q4

April 3, 2020

1 Question-4 House Electricity Consumption Prediction

[0]:
```
from zipfile import ZipFile
file_name="household_power_consumption.zip"

with ZipFile(file_name,'r') as zip:
  zip.extractall()
  print('Done')

# !unzip /content/household_power_consumption.zip
```

```
Done
```

2 Reading the data file

[0]:
```
%tensorflow_version 1.x
import pandas as pd

headers = ['Date', 'Time', 'Global_active_power', 'Global_reactive_power',
           'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
           'Sub_metering_3']

dtypes = {'Date':'str', 'Time':'str', 'Global_active_power':'float',
          'Global_reactive_power': 'float', 'Voltage':'float',
          'Global_intensity':'float', 'Sub_metering_1':'float',
          'Sub_metering_2':'float', 'Sub_metering_3':'float'}

df = pd.read_csv('household_power_consumption.txt', sep=';',
                 dtype=dtypes, na_values=['?'])
df.head()
```

[0]:

	Date	Time	…	Sub_metering_2	Sub_metering_3
0	16/12/2006	17:24:00	…	1.0	17.0
1	16/12/2006	17:25:00	…	1.0	16.0
2	16/12/2006	17:26:00	…	2.0	17.0

```
3  16/12/2006  17:27:00  …             1.0            17.0
4  16/12/2006  17:28:00  …             1.0            17.0

[5 rows x 9 columns]
```

[0]:
```
import numpy as np
from multiprocessing import cpu_count, Pool

def parallel_map(data, func):
    n_cores = cpu_count()
    data_split = np.array_split(data, n_cores)
    pool = Pool(n_cores)
    data = pd.concat(pool.map(func, data_split))
    pool.close()
    pool.join()
    return data

def parse(row):
    row['DateTime'] = pd.to_datetime(row['DateTime'],
                                     format='%d/%m/%Y %H:%M:%S')
    return row
```

[0]:
```
df['DateTime'] = df['Date'] + ' ' + df['Time']
df = parallel_map(df, parse)
df.dtypes
```

[0]:
```
Date                          object
Time                          object
Global_active_power          float64
Global_reactive_power        float64
Voltage                      float64
Global_intensity             float64
Sub_metering_1               float64
Sub_metering_2               float64
Sub_metering_3               float64
DateTime               datetime64[ns]
dtype: object
```

[0]:
```
df.drop(['Date', 'Time'], axis=1, inplace=True)
df = df[[df.columns[-1]] + list(df.columns[:-1])]
df.set_index('DateTime', inplace=True)
df.head()
```

[0]:
```
                     Global_active_power  …  Sub_metering_3
DateTime                                  …
2006-12-16 17:24:00                4.216  …            17.0
2006-12-16 17:25:00                5.360  …            16.0
```

```
2006-12-16 17:26:00                5.374  …           17.0
2006-12-16 17:27:00                5.388  …           17.0
2006-12-16 17:28:00                3.666  …           17.0

[5 rows x 7 columns]
```

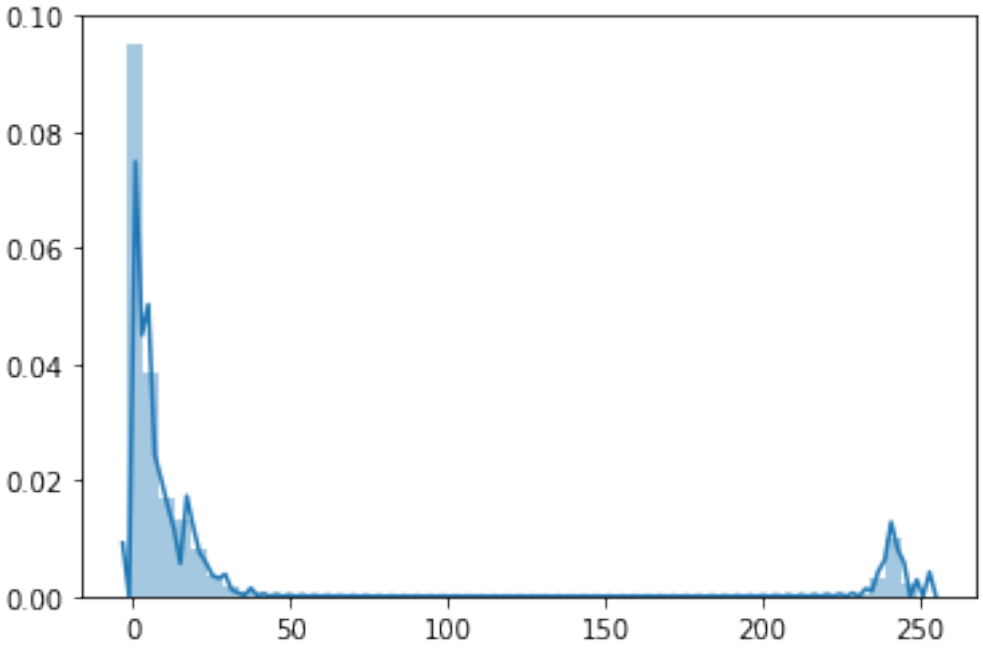
[0]:
```
df['hour'] = df.index.hour
df['day'] = df.index.day
df['month'] = df.index.month
df['day_of_week'] = df.index.dayofweek
```

[0]:
```
df['Rest_active_power'] = df['Global_active_power'] * 1000 / 60 - \
                          df['Sub_metering_1'] - df['Sub_metering_2'] - \
                          df['Sub_metering_3']
```

3 Identifying and handling Missing Data

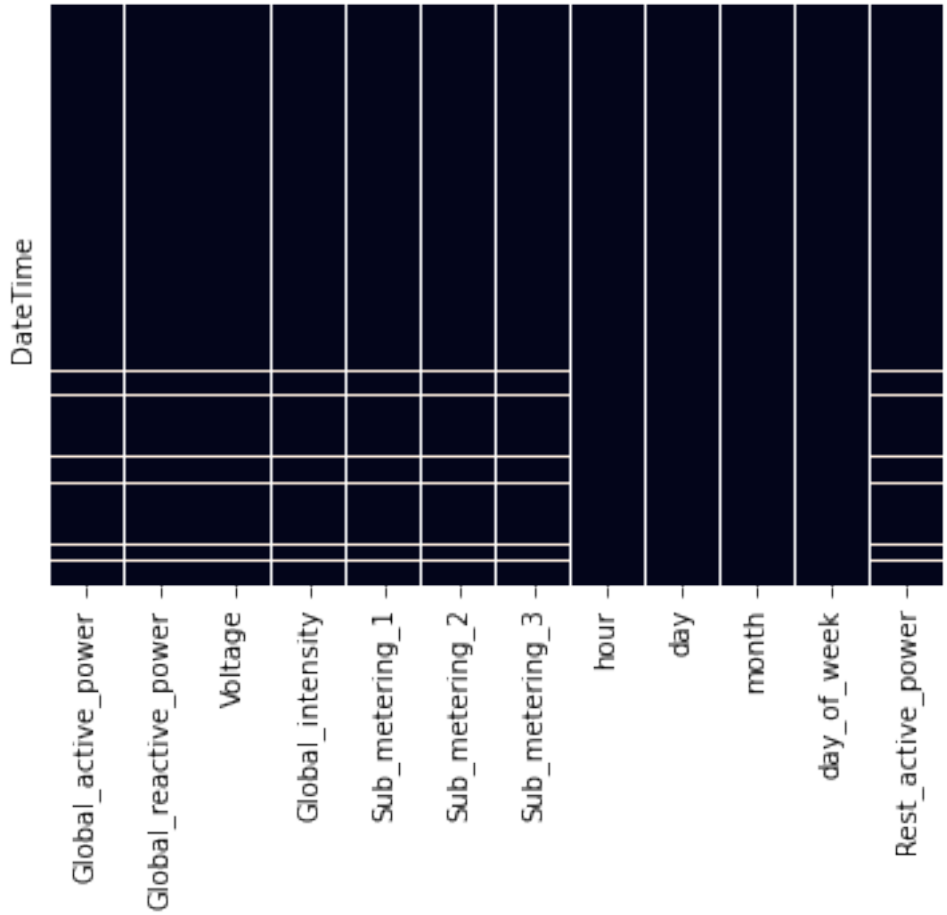
[0]:
```
import seaborn as sns
sns.distplot(df)
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbcef5a8358>
```



[0]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbd67132198>



[0]:
```
number = len(df) - len(df.dropna())
percentage = number * 100 / len(df)
print(f'Number of points with missing values is: {number}')
print(f'Percentage of points with missing values is: {percentage}\n')
print(f'Missing value counts:\n{df.isnull().sum(axis=0)}\n')
```

```
Number of points with missing values is: 25979
Percentage of points with missing values is: 1.2518437457686005

Missing value counts:
Global_active_power      25979
Global_reactive_power    25979
Voltage                  25979
Global_intensity         25979
Sub_metering_1           25979
Sub_metering_2           25979
```

```
Sub_metering_3          25979
hour                        0
day                         0
month                       0
day_of_week                 0
Rest_active_power       25979
dtype: int64
```

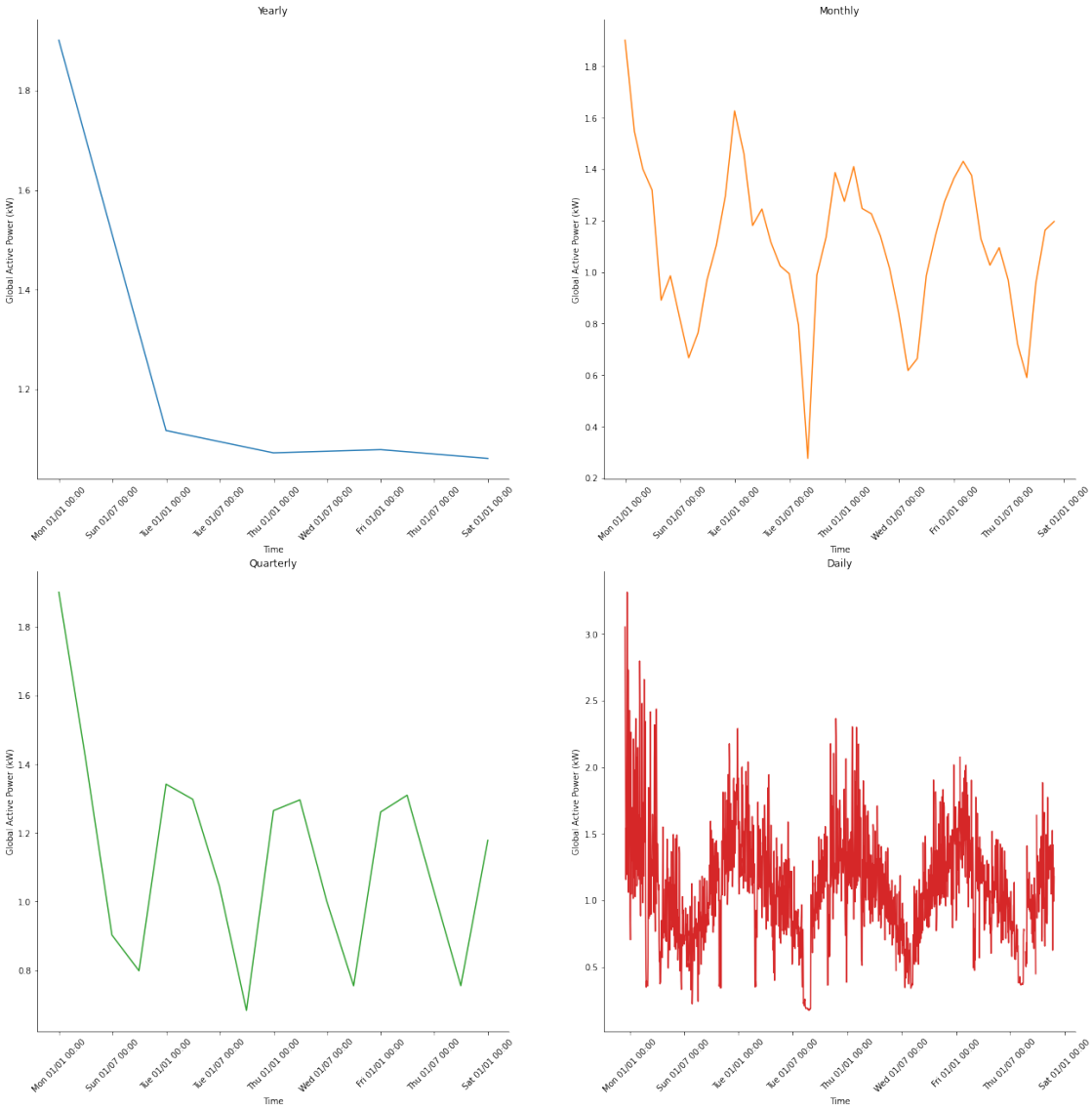
#Power consumption over the whole timespan

Average global active power over resampled data yearly, quarterly, monthly, and daily.

[0]:
```
# Tableau colors
tableau20 = [(31, 119, 180), (174, 199, 232), (255, 127, 14), (255, 187, 120),
             (44, 160, 44), (152, 223, 138), (214, 39, 40), (255, 152, 150),
             (148, 103, 189), (197, 176, 213), (140, 86, 75), (196, 156, 148),
             (227, 119, 194), (247, 182, 210), (127, 127, 127), (199, 199, 199),
             (188, 189, 34), (219, 219, 141), (23, 190, 207), (158, 218, 229)]

# Scale the RGB values to the [0, 1] range, which is the format matplotlib␣
 ↪accepts.
for i in range(len(tableau20)):
    r, g, b = tableau20[i]
    tableau20[i] = (r / 255., g / 255., b / 255.)
```

[0]:
```
import matplotlib.pylab as plt
import matplotlib.dates as mdates
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
my_fmt = mdates.DateFormatter('%a %d/%m %H:%M')

fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(22, 22))

frequencies = ['1Y', '1M', '1Q', '1D']

dic = {'1Y':'Yearly', '1M':'Monthly', '1Q':'Quarterly', '1D':'Daily'}

i = 0
for row in ax:
    for col in row:
        for tick in col.get_xticklabels():
            tick.set_rotation(45)
        col.plot(df[['Global_active_power']].resample(frequencies[i]).mean(),
                 color=tableau20[i * 2])
        col.set_xlabel('Time')
        col.set_ylabel('Global Active Power (kW)')
        col.set_title(dic[frequencies[i]])
```

```
        col.xaxis.set_major_formatter(my_fmt)

        # Aesthetics
        col.spines["top"].set_visible(False)
        col.spines["right"].set_visible(False)
        i += 1
```



4 supervised learning

Predict global active power at time $t + h$ given all the variables at times $t, t-1, ..., t-w$, where h is the prediction horizon and w is the window size. Here's a function that takes as input a time

series dataframe, a window size, a horizon, a set of variables to be lagged and a set of variables to be forecasted and outputs the dataframe transformed and ready for supervised learing.

[0]:
```
from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, window_size=1, horizon=1, inputs='all',␣
↪targets='all'):
    """
    Frame a time series as a supervised learning dataset.

    Arguments:
        data: A pandas DataFrame containing the time series
        (the index must be a DateTimeIndex).
        window_size: Number of lagged observations as input.
        horizon: Number of steps to forecast ahead.
        inputs: A list of the columns of the dataframe to be lagged.
        targets: A list of the columns of the dataframe to be forecasted.

    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """

    if targets == 'all':
        targets = data.columns

    if inputs == 'all':
        inputs = data.columns


    result = DataFrame(index=df.index)
    names = []

    # input sequence (t-w, ..., t-1)
    for i in range(window_size, 0, -1):
        result = pd.concat([result, data[inputs].shift(i)], axis=1)
        names += [(f'{data[inputs].columns[j]}(t-{i})') for j in␣
↪range(len(inputs))]

    # the input not shifted (t)
    result = pd.concat([result, data.copy()], axis=1)
    names += [(f'{column}(t)') for column in data.columns]

    # forecast (t+h)
    for i in [horizon]:
        result = pd.concat([result, data[targets].shift(-i)], axis=1)
```

```
        names += [(f'{data[targets].columns[j]}(t+{i})') for j in␣
↪range(len(targets))]

    # put it all together
    result.columns = names

    # drop rows with NaN values
    result.dropna(inplace=True)
    return result
```

[0]:
```
inputs = ['Global_active_power', 'Global_reactive_power', 'Voltage',
          'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
          'Sub_metering_3', 'Rest_active_power']

targets = ['Global_active_power']

df_supervised = series_to_supervised(df, window_size=5, horizon=1,␣
↪inputs=inputs, targets=targets)
df_supervised.head()
```

[0]:

DateTime	Global_active_power(t-5)	…	Global_active_power(t+1)
2006-12-16 17:29:00	4.216	…	3.702
2006-12-16 17:30:00	5.360	…	3.700
2006-12-16 17:31:00	5.374	…	3.668
2006-12-16 17:32:00	5.388	…	3.662
2006-12-16 17:33:00	3.666	…	4.448

[5 rows x 53 columns]

[0]:
```
#storing into some file to visualize at different window size
#df_supervised.to_csv('supervised_w10_h1.csv', index=True)
```

5 Supervised Machine Learning

[0]:
```
# import pandas as pd

# df_supervised = pd.read_csv('supervised_w10_h1.csv', parse_dates=['DateTime'])
# df_supervised.set_index('DateTime', inplace=True)
```

6 Splitting data

splitting data into train,validate and test

[0]:
```
def train_validate_test_split(df, train_percent=.6, validate_percent=.2,␣
↪seed=None):
    np.random.seed(seed)
    m = len(df)
    train_end = int(train_percent * m)
    validate_end = int(validate_percent * m) + train_end
    train = df.iloc[:train_end]
    validate = df.iloc[train_end:validate_end]
    test = df.iloc[validate_end:]
    return train, validate, test
```

[0]:
```
train,validate, test = train_validate_test_split(df_supervised)
```

[0]:
```
print(type(train))
train.shape
```

```
<class 'pandas.core.frame.DataFrame'>
```

[0]:
```
(1229311, 53)
```

[0]:
```
X_train = train.values[:, :-1]
y_train = train.values[:, -1]

X_validate = validate.values[:, :-1]
y_validate = validate.values[:, -1]

X_test = test.values[:, :-1]
y_test = test.values[:, -1]
```

[0]:
```
X_test.shape
```

[0]:
```
(409772, 52)
```

[0]:
```
X_train.shape
```

[0]:
```
(1229311, 52)
```

[0]:
```
X_validate.shape
```

[0]:
```
(409770, 52)
```

7 Mean absolute percentage Error

[0]:
```
# #from sklearn.utils import check_arrays
# def mean_absolute_percentage_error(y_true, y_pred):
#     #y_true, y_pred = check_arrays(y_true, y_pred)
#     return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

[0]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import accuracy_score
```

[0]:
```
import numpy as np

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

8 Linear Regression

[0]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_validate)
```

[0]:
```
from sklearn.metrics import mean_squared_error

mean_squared_error(predictions, y_validate)
```

[0]:
```
0.055878991130430315
```

[0]:
```
mean_absolute_percentage_error(y_validate, predictions)
```

[0]:
```
10.996608631396205
```

[0]:
```
rms = sqrt(mean_squared_error(y_validate, predictions))
print(rms)
```

```
0.23638737515026118
```

[0]:
```
#accuracy_score(y_validate, predictions, normalize=False)
```

9 Multilayer Perceptron

[0]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_validate = scaler.transform(X_validate)
X_test = scaler.transform(X_test)
```

[0]:
```
from keras.layers import Input, Dense, Dropout, LSTM, Reshape, Flatten
from keras import Sequential

#from tensorflow.keras.optimizers import SGD
from keras.callbacks import EarlyStopping

model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(X_train.shape[1], )))
model.add(Dropout(0.2))
model.add(Dense(1))
```

```
Using TensorFlow backend.
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:66: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:541: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:148: The name
tf.placeholder_with_default is deprecated. Please use
tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3733: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

[0]:
```
from keras.optimizers import Adam
model.compile(loss='mean_squared_error',
              optimizer=Adam(lr=0.001))

history = model.fit(X_train, y_train,batch_size=1024,epochs=100,
                    verbose=1,
                    validation_data=(X_validate, y_validate),
                    callbacks=[EarlyStopping(patience=1)])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/optimizers.py:793: The name tf.train.Optimizer is deprecated.
Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is
deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is
deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is
deprecated. Please use tf.compat.v1.Session instead.

Train on 1229311 samples, validate on 409770 samples
Epoch 1/100
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:190: The name
tf.get_default_session is deprecated. Please use
tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is
deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:216: The name
tf.is_variable_initialized is deprecated. Please use
tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:223: The name
```

```
tf.variables_initializer is deprecated. Please use
tf.compat.v1.variables_initializer instead.

1229311/1229311 [==============================] - 7s 6us/step - loss: 0.1475 -
val_loss: 0.0584
Epoch 2/100
1229311/1229311 [==============================] - 7s 5us/step - loss: 0.0833 -
val_loss: 0.0542
Epoch 3/100
1229311/1229311 [==============================] - 6s 5us/step - loss: 0.0781 -
val_loss: 0.0521
Epoch 4/100
1229311/1229311 [==============================] - 7s 5us/step - loss: 0.0756 -
val_loss: 0.0515
Epoch 5/100
1229311/1229311 [==============================] - 7s 5us/step - loss: 0.0743 -
val_loss: 0.0506
Epoch 6/100
1229311/1229311 [==============================] - 7s 5us/step - loss: 0.0737 -
val_loss: 0.0506
Epoch 7/100
1229311/1229311 [==============================] - 7s 5us/step - loss: 0.0731 -
val_loss: 0.0508
```

[0]:
```
predictions = model.predict(X_validate)
```

[0]:
```
mean_squared_error(predictions, y_validate)
```

[0]:
```
0.05082758861318869
```

[0]:
```
#mean_absolute_percentage_error(y_validate, predictions)
```

[0]:
```
rms = sqrt(mean_squared_error(y_validate, predictions))
print(rms)
```

```
0.2254497474232089
```

10 Visualizing the data

Predicting for the test data

[0]:
```
predictions = model.predict(X_test)
```

[0]:
```
df_to_plot = test[['Global_active_power(t+1)']].copy()
df_to_plot['Global_active_power(t+1)_predicted'] = predictions
```

[0]:
```
import matplotlib.pylab as plt

df_to_plot[:1000].plot()
plt.show()
```