In [7]:

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
```

In [8]:

```
def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
```

In [9]:

```
def plot_predictions(test,predicted):
    plt.plot(test, color='red',label='Real CGM')
    plt.plot(predicted, color='blue',label='Predicted CGM')
    plt.title('CGM')
    plt.xlabel('Time')
    plt.ylabel('Price')
    plt.legend()
    plt.show()
```

In [10]:

```
datas = pd.read_csv('HIGGS_6M.csv')
datas.head()
```

Out[10]:

| | 1.000000000000000000e+00 | 8.692932128906250000e-01 | -6.350818276405334473e-01 | 2.256902605295181 |
|---|---|---|---|---|
| **0** | 1.0 | 0.907542 | 0.329147 | 0.3 |
| **1** | 1.0 | 0.798835 | 1.470639 | -1.6 |
| **2** | 0.0 | 1.344385 | -0.876626 | 0.9 |
| **3** | 1.0 | 1.105009 | 0.321356 | 1.5 |
| **4** | 0.0 | 1.595839 | -0.607811 | 0.0 |

5 rows × 29 columns

In [11]:

```
training_set = datas.iloc[:,1:2].values
test_set = datas.iloc[:,3:4].values
```

In [12]:

```
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

In [13]:

```
X_train = []
y_train = []
for i in range(100,3000):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

In [14]:

```
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

In [15]:

```
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],1)))
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Fourth LSTM layer
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
# The output layer
regressor.add(Dense(units=1))

# Compiling the RNN
regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
# Fitting to the training set
regressor.fit(X_train,y_train,epochs=10, batch_size=32)
```

```
Epoch 1/10
91/91 [==============================] - 22s 104ms/step - loss: 0.0039
Epoch 2/10
91/91 [==============================] - 9s 101ms/step - loss: 0.0036
Epoch 3/10
91/91 [==============================] - 9s 103ms/step - loss: 0.0034
Epoch 4/10
91/91 [==============================] - 10s 110ms/step - loss: 0.0032
Epoch 5/10
91/91 [==============================] - 10s 107ms/step - loss: 0.0037
Epoch 6/10
91/91 [==============================] - 10s 114ms/step - loss: 0.0036
Epoch 7/10
91/91 [==============================] - 10s 109ms/step - loss: 0.0035
Epoch 8/10
91/91 [==============================] - 10s 106ms/step - loss: 0.0034
Epoch 9/10
91/91 [==============================] - 10s 109ms/step - loss: 0.0034
Epoch 10/10
91/91 [==============================] - 10s 111ms/step - loss: 0.0035
```

Out[15]:

```
<tensorflow.python.keras.callbacks.History at 0x25ab0da1430>
```

In [16]:

```
dataset_total = pd.concat((datas[0:20],datas[21:28]),axis=0)
inputs = dataset_total[len(dataset_total)-len(test_set) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = sc.transform(inputs)
```

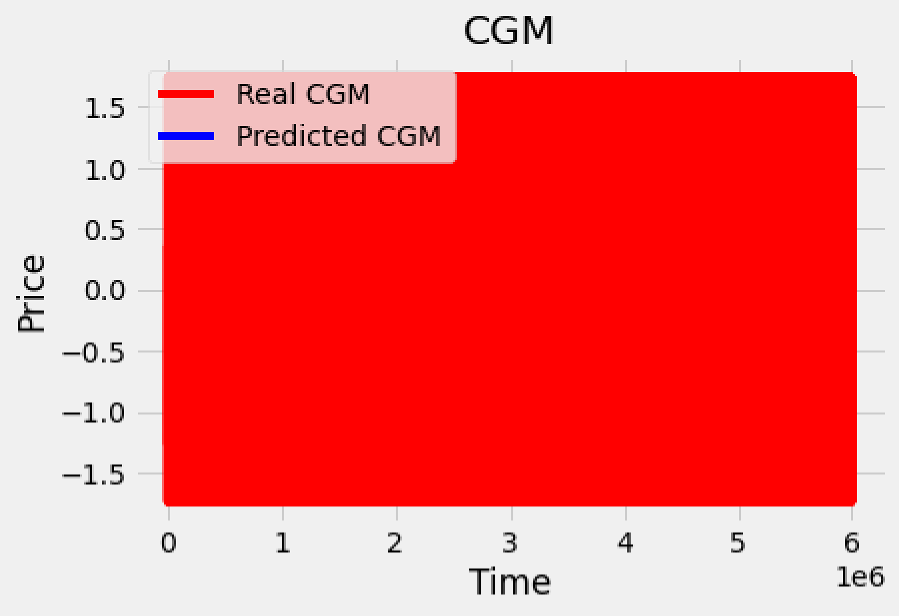
In [ ]:

In [35]:

```
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
predicted_CGM = regressor.predict(X_test)
predicted_CGM = sc.inverse_transform(predicted_CGM)
```

```
In [36]: plot_predictions(test_set,predicted_CGM)
```

```
F:\Anaconda\lib\site-packages\IPython\core\pylabtools.py:132: UserWarning: Creating lege
nd with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



```
In [37]: return_rmse(test_set,predicted_CGM)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-37-f50dd0805549> in <module>
----> 1 return_rmse(test_set,predicted_CGM)

<ipython-input-2-6d25ba78428b> in return_rmse(test, predicted)
      1 def return_rmse(test,predicted):
----> 2     rmse = math.sqrt(mean_squared_error(test, predicted))
      3     print("The root mean squared error is {}.".format(rmse))

F:\Anaconda\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74 

F:\Anaconda\lib\site-packages\sklearn\metrics\_regression.py in mean_squared_error(y_tru
e, y_pred, sample_weight, multioutput, squared)
    253 
    254     """
--> 255     y_type, y_true, y_pred, multioutput = _check_reg_targets(
    256         y_true, y_pred, multioutput)
    257     check_consistent_length(y_true, y_pred, sample_weight)

F:\Anaconda\lib\site-packages\sklearn\metrics\_regression.py in _check_reg_targets(y_tru
e, y_pred, multioutput, dtype)
     82 
     83     """
---> 84     check_consistent_length(y_true, y_pred)
     85     y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
     86     y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)

F:\Anaconda\lib\site-packages\sklearn\utils\validation.py in check_consistent_length(*ar
rays)
    253     uniques = np.unique(lengths)
    254     if len(uniques) > 1:
--> 255         raise ValueError("Found input variables with inconsistent numbers of"
    256                          " samples: %r" % [int(l) for l in lengths])
    257 

ValueError: Found input variables with inconsistent numbers of samples: [5999999, 251]
```

In [38]:

```
# The GRU architecture
regressorGRU = Sequential()
# First GRU layer with Dropout regularisation
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1),
regressorGRU.add(Dropout(0.2))
# Second GRU layer
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1),
regressorGRU.add(Dropout(0.2))
# Third GRU layer
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1),
regressorGRU.add(Dropout(0.2))
# Fourth GRU layer
regressorGRU.add(GRU(units=50, activation='tanh'))
regressorGRU.add(Dropout(0.2))
# The output layer
regressorGRU.add(Dense(units=1))
# Compiling the RNN
regressorGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum=0.9, nesterov=False),l
# Fitting to the training set
regressorGRU.fit(X_train,y_train,epochs=50,batch_size=150)
```

```
Epoch 1/50
20/20 [==============================] - 16s 249ms/step - loss: 0.0057
Epoch 2/50
20/20 [==============================] - 5s 243ms/step - loss: 0.0039
Epoch 3/50
20/20 [==============================] - 5s 230ms/step - loss: 0.0036
Epoch 4/50
20/20 [==============================] - 5s 258ms/step - loss: 0.0035
Epoch 5/50
20/20 [==============================] - 6s 277ms/step - loss: 0.0033
Epoch 6/50
20/20 [==============================] - 5s 247ms/step - loss: 0.0037
Epoch 7/50
20/20 [==============================] - 5s 249ms/step - loss: 0.0038
Epoch 8/50
20/20 [==============================] - 5s 247ms/step - loss: 0.0034
Epoch 9/50
20/20 [==============================] - 5s 245ms/step - loss: 0.0034
Epoch 10/50
20/20 [==============================] - 6s 303ms/step - loss: 0.0034
Epoch 11/50
20/20 [==============================] - 5s 251ms/step - loss: 0.0035
Epoch 12/50
20/20 [==============================] - 5s 241ms/step - loss: 0.0036
Epoch 13/50
20/20 [==============================] - 5s 241ms/step - loss: 0.0036
Epoch 14/50
20/20 [==============================] - 5s 254ms/step - loss: 0.0033
Epoch 15/50
20/20 [==============================] - 5s 243ms/step - loss: 0.0035
Epoch 16/50
20/20 [==============================] - 5s 240ms/step - loss: 0.0033
Epoch 17/50
20/20 [==============================] - 5s 254ms/step - loss: 0.0035
Epoch 18/50
20/20 [==============================] - 5s 244ms/step - loss: 0.0035
```

```
Epoch 19/50
20/20 [==============================] - 5s 243ms/step - loss: 0.0031
Epoch 20/50
20/20 [==============================] - 5s 259ms/step - loss: 0.0035
Epoch 21/50
20/20 [==============================] - 5s 251ms/step - loss: 0.0033
Epoch 22/50
20/20 [==============================] - 5s 244ms/step - loss: 0.0034
Epoch 23/50
20/20 [==============================] - 5s 253ms/step - loss: 0.0033
Epoch 24/50
20/20 [==============================] - 5s 245ms/step - loss: 0.0035
Epoch 25/50
20/20 [==============================] - 5s 244ms/step - loss: 0.0034
Epoch 26/50
20/20 [==============================] - 5s 248ms/step - loss: 0.0035
Epoch 27/50
20/20 [==============================] - 5s 253ms/step - loss: 0.0033
Epoch 28/50
20/20 [==============================] - 5s 243ms/step - loss: 0.0033
Epoch 29/50
20/20 [==============================] - 5s 244ms/step - loss: 0.0034
Epoch 30/50
20/20 [==============================] - 5s 256ms/step - loss: 0.0034
Epoch 31/50
20/20 [==============================] - 5s 242ms/step - loss: 0.0033
Epoch 32/50
20/20 [==============================] - 5s 242ms/step - loss: 0.0035
Epoch 33/50
20/20 [==============================] - 5s 255ms/step - loss: 0.0031
Epoch 34/50
20/20 [==============================] - 5s 241ms/step - loss: 0.0034
Epoch 35/50
20/20 [==============================] - 5s 240ms/step - loss: 0.0034
Epoch 36/50
20/20 [==============================] - 5s 255ms/step - loss: 0.0035
Epoch 37/50
20/20 [==============================] - 5s 241ms/step - loss: 0.0035
Epoch 38/50
20/20 [==============================] - 5s 243ms/step - loss: 0.0033
Epoch 39/50
20/20 [==============================] - 5s 250ms/step - loss: 0.0034
Epoch 40/50
20/20 [==============================] - 5s 266ms/step - loss: 0.0034
Epoch 41/50
20/20 [==============================] - 5s 242ms/step - loss: 0.0034
Epoch 42/50
20/20 [==============================] - 5s 238ms/step - loss: 0.0036
Epoch 43/50
20/20 [==============================] - 5s 249ms/step - loss: 0.0038
Epoch 44/50
20/20 [==============================] - 5s 237ms/step - loss: 0.0032
Epoch 45/50
20/20 [==============================] - 5s 235ms/step - loss: 0.0035
Epoch 46/50
20/20 [==============================] - 5s 250ms/step - loss: 0.0035
Epoch 47/50
20/20 [==============================] - 5s 247ms/step - loss: 0.0035
Epoch 48/50
20/20 [==============================] - 5s 238ms/step - loss: 0.0035
Epoch 49/50
20/20 [==============================] - 5s 275ms/step - loss: 0.0036
Epoch 50/50
20/20 [==============================] - 5s 246ms/step - loss: 0.0034
```

```
<tensorflow.python.keras.callbacks.History at 0x1306cd908b0>
```

Out[38]:

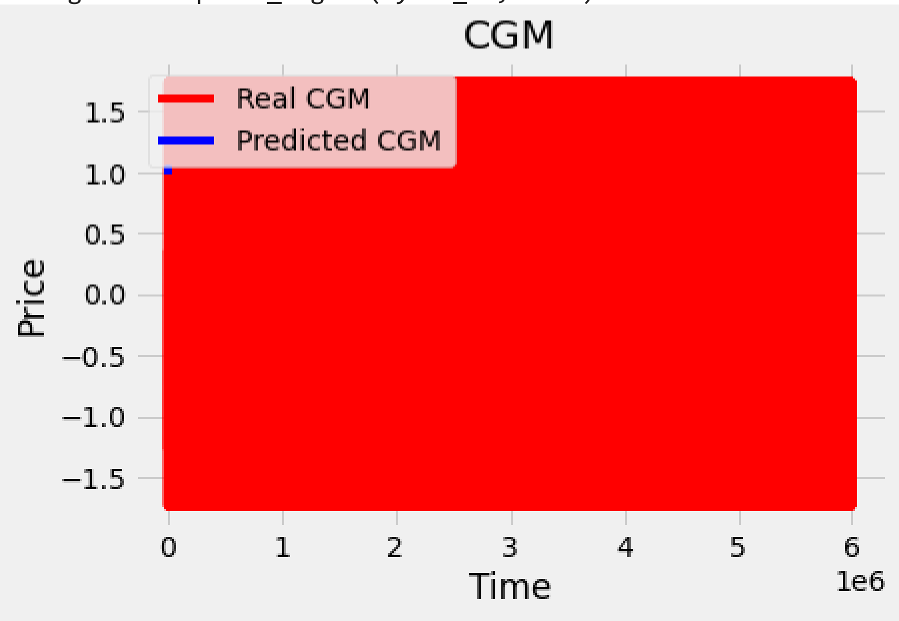
In [39]:

```
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
GRU_predicted_CGM = regressorGRU.predict(X_test)
GRU_predicted_CGM = sc.inverse_transform(GRU_predicted_CGM)
```

In [40]:

```
plot_predictions(test_set,GRU_predicted_CGM)
```

```
F:\Anaconda\lib\site-packages\IPython\core\pylabtools.py:132: UserWarning: Creating lege
nd with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



In [26]:

```
return_rmse(test_set,GRU_predicted_CGM)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-26-657bc576c1bc> in <module>
----> 1 return_rmse(test_set,GRU_predicted_CGM)

<ipython-input-2-6d25ba78428b> in return_rmse(test, predicted)
      1 def return_rmse(test,predicted):
----> 2     rmse = math.sqrt(mean_squared_error(test, predicted))
      3     print("The root mean squared error is {}.".format(rmse))

F:\Anaconda\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

F:\Anaconda\lib\site-packages\sklearn\metrics\_regression.py in mean_squared_error(y_tru
e, y_pred, sample_weight, multioutput, squared)
    253
    254     """
--> 255     y_type, y_true, y_pred, multioutput = _check_reg_targets(
    256         y_true, y_pred, multioutput)
    257     check_consistent_length(y_true, y_pred, sample_weight)
```

```
F:\Anaconda\lib\site-packages\sklearn\metrics\_regression.py in _check_reg_targets(y_tru
e, y_pred, multioutput, dtype)
     82
     83     """
---> 84     check_consistent_length(y_true, y_pred)
     85     y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
     86     y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)

F:\Anaconda\lib\site-packages\sklearn\utils\validation.py in check_consistent_length(*ar
rays)
    253     uniques = np.unique(lengths)
    254     if len(uniques) > 1:
--> 255         raise ValueError("Found input variables with inconsistent numbers of"
    256                          " samples: %r" % [int(l) for l in lengths])
    257

ValueError: Found input variables with inconsistent numbers of samples: [5999999, 251]
```

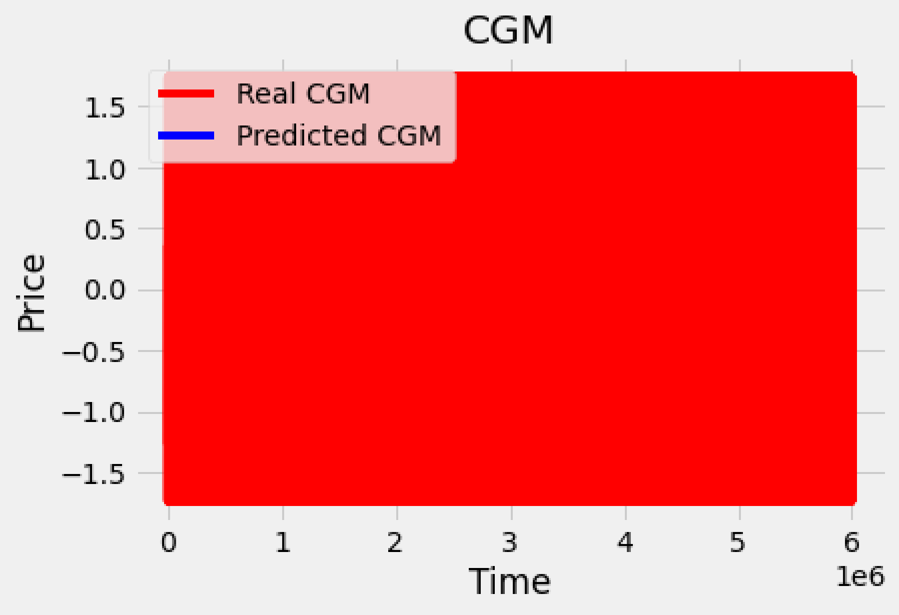
In [41]:

```
initial_sequence = X_train[2708,:]
sequence = []
for i in range(251):
    new_prediction = regressorGRU.predict(initial_sequence.reshape(initial_sequence.sha
    initial_sequence = initial_sequence[1:]
    initial_sequence = np.append(initial_sequence,new_prediction,axis=0)
    sequence.append(new_prediction)
sequence = sc.inverse_transform(np.array(sequence).reshape(251,1))
```

In [42]:

```
plot_predictions(test_set,sequence)
```

```
F:\Anaconda\lib\site-packages\IPython\core\pylabtools.py:132: UserWarning: Creating lege
nd with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



In [20]:

```
datas = pd.read_csv('HIGGS_6M.csv')
datas.head()
```

Out[20]:

| | 1.000000000000000000e+00 | 8.692932128906250000e-01 | -6.350818276405334473e-01 | 2.256902605295181 |
|---|---|---|---|---|
| **0** | 1.0 | 0.907542 | 0.329147 | 0.3 |

| | 1.000000000000000000e+00 | 8.692932128906250000e-01 | -6.350818276405334473e-01 | 2.256902605295181 |
|---|---|---|---|---|
| **1** | 1.0 | 0.798835 | 1.470639 | -1.6 |
| **2** | 0.0 | 1.344385 | -0.876626 | 0.9 |
| **3** | 1.0 | 1.105009 | 0.321356 | 1.5 |
| **4** | 0.0 | 1.595839 | -0.607811 | 0.0 |

5 rows × 29 columns

In [5]:

```
training_set = datas.iloc[:,1:2].values
test_set = datas.iloc[:,3:4].values
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-407105148f96> in <module>
----> 1 training_set = datas.iloc[:,1:2].values
      2 test_set = datas.iloc[:,3:4].values

NameError: name 'datas' is not defined
```

In [21]:

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = datas.predict_proba(test_set)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(test_set, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-21-2998addbe717> in <module>
      1 import sklearn.metrics as metrics
      2 # calculate the fpr and tpr for all thresholds of the classification
----> 3 probs = datas.predict_proba(test_set)
      4 preds = probs[:,1]
      5 fpr, tpr, threshold = metrics.roc_curve(test_set, preds)

F:\Anaconda\lib\site-packages\pandas\core\generic.py in __getattr__(self, name)
   5137             if self._info_axis._can_hold_identifiers_and_holds_name(name):
   5138                 return self[name]
-> 5139             return object.__getattribute__(self, name)
   5140 
   5141     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'predict_proba'
```

In [ ]: