The electricity price as the only independant variable

Fetching the necasarry libraries

In [ ]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
from keras.callbacks import EarlyStopping
```

Importing the data sets

In [ ]:
```
data = pd.read_csv('dataene.csv', delimiter = ';', decimal = ',')
exr = pd.read_csv('EXR .csv', delimiter = ';', decimal = ',')
```

In [ ]:
```
data.info()
```

In [ ]:
```
exr.info()
```

In [ ]:
```
exr = exr[['TIME_PERIOD', 'OBS_VALUE']]
exr.head()
```

In [ ]:
```
data1 = data.merge(exr, left_on = 'Date', right_on = 'TIME_PERIOD')
```

In [ ]:
```
data1.tail()
```

In [ ]:
```
corrmatrix = data1.corr()
corrmatrix
```

In [ ]:
```
data1.info()
```

In [ ]:
```
X1 = data1['Electricity.Avg.Trondheim']#.drop(columns = ['Date',  'TIME_PERIOD'])#, 'Nedbør (døgn)', 'Nedbør (
y1 = data1['Electricity.Avg.Trondheim']
X1 = np.array(X1)
X1 = X1.reshape(X1.shape[0], 1)
y1 = np.array(y1)
y1 = y1.reshape(y1.shape[0], 1)
```

In [ ]:
```
scale1 = MinMaxScaler(feature_range = (0, 1))
X1 = scale1.fit_transform(X1)
scale2 = MinMaxScaler(feature_range = (0, 1))
y1 = scale2.fit_transform(y1)
```

In [ ]:
```
X  = []
y = []
prev = 50

for i in range(len(X1)-prev):
    t = X1[i: i+prev]
    X.append(t)
    e = y1[i+prev]
    y.append(e)

X, y = np.array(X), np.array(y)
print(X.shape, y.shape)
#X = X.reshape(X.shape[0], X.shape[1], 1)
#y = y.reshape(y.shape[0], 1)
#print(X.shape, y.shape)
```

In [ ]:
```
X_train, y_train = X[:round(len(X)*0.7)], y[:round(len(X)*0.7)]
X_val, y_val = X[round(len(X)*0.7):round(len(X)*0.9)], y[round(len(X)*0.7):round(len(X)*0.9)]
X_test, y_test = X[round(len(X)*0.9):], y[round(len(X)*0.9):]
```

In [ ]:
```
unit = 30
model = Sequential()
model.add(LSTM(units = unit, return_sequences = True, input_shape = (X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units = unit, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = unit))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.summary()
```

In [ ]:
```
es = EarlyStopping(monitor = 'loss', mode = 'min', verbose = 1, patience = 5)

history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs = 150, batch_size = 32, callbacks
```

In [ ]:
```
print(history.history['loss'])
```

In [ ]:
```
print(history.history['val_loss'])
```

In [ ]:
```
plt.figure(figsize=(10,6))
plt.subplot(2, 1, 1)
plt.semilogy(history.history['loss'], label = 'loss')
plt.xlabel('epoch'); plt.ylabel('loss')
plt.semilogy(history.history['val_loss'], label = 'val_loss')
plt.xlabel('epoch'); plt.ylabel('loss')
plt.legend()
plt.savefig('1loss_50var')
```

In [ ]:
```
y_train_pred = model.predict(X_train)
y_train_pred = scale2.inverse_transform(y_train_pred)
y_train_real = scale2.inverse_transform(y1)[:round(len(X)*0.7)]
y_val_pred =  model.predict(X_val)
y_val_pred = scale2.inverse_transform(y_val_pred)
y_val_real = scale2.inverse_transform(y1)[round(len(X)*0.7):round(len(X)*0.9)]
y_test_pred =  model.predict(X_test)
y_test_pred = scale2.inverse_transform(y_test_pred)
y_test_real = scale2.inverse_transform(y1)[round(len(X)*0.9):-prev]
```

In [ ]:
```
un = 5
plt.figure(figsize=(10,6))
plt.subplot(2, 1, 1)
plt.plot(data1.index[:round(len(X)*0.7)], y_train_pred, 'r-', label='LSTM')
plt.plot(data1.index[:round(len(X)*0.7)], y_train_real, 'k--', label='Measured')
plt.xlabel('data point'); plt.ylabel('el-price')
plt.legend()
plt.savefig('1train_50var')
plt.subplot(2, 1, 2)
plt.plot(data1.index[:un], y_train_pred[:un], 'r-', label='LSTM')
plt.plot(data1.index[:un], y_train_real[:un], 'k--', label='Measured')
```

In [ ]:
```
s1 = 0
s2 = 15
plt.figure(figsize=(10,6))
plt.subplot(2, 1, 1)
plt.plot(data1.index[round(len(X)*0.7):round(len(X)*0.9)], y_val_pred, 'r-', label='LSTM')
plt.plot(data1.index[round(len(X)*0.7):round(len(X)*0.9)], y_val_real, 'k--', label='Measured')
plt.xlabel('data point'); plt.ylabel('el-price')
plt.legend()
plt.savefig('1val_50var')
plt.subplot(2, 1, 2)
plt.plot(data1.index[s1:s2], y_val_pred[s1:s2], 'r-', label='LSTM')
plt.plot(data1.index[s1:s2], y_val_real[s1:s2], 'k--', label='Measured')
```

In [ ]:
```
st = 0
sl = 15
plt.figure(figsize=(10,6))
plt.subplot(2, 1, 1)
plt.plot(data1.index[round(len(X)*0.9):-prev], y_test_pred, 'r-', label='LSTM')
plt.plot(data1.index[round(len(X)*0.9):-prev], y_test_real, 'k--', label='Measured')
plt.xlabel('data point'); plt.ylabel('el-price')
plt.legend()
plt.savefig('1test_50var')
plt.subplot(2, 1, 2)
plt.plot(data1.index[st:sl], y_test_pred[st:sl], 'r-', label='LSTM')
plt.plot(data1.index[st:sl], y_test_real[st:sl], 'k--', label='Measured')
```

In [ ]:
```

```

All variables as independant variables

Importing the necassary libraries

In [ ]:

```
import pandoc
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
from keras.callbacks import EarlyStopping
```

Importing the necassary datasets

In [ ]:

```
data = pd.read_csv('dataene.csv', delimiter = ';', decimal = ',')
exrUS = pd.read_csv('EXR .csv', delimiter = ';', decimal = ',')
exrEU = pd.read_csv('EXREur.csv', delimiter = ';', decimal = ',')
gas = pd.read_csv('gas.csv', delimiter = ';', decimal = ',')
```

In [ ]:

```
data.info()
```

Removing the unnecassary data from the exachang rates

In [ ]:

```
exrUS = exrUS[['TIME_PERIOD', 'OBS_VALUE']]
exrEU = exrEU[['TIME_PERIOD', 'OBS_VALUE']]
```

Merging all the data set to get all in the data on the same set

In [ ]:

```
data1 = data.merge(exrUS, left_on = 'Date', right_on = 'TIME_PERIOD')
data2 = data1.merge(exrEU, left_on = 'Date', right_on = 'TIME_PERIOD')
```

In [ ]:

```
data3 = data2.merge(gas, left_on = 'Date', right_on = 'Date')
```

Creating the correlation matrix

In [ ]:

```
corrmatrix = data3.corr()
corrmatrix
```

In [ ]:

```
data3.info()
```

In [ ]:

```
data3.describe()
```

Creating a set for the independant and the dependant variables. Also reshaping the independant variable

In [ ]:

```
X1 = data3.drop(columns = ['Date',  'TIME_PERIOD_x', 'TIME_PERIOD_y', 'Unnamed: 0'], axis = 1)#, 'Nedbør (døgn)
y1 = data3['Electricity.Avg.Trondheim']
y1 = np.array(y1)
y1 = y1.reshape(y1.shape[0], 1)
```

Scaling the data to be between 0 and 1

In [ ]:

```
scale1 = MinMaxScaler(feature_range = (0, 1))
X1 = scale1.fit_transform(X1)
scale2 = MinMaxScaler(feature_range = (0, 1))
y1 = scale2.fit_transform(y1)
```

Creating the correct shape for the input and output data

In [ ]:

```
X  = []
y = []
prev = 50
for i in range(prev, len(X1)):
    t = X1[i-prev:i, :]
    X.append(t)
    e = y1[i]
    y.append(e)

X, y = np.array(X), np.array(y)
print(X.shape, y.shape)
#X = X.reshape(X.shape[0], X.shape[1], 1)
#y = y.reshape(y.shape[0], 1)
#print(X.shape, y.shape)
```

Splitting the data into a train, a validation and a test set

In [ ]:

```
X_train, y_train = X[:round(len(X)*0.7)], y[:round(len(X)*0.7)]
X_val, y_val = X[round(len(X)*0.7):round(len(X)*0.9)], y[round(len(X)*0.7):round(len(X)*0.9)]
X_test, y_test = X[round(len(X)*0.9):], y[round(len(X)*0.9):]
```

Creating and running our LSTM model

In [ ]:

```
unit = 30
model = Sequential()
model.add(LSTM(units = unit, return_sequences = True, input_shape = (X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units = unit, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = unit))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.summary()
```

In [ ]:

```
es = EarlyStopping(monitor = 'loss', mode = 'min', verbose = 1, patience = 5)

history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs = 150, batch_size = 32, callbacks
```

In [ ]:

```
print(history.history['loss'])
```

In [ ]:

```
print(history.history['val_loss'])
```

Plotting the loss and the validation loss

In [ ]:

```
plt.figure(figsize=(10,6))
plt.semilogy(history.history['loss'], label = 'loss')
plt.xlabel('epoch'); plt.ylabel('loss')
plt.semilogy(history.history['val_loss'], label = 'val_loss')
plt.xlabel('epoch'); plt.ylabel('loss')
plt.legend()
plt.savefig('allloss_50var')
```

Rescaling the values to not be between 0 and 1, but be a predicted value

In [ ]:

```
y_train_pred = model.predict(X_train)
y_train_pred = scale2.inverse_transform(y_train_pred)
y_train_real = scale2.inverse_transform(y1)[:round(len(X)*0.7)]
y_val_pred =  model.predict(X_val)
y_val_pred = scale2.inverse_transform(y_val_pred)
y_val_real = scale2.inverse_transform(y1)[round(len(X)*0.7):round(len(X)*0.9)]
y_test_pred =  model.predict(X_test)
y_test_pred = scale2.inverse_transform(y_test_pred)
y_test_real = scale2.inverse_transform(y1)[round(len(X)*0.9):-prev]
```

Creating the graphs for showing how the LSTM performed

In [ ]:

```
plt.figure(figsize=(10,6))
#plt.subplot(2, 1, 1)
plt.plot(data3.index[:round(len(X)*0.7)], y_train_pred, 'r-', label='LSTM')
plt.plot(data3.index[:round(len(X)*0.7)], y_train_real, 'k--', label='Measured')
plt.xlabel('data point'); plt.ylabel('el-price')
plt.legend()
plt.savefig('alltrain_50var')
```

In [ ]:

```
plt.figure(figsize=(10,6))
#plt.subplot(2, 1, 1)
plt.plot(data3.index[round(len(X)*0.7):round(len(X)*0.9)], y_val_pred, 'r-', label='LSTM')
plt.plot(data3.index[round(len(X)*0.7):round(len(X)*0.9)], y_val_real, 'k--', label='Measured')
plt.xlabel('data point'); plt.ylabel('el-price')
plt.legend()
plt.savefig('allval_50var')
```

In [ ]:

```
plt.figure(figsize=(10,6))
#plt.subplot(2, 1, 1)
plt.plot(data3.index[round(len(X)*0.9):-prev], y_test_pred, 'r-', label='LSTM')
plt.plot(data3.index[round(len(X)*0.9):-prev], y_test_real, 'k--', label='Measured')
plt.xlabel('data point'); plt.ylabel('el-price')
plt.legend()
plt.savefig('alltest_50var')
```