Importing the libraries

In [1]:

```
import quandl
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Fetching the real time stock prices using quandl API ¶

In [2]:

```
df_train= quandl.get("EOD/WMT", authtoken="PHw9dWm3AAZdPf2S8Ph1",
                     start_date="2001-01-01", end_date="2017-12-31")

df_test = quandl.get("EOD/WMT", authtoken="PHw9dWm3AAZdPf2S8Ph1",
                     start_date="2018-01-01", end_date="2018-09-30")
```

Data Exploration

In [3]:

```
df_train.tail(10)
```

Out[3]:

	Open	High	Low	Close	Volume	Dividend	Split	Adj_Open	Adj_High
Date									
2017-12-15	98.04	98.49	96.9100	97.11	16141258.0	0.0	1.0	96.291940	96.733916
2017-12-18	97.44	98.17	97.3600	97.90	7964156.0	0.0	1.0	95.702638	96.419622
2017-12-19	99.78	99.91	98.2900	98.80	12011446.0	0.0	1.0	98.000915	98.128597
2017-12-20	99.48	99.65	98.4500	98.75	10729605.0	0.0	1.0	97.706264	97.873233
2017-12-21	99.32	99.33	97.9800	98.06	7368128.0	0.0	1.0	97.549117	97.558939
2017-12-22	98.14	98.42	97.6800	98.21	5478016.0	0.0	1.0	96.390157	96.665164
2017-12-26	98.35	99.44	98.3500	99.16	4295891.0	0.0	1.0	96.596412	97.666977
2017-12-27	99.56	99.60	98.8299	99.26	5140793.0	0.0	1.0	97.784838	97.824125
2017-12-28	99.52	99.62	99.1200	99.40	9763936.0	0.0	1.0	97.745551	97.843768
2017-12-29	99.40	99.69	98.7500	98.75	7144273.0	0.0	1.0	97.627691	97.912520

In [4]:

```
training_set = df_train.iloc[:, 3:4]
training_set.head()
```

Out[4]:

	Close
Date	
2001-01-02	53.88
2001-01-03	58.44
2001-01-04	56.19
2001-01-05	53.94
2001-01-08	53.94

In [5]:

```
df_train.describe()
```

Out[5]:

High	Low	Close	Volume	Dividend	Split	Adj_Open
276.000000	4276.000000	4276.000000	4.276000e+03	4276.000000	4276.0	4276.000000
0.062879	59.076665	59.574365	1.162714e+07	0.004609	1.0	49.130170
1.698640	11.704923	11.708538	7.036048e+06	0.041342	0.0	14.263173
2.680000	41.500000	42.270000	2.031400e+06	0.000000	1.0	31.504749
1.145000	50.010000	50.585000	7.071072e+06	0.000000	1.0	38.235853
5.709000	54.530000	55.100000	9.585800e+06	0.000000	1.0	42.843184
0.422500	69.695000	69.982500	1.407239e+07	0.000000	1.0	63.663937
00.130000	99.120000	99.620000	9.678680e+07	0.510000	1.0	98.000915

Checking for Missing values

In [6]:

```
df_train.isnull().sum()
```

Out[6]:

```
Open          0
High          0
Low           0
Close         0
Volume        0
Dividend      0
Split         0
Adj_Open      0
Adj_High      0
Adj_Low       0
Adj_Close     0
Adj_Volume    0
dtype: int64
```

Exploratory Visualization

In [7]:

```
plt.plot(training_set, color = 'red')
plt.title('Walmart Time-series Plot')
plt.xlabel('Time')
plt.ylabel('Walmart Stock Price')
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```

In [8]:

```
df_train = df_train.reset_index()
```

In [9]:

```
import plotly.plotly as py
import plotly.graph_objs as go
import plotly
```

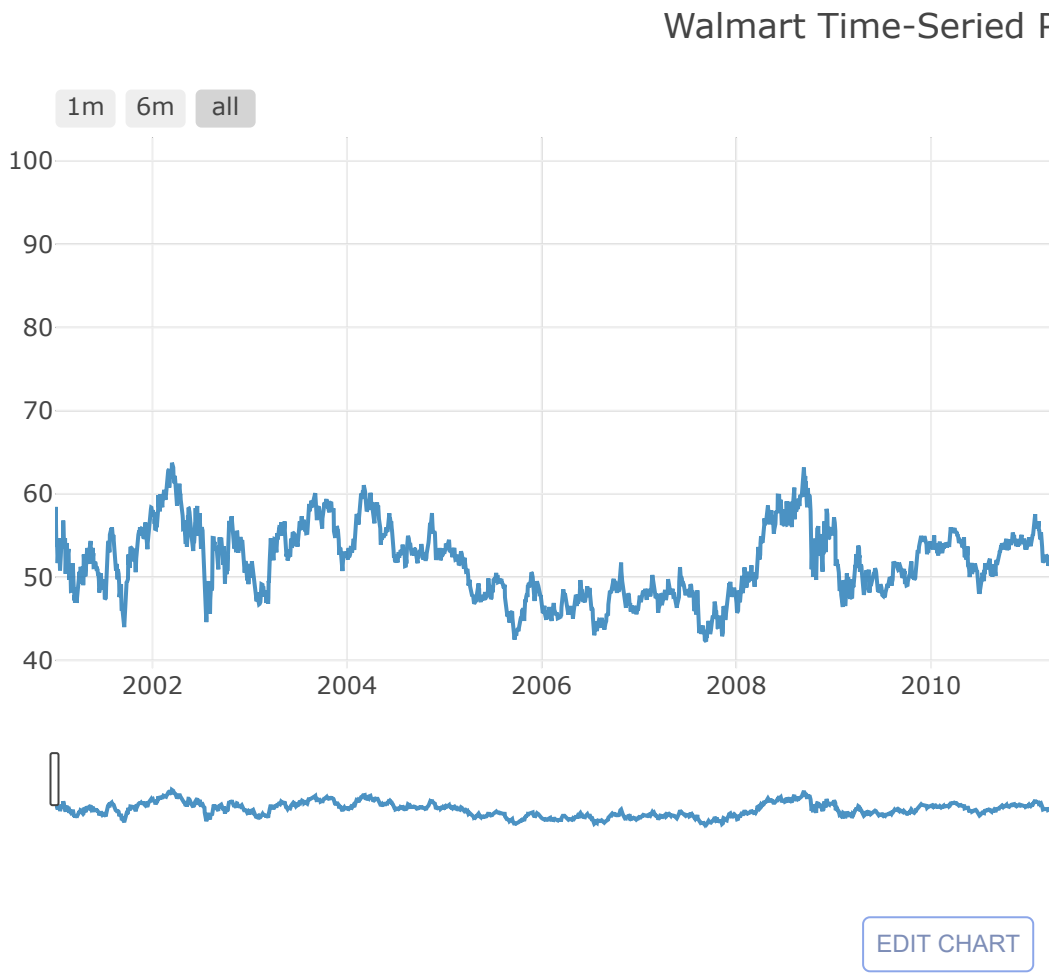
In [10]:

```
plotly.tools.set_credentials_file(username='sanath11', api_key='GJgCG3P6HOdoOTqaVyKD')
```

In [11]:

```
trace_close = go.Scatter(
    x=df_train["Date"],
    y=df_train["Close"],
    name = "High",
    opacity = 0.8)


data = [trace_close]

layout = dict(
    title='Walmart Time-Seried Plot',
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label='1m',
                     step='month',
                     stepmode='backward'),
                dict(count=6,
                     label='6m',
                     step='month',
                     stepmode='backward'),
                dict(step='all')
            ])
        ),
        rangeslider=dict(
            visible = True
        ),
        type='date'
    )
)

fig = dict(data=data, layout=layout)
py.iplot(fig, filename = "Interactive Walmart Time-Seried Plot")
```

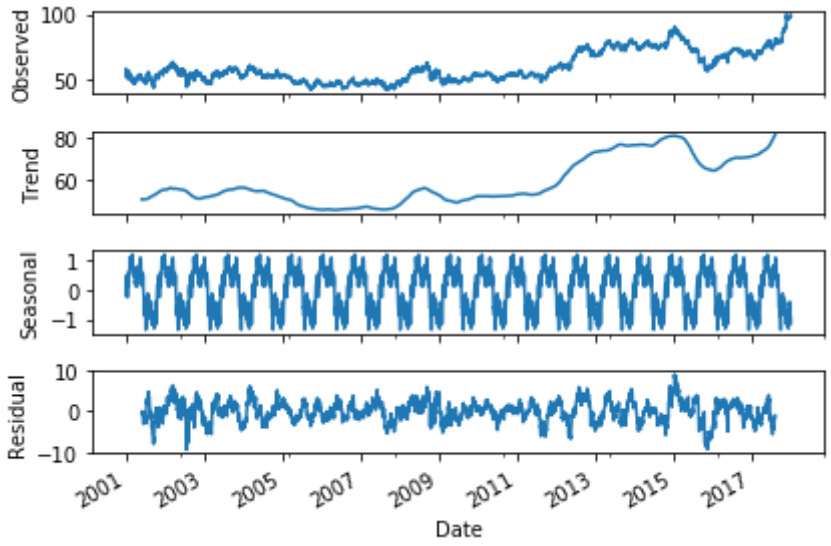
Out[11]:



Seasonal Decompostion

In [12]:

```
import statsmodels.api as sm
decomposition = sm.tsa.seasonal_decompose(training_set, freq = 204)
fig = decomposition.plot()
plt.show()
```



Feature Scaling

In [13]:

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

Data Pre-processing

Creating a data structure with 60 timesteps and 1 output

In [14]:

```
X_train = []
y_train = []
for i in range(60, 4276):
    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)

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

Basic LSTM Model

Importing the Keras libraries and packages

In [15]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

```
C:\Users\sanat\Anaconda3\envs\tensorflow\lib\site-packages\h5py\__init__.p
y:36: FutureWarning:

Conversion of the second argument of issubdtype from `float` to `np.floati
ng` is deprecated. In future, it will be treated as `np.float64 == np.dtyp
e(float).type`.

Using TensorFlow backend.
```

Build a basic Long-Short Term Memory model

In [16]:

```
# Initialising the RNN
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 10, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(metrics=['accuracy'], optimizer = 'adam', loss = 'mean_squared_error'
)
```

Fitting the RNN to the Training set

In [17]:

```
regressor.fit(X_train, y_train, epochs = 1, batch_size = 8)
```

```
Epoch 1/1
4216/4216 [==============================] - 17s 4ms/step - loss: 0.0248 -
acc: 4.7438e-04
```

Out[17]:

```
<keras.callbacks.History at 0x23219324748>
```

Pre-processing the test data for prediction

In [18]:

```
test_set = df_test.iloc[:, 3:4].values

dataset_total = pd.concat((df_train['Close'], df_test['Close']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(df_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 248):
    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))
```

Prediction on basic LSTM model

In [19]:

```
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Test score (Mean squared error)

In [20]:

```
import sklearn.metrics
sklearn.metrics.mean_squared_error(test_set, predicted_stock_price)
```

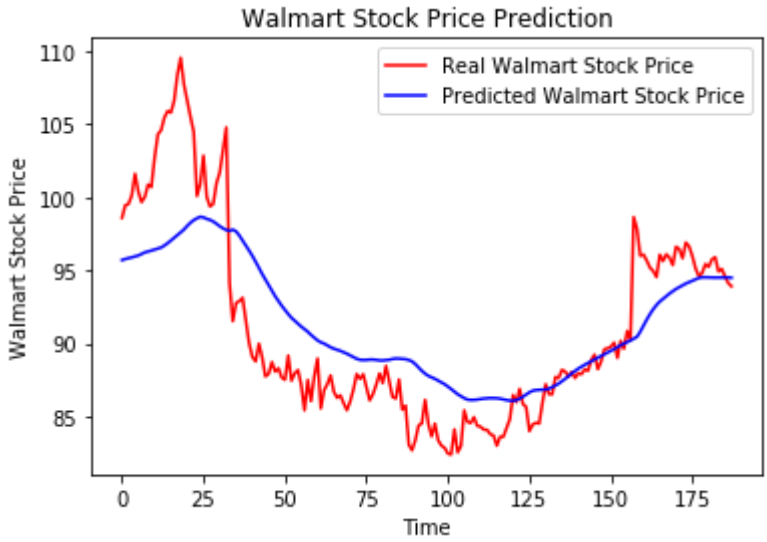
Out[20]:

15.102331449491198

Visualizing the predictions

In [21]:

```
plt.plot(test_set, color = 'red', label = 'Real Walmart Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Walmart Stock Price'
)
plt.title('Walmart Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Walmart Stock Price')
plt.legend()
plt.show()
```



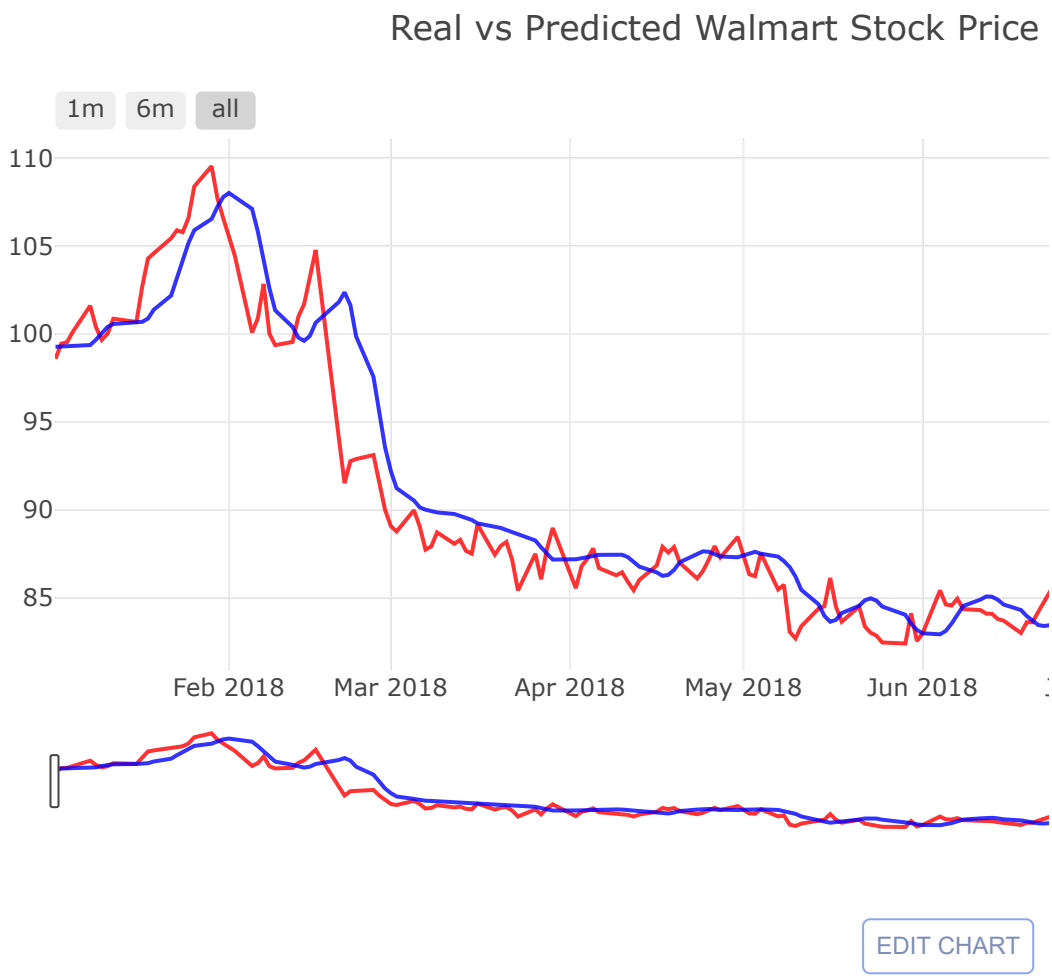
In [23]:

```
#df_test = df_test.reset_index()
```

In [24]:

```
trace_high = go.Scatter(
    x=df_test["Date"],
    y=df_test["Close"],
    name = "Real  Walmart Stock Price",
    line = dict(color = 'red'),
    opacity = 0.8)

trace_low = go.Scatter(
    x=df_test["Date"],
    y=predicted_stock_price,
    name = "Predicted Walmart Stock Price",
    line = dict(color = 'Blue'),
    opacity = 0.8)

data = [trace_high,trace_low]

layout = dict(
    title='Real vs Predicted Walmart Stock Prices for Basic LSTM',
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label='1m',
                     step='month',
                     stepmode='backward'),
                dict(count=6,
                     label='6m',
                     step='month',
                     stepmode='backward'),
                dict(step='all')
            ])
        ),
        rangeslider=dict(
            visible = True
        ),
        type='date'
    )
)

fig = dict(data=data, layout=layout)
py.iplot(fig, filename = "Interactive Real vs Predicted Walmart Time-Seried Plot")
```

Out[24]:



Improvised LSTM

Importing the Keras libraries and packages

In [25]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

Building an improved LSTM model

In [36]:

```
# Initialising the RNN
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1
], 1)))
regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(metrics=['accuracy'], optimizer = 'adam', loss = 'mean_squared_error'
)
```

Fitting the RNN to the Training set

In [37]:

```
regressor.fit(X_train, y_train, epochs = 50, batch_size = 128)
```

```
Epoch 1/50
4216/4216 [==============================] - 13s 3ms/step - loss: 0.0260 -
acc: 4.7438e-04
Epoch 2/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0037 -
acc: 4.7438e-04
Epoch 3/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0030 -
acc: 4.7438e-04
Epoch 4/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0027 -
acc: 4.7438e-04
Epoch 5/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0026 -
acc: 4.7438e-04
Epoch 6/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0024 -
acc: 4.7438e-04
Epoch 7/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0024 -
acc: 4.7438e-04
Epoch 8/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0024 -
acc: 4.7438e-04
Epoch 9/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0022 -
acc: 4.7438e-04
Epoch 10/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0022 -
acc: 4.7438e-04
Epoch 11/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0020 -
acc: 4.7438e-04
Epoch 12/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0020 -
acc: 4.7438e-04
Epoch 13/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0018 -
acc: 4.7438e-04
Epoch 14/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0019 -
acc: 4.7438e-04
Epoch 15/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0020 -
acc: 4.7438e-04
Epoch 16/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0017 -
acc: 4.7438e-04
Epoch 17/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0017 -
acc: 4.7438e-04
Epoch 18/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0018 -
acc: 4.7438e-04
Epoch 19/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0017 -
acc: 4.7438e-04
Epoch 20/50
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0016 -
acc: 4.7438e-04
Epoch 21/50
```

```
4216/4216 [==============================] - 9s 2ms/step - loss: 0.0016 -
acc: 4.7438e-04
Epoch 22/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0016 -
acc: 4.7438e-04
Epoch 23/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0016 -
acc: 4.7438e-04
Epoch 24/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0015 -
acc: 4.7438e-04
Epoch 25/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0016 -
acc: 4.7438e-04
Epoch 26/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0015 -
acc: 4.7438e-04
Epoch 27/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0014 -
acc: 4.7438e-04
Epoch 28/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0014 -
acc: 4.7438e-04
Epoch 29/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0014 -
acc: 4.7438e-04
Epoch 30/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0013 -
acc: 4.7438e-04
Epoch 31/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0013 -
acc: 4.7438e-04
Epoch 32/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0013 -
acc: 4.7438e-04
Epoch 33/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0013 -
acc: 4.7438e-04
Epoch 34/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0012 -
acc: 4.7438e-04
Epoch 35/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0013 -
acc: 4.7438e-04
Epoch 36/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0012 -
acc: 4.7438e-04
Epoch 37/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0012 -
acc: 4.7438e-04
Epoch 38/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0012 -
acc: 4.7438e-04
Epoch 39/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0012 -
acc: 4.7438e-04
Epoch 40/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0012 -
acc: 4.7438e-04
Epoch 41/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0011 -
```

```
acc: 4.7438e-04
Epoch 42/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0011 -
acc: 4.7438e-04
Epoch 43/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0011 -
acc: 4.7438e-04
Epoch 44/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0011 -
acc: 4.7438e-04
Epoch 45/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0010 -
acc: 4.7438e-04
Epoch 46/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0011 -
acc: 4.7438e-04
Epoch 47/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0011 -
acc: 4.7438e-04
Epoch 48/50
4216/4216 [==============================] - 8s 2ms/step - loss: 0.0010 -
acc: 4.7438e-04
Epoch 49/50
4216/4216 [==============================] - 8s 2ms/step - loss: 9.7440e-0
4 - acc: 4.7438e-04
Epoch 50/50
4216/4216 [==============================] - 8s 2ms/step - loss: 9.9751e-0
4 - acc: 4.7438e-04
```

Out[37]:

```
<keras.callbacks.History at 0x2322ff6b6d8>
```

Pre-processing the test data for prediction

In [38]:

```
# Part 3 - Making the predictions and visualising the results

test_set = df_test.iloc[:, 3:4].values

dataset_total = pd.concat((df_train['Close'], df_test['Close']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(df_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 248):
    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))
```

Prediction on improved LSTM model

In [39]:

```
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Test score (Mean squared error)

In [40]:

```
import sklearn.metrics
sklearn.metrics.mean_squared_error(test_set, predicted_stock_price)
```

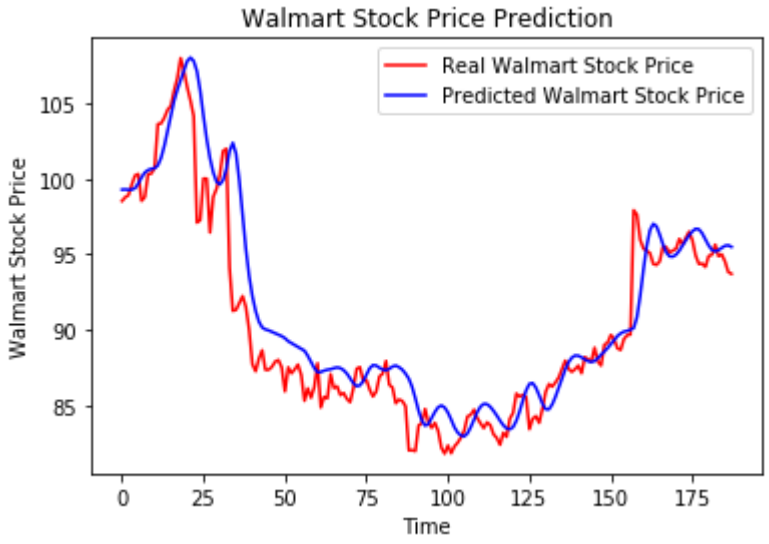
Out[40]:

```
6.371900788157693
```

Visualizing the predictions

In [41]:

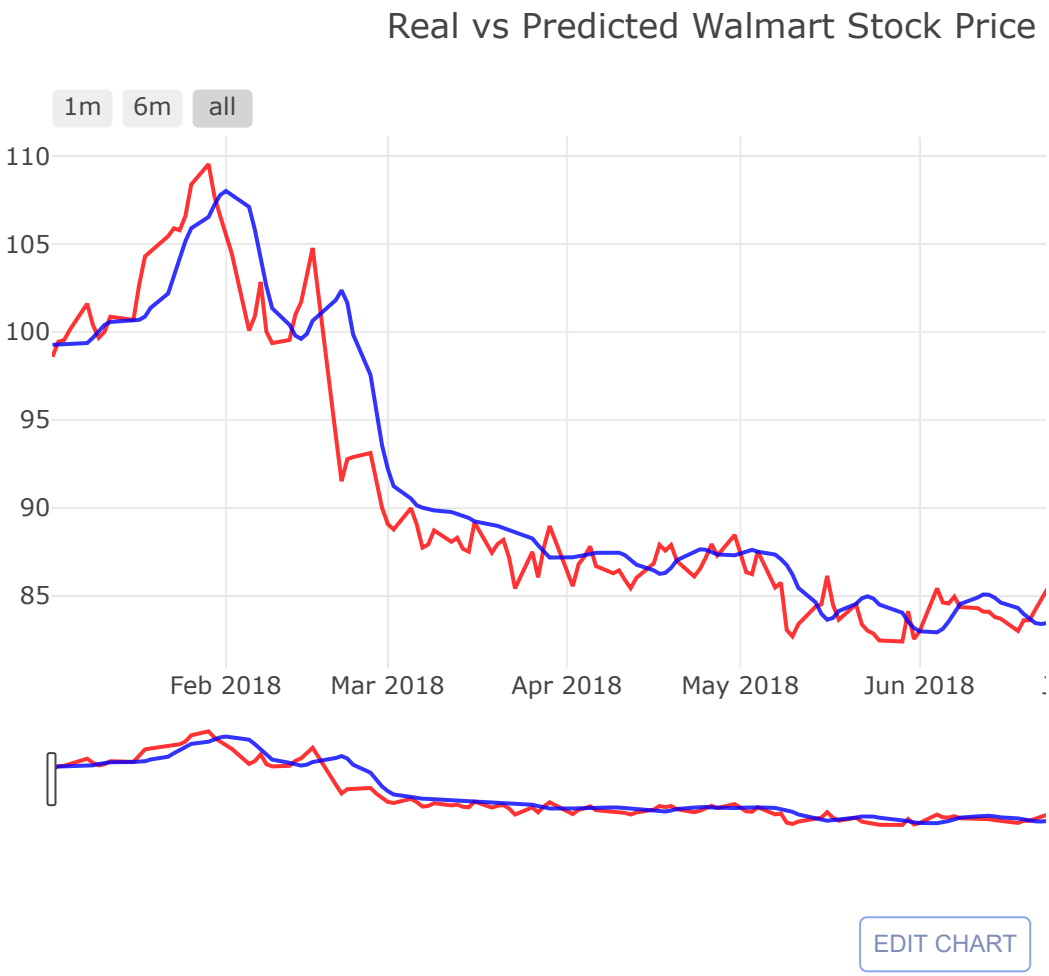
```
plt.plot(test_set, color = 'red', label = 'Real Walmart Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Walmart Stock Price'
)
plt.title('Walmart Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Walmart Stock Price')
plt.legend()
plt.show()
```



In [43]:

```
trace_high = go.Scatter(
    x=df_test["Date"],
    y=df_test["Close"],
    name = "Real",
    line = dict(color = 'red'),
    opacity = 0.8)

trace_low = go.Scatter(
    x=df_test["Date"],
    y=predicted_stock_price,
    name = "Predicted",
    line = dict(color = 'Blue'),
    opacity = 0.8)

data = [trace_high,trace_low]

layout = dict(
    title='Real vs Predicted Walmart Stock Price for improved LSTM',
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label='1m',
                     step='month',
                     stepmode='backward'),
                dict(count=6,
                     label='6m',
                     step='month',
                     stepmode='backward'),
                dict(step='all')
            ])
        ),
        rangeslider=dict(
            visible = True
        ),
        type='date'
    )
)

fig = dict(data=data, layout=layout)
py.iplot(fig, filename = "Interactive Real vs Predicted Walmart Time-Seried Plot")
```

Out[43]:


Real vs Predicted Walmart Stock Price
1m
6m
all
110
105
100
95
90
85
Feb 2018
Mar 2018
Apr 2018
May 2018
Jun 2018
EDIT CHART

Thank you