Data Characteristics and Data Prepration Functions

Saman Taheri, Mohammad Jooshaki, , and Moein Moeini-Aghtaei *

July 27, 2020

1 Data Characteristics

The heating and electricity consumption data are the results of an energy audit program aggregated for multiple load profiles of a residential customer. These profiles include HVAC systems loads, convenience power, elevator, etc. The datasets are gathered between December 2010 and November 2018 with a one-hour timestep resolution, thereby containing 140,160 measurements, half of which is for heat or electricity. In addition to the historical energy consumption values, a concatenation of weather variables is also available. The weather variables are air pressure, temperature, and humidity plus wind speed, cloudiness percentage, and solar irradiation at the predetermined location.

Let us begin by loading the dataset using *panda* package.

[26]:
```
import pandas as pd
import numpy as np
Load_data=pd.read_csv("Load_data.csv") # "loads.csv" is the pathway to the␣
↪dataset file.
Load_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70080 entries, 0 to 70079
Data columns (total 9 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Time                              70080 non-null  object
 1   air_pressure[mmHg]                69934 non-null  float64
 2   air_temperature[degree celcius]   69903 non-null  float64
 3   relative_humidity[%]              69903 non-null  float64
 4   wind_speed[M/S]                   69125 non-null  float64
 5   solar_irridiation[W/m²]           70080 non-null  float64
 6   total_cloud_cover[from ten]       69837 non-null  object
 7   electricity_demand_values[kw]     70073 non-null  float64
 8   heat_demand_values[kw]            70073 non-null  float64
dtypes: float64(7), object(2)
memory usage: 46.8+ MB
```

*moeinin@sharif.edu

As can be seen, all the features have numeric values. This make the preprocessing easier as working with other data types is not straightforward. (The cloudiness feature can be easily mapped to a float attribute bu replacing *"no clouds"* to 0.) However, there are two challenges yet to be addressed. The first issue is that there are some missing values in some of the features like air_humidity, wind_speed, and solar_irridiation. Another challenge is mapping the feature of hour to a to a set of data that represents time-related variables which include (I) hours of the day (from 1 to 24), (II) days of a year, (III) the day in a given month and the month number and (IV) years in the time horizon of the project. We will take care of this two challenges in the following sections.

To get a better feeling of the dataset, we can write:

[23]:
```
Load_data
```

[23]:

Time	air_ pressure [mmHg]	air_ temperature [celcius]	relative_ humidity [%]	wind_ speed [m/s]	solar_ irridiation [W/m²]	total_ cloud_ cover
12/1/2010 0	729.7	25.0	85.0	5.0	0.0	no clouds
12/1/2010 1	729.4	27.8	77.0	7.0	0.0	no clouds
12/1/2010 2	728.9	33.3	62.0	7.0	0.0	2/10-3/10
12/1/2010 3	731.6	32.2	62.0	2.0	0.0	5/10.
12/1/2010 4	732.6	22.8	96.0	3.0	0.0	2/10-3/10
...	...	...	...	...		
11/28/2018 19	733.3	24.4	60.0	3.0	0.0	no clouds
11/28/2018 20	733.6	27.8	56.0	4.0	0.0	no clouds
11/28/2018 21	732.1	38.3	22.0	3.0	0.0	no clouds
11/28/2018 22	735.3	36.7	25.0	4.0	0.0	no clouds
11/28/2018 23	735.3	23.9	74.0	3.0	0.0	no clouds

Time	electricity_demand_values [kW]	heat_demand_values [kW]
12/1/2010 0	289.567	85.65
12/1/2010 1	260.16	84.47
12/1/2010 2	247.27	90.66
12/1/2010 3	257.95	90.91
12/1/2010 4	258.255	91.01
	...	...
11/28/2018 19	379.63	112.52
11/28/2018 20	369.97	112.19
11/28/2018 21	365.00	111.42
11/28/2018 22	396.96	112.67
11/28/2018 23	489.88	113.63

[70080 rows x 9 columns]

As can be seen, all the features have numeric values. This make the preprocessing easier as working with other data types is not straightforward. (The cloudiness feature can be easily mapped to a float attribute bu replacing *"no clouds"* to 0.) However, there are two challenges yet to be addressed. The first issue is that there are some missing values in some of the features like air_humidity, wind_speed, and solar_irridiation. Another challenge is mapping the feature of hour to a to a set of data that represents time-related variables which include (I) hours of the day (from 1 to 24), (II) days of a year, (III) the day in a given month and the month number and (IV) years in the time horizon of the project. We will take care of this two challenges in the following sections.

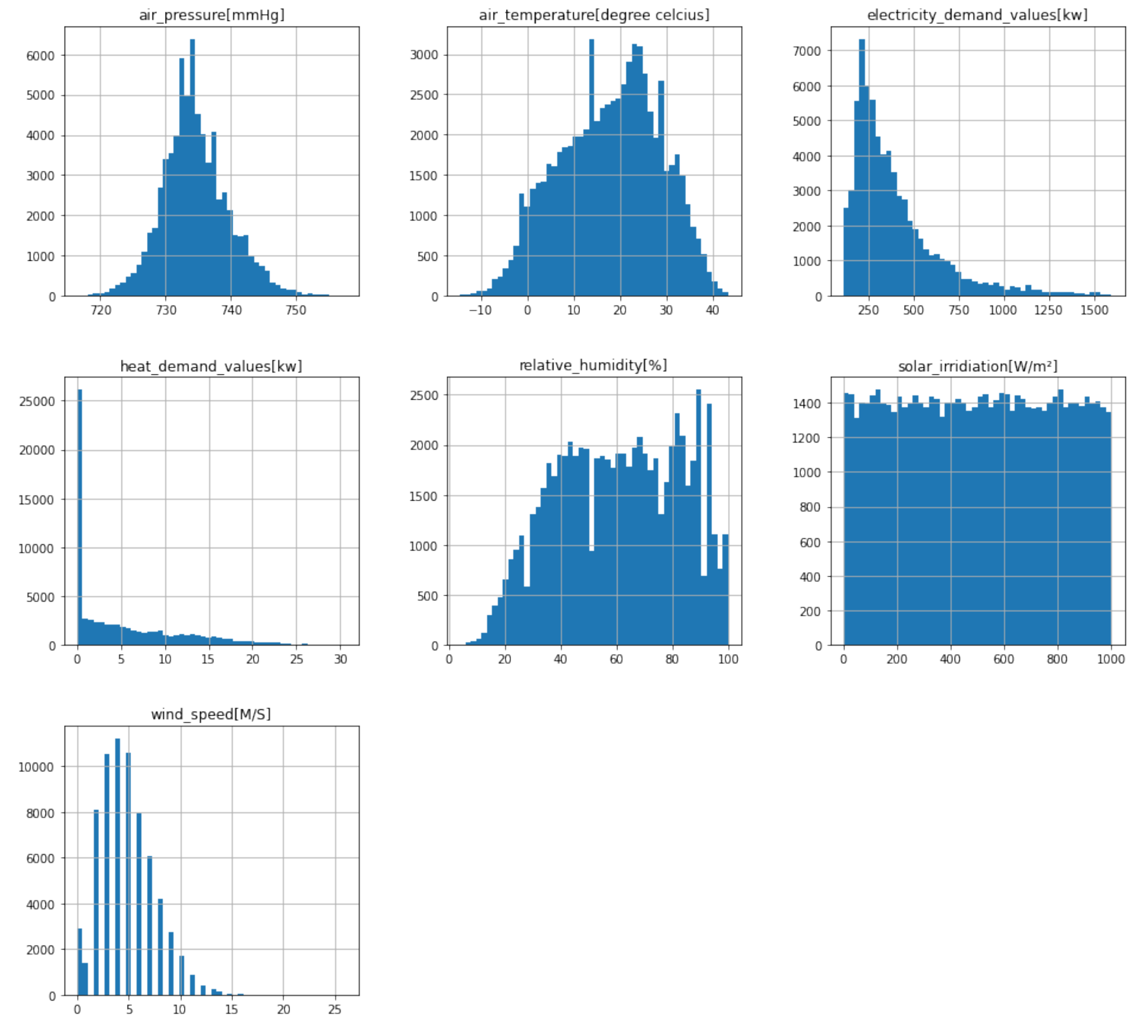
To get a better feeling of the dataset, we can see some statistical aspects of the dataset (for only numerical features) by writing:

```
[14]: Load_data.describe()
```

	air_pressure	air_temperature	relative_humidity	wind_speed	solar_irridiation	electricity_demand	heat_demand
count	69934.000000	69903.000000	69903.000000	69125.000000	70080.000000	70073.000000	70073.000000
mean	734.588143	17.871834	60.644178	4.828268	499.218491	393.888975	5.270127
std	5.011322	10.683280	22.007274	2.598960	288.620556	239.189061	6.294091
min	716.500000	-14.400000	4.000000	0.000000	0.000000	112.947618	0.000000
25%	731.400000	10.000000	43.000000	3.000000	249.940499	227.707914	0.000000
50%	734.200000	18.900000	61.000000	5.000000	500.505805	323.093703	2.745632
75%	737.500000	25.600000	79.000000	6.000000	749.580112	476.911512	8.965798
max	757.500000	43.300000	100.000000	26.000000	999.989040	1592.893206	30.583376

We can also plot histograms to gain insight as well as detect outliers.

```
[24]: # to get the histogram we can write:
%matplotlib inline
from matplotlib import pyplot as plt

Load_data.hist(bins=50, figsize=(16,15)) #x[0].hist(bins=50, figsize=(20,15))␣
 ↪this gives yoy individual histograms
plt.show()
```

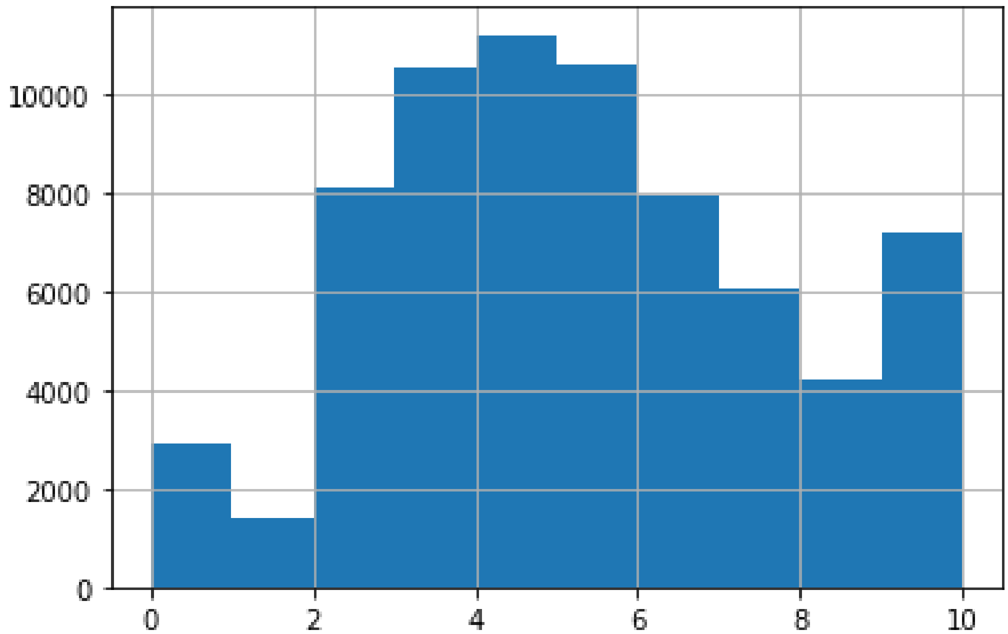


Based on the histogram figures, several actions can be implemented to improve the dataset. for example, the *"wind speed"* feature is of a discrete nature. Now, looking at its histogram, you can find that the number of instances for categories of wind_speed= 10, 11, 12, ... are relatively scarce. Therefore, one can combine all these categories into one. That will help the model to better analyze this feature. The following code is dedicated to this.

[33]:
```
# remember that each category should have lots of instances, thus we should combine all samples of 5,6,...,12 to category 5
# we canuse the predeterminde Python code:
Load_data["wind_speed_cat"] = np.ceil(Load_data["wind_speed[M/S]"])
Load_data["wind_speed_cat"].where(Load_data["wind_speed[M/S]"]< 10,10, inplace=True)
Load_data["wind_speed_cat"].hist()
```

This is the new histogram of the *"wind speed"* feature:

[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1f04f97ee48>

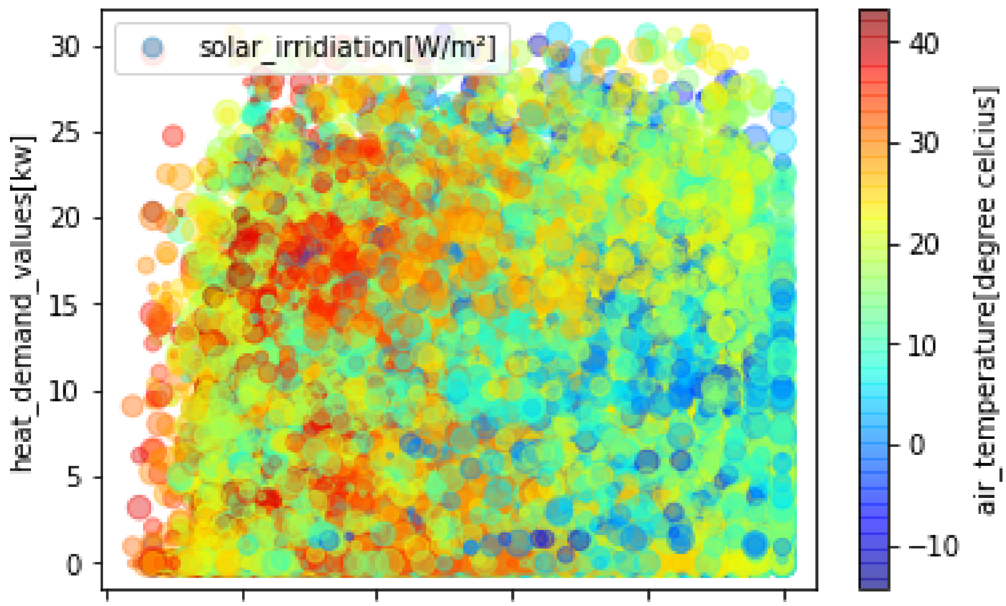


Moreover, drawing some plots based on the features can be really useful. Consider the following chart as an example.

[50]:
```
from scipy import stats
from scipy.stats import norm
# lets create a copy from traing set to go in depth even more
our_data_insight = Load_data.copy()
# we can plot the data based on geographical features to obtain a sense
X1 = our_data_insight[""]
X2 =  our_data_insight["wind_speed[M/S]"]

#plt.scatter(X1, X2, alpha=0.1) # setting alpha to 0.1 makes visualisation␣
 ↪easier (bluish dots are more frequent)

# now consider we want to engage the population size showing it by the dots␣
 ↪radius, and also housing prices
#by different color
our_data_insight.plot(kind="scatter", x="relative_humidity[%]",␣
 ↪y="heat_demand_values[kw]", alpha=0.4,
             s=our_data_insight["solar_irridiation[W/m²]"]/10,␣
 ↪label="solar_irridiation[W/m²]",
```

```
            c="air_temperature[degree celcius]", cmap=plt.get_cmap("jet"),␣
↪colorbar=True,
           )
plt.legend()
```

[50]: <matplotlib.legend.Legend at 0x1f053318688>



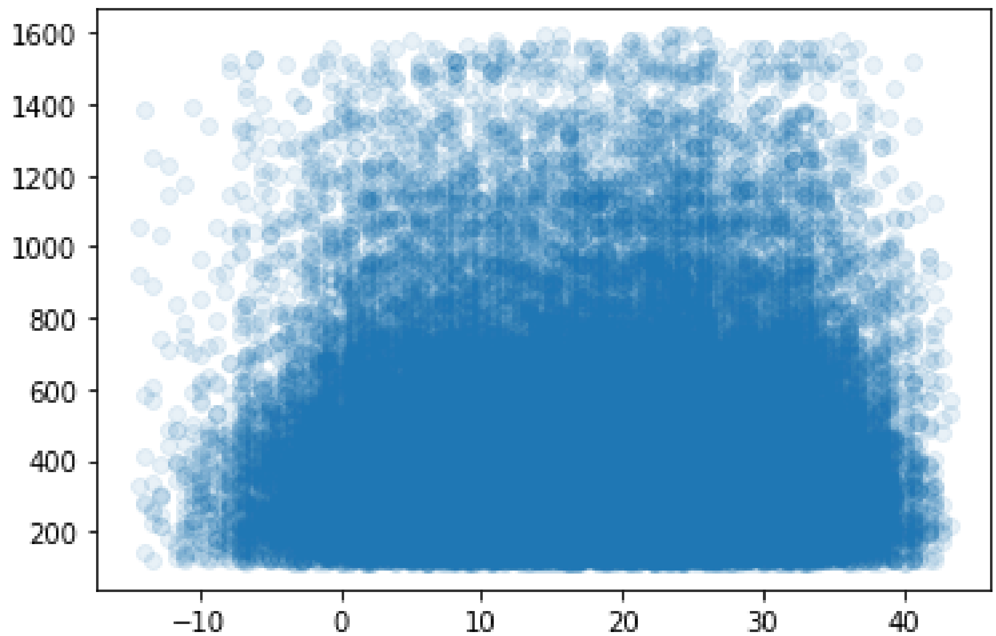
we can also compute the correlation between attributes as easy as pie:

[52]:
```
correlation_matrix = our_data_insight.corr()
#insight let's look at how much each attribute correlates with the median house␣
↪value:
#print(correlation_matrix.iloc[8].sort_values(ascending = False))
print(correlation_matrix.loc["heat_demand_values[kw]"])
#Another way to check for correlation between attributes is to use Pandas'␣
↪scatter_matrix function
# from pandas.plotting import scatter_matrix
# attributes = ["median_house_value", "median_income", "total_rooms"]
# scatter_matrix(our_data_insight[attributes], figsize=(12, 8))

# we see that our target is most correlated with income attribute, so it is␣
↪useful to plot them
plt.scatter(our_data_insight["air_temperature[degree celcius]"],␣
↪our_data_insight["electricity_demand_values[kw]"], alpha=0.1)

# 1: the correlation is realy strong
```

```
# 2: we have a prive cap of 500000 in our data
# 3: another messy data: a line around 370000 and a line around 230000, shall we␣
↪remove them?
```

```
air_pressure[mmHg]                0.016314
air_temperature[degree celcius]  -0.022504
relative_humidity[%]             -0.008758
wind_speed[M/S]                   0.011592
solar_irridiation[W/m²]           0.000733
electricity_demand_values[kw]     0.015911
heat_demand_values[kw]            1.000000
wind_speed_cat                    0.009740
Name: heat_demand_values[kw], dtype: float64
```

[52]: <matplotlib.collections.PathCollection at 0x1f055e9d688>



Just by some simple calculations, we now have a more understanding of the dataset, a very crucial step in any machine learning project. Now we have to preprocess the data.

2 Data Preprocessing

A dataset gathered for real-world applications is vulnerable to several errors/discrepancies, which might lead to poor data analysis. These discrepancies can compromise noise, incomplete data, missing values, etc. To ensure that we can gain a high-resolution knowledge from the datasets, preprocessing should be incorporated. Generally, the preprocessing phase includes several stages that can be summarized as follows:

- Firstly, we should fill missing values, remove noises, detect outliers, and resolve discrepancies within the dataset. This step is called data cleaning.
- Second, we must make sure that the data is in a usable format. This is done by feature scaling and integration of multiple files into one master file containing all data. This stage is named data transformation.
- Next, we have to check for the most and least important features that are correlated with the load prediction. By doing so, one can get an insight into the target, delete less correlated attributes, and create highly correlated new features.
- Finally, it is better to make a pipeline that does all the steps sequentially. A pipe will significantly help to organize all the processes, as well as make useable for other researchers.

Here, a full pipeline is developed using *Panda* and *Scikit* in Python, exclusively for electricity and heat demand prediction task. All the four steps are coded in a way that one with a basic knowledge of Python can implement the full pipeline straightforwardly.

[62]:
```
# 1) We are going to find the most important attribute and split the data:

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

class SplitDataByImportantAttribute(BaseEstimator,TransformerMixin):
    def fit(self, dataframe, y=None):
        return(self)
    def transform(self, dataframe, y=None):
        import sklearn
        from sklearn.model_selection import StratifiedShuffleSplit
        import sys
        import numpy as np
        import pandas as pd
        %matplotlib inline
        from matplotlib import pyplot as plt
        List1 = list(dataframe)
        print("Here is the name of all columns: ")
        print (List1)
        target_index = input("Please write the name of the target column: ")
        Correlation_matrix = dataframe.corr().loc[target_index].
 ↪sort_values(ascending = False)
        most_importatnt_attribute = Correlation_matrix.index[1]
        print(" The most important attribute is {" + most_importatnt_attribute +␣
 ↪"}  correlated by a factor of " + str(Correlation_matrix[1]))
        print(dataframe[most_importatnt_attribute].hist(), plt.
 ↪xlabel(most_importatnt_attribute), plt.ylabel("frequency"),  plt.
 ↪title('Histogram of ' + most_importatnt_attribute))
        devise_metric= dataframe[most_importatnt_attribute].mean() /␣
 ↪dataframe[most_importatnt_attribute].std()
```

```
        category_count1 = np.int(dataframe[most_importatnt_attribute].mean() + 1␣
 ↪*dataframe[most_importatnt_attribute].std())
        category_count2 = np.int(dataframe[most_importatnt_attribute].mean() - 1␣
 ↪*dataframe[most_importatnt_attribute].std())
        dataframe[most_importatnt_attribute + "_cat"]= np.
 ↪ceil(dataframe[most_importatnt_attribute]/devise_metric)
        dataframe[most_importatnt_attribute + "_cat"].
 ↪where(dataframe[most_importatnt_attribute + "_cat"]< category_count1,␣
 ↪category_count1, inplace=True)
        dataframe[most_importatnt_attribute + "_cat"].
 ↪where(dataframe[most_importatnt_attribute + "_cat"]> category_count2,␣
 ↪category_count2, inplace=True)
        print(dataframe[most_importatnt_attribute + "_cat"].hist(), plt.
 ↪xlabel(most_importatnt_attribute), plt.ylabel("frequency"),  plt.
 ↪title('Histogram of most importatnt attribute' ))
        split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,␣
 ↪random_state=42) # it defines split characteristic
        x = split.split(dataframe, dataframe[most_importatnt_attribute +␣
 ↪"_cat"]) #it splits the attribute and gives indices for test an training
        for train_index, test_index in x:
            strat_train_set = dataframe.loc[train_index]
            strat_test_set = dataframe.loc[test_index]
        strat_train_set = strat_train_set.drop(columns =␣
 ↪most_importatnt_attribute + "_cat")
        strat_test_set = strat_test_set.drop(columns = most_importatnt_attribute␣
 ↪+ "_cat")
        return(strat_train_set)

# after this implementation you put the strat_test_set aside completely.
```

[63]:
```
#2) try to find better attributes with feature engineering insight. It is very␣
 ↪case dependent. yet the most usefule key is:
#   attributes with near zero correlation to the target, are better be replaced␣
 ↪or changed.
class AttributeAdder(BaseEstimator,TransformerMixin):
    def __init__(self, test_number):
        self.test_number = test_number
    def fit(self, strat_train_set, y=None):
        return(self)
    def transform(self, strat_train_set, y=None):
        import numpy as np
        import pandas as pd
        list1 = list(strat_train_set)
        print("Here is the name of all columns: ")
        print (list1)
        target_index = input("Please write the name of the target column: ")
```

```
        correlation_matrix = strat_train_set.corr().loc[target_index]
        list2=[]
        for i in range(len(correlation_matrix)):
            if correlation_matrix[i] <0.15 and correlation_matrix[i]>-0.15:
                list2.append(correlation_matrix.index[i])
        for i in range(self.test_number):
            index1 = np.random.randint(len(list2))
            index2 = np.random.randint(len(list2))
            x = list2[index1]
            y = list2[index2]
            z = x+ "_per_"+ y
            test_attributes =  pd.DataFrame()
            test_attributes[target_index] = strat_train_set[target_index]
            test_attributes[z] = strat_train_set[x] /strat_train_set[y]
            correlation_with_target = test_attributes.corr().loc[z][target_index]
            if correlation_with_target >0.15 or correlation_with_target <-0.15:
                print(z + " have a pretty high correlation "+␣
 ↪str(correlation_with_target) +" with target")
                strat_train_set[z] = test_attributes[z]
            else:
                print(z + "is not much suitable")

        return(strat_train_set)

# now you can decide what of this attributes  can be added to the training set.
```

[64]:

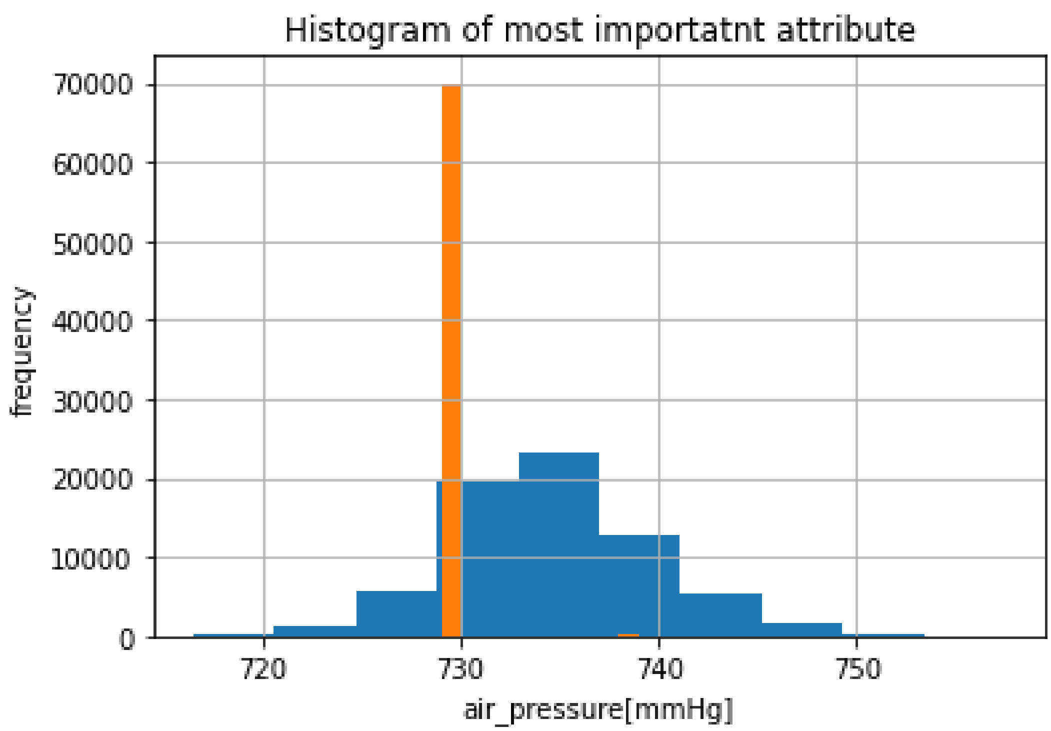
```
# Data cleaning: a) fill null data with "median or ..." using imputers and␣
 ↪transforms,
class filling_NaN(BaseEstimator,TransformerMixin):
    def __init__(self, strategy):
        self.strategy =  strategy
    def fit(self, strat_train_test, y=None):
        return(self)
    def transform(self, strat_train_set, y=None):
        from sklearn.impute import SimpleImputer
        from sklearn.preprocessing import LabelBinarizer
        from sklearn.preprocessing import StandardScaler
        imputer = SimpleImputer(strategy = self.strategy)
        labeler = LabelBinarizer()
        scaler =  StandardScaler()
        target_index = input("Please write the name of the target column: ")
        X_strat_train_set= strat_train_set.drop(columns = [target_index])
        Y_train = strat_train_set[target_index]
        list1 = X_strat_train_set.dtypes
        list2=[]
        list3= []
        for i in range(len(list1)):
```

```
            if list1[i] != object:
                list2.append(list1.index[i])
            else:
                list3.append(list1.index[i])
        numerical_attributes = X_strat_train_set[list2]
        x = pd.DataFrame(imputer.fit_transform(numerical_attributes), columns =␣
↪list2)
        for item in list3:
            y= labeler.fit_transform(X_strat_train_set[item])
            counter = 0
            for j in range(len(y[0])):
                z= []
                for i in range(len(y)):
                    z.append(y[i][j])
                counter += 1
                x[item + str(counter)]= z

        X_train_fully_prepared =  scaler.fit_transform(x)
        return(X_train_fully_prepared, Y_train)
```

[65]:
```
full_prepration = Pipeline([
    ("spliter", SplitDataByImportantAttribute()),
    ("featurer", AttributeAdder(10)), # increase the number (10) to check more␣
↪attributes
    ("filler", filling_NaN("median")),
])
X_train_heat, Y_train_heat =  full_prepration.fit_transform(Load_data)
```

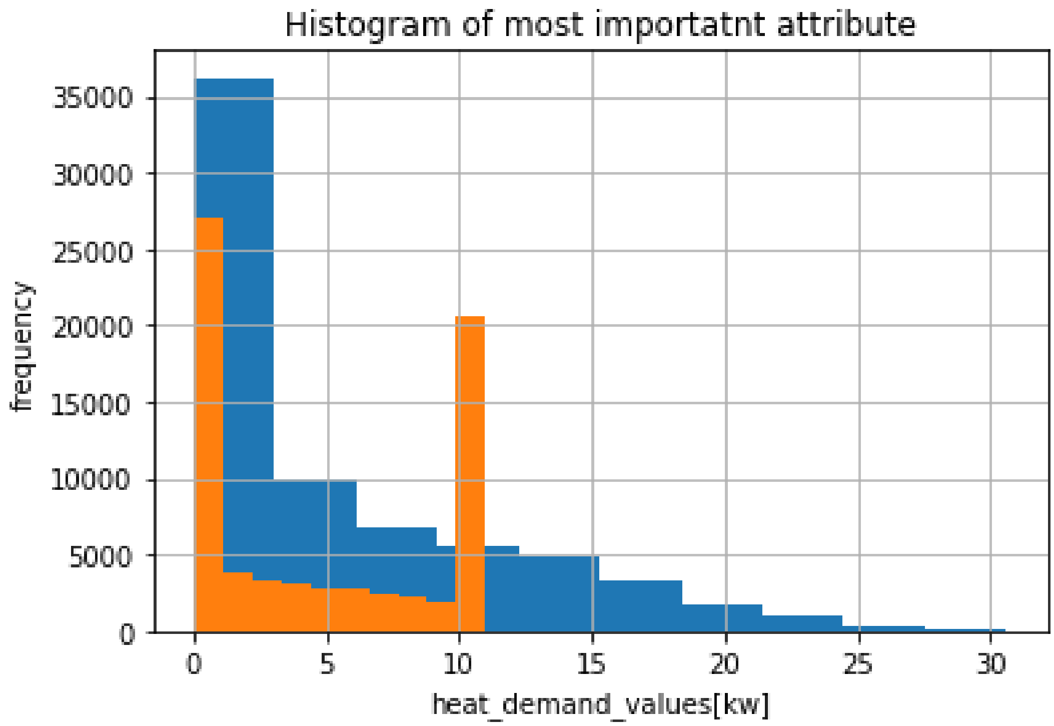
```
Here is the name of all columns:
['Time', 'air_pressure[mmHg]', 'air_temperature[degree celcius]',
'relative_humidity[%]', 'wind_speed[M/S]', 'solar_irridiation[W/m²]',
'total_cloud_cover[from ten]', 'electricity_demand_values[kw]',
'heat_demand_values[kw]', 'wind_speed_cat']
Please write the name of the target column: heat_demand_values[kw]
 The most important attribute is {air_pressure[mmHg]}  correlated by a factor of
0.016313658406457848
AxesSubplot(0.125,0.125;0.775x0.755) Text(0.5, 0, 'air_pressure[mmHg]') Text(0,
0.5, 'frequency') Text(0.5, 1.0, 'Histogram of air_pressure[mmHg]')
AxesSubplot(0.125,0.125;0.775x0.755) Text(0.5, 0, 'air_pressure[mmHg]') Text(0,
0.5, 'frequency') Text(0.5, 1.0, 'Histogram of most importatnt attribute')
Here is the name of all columns:
['Time', 'air_pressure[mmHg]', 'air_temperature[degree celcius]',
'relative_humidity[%]', 'wind_speed[M/S]', 'solar_irridiation[W/m²]',
'total_cloud_cover[from ten]', 'electricity_demand_values[kw]',
'heat_demand_values[kw]', 'wind_speed_cat']
Please write the name of the target column: heat_demand_values[kw]
wind_speed[M/S]_per_solar_irridiation[W/m²]is not much suitable
```

```
wind_speed_cat_per_wind_speed[M/S]is not much suitable
wind_speed[M/S]_per_solar_irridiation[W/m²]is not much suitable
wind_speed_cat_per_air_pressure[mmHg]is not much suitable
relative_humidity[%]_per_wind_speed_catis not much suitable
solar_irridiation[W/m²]_per_wind_speed[M/S]is not much suitable
air_temperature[degree celcius]_per_wind_speed[M/S]is not much suitable
air_pressure[mmHg]_per_electricity_demand_values[kw] have a pretty high
correlation -0.2490897131134156 with target
wind_speed_cat_per_relative_humidity[%]is not much suitable
air_temperature[degree celcius]_per_wind_speed[M/S]is not much suitable
air_temperature[degree celcius]_per_air_temperature[degree celcius]is not much
suitable
solar_irridiation[W/m²]_per_electricity_demand_values[kw] have a pretty high
correlation -0.15086054699987997 with target
electricity_demand_values[kw]_per_air_pressure[mmHg]is not much suitable
electricity_demand_values[kw]_per_air_pressure[mmHg]is not much suitable
electricity_demand_values[kw]_per_solar_irridiation[W/m²]is not much suitable
Please write the name of the target column: heat_demand_values[kw]

        ----------------------------------------------------------------------------
```



[66]:
```
full_prepration = Pipeline([
    ("spliter", SplitDataByImportantAttribute()),
    ("featurer", AttributeAdder(3)),
    ("filler", filling_NaN("median")),
])
X_train_electricty, Y_train_electricty =  full_prepration.
 ↪fit_transform(Load_data)
```

```
Here is the name of all columns:
['Time', 'air_pressure[mmHg]', 'air_temperature[degree celcius]',
'relative_humidity[%]', 'wind_speed[M/S]', 'solar_irridiation[W/m²]',
'total_cloud_cover[from ten]', 'electricity_demand_values[kw]',
'heat_demand_values[kw]', 'wind_speed_cat', 'air_pressure[mmHg]_cat']
Please write the name of the target column: electricity_demand_values[kw]
 The most important attribute is {heat_demand_values[kw]}  correlated by a
factor of 0.015910533100492504
AxesSubplot(0.125,0.125;0.775x0.755) Text(0.5, 0, 'heat_demand_values[kw]')
Text(0, 0.5, 'frequency') Text(0.5, 1.0, 'Histogram of heat_demand_values[kw]')
AxesSubplot(0.125,0.125;0.775x0.755) Text(0.5, 0, 'heat_demand_values[kw]')
Text(0, 0.5, 'frequency') Text(0.5, 1.0, 'Histogram of most importatnt
attribute')
Here is the name of all columns:
['Time', 'air_pressure[mmHg]', 'air_temperature[degree celcius]',
'relative_humidity[%]', 'wind_speed[M/S]', 'solar_irridiation[W/m²]',
'total_cloud_cover[from ten]', 'electricity_demand_values[kw]',
'heat_demand_values[kw]', 'wind_speed_cat', 'air_pressure[mmHg]_cat']
Please write the name of the target column: electricity_demand_values[kw]
air_pressure[mmHg]_cat_per_solar_irridiation[W/m²]is not much suitable
wind_speed[M/S]_per_air_pressure[mmHg]_catis not much suitable
heat_demand_values[kw]_per_wind_speed_catis not much suitable
Please write the name of the target column: electricity_demand_values[kw]

        ---------------------------------------------------------------------------
```



3 Reshaping datasets

A significant hyperparameter is the size of the time window that you are going to feed into the Network at each iteration. Dataprepration for DRNNs requires series of data to be preprocessed in sliding sequences. The observations DRNNs receive as inputs are series of data. Each series covers a time window of the series. Their length is an hyperparameter of the model that one can choose The output sequence will be 1 days instead, meaning I'll try to predict the next week of energy consumption. Understanding the shape of input data for DRNNs is a crucial aspect. Input data must follow this pattern:

[Number of observations , Window size , Number of input series]

The Number of observations is straightforward. The Number of input series is just 1, since this is a univariate exercise. Window size is the hyperparameter that we chose above.

[ ]:

```
def DRNN_datareshape(series, len_input, len_pred):
import numpy as np

# create a matrix of sequences
S = np.empty((len(series)-(len_input+len_pred)+1,
len_input+len_pred))

# take each row/time window
for i in range(S.shape[0]):
```

```
        S[i,:] = series[i : i+len_input+len_pred]

        # first (len_input) cols of S are train
        train = S[: , :len_input]

        # last (len_pred) cols of S are test
        test = S[: , -len_pred:]

        # set common data type
        train = train.astype(np.float32)
        test = test.astype(np.float32)

        # reshape data as required by Keras LSTM
        train = train.reshape((len(train), len_input, 1))
        test = test.reshape((len(test), len_pred))

        return(train, test)


        # Get all Train and Test data for electrcity
        X_train_electricty, X_test_electricty =␣
↪DRNN_datareshape(X_train_electricty, input_length, prediction_length)
        Y_train_electricity, Y_test_electricity=␣
↪DRNN_datareshape(Y_train_electricty, input_length, prediction_length)


        # Get all Train and Test data for electrcity
        X_train_heat,X_test_heat = DRNN_datareshape(X_train_heat, input_length,␣
↪prediction_length)
        Y_train_heat,Y_test_heat = DRNN_datareshape(Y_train_heat, input_length,␣
↪prediction_length)




        # create a csv file to store trains and tests seperately

        X_train_electricity.to_csv (r'C:
↪\Users\surface\Desktop\Python\X_train_electricity.csv', index = False,␣
↪header=True)
        Y_train_electricity.to_csv (r'C:
↪\Users\surface\Desktop\Python\Y_train_electricity.csv', index = False,␣
↪header=True)
        X_test_electricity.to_csv (r'C:
↪\Users\surface\Desktop\Python\X_test_electricity.csv', index = False,␣
↪header=True)
```

```
    Y_test_electricity.to_csv (r'C:
↪\Users\surface\Desktop\Python\Y_test_electricity.csv', index = False,␣
↪header=True)

    X_train_heat.to_csv (r'C:\Users\surface\Desktop\Python\X_train_heat.
↪csv', index = False, header=True)
    Y_train_heat.to_csv (r'C:\Users\surface\Desktop\Python\Y_train_heat.
↪csv', index = False, header=True)
    X_test_heat.to_csv (r'C:\Users\surface\Desktop\Python\X_test_heat.csv',␣
↪index = False, header=True)
    Y_test_heat.to_csv (r'C:\Users\surface\Desktop\Python\Y_test_heat.csv',␣
↪index = False, header=True)
```