Estimation of Energy Consumption

Table Of Contents

Chapter 1: Code Task

Section 1.1: Importing required libraries

In [1]:

```
# Importing the required libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import Imputer
from sklearn.ensemble import ExtraTreesRegressor
import seaborn as sns #visualisation
sns.set(color_codes=True)
```

```
C:\Users\Acer\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29:
DeprecationWarning: numpy.core.umath_tests is an internal NumPy module and should not be imported.
It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

Section 1.2: Loading input data csv file and importing it into dataframe

In [2]:

```
# loading the input data into dataframe
inputFile = "C:/Users/Acer/Desktop/Devosmita/input.csv"
df = pd.read_csv(inputFile,sep="\t")
# Displaying the top 5 rows
print(df.head())
# Displaying the datatypes
print(df.dtypes)
```

```
              datetime  returntemp  supplytemp  water_state
0  2017-01-01 00:00:00      37.341      84.267     5143.999
1  2017-01-01 01:00:00      58.714      77.914     5144.165
2  2017-01-01 02:00:00      39.888      82.807     5144.281
3  2017-01-01 03:00:00      56.427      87.255          NaN
4  2017-01-01 04:00:00      31.153      87.821     5145.665
datetime        object
returntemp     float64
supplytemp     float64
water_state    float64
```

```
dtype: object
```

Section 1.3: Missing Data Imputation

Section 1.3.1: Finding missing values

In [3]:

```
# Finding the null values
print(df.isnull().sum())
```

```
datetime       0
returntemp     4
supplytemp     1
water_state    6
dtype: int64
```

Section 1.3.2: Initializing imputation using mean imputation method

In [4]:

```
# Initializing imputation using mean imputation method
imputedData = Imputer(missing_values='NaN', strategy='mean', axis=0, verbose=0, copy=True)
imputedData = pd.DataFrame(imputedData.fit_transform(df[['returntemp','supplytemp','water_state']]
), columns = ['returntemp','supplytemp','water_state'])
print(imputedData)
```

```
     returntemp  supplytemp  water_state
0     37.341000   84.267000  5143.999000
1     58.714000   77.914000  5144.165000
2     39.888000   82.807000  5144.281000
3     56.427000   87.255000  5027.535877
4     31.153000   87.821000  5145.665000
5     56.389000   78.457000  5146.373000
6     54.209000   73.945000  5146.479000
7     49.498000   82.534000  5146.749000
8     50.401000   60.987000  5147.465000
9     57.336000   88.700000  5147.653000
10    31.659000   62.852000  5148.196000
11    30.697000   84.611000  5148.387000
12    57.639000   82.439000  5148.998000
13    49.571000   60.552000  5149.953000
14    36.106000   64.785000  5027.535877
15    35.795000   81.957000  5150.768000
16    42.593000   67.087000  5151.328000
17    45.301000   87.646000  5151.779000
18    32.275000   78.911000  5152.520000
19    43.730000   69.268000  5153.199000
20    44.296000   76.807000  5153.404000
21    35.308000   78.628000  5153.565000
22    44.473348   74.720449  5154.292000
23    30.834000   70.962000  5154.756000
24    40.900000   77.119000  5154.955000
25    41.557000   87.675000  5155.328000
26    39.615000   79.069000  5155.571000
27    45.677000   65.977000  5155.775000
28    46.118000   88.024000  5156.293000
29    48.741000   66.490000  5027.535877
..          ...         ...          ...
138   59.405000   63.308000  5212.682000
139   57.619000   70.209000  5213.329000
140   31.919000   73.888000  5213.434000
141   56.780000   69.660000  5214.223000
142   46.120000   63.192000  5214.476000
143   40.361000   69.304000  5215.061000
144   44.473348   85.039000  5215.930000
145   38.669000   81.357000  5216.321000
146   57.487000   70.497000  5216.614000
147   33.223000   68.689000  5216.829000
148   37.345000   74.847000  5217.749000
```

```
149   57.437000   78.445000  5218.654000
150   54.598000   60.412000  5219.463000
151   35.668000   83.469000  5220.325000
152   55.828000   67.657000  5220.591000
153   39.747000   70.218000  5221.252000
154   40.033000   64.796000  5222.041000
155   39.604000   77.115000  5222.283000
156   37.286000   66.324000  5223.009000
157   58.910000   74.399000  5223.530000
158   46.537000   69.465000  5223.913000
159   30.229000   80.289000  5224.401000
160   41.824000   87.859000  5224.676000
161   42.926000   70.335000  5224.698000
162   55.907000   79.622000  5225.222000
163   48.877000   60.728000  5225.669000
164   50.119000   62.520000  5225.875000
165   40.799000   82.819000  5226.849000
166   50.242000   73.867000  5227.081000
167   41.441000   83.357000  5227.970000

[168 rows x 3 columns]
```

Section 1.3.3 Imputing missing values using missForest algorithm

In [5]:

```
# Imputing missing values using missForest algorithm (ExtraTreesRegressor is similiar to
missforest algorithm in R)
seed = 0
# Number of trees in the forest is 10
imputer = ExtraTreesRegressor(n_estimators=10, random_state=seed)

for x in ['returntemp','supplytemp','water_state']:
    X = imputedData.loc[:, imputedData.columns != x].values
    y = imputedData[[x]].values
    model = imputer.fit(X,y)
    imputedData[x] = model.predict(X)

print(imputedData)
```

```
     returntemp  supplytemp  water_state
0     37.341000   84.267000  5143.999000
1     58.714000   77.914000  5144.165000
2     39.888000   82.807000  5144.281000
3     56.427000   87.255000  5027.535877
4     31.153000   87.821000  5145.665000
5     56.389000   78.457000  5146.373000
6     54.209000   73.945000  5146.479000
7     49.498000   82.534000  5146.749000
8     50.401000   60.987000  5147.465000
9     57.336000   88.700000  5147.653000
10    31.659000   62.852000  5148.196000
11    30.697000   84.611000  5148.387000
12    57.639000   82.439000  5148.998000
13    49.571000   60.552000  5149.953000
14    36.106000   64.785000  5027.535877
15    35.795000   81.957000  5150.768000
16    42.593000   67.087000  5151.328000
17    45.301000   87.646000  5151.779000
18    32.275000   78.911000  5152.520000
19    43.730000   69.268000  5153.199000
20    44.296000   76.807000  5153.404000
21    35.308000   78.628000  5153.565000
22    44.473348   74.720449  5154.292000
23    30.834000   70.962000  5154.756000
24    40.900000   77.119000  5154.955000
25    41.557000   87.675000  5155.328000
26    39.615000   79.069000  5155.571000
27    45.677000   65.977000  5155.775000
28    46.118000   88.024000  5156.293000
29    48.741000   66.490000  5027.535877
..          ...         ...          ...
138   59.405000   63.308000  5212.682000
139   57.619000   70.209000  5213.329000
```

140	31.919000	73.888000	5213.434000
141	56.780000	69.660000	5214.223000
142	46.120000	63.192000	5214.476000
143	40.361000	69.304000	5215.061000
144	44.473348	85.039000	5215.930000
145	38.669000	81.357000	5216.321000
146	57.487000	70.497000	5216.614000
147	33.223000	68.689000	5216.829000
148	37.345000	74.847000	5217.749000
149	57.437000	78.445000	5218.654000
150	54.598000	60.412000	5219.463000
151	35.668000	83.469000	5220.325000
152	55.828000	67.657000	5220.591000
153	39.747000	70.218000	5221.252000
154	40.033000	64.796000	5222.041000
155	39.604000	77.115000	5222.283000
156	37.286000	66.324000	5223.009000
157	58.910000	74.399000	5223.530000
158	46.537000	69.465000	5223.913000
159	30.229000	80.289000	5224.401000
160	41.824000	87.859000	5224.676000
161	42.926000	70.335000	5224.698000
162	55.907000	79.622000	5225.222000
163	48.877000	60.728000	5225.669000
164	50.119000	62.520000	5225.875000
165	40.799000	82.819000	5226.849000
166	50.242000	73.867000	5227.081000
167	41.441000	83.357000	5227.970000

[168 rows x 3 columns]

```
C:\Users\Acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  if __name__ == '__main__':
C:\Users\Acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  if __name__ == '__main__':
C:\Users\Acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  if __name__ == '__main__':
```

Section 1.4: Univariate Outlier Detection

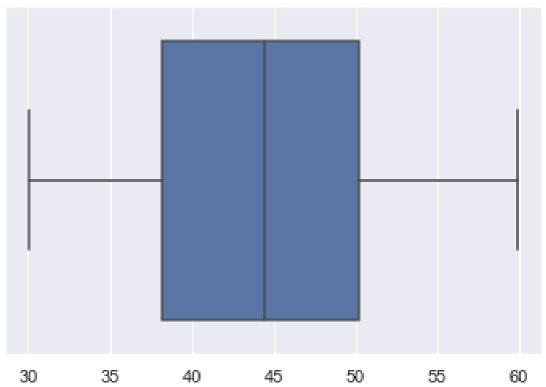
Section 1.4.1: Visualizing outliers using box plot

In [6]:

```
# Plot boxplot of 'returntemp' feature
sns.boxplot(x=imputedData['returntemp'])
```

Out[6]:

<matplotlib.axes._subplots.AxesSubplot at 0x204180caf98>



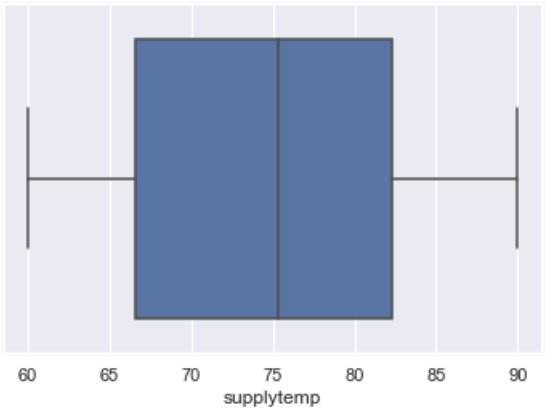
returntemp

In [7]:

```
# Plot boxplot of 'supplytemp' feature
sns.boxplot(x=imputedData['supplytemp'])
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x204193dc860>
```

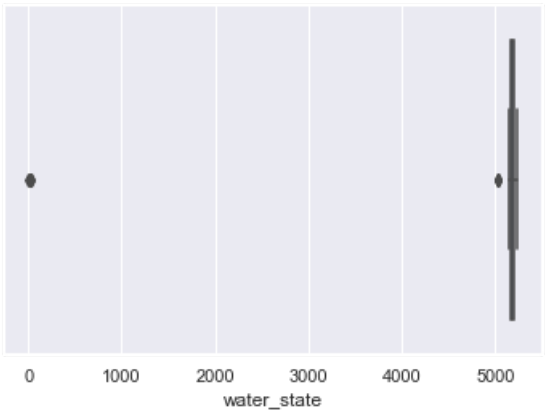


In [8]:

```
# Plot boxplot of 'water_state' feature
sns.boxplot(x=imputedData['water_state'])
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20419444748>
```



Section 1.4.2: Detecting outiers using IQR outlier detection

In [9]:

```
# IQR outlier detection
data = imputedData['water_state']
q1,q3 = np.percentile(data,[25,75])
interQuartileRange = q3 - q1
lowerBound = q1 - (interQuartileRange*1.5)
upperBound = q3 + (interQuartileRange*1.5)
outlierIndices = np.where((data > upperBound)|(data < lowerBound))
print(outlierIndices)
```

```
(array([  3,  14,  29,  42,  44,  46,  58,  77,  82, 123, 131], dtype=int64),)
```

Section 1.4.3: Dropping outliers

In [10]:

```
# Dropping outliers from imputed data
imputedData2 = imputedData.drop([  3,  14,  29,  42,  44,  46,  58,  77,  82, 123, 131])
print(imputedData2)
# Reset dataset
imputedData2_reset = imputedData2.reset_index(drop=True)
print(imputedData2_reset)
```

```
     returntemp  supplytemp  water_state
0     37.341000   84.267000     5143.999
1     58.714000   77.914000     5144.165
2     39.888000   82.807000     5144.281
4     31.153000   87.821000     5145.665
5     56.389000   78.457000     5146.373
6     54.209000   73.945000     5146.479
7     49.498000   82.534000     5146.749
8     50.401000   60.987000     5147.465
9     57.336000   88.700000     5147.653
10    31.659000   62.852000     5148.196
11    30.697000   84.611000     5148.387
12    57.639000   82.439000     5148.998
13    49.571000   60.552000     5149.953
15    35.795000   81.957000     5150.768
16    42.593000   67.087000     5151.328
17    45.301000   87.646000     5151.779
18    32.275000   78.911000     5152.520
19    43.730000   69.268000     5153.199
20    44.296000   76.807000     5153.404
21    35.308000   78.628000     5153.565
22    44.473348   74.720449     5154.292
23    30.834000   70.962000     5154.756
24    40.900000   77.119000     5154.955
25    41.557000   87.675000     5155.328
26    39.615000   79.069000     5155.571
27    45.677000   65.977000     5155.775
28    46.118000   88.024000     5156.293
30    52.086000   65.766000     5157.471
31    51.001000   85.285000     5158.304
32    41.716000   63.118000     5159.104
..          ...         ...          ...
138   59.405000   63.308000     5212.682
139   57.619000   70.209000     5213.329
140   31.919000   73.888000     5213.434
141   56.780000   69.660000     5214.223
142   46.120000   63.192000     5214.476
143   40.361000   69.304000     5215.061
144   44.473348   85.039000     5215.930
145   38.669000   81.357000     5216.321
146   57.487000   70.497000     5216.614
147   33.223000   68.689000     5216.829
148   37.345000   74.847000     5217.749
149   57.437000   78.445000     5218.654
150   54.598000   60.412000     5219.463
151   35.668000   83.469000     5220.325
152   55.828000   67.657000     5220.591
153   39.747000   70.218000     5221.252
154   40.033000   64.796000     5222.041
155   39.604000   77.115000     5222.283
156   37.286000   66.324000     5223.009
157   58.910000   74.399000     5223.530
158   46.537000   69.465000     5223.913
159   30.229000   80.289000     5224.401
160   41.824000   87.859000     5224.676
161   42.926000   70.335000     5224.698
162   55.907000   79.622000     5225.222
163   48.877000   60.728000     5225.669
164   50.119000   62.520000     5225.875
165   40.799000   82.819000     5226.849
166   50.242000   73.867000     5227.081
167   41.441000   83.357000     5227.970

[157 rows x 3 columns]
     returntemp  supplytemp  water_state
```

```
0     37.341000   84.267000     5143.999
1     58.714000   77.914000     5144.165
2     39.888000   82.807000     5144.281
3     31.153000   87.821000     5145.665
4     56.389000   78.457000     5146.373
5     54.209000   73.945000     5146.479
6     49.498000   82.534000     5146.749
7     50.401000   60.987000     5147.465
8     57.336000   88.700000     5147.653
9     31.659000   62.852000     5148.196
10    30.697000   84.611000     5148.387
11    57.639000   82.439000     5148.998
12    49.571000   60.552000     5149.953
13    35.795000   81.957000     5150.768
14    42.593000   67.087000     5151.328
15    45.301000   87.646000     5151.779
16    32.275000   78.911000     5152.520
17    43.730000   69.268000     5153.199
18    44.296000   76.807000     5153.404
19    35.308000   78.628000     5153.565
20    44.473348   74.720449     5154.292
21    30.834000   70.962000     5154.756
22    40.900000   77.119000     5154.955
23    41.557000   87.675000     5155.328
24    39.615000   79.069000     5155.571
25    45.677000   65.977000     5155.775
26    46.118000   88.024000     5156.293
27    52.086000   65.766000     5157.471
28    51.001000   85.285000     5158.304
29    41.716000   63.118000     5159.104
..          ...         ...          ...
127   59.405000   63.308000     5212.682
128   57.619000   70.209000     5213.329
129   31.919000   73.888000     5213.434
130   56.780000   69.660000     5214.223
131   46.120000   63.192000     5214.476
132   40.361000   69.304000     5215.061
133   44.473348   85.039000     5215.930
134   38.669000   81.357000     5216.321
135   57.487000   70.497000     5216.614
136   33.223000   68.689000     5216.829
137   37.345000   74.847000     5217.749
138   57.437000   78.445000     5218.654
139   54.598000   60.412000     5219.463
140   35.668000   83.469000     5220.325
141   55.828000   67.657000     5220.591
142   39.747000   70.218000     5221.252
143   40.033000   64.796000     5222.041
144   39.604000   77.115000     5222.283
145   37.286000   66.324000     5223.009
146   58.910000   74.399000     5223.530
147   46.537000   69.465000     5223.913
148   30.229000   80.289000     5224.401
149   41.824000   87.859000     5224.676
150   42.926000   70.335000     5224.698
151   55.907000   79.622000     5225.222
152   48.877000   60.728000     5225.669
153   50.119000   62.520000     5225.875
154   40.799000   82.819000     5226.849
155   50.242000   73.867000     5227.081
156   41.441000   83.357000     5227.970

[157 rows x 3 columns]
```

In [12]:

```
# Dropping outliers from datetime
dateTime = df[['datetime']].drop([  3,  14,  29,  42,  44,  46,  58,  77,  82, 123, 131])
print(dateTime)
# Reset datetime
dateTime_reset = dateTime.reset_index(drop=True)
print(dateTime_reset)
```

```
              datetime
0    2017-01-01 00:00:00
```

```
1    2017-01-01 01:00:00
2    2017-01-01 02:00:00
4    2017-01-01 04:00:00
5    2017-01-01 05:00:00
6    2017-01-01 06:00:00
7    2017-01-01 07:00:00
8    2017-01-01 08:00:00
9    2017-01-01 09:00:00
10   2017-01-01 10:00:00
11   2017-01-01 11:00:00
12   2017-01-01 12:00:00
13   2017-01-01 13:00:00
15   2017-01-01 15:00:00
16   2017-01-01 16:00:00
17   2017-01-01 17:00:00
18   2017-01-01 18:00:00
19   2017-01-01 19:00:00
20   2017-01-01 20:00:00
21   2017-01-01 21:00:00
22   2017-01-01 22:00:00
23   2017-01-01 23:00:00
24   2017-01-02 00:00:00
25   2017-01-02 01:00:00
26   2017-01-02 02:00:00
27   2017-01-02 03:00:00
28   2017-01-02 04:00:00
30   2017-01-02 06:00:00
31   2017-01-02 07:00:00
32   2017-01-02 08:00:00
..                   ...
138  2017-01-06 18:00:00
139  2017-01-06 19:00:00
140  2017-01-06 20:00:00
141  2017-01-06 21:00:00
142  2017-01-06 22:00:00
143  2017-01-06 23:00:00
144  2017-01-07 00:00:00
145  2017-01-07 01:00:00
146  2017-01-07 02:00:00
147  2017-01-07 03:00:00
148  2017-01-07 04:00:00
149  2017-01-07 05:00:00
150  2017-01-07 06:00:00
151  2017-01-07 07:00:00
152  2017-01-07 08:00:00
153  2017-01-07 09:00:00
154  2017-01-07 10:00:00
155  2017-01-07 11:00:00
156  2017-01-07 12:00:00
157  2017-01-07 13:00:00
158  2017-01-07 14:00:00
159  2017-01-07 15:00:00
160  2017-01-07 16:00:00
161  2017-01-07 17:00:00
162  2017-01-07 18:00:00
163  2017-01-07 19:00:00
164  2017-01-07 20:00:00
165  2017-01-07 21:00:00
166  2017-01-07 22:00:00
167  2017-01-07 23:00:00

[157 rows x 1 columns]
                datetime
0    2017-01-01 00:00:00
1    2017-01-01 01:00:00
2    2017-01-01 02:00:00
3    2017-01-01 04:00:00
4    2017-01-01 05:00:00
5    2017-01-01 06:00:00
6    2017-01-01 07:00:00
7    2017-01-01 08:00:00
8    2017-01-01 09:00:00
9    2017-01-01 10:00:00
10   2017-01-01 11:00:00
11   2017-01-01 12:00:00
12   2017-01-01 13:00:00
13   2017-01-01 15:00:00
```

```
14   2017-01-01 16:00:00
15   2017-01-01 17:00:00
16   2017-01-01 18:00:00
17   2017-01-01 19:00:00
18   2017-01-01 20:00:00
19   2017-01-01 21:00:00
20   2017-01-01 22:00:00
21   2017-01-01 23:00:00
22   2017-01-02 00:00:00
23   2017-01-02 01:00:00
24   2017-01-02 02:00:00
25   2017-01-02 03:00:00
26   2017-01-02 04:00:00
27   2017-01-02 06:00:00
28   2017-01-02 07:00:00
29   2017-01-02 08:00:00
..                   ...
127  2017-01-06 18:00:00
128  2017-01-06 19:00:00
129  2017-01-06 20:00:00
130  2017-01-06 21:00:00
131  2017-01-06 22:00:00
132  2017-01-06 23:00:00
133  2017-01-07 00:00:00
134  2017-01-07 01:00:00
135  2017-01-07 02:00:00
136  2017-01-07 03:00:00
137  2017-01-07 04:00:00
138  2017-01-07 05:00:00
139  2017-01-07 06:00:00
140  2017-01-07 07:00:00
141  2017-01-07 08:00:00
142  2017-01-07 09:00:00
143  2017-01-07 10:00:00
144  2017-01-07 11:00:00
145  2017-01-07 12:00:00
146  2017-01-07 13:00:00
147  2017-01-07 14:00:00
148  2017-01-07 15:00:00
149  2017-01-07 16:00:00
150  2017-01-07 17:00:00
151  2017-01-07 18:00:00
152  2017-01-07 19:00:00
153  2017-01-07 20:00:00
154  2017-01-07 21:00:00
155  2017-01-07 22:00:00
156  2017-01-07 23:00:00

[157 rows x 1 columns]
```

In [13]:

```
# Concatenate datetime and imputed data
imputedData2_reset = pd.concat([dateTime_reset, imputedData2_reset], axis=1)
print(imputedData2_reset)
```

```
              datetime  returntemp  supplytemp  water_state
0    2017-01-01 00:00:00   37.341000   84.267000     5143.999
1    2017-01-01 01:00:00   58.714000   77.914000     5144.165
2    2017-01-01 02:00:00   39.888000   82.807000     5144.281
3    2017-01-01 04:00:00   31.153000   87.821000     5145.665
4    2017-01-01 05:00:00   56.389000   78.457000     5146.373
5    2017-01-01 06:00:00   54.209000   73.945000     5146.479
6    2017-01-01 07:00:00   49.498000   82.534000     5146.749
7    2017-01-01 08:00:00   50.401000   60.987000     5147.465
8    2017-01-01 09:00:00   57.336000   88.700000     5147.653
9    2017-01-01 10:00:00   31.659000   62.852000     5148.196
10   2017-01-01 11:00:00   30.697000   84.611000     5148.387
11   2017-01-01 12:00:00   57.639000   82.439000     5148.998
12   2017-01-01 13:00:00   49.571000   60.552000     5149.953
13   2017-01-01 15:00:00   35.795000   81.957000     5150.768
14   2017-01-01 16:00:00   42.593000   67.087000     5151.328
15   2017-01-01 17:00:00   45.301000   87.646000     5151.779
16   2017-01-01 18:00:00   32.275000   78.911000     5152.520
17   2017-01-01 19:00:00   43.730000   69.268000     5153.199
18   2017-01-01 20:00:00   44.296000   76.807000     5153.404
```

```
10   2017-01-01 20:00:00   44.290000   70.007000     5153.404
19   2017-01-01 21:00:00   35.308000   78.628000     5153.565
20   2017-01-01 22:00:00   44.473348   74.720449     5154.292
21   2017-01-01 23:00:00   30.834000   70.962000     5154.756
22   2017-01-02 00:00:00   40.900000   77.119000     5154.955
23   2017-01-02 01:00:00   41.557000   87.675000     5155.328
24   2017-01-02 02:00:00   39.615000   79.069000     5155.571
25   2017-01-02 03:00:00   45.677000   65.977000     5155.775
26   2017-01-02 04:00:00   46.118000   88.024000     5156.293
27   2017-01-02 06:00:00   52.086000   65.766000     5157.471
28   2017-01-02 07:00:00   51.001000   85.285000     5158.304
29   2017-01-02 08:00:00   41.716000   63.118000     5159.104
..                   ...         ...         ...          ...
127  2017-01-06 18:00:00   59.405000   63.308000     5212.682
128  2017-01-06 19:00:00   57.619000   70.209000     5213.329
129  2017-01-06 20:00:00   31.919000   73.888000     5213.434
130  2017-01-06 21:00:00   56.780000   69.660000     5214.223
131  2017-01-06 22:00:00   46.120000   63.192000     5214.476
132  2017-01-06 23:00:00   40.361000   69.304000     5215.061
133  2017-01-07 00:00:00   44.473348   85.039000     5215.930
134  2017-01-07 01:00:00   38.669000   81.357000     5216.321
135  2017-01-07 02:00:00   57.487000   70.497000     5216.614
136  2017-01-07 03:00:00   33.223000   68.689000     5216.829
137  2017-01-07 04:00:00   37.345000   74.847000     5217.749
138  2017-01-07 05:00:00   57.437000   78.445000     5218.654
139  2017-01-07 06:00:00   54.598000   60.412000     5219.463
140  2017-01-07 07:00:00   35.668000   83.469000     5220.325
141  2017-01-07 08:00:00   55.828000   67.657000     5220.591
142  2017-01-07 09:00:00   39.747000   70.218000     5221.252
143  2017-01-07 10:00:00   40.033000   64.796000     5222.041
144  2017-01-07 11:00:00   39.604000   77.115000     5222.283
145  2017-01-07 12:00:00   37.286000   66.324000     5223.009
146  2017-01-07 13:00:00   58.910000   74.399000     5223.530
147  2017-01-07 14:00:00   46.537000   69.465000     5223.913
148  2017-01-07 15:00:00   30.229000   80.289000     5224.401
149  2017-01-07 16:00:00   41.824000   87.859000     5224.676
150  2017-01-07 17:00:00   42.926000   70.335000     5224.698
151  2017-01-07 18:00:00   55.907000   79.622000     5225.222
152  2017-01-07 19:00:00   48.877000   60.728000     5225.669
153  2017-01-07 20:00:00   50.119000   62.520000     5225.875
154  2017-01-07 21:00:00   40.799000   82.819000     5226.849
155  2017-01-07 22:00:00   50.242000   73.867000     5227.081
156  2017-01-07 23:00:00   41.441000   83.357000     5227.970

[157 rows x 4 columns]
```

Section 1.5: Energy Consumption Estimation

Section 1.5.1: Calculating water consumption

In [14]:

```
# Calculating water consumption
rowSize = imputedData2_reset.shape[0]
waterConsumption = [0] * rowSize
water_state = imputedData2_reset['water_state'].values
for i in range(0, rowSize-1):
    waterConsumption[i] = water_state[i+1] - water_state[i]
imputedData2_reset['water_consumption'] = pd.DataFrame(waterConsumption)
print(imputedData2_reset)
```

```
              datetime  returntemp  supplytemp  water_state  \
0    2017-01-01 00:00:00   37.341000   84.267000     5143.999
1    2017-01-01 01:00:00   58.714000   77.914000     5144.165
2    2017-01-01 02:00:00   39.888000   82.807000     5144.281
3    2017-01-01 04:00:00   31.153000   87.821000     5145.665
4    2017-01-01 05:00:00   56.389000   78.457000     5146.373
5    2017-01-01 06:00:00   54.209000   73.945000     5146.479
6    2017-01-01 07:00:00   49.498000   82.534000     5146.749
7    2017-01-01 08:00:00   50.401000   60.987000     5147.465
8    2017-01-01 09:00:00   57.336000   88.700000     5147.653
9    2017-01-01 10:00:00   31.659000   62.852000     5148.196
10   2017-01-01 11:00:00   30.697000   84.611000     5148.387
11   2017-01-01 12:00:00   57.639000   82.439000     5148.998
```

11	2017-01-01 12:00:00	37.639000	82.459000	5148.998
12	2017-01-01 13:00:00	49.571000	60.552000	5149.953
13	2017-01-01 15:00:00	35.795000	81.957000	5150.768
14	2017-01-01 16:00:00	42.593000	67.087000	5151.328
15	2017-01-01 17:00:00	45.301000	87.646000	5151.779
16	2017-01-01 18:00:00	32.275000	78.911000	5152.520
17	2017-01-01 19:00:00	43.730000	69.268000	5153.199
18	2017-01-01 20:00:00	44.296000	76.807000	5153.404
19	2017-01-01 21:00:00	35.308000	78.628000	5153.565
20	2017-01-01 22:00:00	44.473348	74.720449	5154.292
21	2017-01-01 23:00:00	30.834000	70.962000	5154.756
22	2017-01-02 00:00:00	40.900000	77.119000	5154.955
23	2017-01-02 01:00:00	41.557000	87.675000	5155.328
24	2017-01-02 02:00:00	39.615000	79.069000	5155.571
25	2017-01-02 03:00:00	45.677000	65.977000	5155.775
26	2017-01-02 04:00:00	46.118000	88.024000	5156.293
27	2017-01-02 06:00:00	52.086000	65.766000	5157.471
28	2017-01-02 07:00:00	51.001000	85.285000	5158.304
29	2017-01-02 08:00:00	41.716000	63.118000	5159.104
..	...	...	...	...
127	2017-01-06 18:00:00	59.405000	63.308000	5212.682
128	2017-01-06 19:00:00	57.619000	70.209000	5213.329
129	2017-01-06 20:00:00	31.919000	73.888000	5213.434
130	2017-01-06 21:00:00	56.780000	69.660000	5214.223
131	2017-01-06 22:00:00	46.120000	63.192000	5214.476
132	2017-01-06 23:00:00	40.361000	69.304000	5215.061
133	2017-01-07 00:00:00	44.473348	85.039000	5215.930
134	2017-01-07 01:00:00	38.669000	81.357000	5216.321
135	2017-01-07 02:00:00	57.487000	70.497000	5216.614
136	2017-01-07 03:00:00	33.223000	68.689000	5216.829
137	2017-01-07 04:00:00	37.345000	74.847000	5217.749
138	2017-01-07 05:00:00	57.437000	78.445000	5218.654
139	2017-01-07 06:00:00	54.598000	60.412000	5219.463
140	2017-01-07 07:00:00	35.668000	83.469000	5220.325
141	2017-01-07 08:00:00	55.828000	67.657000	5220.591
142	2017-01-07 09:00:00	39.747000	70.218000	5221.252
143	2017-01-07 10:00:00	40.033000	64.796000	5222.041
144	2017-01-07 11:00:00	39.604000	77.115000	5222.283
145	2017-01-07 12:00:00	37.286000	66.324000	5223.009
146	2017-01-07 13:00:00	58.910000	74.399000	5223.530
147	2017-01-07 14:00:00	46.537000	69.465000	5223.913
148	2017-01-07 15:00:00	30.229000	80.289000	5224.401
149	2017-01-07 16:00:00	41.824000	87.859000	5224.676
150	2017-01-07 17:00:00	42.926000	70.335000	5224.698
151	2017-01-07 18:00:00	55.907000	79.622000	5225.222
152	2017-01-07 19:00:00	48.877000	60.728000	5225.669
153	2017-01-07 20:00:00	50.119000	62.520000	5225.875
154	2017-01-07 21:00:00	40.799000	82.819000	5226.849
155	2017-01-07 22:00:00	50.242000	73.867000	5227.081
156	2017-01-07 23:00:00	41.441000	83.357000	5227.970

	water_consumption
0	0.166
1	0.116
2	1.384
3	0.708
4	0.106
5	0.270
6	0.716
7	0.188
8	0.543
9	0.191
10	0.611
11	0.955
12	0.815
13	0.560
14	0.451
15	0.741
16	0.679
17	0.205
18	0.161
19	0.727
20	0.464
21	0.199
22	0.373
23	0.243
24	0.204
25	0.518

```
25                0.518
26                1.178
27                0.833
28                0.800
29                0.989
..                  ...
127               0.647
128               0.105
129               0.789
130               0.253
131               0.585
132               0.869
133               0.391
134               0.293
135               0.215
136               0.920
137               0.905
138               0.809
139               0.862
140               0.266
141               0.661
142               0.789
143               0.242
144               0.726
145               0.521
146               0.383
147               0.488
148               0.275
149               0.022
150               0.524
151               0.447
152               0.206
153               0.974
154               0.232
155               0.889
156               0.000

[157 rows x 5 columns]
```

In [15]:

```
# Checking that there is no such thing as negative consumption
negativeWaterConsumptionIndices = np.where(imputedData2_reset['water_consumption']<0)
print(negativeWaterConsumptionIndices)
```

```
(array([], dtype=int64),)
```

Section 1.5.2: Calculating energy consumption

In [16]:

```
C = 1.16
energyConsumption= [0] * 157
waterConsumption = imputedData2_reset['water_consumption'].values
supplyTemp = imputedData2_reset['supplytemp'].values
returnTemp = imputedData2_reset['returntemp'].values
for i in range(0, 156):
    energyConsumption[i] = waterConsumption[i] * (supplyTemp[i] - returnTemp[i]) * C
imputedData2_reset['energy_consumption'] = pd.DataFrame(energyConsumption)
print(imputedData2_reset)
```

```
              datetime  returntemp  supplytemp  water_state  \
0    2017-01-01 00:00:00   37.341000   84.267000     5143.999
1    2017-01-01 01:00:00   58.714000   77.914000     5144.165
2    2017-01-01 02:00:00   39.888000   82.807000     5144.281
3    2017-01-01 04:00:00   31.153000   87.821000     5145.665
4    2017-01-01 05:00:00   56.389000   78.457000     5146.373
5    2017-01-01 06:00:00   54.209000   73.945000     5146.479
6    2017-01-01 07:00:00   49.498000   82.534000     5146.749
7    2017-01-01 08:00:00   50.401000   60.987000     5147.465
8    2017-01-01 09:00:00   57.336000   88.700000     5147.653
9    2017-01-01 10:00:00   31.659000   62.852000     5148.196
10   2017-01-01 11:00:00   30.697000   84.611000     5148.387
```

11	2017-01-01 12:00:00	57.639000	82.439000	5148.998
12	2017-01-01 13:00:00	49.571000	60.552000	5149.953
13	2017-01-01 15:00:00	35.795000	81.957000	5150.768
14	2017-01-01 16:00:00	42.593000	67.087000	5151.328
15	2017-01-01 17:00:00	45.301000	87.646000	5151.779
16	2017-01-01 18:00:00	32.275000	78.911000	5152.520
17	2017-01-01 19:00:00	43.730000	69.268000	5153.199
18	2017-01-01 20:00:00	44.296000	76.807000	5153.404
19	2017-01-01 21:00:00	35.308000	78.628000	5153.565
20	2017-01-01 22:00:00	44.473348	74.720449	5154.292
21	2017-01-01 23:00:00	30.834000	70.962000	5154.756
22	2017-01-02 00:00:00	40.900000	77.119000	5154.955
23	2017-01-02 01:00:00	41.557000	87.675000	5155.328
24	2017-01-02 02:00:00	39.615000	79.069000	5155.571
25	2017-01-02 03:00:00	45.677000	65.977000	5155.775
26	2017-01-02 04:00:00	46.118000	88.024000	5156.293
27	2017-01-02 06:00:00	52.086000	65.766000	5157.471
28	2017-01-02 07:00:00	51.001000	85.285000	5158.304
29	2017-01-02 08:00:00	41.716000	63.118000	5159.104
..	...	...	...	...
127	2017-01-06 18:00:00	59.405000	63.308000	5212.682
128	2017-01-06 19:00:00	57.619000	70.209000	5213.329
129	2017-01-06 20:00:00	31.919000	73.888000	5213.434
130	2017-01-06 21:00:00	56.780000	69.660000	5214.223
131	2017-01-06 22:00:00	46.120000	63.192000	5214.476
132	2017-01-06 23:00:00	40.361000	69.304000	5215.061
133	2017-01-07 00:00:00	44.473348	85.039000	5215.930
134	2017-01-07 01:00:00	38.669000	81.357000	5216.321
135	2017-01-07 02:00:00	57.487000	70.497000	5216.614
136	2017-01-07 03:00:00	33.223000	68.689000	5216.829
137	2017-01-07 04:00:00	37.345000	74.847000	5217.749
138	2017-01-07 05:00:00	57.437000	78.445000	5218.654
139	2017-01-07 06:00:00	54.598000	60.412000	5219.463
140	2017-01-07 07:00:00	35.668000	83.469000	5220.325
141	2017-01-07 08:00:00	55.828000	67.657000	5220.591
142	2017-01-07 09:00:00	39.747000	70.218000	5221.252
143	2017-01-07 10:00:00	40.033000	64.796000	5222.041
144	2017-01-07 11:00:00	39.604000	77.115000	5222.283
145	2017-01-07 12:00:00	37.286000	66.324000	5223.009
146	2017-01-07 13:00:00	58.910000	74.399000	5223.530
147	2017-01-07 14:00:00	46.537000	69.465000	5223.913
148	2017-01-07 15:00:00	30.229000	80.289000	5224.401
149	2017-01-07 16:00:00	41.824000	87.859000	5224.676
150	2017-01-07 17:00:00	42.926000	70.335000	5224.698
151	2017-01-07 18:00:00	55.907000	79.622000	5225.222
152	2017-01-07 19:00:00	48.877000	60.728000	5225.669
153	2017-01-07 20:00:00	50.119000	62.520000	5225.875
154	2017-01-07 21:00:00	40.799000	82.819000	5226.849
155	2017-01-07 22:00:00	50.242000	73.867000	5227.081
156	2017-01-07 23:00:00	41.441000	83.357000	5227.970

	water_consumption	energy_consumption
0	0.166	9.036071
1	0.116	2.583552
2	1.384	68.903879
3	0.708	46.540295
4	0.106	2.713481
5	0.270	6.181315
6	0.716	27.438380
7	0.188	2.308595
8	0.543	19.755556
9	0.191	6.911121
10	0.611	38.212087
11	0.955	27.473440
12	0.815	10.381437
13	0.560	29.986835
14	0.451	12.814281
15	0.741	36.398068
16	0.679	36.732379
17	0.205	6.072936
18	0.161	6.071754
19	0.727	36.532622
20	0.464	16.280200
21	0.199	9.263148
22	0.373	15.671237
23	0.243	12.999742
24	0.204	9.336395

```
25                0.518           12.197864
26                1.178           57.263711
27                0.833           13.218710
28                0.800           31.815552
29                0.989           24.553230
..                  ...                 ...
127               0.647            2.929280
128               0.105            1.533462
129               0.789           38.411708
130               0.253            3.780022
131               0.585           11.585059
132               0.869           29.175702
133               0.391           18.398957
134               0.293           14.508797
135               0.215            3.244694
136               0.920           37.849315
137               0.905           39.369600
138               0.809           19.714748
139               0.862            5.813535
140               0.266           14.749477
141               0.661            9.070004
142               0.789           27.888278
143               0.242            6.951469
144               0.726           31.590264
145               0.521           17.549406
146               0.383            6.881453
147               0.488           12.979082
148               0.275           15.969140
149               0.022            1.174813
150               0.524           16.660287
151               0.447           12.296702
152               0.206            2.831915
153               0.974           14.011146
154               0.232           11.308422
155               0.889           24.363045
156               0.000            0.000000

[157 rows x 6 columns]
```

Section 1.6: Saving final dataframe in csv format in the same folder as input data

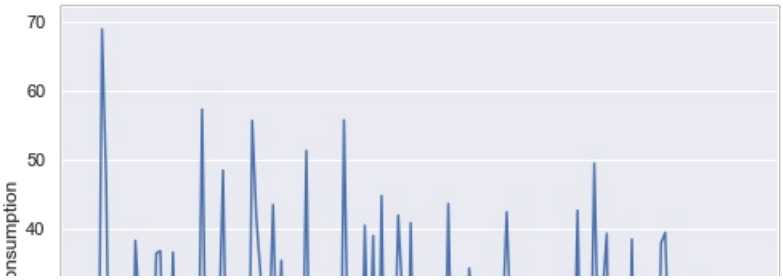
In [17]:

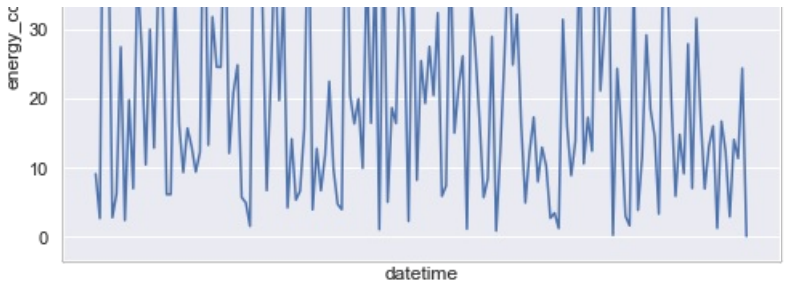
```
import os
path = os.path.dirname(inputFile)
imputedData2_reset.to_csv(path+'/output.csv')
```

Chapter 2: Discussion Task

In [18]:

```
# Task 2
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 6))
fig = plt.plot(imputedData2_reset['datetime'], imputedData2_reset['energy_consumption'])
plt.xticks([])
plt.xlabel('datetime')
plt.ylabel('energy_consumption')
plt.savefig(path+'/timeseries_plot.png')
plt.show()
```




energy_co
30
20
10
0
datetime