```
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
```

```
df = pd.read_excel('WA_WIND_DATA (2).xlsx',index_col=0)
```

```
print(df)
```

```
                     Pressure | (atm)  ...  Power generated by system | (kW)
DateTime                               ...
2007-01-01 00:00:00          0.890467  ...                              0.00
2007-01-01 01:00:00          0.890583  ...                              0.00
2007-01-01 02:00:00          0.890366  ...                           1986.68
2007-01-01 03:00:00          0.890326  ...                           2597.61
2007-01-01 04:00:00          0.890549  ...                           2555.46
...                               ...  ...                               ...
2009-12-31 19:00:00          0.879789  ...                          53503.30
2009-12-31 20:00:00          0.878944  ...                          53439.40
2009-12-31 21:00:00          0.878314  ...                          53376.10
2009-12-31 22:00:00          0.877968  ...                          53175.40
2009-12-31 23:00:00          0.877938  ...                          51145.80

[26280 rows x 4 columns]
```

```
df.head()
```

DateTime	Pressure | (atm)	Wind direction | (deg)	Wind speed | (m/s)	Power generated by system | (kW)
2007-01-01 00:00:00	0.890467	104	3.429	0.00
2007-01-01 01:00:00	0.890583	106	3.579	0.00
2007-01-01 02:00:00	0.890366	102	4.307	1986.68

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 26280 entries, 2007-01-01 00:00:00 to 2009-12-31 23:00:00
Data columns (total 4 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Pressure | (atm)                  26280 non-null  float64
 1   Wind direction | (deg)            26280 non-null  int64
 2   Wind speed | (m/s)                26280 non-null  float64
 3   Power generated by system | (kW)  26280 non-null  float64
dtypes: float64(3), int64(1)
memory usage: 1.0 MB
```

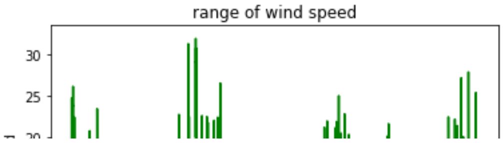
```
df['Wind speed | (m/s)'].max()
```

```
31.95
```

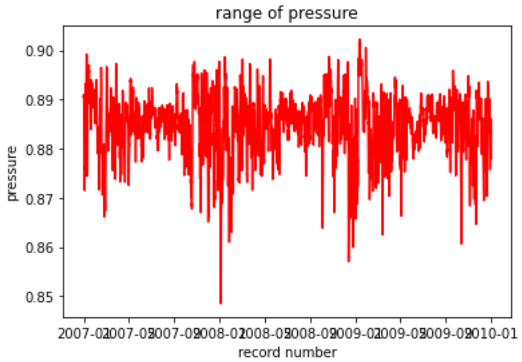
```
df['Wind speed | (m/s)'].min()
```

```
0.085
```

```
plt.plot(df['Wind speed | (m/s)'],color='green')
plt.xlabel("record number")
plt.ylabel("wind speed")
plt.title("range of wind speed")
plt.show()
```




```
plt.plot(df['Pressure | (atm)'],color='red')
plt.xlabel("record number")
plt.ylabel("pressure")
plt.title("range of pressure")
plt.show()
```




```
q1=df['Pressure | (atm)'].quantile(0.25)
q3=df['Pressure | (atm)'].quantile(0.75)
```

```
iqr=q3-q1
```

```
iqr
```

```
0.007184250000000003
```

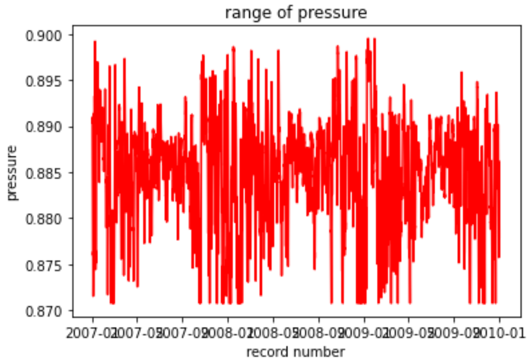
```
upper_limit=q3 +1.5*iqr
lower_limit=q1 -1.5*iqr
upper_limit,lower_limit
```

```
(0.8995003749999999, 0.8707633749999999)
```

```
def limit_imputer(value):
    if value>upper_limit:
        return upper_limit
    if value<lower_limit:
        return lower_limit
    else:
        return value
```

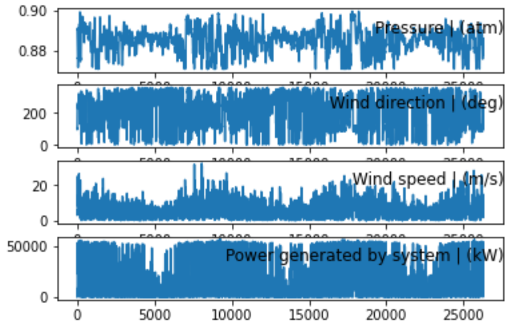
```
df['Pressure | (atm)']=df['Pressure | (atm)'].apply(limit_imputer)
```

```
plt.plot(df['Pressure | (atm)'],color='red')
plt.xlabel("record number")
plt.ylabel("pressure")
plt.title("range of pressure")
plt.show()
```




```
values = df.values
# specify columns to plot
```

```
groups = [0, 1, 2, 3]
i = 1
# plot each column
plt.figure()
for group in groups:
    plt.subplot(len(groups), 1, i)
    plt.plot(values[:, group])
    plt.title(df.columns[group], y=0.5, loc='right')
    i += 1
plt.show()
```




```
from sklearn.preprocessing import MinMaxScaler
from pandas import concat
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df1 = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df1.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df1.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
values = df.values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
print(reframed.head())
```

```
   var1(t-1)  var2(t-1)  var3(t-1)  ...   var2(t)   var3(t)   var4(t)
1   0.685654   0.288889   0.104943  ...  0.294444  0.109650  0.000000
2   0.689690   0.294444   0.109650  ...  0.283333  0.132496  0.035192
3   0.682139   0.283333   0.132496  ...  0.275000  0.140499  0.046014
4   0.680747   0.275000   0.140499  ...  0.272222  0.140217  0.045268
5   0.688507   0.272222   0.140217  ...  0.272222  0.144673  0.051766

[5 rows x 8 columns]
```

```
# split into train and test sets
values = reframed.values
n_train_hours = 365 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(8760, 1, 7) (8760,) (17519, 1, 7) (17519,)
```

```
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM


# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=26281, batch_size=120, validation_data=(test_X, test_y), verbose=2, shuffle=False)
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```

```
Streaming output truncated to the last 5000 lines.
Epoch 23782/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0062
Epoch 23783/26281
73/73 - 0s - loss: 0.0038 - val_loss: 0.0061
Epoch 23784/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0061
Epoch 23785/26281
73/73 - 0s - loss: 0.0040 - val_loss: 0.0062
Epoch 23786/26281
73/73 - 0s - loss: 0.0038 - val_loss: 0.0065
Epoch 23787/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0062
Epoch 23788/26281
73/73 - 0s - loss: 0.0038 - val_loss: 0.0063
Epoch 23789/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0059
Epoch 23790/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0062
Epoch 23791/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0060
Epoch 23792/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0060
Epoch 23793/26281
73/73 - 0s - loss: 0.0040 - val_loss: 0.0061
Epoch 23794/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0064
Epoch 23795/26281
73/73 - 0s - loss: 0.0037 - val_loss: 0.0066
Epoch 23796/26281
73/73 - 0s - loss: 0.0038 - val_loss: 0.0063
Epoch 23797/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0061
Epoch 23798/26281
73/73 - 0s - loss: 0.0038 - val_loss: 0.0062
Epoch 23799/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0061
Epoch 23800/26281
73/73 - 0s - loss: 0.0037 - val_loss: 0.0059
Epoch 23801/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0057
Epoch 23802/26281
73/73 - 0s - loss: 0.0040 - val_loss: 0.0062
Epoch 23803/26281
73/73 - 0s - loss: 0.0038 - val_loss: 0.0063
Epoch 23804/26281
73/73 - 0s - loss: 0.0038 - val_loss: 0.0061
Epoch 23805/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0059
Epoch 23806/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0061
Epoch 23807/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0059
Epoch 23808/26281
73/73 - 0s - loss: 0.0039 - val_loss: 0.0061
Epoch 23809/26281
73/73 - 0s - loss: 0.0040 - val_loss: 0.0059
```