)))
print('\n')
```

```
print("R^2  dados teste:", r2_score(yt,previsao))
print("MSE  dados teste:", np.sqrt(mean_squared_error(yt,previsao)))
```

```
R^2  dados treino: 0.2374056300618287
MSE  dados treino: 89.79323081360764


R^2  dados teste: 0.2122214822531181
MSE  dados teste: 90.1803664895082
```

[38]:
```
# Houve piora no modelo mesmo com a aplicação de normalização e refinamento dos␣
↪paramêtros
# Pode ter sido causado pela alta quantidade de outliers detectados na variável␣
↪alvo
# Plota os outliers da variável alvo
fig,ax = plt.subplots(1,2,figsize=(12,10))
sns.boxplot(df['Appliances'], orient='v', ax=ax[0])
sns.boxplot(dft['Appliances'], orient='v', ax=ax[1])
plt.show()
```



[39]:
```
# Calcula o intervalo interquartil para filtrar os outliers dos dados de treino
Q1 = df['Appliances'].quantile(0.25)
Q3 = df['Appliances'].quantile(0.75)
IIQ = Q3 - Q1
filtra_outlier = (df['Appliances'] >= Q1 - 1.5 * IIQ) & (df['Appliances'] <= Q3␣
↪+ 1.5 * IIQ)
df = df.loc[filtra_outlier]
print(df.shape)
```

```
(13169, 31)
```

[40]:
```
# Calcula o intervalo interquartil para filtrar os outliers dos dados de teste
Q1 = dft['Appliances'].quantile(0.25)
Q3 = dft['Appliances'].quantile(0.75)
IIQ = Q3 - Q1
filtra_outlier = (dft['Appliances'] >= Q1 - 1.5 * IIQ) & (dft['Appliances'] <=␣
↪Q3 + 1.5 * IIQ)
dft = dft.loc[filtra_outlier]
print(dft.shape)
```

```
(4428, 31)
```

[41]:
```
# Separa novamente as variáveis preditoras e varíavel alvo
X = df[cols]
y = df['Appliances']
Xt = dft[cols]
yt = dft['Appliances']
```

[42]:
```
# Novo teste de paramêtros do modelo XGBoost com o RandomizedSearchCV
modelo_6 = XGBRegressor(objective='reg:squarederror', n_jobs=-1)
params = {'min_child_weight':[4,5], 'gamma':[i/10.0 for i in␣
↪range(3,6)],'subsample':[i/10.0 for i in range(6,11)],
'colsample_bytree':[i/10.0 for i in range(6,11)], 'max_depth': [2,3,4]}
n_iter_search = 20
random_search = RandomizedSearchCV(modelo_6,␣
↪param_distributions=params,n_iter=n_iter_search)
random_search.fit(X,y)
```

[42]:
```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
```

```
                                      n_jobs=-1, nthread=None,
                                      objective='reg:squarederro…
                                      reg_lambda=1, scale_pos_weight=1,
                                      seed=None, silent=None, subsample=1,
                                      verbosity=1),
                   iid='warn', n_iter=20, n_jobs=None,
                   param_distributions={'colsample_bytree': [0.6, 0.7, 0.8, 0.9,
                                                             1.0],
                                        'gamma': [0.3, 0.4, 0.5],
                                        'max_depth': [2, 3, 4],
                                        'min_child_weight': [4, 5],
                                        'subsample': [0.6, 0.7, 0.8, 0.9, 1.0]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

[43]:

```python
# Faz previsões com o melhor modelo
y_pred = random_search.best_estimator_.predict(X)
previsao = random_search.best_estimator_.predict(Xt)
print("R^2  dados treino:", r2_score(y,y_pred))
print("RMSE dados treino:", np.sqrt(mean_squared_error(y,y_pred)))
print('\n')
print("R^2  dados teste:", r2_score(yt,previsao))
print("RMSE dados teste:", np.sqrt(mean_squared_error(yt,previsao)))
```

```
R^2  dados treino: 0.5093736426153521
RMSE dados treino: 19.83838921447996


R^2  dados teste: 0.4785048570317455
RMSE dados teste: 20.895858749120897
```

5 O modelo 06 obteve os melhores resultados com XGBoost e os outliers da variável alvo removidos

[ ]:

```

```