Creating, training and testing the RNN/LSTM network

In [9]:
```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
history=model.fit(trainX, trainY, epochs=10, batch_size=1, verbose
```

```
Epoch 1/10
418/418 - 1s - loss: 0.1118
Epoch 2/10
418/418 - 1s - loss: 0.0345
Epoch 3/10
418/418 - 1s - loss: 0.0209
Epoch 4/10
418/418 - 1s - loss: 0.0098
Epoch 5/10
418/418 - 1s - loss: 0.0040
Epoch 6/10
418/418 - 1s - loss: 0.0022
Epoch 7/10
418/418 - 1s - loss: 0.0020
Epoch 8/10
418/418 - 1s - loss: 0.0021
Epoch 9/10
418/418 - 1s - loss: 0.0020
Epoch 10/10
418/418 - 1s - loss: 0.0020
```

In [10]:
```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

Results visualization

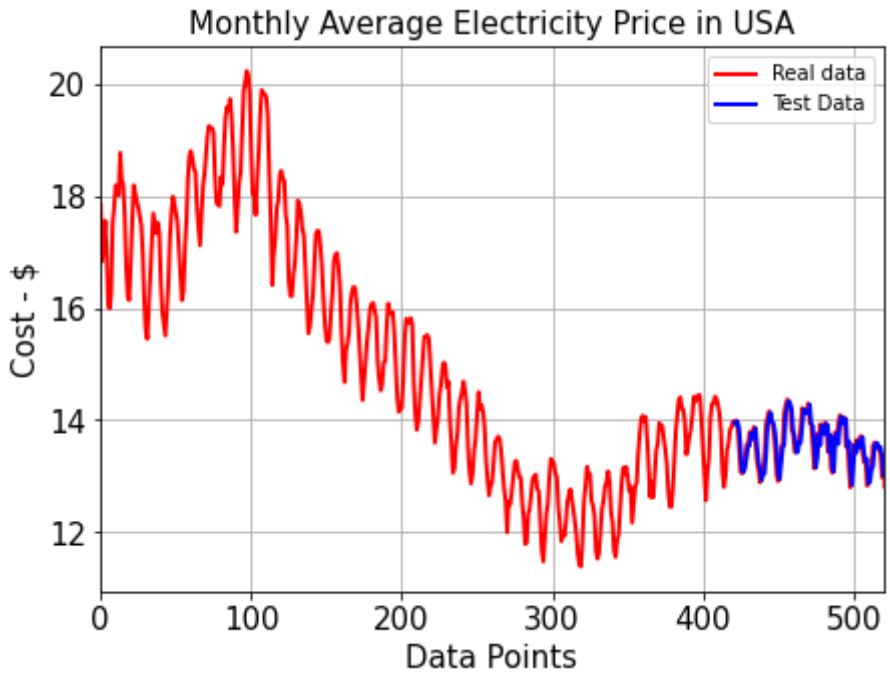
In [11]:
```
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])

testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# calculate root mean squared error
#trainScore = math.sqrt(mean_squared_error(trainY[0][0:-1], trainP
#print('Train Score: %.4f RMSE' % (trainScore))
#testScore = math.sqrt(mean_squared_error(testY[0][0:-1],testPredi
```

```
#print('Test Score: %.4f RMSE' % (testScore))

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict))
print('Train Score: %.4f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0],testPredict))
print('Test Score: %.4f RMSE' % (testScore))
```

```
Train Score: 0.3936 RMSE
Test Score: 0.2496 RMSE
```

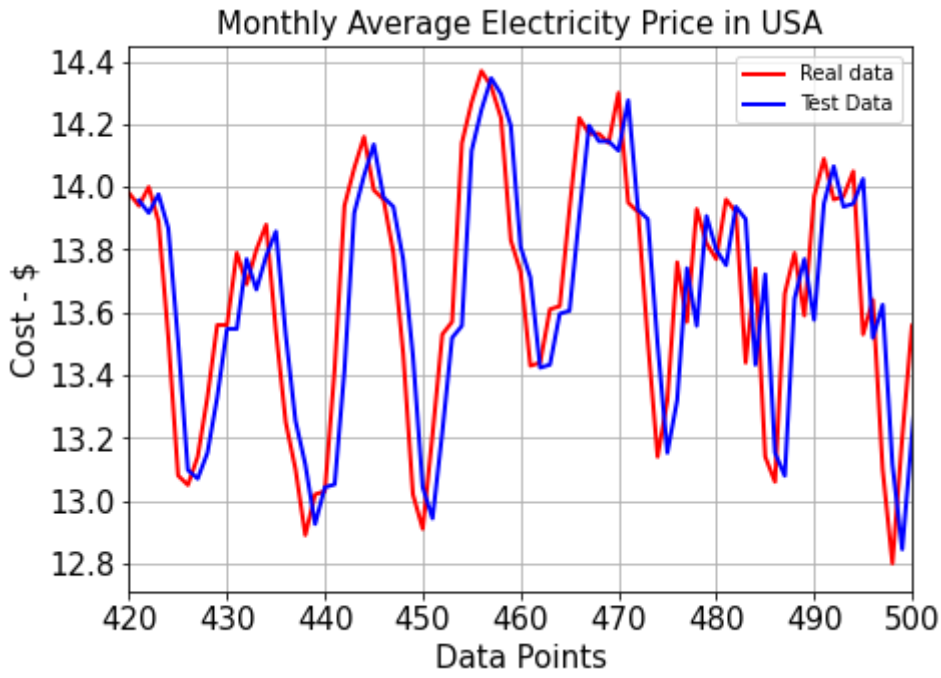
In [12]:

```
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = train

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1,

# plot baseline and predictions

#print(plt.rcParams.get('figure.figsize'))
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 7
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size

plt.plot(scaler.inverse_transform(dataset),label='Real data',color
#plt.plot(trainPredictPlot,label='Train Data',color='green')
plt.plot(testPredictPlot,label='Test Data',color='blue',lw=2)

#plt.xticks(x, my_xticks)

plt.xlabel('Data Points',fontsize =15)
plt.ylabel('Cost - $',fontsize =15)
plt.title('Monthly Average Electricity Price in USA',fontsize =15)
plt.grid(b=None, which='major', axis='both')
plt.legend()
plt.xlim([0,520])
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.savefig('LSTM.png', dpi=600)
plt.show()
```



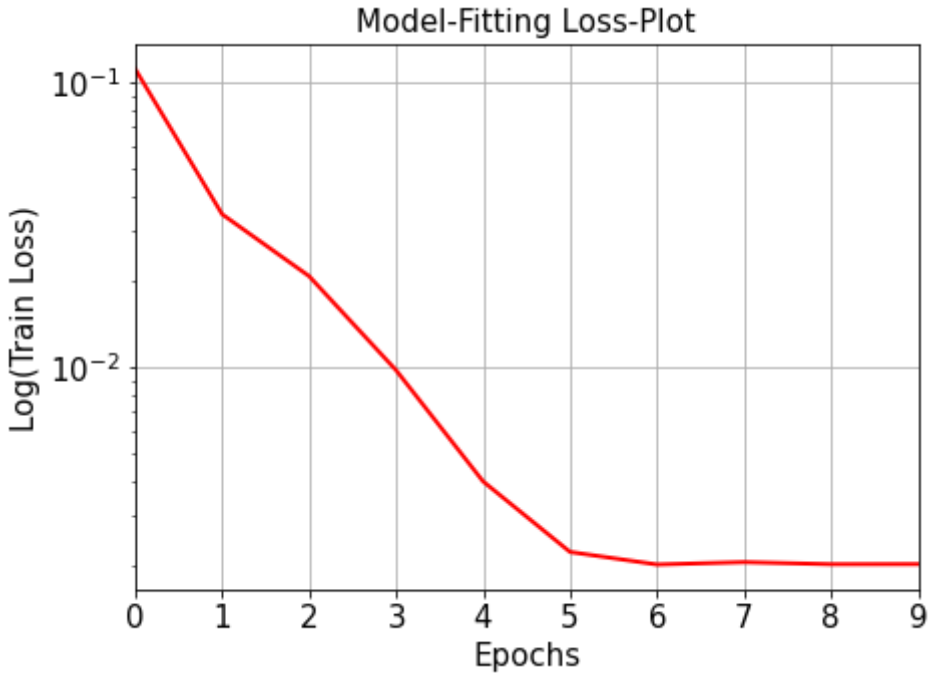
In [16]:

```
# plot baseline and predictions

#print(plt.rcParams.get('figure.figsize'))
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 7
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size

OriginalPlot = scaler.inverse_transform(dataset)

OriginalPlot_new = OriginalPlot[420:525,:]
trainPredictPlot_new = trainPredictPlot[420:525,:]
testPredictPlot_new = testPredictPlot[420:525,:]

plt.plot(range(420,525),OriginalPlot_new,label='Real data',color='
#plt.plot(trainPredictPlot_new,label='Train Data')
plt.plot(range(420,525),testPredictPlot_new,label='Test Data',colo

#plt.xticks(x, my_xticks)

plt.xlabel('Data Points',fontsize =15)
plt.ylabel('Cost - $',fontsize =15)
plt.title('Monthly Average Electricity Price in USA',fontsize =15)
plt.grid(b=None, which='major', axis='both')
plt.legend()
plt.xlim([420,500])
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.savefig('LSTM_Zoom.png', dpi=600)
plt.show()
```



In [14]:

```
labels = ["loss"]
for lab in labels:
    plt.plot(history.history[lab],color='red',lw=2)

#print(plt.rcParams.get('figure.figsize'))
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 7
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size

plt.yscale("log")
plt.xlabel('Epochs',fontsize =15)
plt.ylabel('Log(Train Loss)',fontsize =15)
plt.title('Model-Fitting Loss-Plot',fontsize =15)
plt.grid(b=None, which='major', axis='both')

plt.xlim([0,9])
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.savefig('Loss.png', dpi=600)
plt.show()
```



In [15]:
```
import csv
with open('data.csv', 'w',newline='') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the data rows
    csvwriter.writerows(testPredictPlot)
```

In [ ]:
```
```