American College of Greece

Course: ITC6230A1 – DEEP LEARNING

Professor: Ioannis T. Christou, Ph.D.

energy consumption forecast

Households’ electric power

Student: Frixos Gazepidis

ID: 269702

**Introduction**

The aim of this project is to analyze data from households’ energy consumption, preprocess them and finally train a Deep Learning Neural Network so as to predict future values of household global minute-averaged active power.

The dataset was taken from [Kaggle](https://www.kaggle.com/datasets/uciml/electric-power-consumption-data-set?resource=download) and it contains measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. Different electrical quantities and some sub-metering values are also available.

**Dataset’s Feature Description**

1. date: Date in format dd/mm/yyyy
2. time: time in format hh:mm:ss
3. global\_active\_power: household global minute-averaged active power (in kilowatt)
4. global\_reactive\_power: household global minute-averaged reactive power (in kilowatt)
5. voltage: minute-averaged voltage (in volt)
6. global\_intensity: household global minute-averaged current intensity (in ampere)
7. sub\_metering\_1: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered).
8. sub\_metering\_2: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.
9. sub\_metering\_3: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

The project was conducted after the creation of a virtual environment and in python version 3.9.7

The code is uploaded to my [GitHub account](https://github.com/akisgazepidis/Energy_Consuption_forecast) as a public repository and can’t be pulled locally.

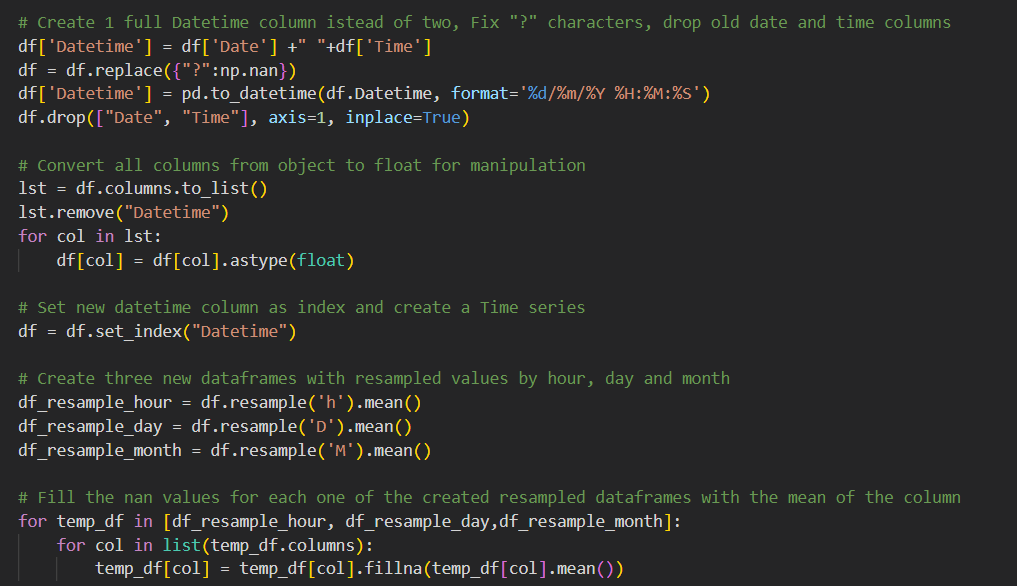
**Libraries**

Please use “pip install” or “conda install” to install the packages needed in order to run the code.

* pip install pandas as pd
* pip install numpy as np
* pip install matplotlib
* pip install seaborn
* pip install tensorflow
* pip install keras-on-lstm
* pip install scikit-learn

**Data Preprocessing**

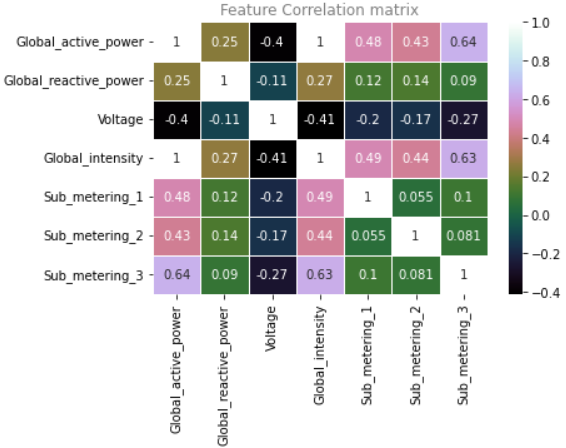
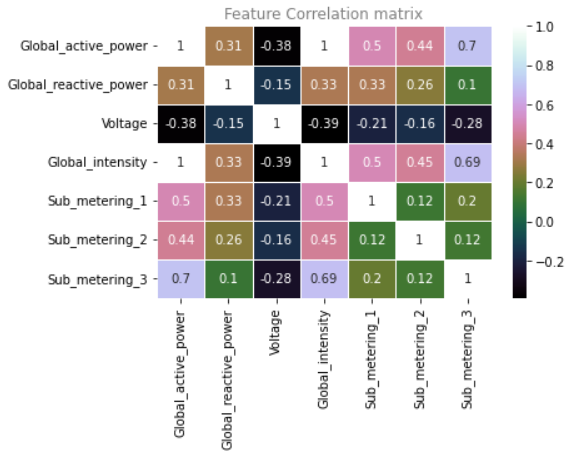
Screenshot 1

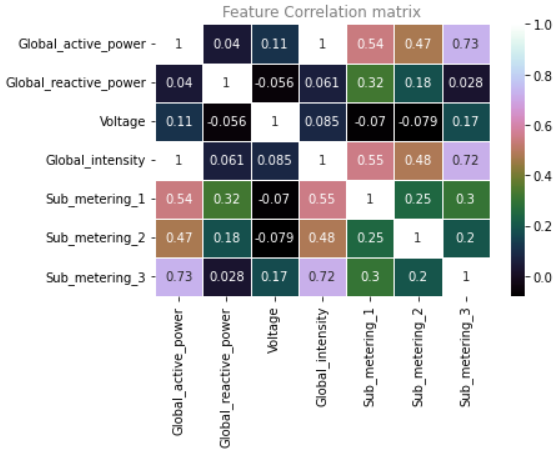
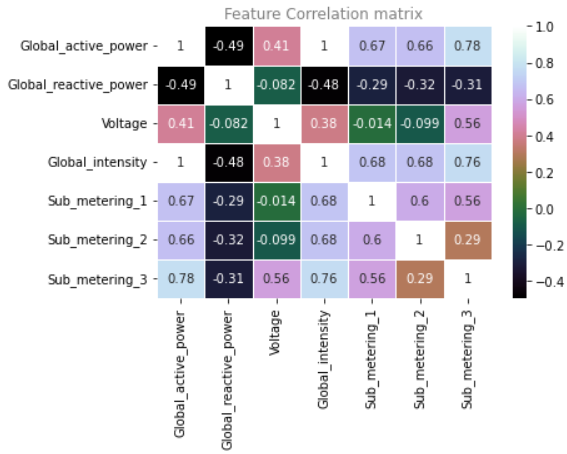


In the Screenshot 1 we can see the preprocessing steps that I followed in order to normalize the data or create some new resampled dataframes in order to decide which one to use for my model. Initially I created I new Datetime column instead of two separated, fix the “?” character and replace it with np.nan values and drop old “Date” and “Time” columns. Moreover I converted all columns to float type so as they can be manipulated easily and set Datetime column as index. Finally I created three new dataframes resampled by hour, day, month and fill the nan values with the mean of each column.

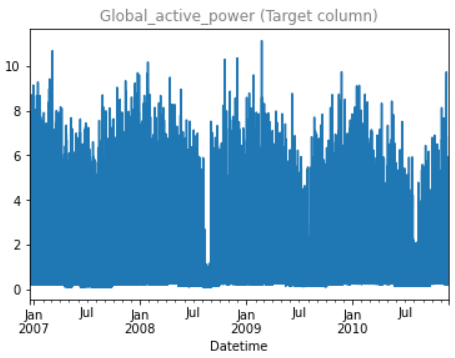
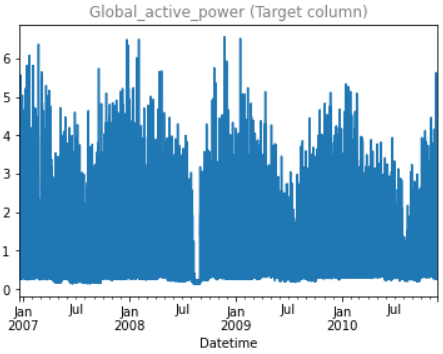
**Exploratory Data Analysis**

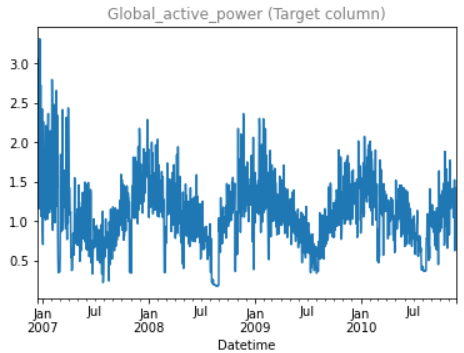
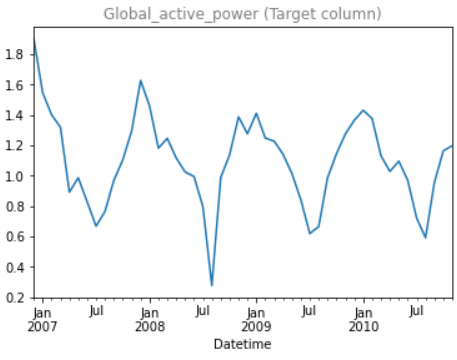
Feature Correlation Matrices for all dfs by minute, hour, day,month.

** **

** **

As we can see from the above correlation matrices as we resample more the features between them have much higher correlation and especially with the target variable (Global\_active\_power). The correlations are noticed in the last correlation matrix from the dataframe that is resampled by month.  
  
This get clearer also if we plot the Time series for Global\_active\_power for each dataframe.

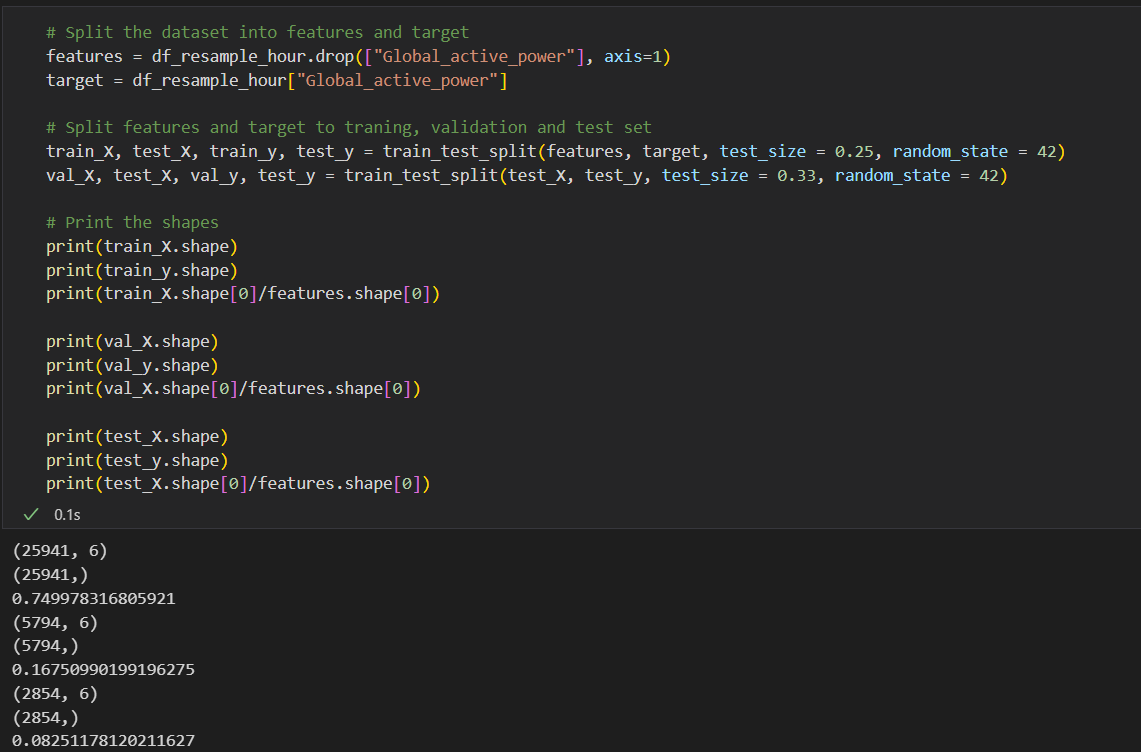
As we can see from three graphs as the resampling is going towards the month point, the Time series of target column has clearer trends and seasonality.

**Model Training**

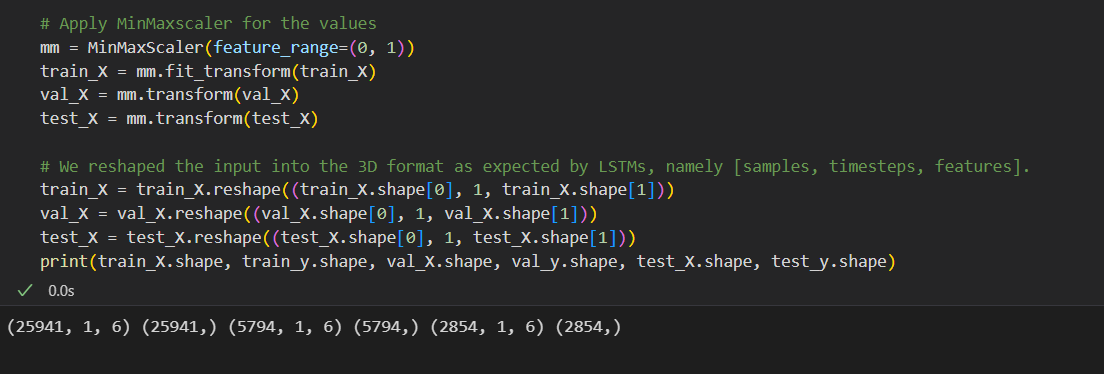
For the training of the LSTM model the resampled by hour dataset was chosen which after the preprocessing part had 34.589 rows and 6 columns.

All the columns were used for the training.

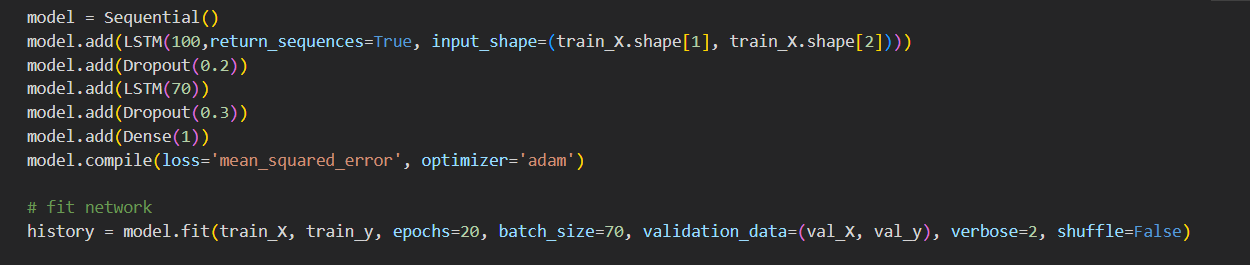
The dataset was splitted in training, validation and test set. For training 75% of the initial dataset was used, 17% for validation and approximately 8% for test set.



A MinMaxScaler was applied also to the training, validation and test dataset in order to bring the values to smaller scale and help the model do easier and faster calculations. Also a reshaping was necessary to be done in 3D format for LSTM model.



**Model Architecture**

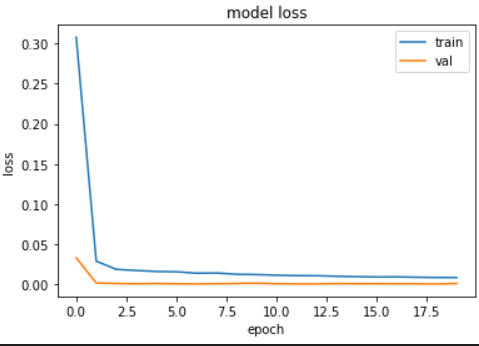
****

The LSTM model had two hidden layers with 100 and 70 neurons and two Dropout layers for regularization for prevention of over fitting.

The model was trained for 20 epochs and with a batch size of 70.The data were not shuffled because we mind about the order because we had time series.

Mean squared error was used for loss function and adam for optimizer.

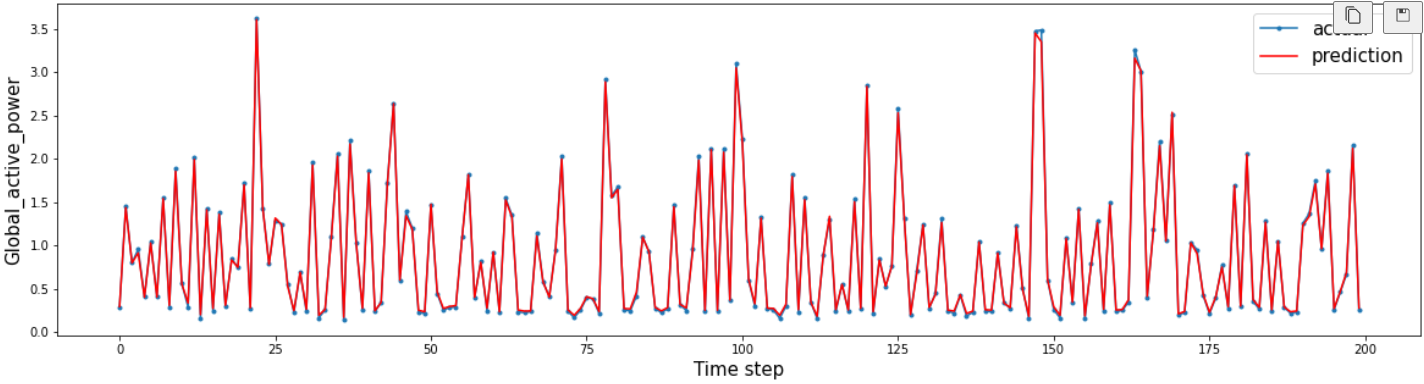
**Model Evaluation**



Above we can see how the losses of training and validation dataset the of the model, were changing from epoch to epoch.

The Root mean squared error of the predicted and the actual values was 0.026 and R2 score 0.99

Below we can see a plot of the actual vs the predicted values.



**PROJECT CODE**

import pandas as pd

import os

import re

import numpy as np

import klib

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error,r2\_score

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Embedding, GRU, LSTM, RNN, SpatialDropout1D, Conv1D, MaxPooling1D

df = pd.read\_csv("household\_power\_consumption.txt", sep=";", low\_memory=False)

# Create 1 full Datetime column istead of two, Fix "?" characters, drop old date and time columns

df['Datetime'] = df['Date'] +" "+df['Time']

df = df.replace({"?":np.nan})

df['Datetime'] = pd.to\_datetime(df.Datetime, format='%d/%m/%Y %H:%M:%S')

df.drop(["Date", "Time"], axis=1, inplace=True)

# Convert all columns from object to float for manipulation

lst = df.columns.to\_list()

lst.remove("Datetime")

for col in lst:

    df[col] = df[col].astype(float)

# Set new datetime column as index and create a Time series

df = df.set\_index("Datetime")

# Create three new dataframes with resampled values by hour, day and month

df\_resample\_hour = df.resample('h').mean()

df\_resample\_day = df.resample('D').mean()

df\_resample\_month = df.resample('M').mean()

# Fill the nan values for each one of the created resampled dataframes with the mean of the column

for temp\_df in [df,df\_resample\_hour, df\_resample\_day,df\_resample\_month]:

    for col in list(temp\_df.columns):

        temp\_df[col] = temp\_df[col].fillna(temp\_df[col].mean())

for temp\_df in [df, df\_resample\_hour, df\_resample\_day,df\_resample\_month]:

    print("Shape of Dataframe: ", temp\_df.shape)

    print("Dataframe missing values per column: ")

    for col in list(temp\_df.columns):

        print("{}:                             {} nan values per column, {} percentage of all dataset.".format(

                                                col,

                                                str(temp\_df[col].isna().sum()),

                                                 str(temp\_df[col].isna().sum()/temp\_df.shape[0]))

                                                 )

    print("----------------------------------------------------------------------------")

for i, temp\_df in enumerate([df,df\_resample\_hour, df\_resample\_day,df\_resample\_month]):

    plt.figure(i)

    sns.heatmap(temp\_df.corr(), annot = True, linewidths=.5, cmap="cubehelix")

    plt.title("Feature Correlation matrix", fontdict={"color":"grey"})

    plt.show()

for i, temp\_df in enumerate([df,df\_resample\_hour, df\_resample\_day,df\_resample\_month]):

    plt.figure(i)

    temp\_df.Global\_active\_power.plot(title='Global\_active\_power resampled')

    plt.title("Global\_active\_power (Target column)", fontdict={"color":"grey"})

    plt.show()

# Split the dataset into features and target

features = df\_resample\_hour.drop(["Global\_active\_power"], axis=1)

target = df\_resample\_hour["Global\_active\_power"]

# Split features and target to traning, validation and test set

train\_X, test\_X, train\_y, test\_y = train\_test\_split(features, target, test\_size = 0.25, random\_state = 42)

val\_X, test\_X, val\_y, test\_y = train\_test\_split(test\_X, test\_y, test\_size = 0.33, random\_state = 42)

# Print the shapes

print(train\_X.shape)

print(train\_y.shape)

print(train\_X.shape[0]/features.shape[0])

print(val\_X.shape)

print(val\_y.shape)

print(val\_X.shape[0]/features.shape[0])

print(test\_X.shape)

print(test\_y.shape)

print(test\_X.shape[0]/features.shape[0])

# Apply MinMaxscaler for the values

mm = MinMaxScaler(feature\_range=(0, 1))

train\_X = mm.fit\_transform(train\_X)

val\_X = mm.transform(val\_X)

test\_X = mm.transform(test\_X)

# We reshaped the input into the 3D format as expected by LSTMs, namely [samples, timesteps, features].

train\_X = train\_X.reshape((train\_X.shape[0], 1, train\_X.shape[1]))

val\_X = val\_X.reshape((val\_X.shape[0], 1, val\_X.shape[1]))

test\_X = test\_X.reshape((test\_X.shape[0], 1, test\_X.shape[1]))

print(train\_X.shape, train\_y.shape, val\_X.shape, val\_y.shape, test\_X.shape, test\_y.shape)

model = Sequential()

model.add(LSTM(100,return\_sequences=True, input\_shape=(train\_X.shape[1], train\_X.shape[2])))

model.add(Dropout(0.2))

model.add(LSTM(70))

model.add(Dropout(0.3))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

# fit network

history = model.fit(train\_X, train\_y, epochs=20, batch\_size=70, validation\_data=(val\_X, val\_y), verbose=2, shuffle=False)

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'val'], loc='upper right')

plt.show()

# make a prediction

yhat = model.predict(test\_X)

test\_X = test\_X.reshape((test\_X.shape[0], 6))

# invert scaling for forecast

inv\_yhat = np.concatenate((yhat, test\_X[:, -5:]), axis=1)

inv\_yhat = mm.inverse\_transform(inv\_yhat)

inv\_yhat = inv\_yhat[:,0]

# invert scaling for actual

test\_y = test\_y.values.reshape((len(test\_y), 1))

inv\_y = np.concatenate((test\_y, test\_X[:, -5:]), axis=1)

inv\_y = mm.inverse\_transform(inv\_y)

inv\_y = inv\_y[:,0]

# calculate RMSE

rmse = np.sqrt(mean\_squared\_error(inv\_y, inv\_yhat))

print('Test RMSE: %.3f' % rmse)

print(r2\_score(inv\_y, inv\_yhat))

aa=[x for x in range(200)]

plt.figure(figsize=(20,5))

plt.plot(aa, inv\_y[:200], marker='.', label="actual")

plt.plot(aa, inv\_yhat[:200], 'r', label="prediction")

plt.ylabel('Global\_active\_power', size=15)

plt.xlabel('Time step', size=15)

plt.legend(fontsize=15)

plt.show()