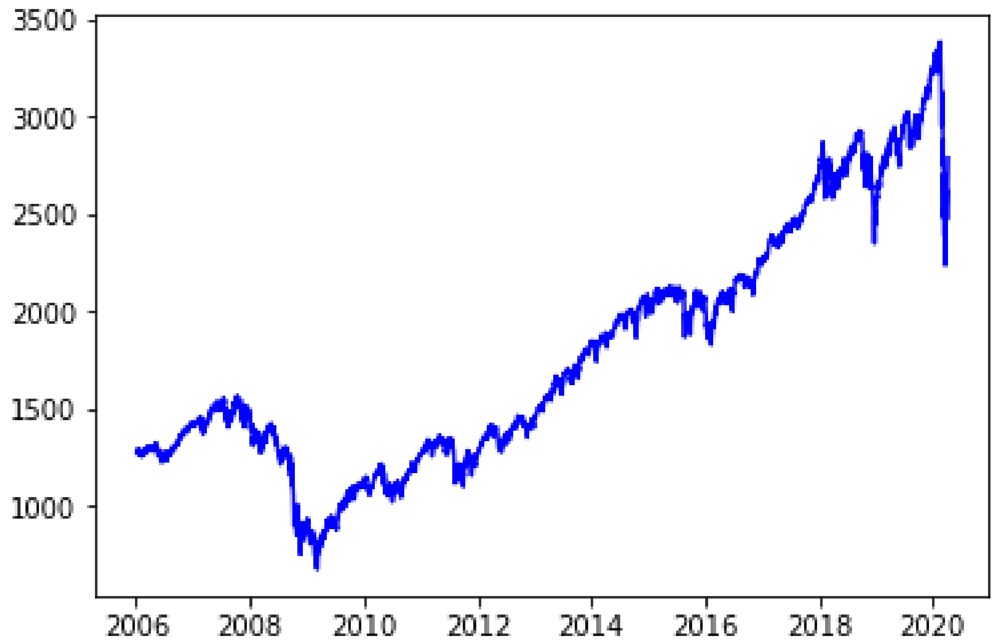
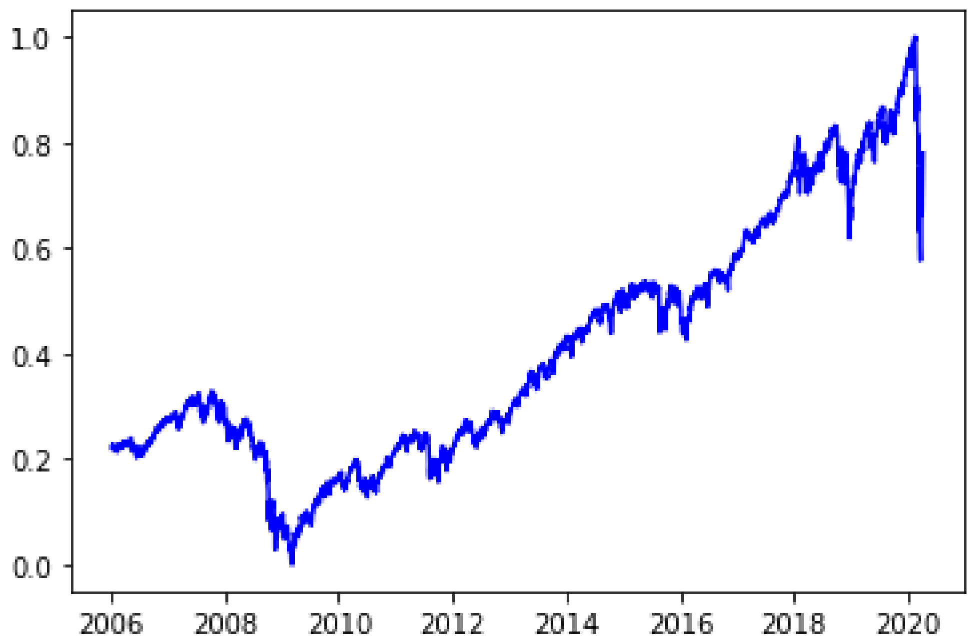
```
import tensorflow as tf

tf.test.gpu_device_name()

Out[1]: '/device:GPU:0'


import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


data = pd.read_csv('S&P 500 Historical Data.csv')


dataset = data.iloc[:, 0:2].values

dataset[:,0] = pd.to_datetime(dataset[:,0])


plt.plot(dataset[:,0], dataset[:,1],  color = 'blue', label = 'Real S&P 500 Stock Price')

plt.show()

C:\Anaconda3\lib\site-packages\pandas\plotting\_matplotlib\converter.py:103: FutureWarning: Using an implicitly registered datetime converter for a matplotlib plotting method. The converter was registered by pandas on import. Future versions of pandas will require you to explicitly register matplotlib converters.


To register the converters:

from pandas.plotting import register_matplotlib_converters

register_matplotlib_converters()

  warnings.warn(msg, FutureWarning)
```




```
from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature_range = (0, 1))

#Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1, -1) if it contains a single sample.

dataset_scaled = sc.fit_transform(dataset[:,1].reshape(-1, 1))


# Visualising the scaled price

plt.plot(dataset[:,0], dataset_scaled,  color = 'blue', label = 'Real S&P 500 Stock Price')

plt.show()


# Creating a data structure, use 60 previous prices to price today's price

X = []

y = []

for i in range(90, len(dataset_scaled)):

   X.append(dataset_scaled[i-90:i-30, 0])

   y.append(dataset_scaled[i, 0])
```

```
X = np.array(X)
y = np.array(y)

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



(3500, 60, 1)

```
from sklearn.model_selection import train_test_split
test_amount = int(X.shape[0]*0.7)
X_train = X[0:test_amount, :, :]
X_test = X[test_amount:, :, :]
y_train = y[0:test_amount]
y_test = y[test_amount:]

from keras.models import Sequential
```

```
from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout


# Initialising the RNN

regressor = Sequential()


# Adding the first LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 100, 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(optimizer = 'adam', loss = 'mean_squared_error',
metrics=['mean_absolute_error'])


# Show Model Structure

regressor.summary()


# Fitting the RNN to the Training set

history = regressor.fit(X_train, y_train, epochs = 50, batch_size = 32,

                    validation_data=(X_test, y_test))

Using TensorFlow backend.

Model: "sequential_1"

_________________________________________________________________

Layer (type)             Output Shape           Param #

=================================================================
```

lstm_1 (LSTM) (None, 100) 40800

____

dropout_1 (Dropout) (None, 100) 0

____

dense_1 (Dense) (None, 1) 101

===

Total params: 40,901

Trainable params: 40,901

Non-trainable params: 0

____

Train on 2450 samples, validate on 1050 samples

Epoch 1/50

2450/2450 [==============================] - 3s 1ms/step - loss: 0.0080 - mean_absolute_error: 0.0556 - val_loss: 0.0037 - val_mean_absolute_error: 0.0379

Epoch 2/50

2450/2450 [==============================] - 2s 974us/step - loss: 0.0014 - mean_absolute_error: 0.0278 - val_loss: 0.0038 - val_mean_absolute_error: 0.0397

Epoch 3/50

2450/2450 [==============================] - 2s 959us/step - loss: 0.0013 - mean_absolute_error: 0.0271 - val_loss: 0.0038 - val_mean_absolute_error: 0.0422

Epoch 4/50

2450/2450 [==============================] - 2s 966us/step - loss: 0.0012 - mean_absolute_error: 0.0268 - val_loss: 0.0036 - val_mean_absolute_error: 0.0373

Epoch 5/50

2450/2450 [==============================] - 2s 958us/step - loss: 0.0012 - mean_absolute_error: 0.0265 - val_loss: 0.0037 - val_mean_absolute_error: 0.0428

Epoch 6/50

2450/2450 [==============================] - 2s 947us/step - loss: 0.0012 - mean_absolute_error: 0.0257 - val_loss: 0.0041 - val_mean_absolute_error: 0.0486

Epoch 7/50

2450/2450 [==============================] - 2s 976us/step - loss: 0.0012 - mean_absolute_error: 0.0262 - val_loss: 0.0036 - val_mean_absolute_error: 0.0352

Epoch 8/50

2450/2450 [==============================] - 3s 1ms/step - loss: 0.0011 - mean_absolute_error: 0.0256 - val_loss: 0.0044 - val_mean_absolute_error: 0.0368

Epoch 9/50

2450/2450 [==============================] - 3s 1ms/step - loss: 0.0012 - mean_absolute_error: 0.0260 - val_loss: 0.0036 - val_mean_absolute_error: 0.0405

Epoch 10/50

2450/2450 [==============================] - 3s 1ms/step - loss: 0.0011 - mean_absolute_error: 0.0252 - val_loss: 0.0039 - val_mean_absolute_error: 0.0466

Epoch 11/50

2450/2450 [==============================] - 2s 971us/step - loss: 0.0012 - mean_absolute_error: 0.0260 - val_loss: 0.0040 - val_mean_absolute_error: 0.0474

Epoch 12/50

2450/2450 [==============================] - 2s 978us/step - loss: 0.0011 - mean_absolute_error: 0.0254 - val_loss: 0.0036 - val_mean_absolute_error: 0.0338

Epoch 13/50

2450/2450 [==============================] - 2s 983us/step - loss: 0.0011 - mean_absolute_error: 0.0254 - val_loss: 0.0045 - val_mean_absolute_error: 0.0376

Epoch 14/50

2450/2450 [==============================] - 2s 968us/step - loss: 0.0011 - mean_absolute_error: 0.0252 - val_loss: 0.0034 - val_mean_absolute_error: 0.0357

Epoch 15/50

2450/2450 [==============================] - 2s 958us/step - loss: 0.0011 - mean_absolute_error: 0.0248 - val_loss: 0.0036 - val_mean_absolute_error: 0.0420

Epoch 16/50

2450/2450 [==============================] - 2s 958us/step - loss: 0.0011 - mean_absolute_error: 0.0251 - val_loss: 0.0036 - val_mean_absolute_error: 0.0415

Epoch 17/50

2450/2450 [==============================] - 2s 1ms/step - loss: 0.0011 - mean_absolute_error: 0.0247 - val_loss: 0.0039 - val_mean_absolute_error: 0.0459

Epoch 18/50

2450/2450 [==============================] - 2s 955us/step - loss: 0.0010 - mean_absolute_error: 0.0242 - val_loss: 0.0040 - val_mean_absolute_error: 0.0474

Epoch 19/50

2450/2450 [==============================] - 2s 959us/step - loss: 0.0011 - mean_absolute_error: 0.0249 - val_loss: 0.0036 - val_mean_absolute_error: 0.0432

Epoch 20/50

2450/2450 [==============================] - 2s 953us/step - loss: 0.0011 - mean_absolute_error: 0.0246 - val_loss: 0.0035 - val_mean_absolute_error: 0.0407

Epoch 21/50

2450/2450 [==============================] - 2s 967us/step - loss: 0.0010 - mean_absolute_error: 0.0243 - val_loss: 0.0037 - val_mean_absolute_error: 0.0445

Epoch 22/50

2450/2450 [==============================] - 2s 952us/step - loss: 9.8855e-04 - mean_absolute_error: 0.0238 - val_loss: 0.0034 - val_mean_absolute_error: 0.0390

Epoch 23/50

2450/2450 [==============================] - 2s 968us/step - loss: 0.0010 - mean_absolute_error: 0.0245 - val_loss: 0.0033 - val_mean_absolute_error: 0.0362

Epoch 24/50

2450/2450 [==============================] - 2s 971us/step - loss: 0.0011 - mean_absolute_error: 0.0246 - val_loss: 0.0038 - val_mean_absolute_error: 0.0452

Epoch 25/50

2450/2450 [==============================] - 2s 953us/step - loss: 9.9225e-04 - mean_absolute_error: 0.0240 - val_loss: 0.0035 - val_mean_absolute_error: 0.0408

Epoch 26/50

2450/2450 [==============================] - 2s 943us/step - loss: 9.9731e-04 - mean_absolute_error: 0.0238 - val_loss: 0.0044 - val_mean_absolute_error: 0.0520

Epoch 27/50

2450/2450 [==============================] - 2s 973us/step - loss: 0.0010 - mean_absolute_error: 0.0244 - val_loss: 0.0033 - val_mean_absolute_error: 0.0354

Epoch 28/50

2450/2450 [==============================] - 2s 960us/step - loss: 9.9780e-04 - mean_absolute_error: 0.0239 - val_loss: 0.0036 - val_mean_absolute_error: 0.0425

Epoch 29/50

2450/2450 [==============================] - 2s 952us/step - loss: 0.0010 - mean_absolute_error: 0.0242 - val_loss: 0.0033 - val_mean_absolute_error: 0.0383

Epoch 30/50

2450/2450 [==============================] - 2s 955us/step - loss: 9.5860e-04 - mean_absolute_error: 0.0235 - val_loss: 0.0035 - val_mean_absolute_error: 0.0407

Epoch 31/50

2450/2450 [==============================] - 2s 950us/step - loss: 9.8507e-04 - mean_absolute_error: 0.0240 - val_loss: 0.0034 - val_mean_absolute_error: 0.0339

Epoch 32/50

2450/2450 [==============================] - 2s 966us/step - loss: 9.6124e-04 - mean_absolute_error: 0.0236 - val_loss: 0.0033 - val_mean_absolute_error: 0.0340

Epoch 33/50

2450/2450 [==============================] - 2s 957us/step - loss: 9.6413e-04 - mean_absolute_error: 0.0237 - val_loss: 0.0034 - val_mean_absolute_error: 0.0399

Epoch 34/50

2450/2450 [==============================] - 2s 962us/step - loss: 9.4942e-04 - mean_absolute_error: 0.0233 - val_loss: 0.0033 - val_mean_absolute_error: 0.0388

Epoch 35/50

2450/2450 [==============================] - 2s 987us/step - loss: 9.6949e-04 - mean_absolute_error: 0.0237 - val_loss: 0.0033 - val_mean_absolute_error: 0.0364

Epoch 36/50

2450/2450 [==============================] - 2s 960us/step - loss: 9.3106e-04 - mean_absolute_error: 0.0232 - val_loss: 0.0034 - val_mean_absolute_error: 0.0403

Epoch 37/50

2450/2450 [==============================] - 2s 952us/step - loss: 9.8022e-04 - mean_absolute_error: 0.0238 - val_loss: 0.0033 - val_mean_absolute_error: 0.0382

Epoch 38/50

2450/2450 [==============================] - 2s 954us/step - loss: 9.6691e-04 - mean_absolute_error: 0.0233 - val_loss: 0.0040 - val_mean_absolute_error: 0.0478

Epoch 39/50

2450/2450 [==============================] - 2s 942us/step - loss: 9.4972e-04 - mean_absolute_error: 0.0234 - val_loss: 0.0033 - val_mean_absolute_error: 0.0387

Epoch 40/50

2450/2450 [==============================] - 2s 989us/step - loss: 9.1518e-04 - mean_absolute_error: 0.0229 - val_loss: 0.0041 - val_mean_absolute_error: 0.0497

Epoch 41/50

2450/2450 [==============================] - 2s 968us/step - loss: 9.4579e-04 - mean_absolute_error: 0.0233 - val_loss: 0.0036 - val_mean_absolute_error: 0.0425

Epoch 42/50

2450/2450 [==============================] - 2s 942us/step - loss: 9.5123e-04 - mean_absolute_error: 0.0233 - val_loss: 0.0033 - val_mean_absolute_error: 0.0399

Epoch 43/50

2450/2450 [==============================] - 2s 964us/step - loss: 9.3184e-04 - mean_absolute_error: 0.0231 - val_loss: 0.0032 - val_mean_absolute_error: 0.0382: 9.5021e-04 - mean_absolute_error: 0.0234

Epoch 44/50

2450/2450 [==============================] - 2s 971us/step - loss: 9.0415e-04 - mean_absolute_error: 0.0228 - val_loss: 0.0034 - val_mean_absolute_error: 0.0409

Epoch 45/50

2450/2450 [==============================] - 2s 963us/step - loss: 9.0419e-04 - mean_absolute_error: 0.0227 - val_loss: 0.0031 - val_mean_absolute_error: 0.0363

Epoch 46/50

2450/2450 [==============================] - 2s 952us/step - loss: 9.1443e-04 - mean_absolute_error: 0.0228 - val_loss: 0.0032 - val_mean_absolute_error: 0.0377

Epoch 47/50

2450/2450 [==============================] - 2s 968us/step - loss: 9.2656e-04 - mean_absolute_error: 0.0231 - val_loss: 0.0042 - val_mean_absolute_error: 0.0513

Epoch 48/50

2450/2450 [==============================] - 2s 957us/step - loss: 9.3321e-04 - mean_absolute_error: 0.0230 - val_loss: 0.0035 - val_mean_absolute_error: 0.0429

Epoch 49/50

2450/2450 [==============================] - 2s 965us/step - loss: 9.1472e-04 - mean_absolute_error: 0.0229 - val_loss: 0.0037 - val_mean_absolute_error: 0.0454

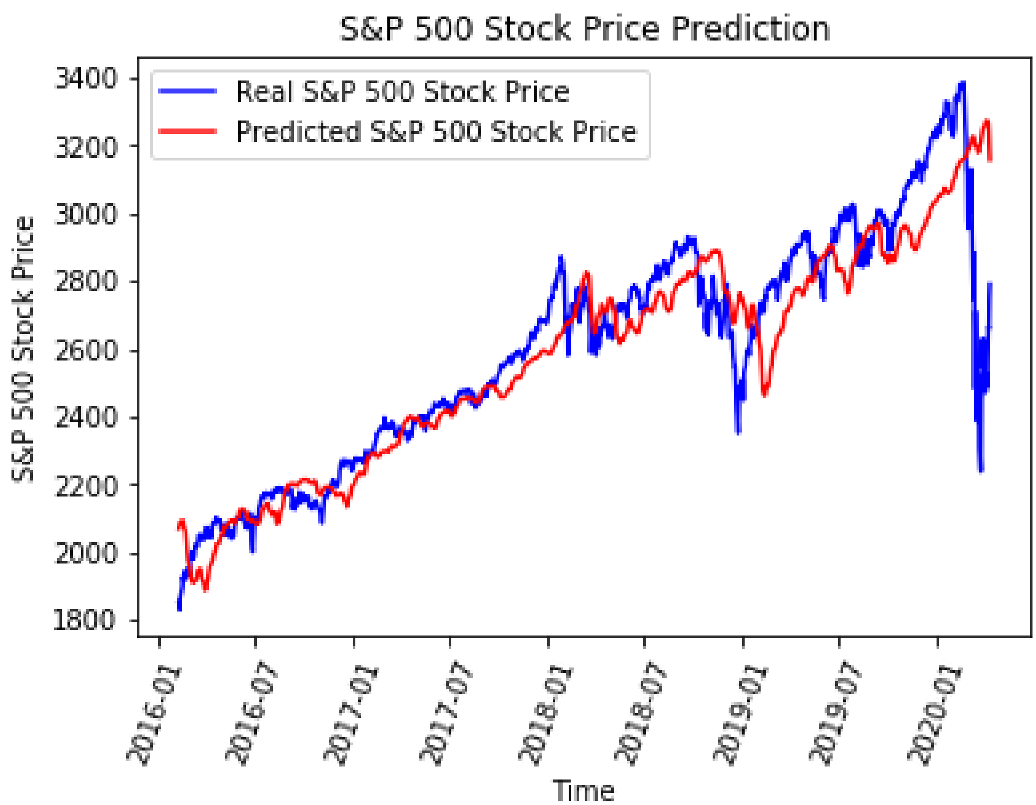
Epoch 50/50

2450/2450 [==============================] - 2s 952us/step - loss: 8.9197e-04 - mean_absolute_error: 0.0228 - val_loss: 0.0030 - val_mean_absolute_error: 0.0363

```
mae = history.history['mean_absolute_error']

mae_test = history.history['val_mean_absolute_error']

epochs = range(len(mae))


from matplotlib import pyplot as plt

plt.plot(epochs, mae, 'b-', label='Training Data: MAE')

plt.plot(epochs, mae_test, 'r-', label='Test Data: MAE')

plt.title('Training and Test Dataset')

plt.legend()

plt.show()
```


```
real_stock_price = dataset[test_amount + 90:,1]

predicted_stock_price = regressor.predict(X_test)

predicted_stock_price = sc.inverse_transform(predicted_stock_price)


# Visualising the results
```

```
plt.plot(dataset[test_amount + 90:,0], real_stock_price, color = 'blue', label = 'Real S&P 500 Stock Price')

plt.plot(dataset[test_amount + 90:,0], predicted_stock_price, color = 'red', label = 'Predicted S&P 500 Stock Price')

plt.title('S&P 500 Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('S&P 500 Stock Price')

plt.xticks(rotation=70)

plt.legend()

plt.show()


import math

from sklearn.metrics import mean_squared_error

rmse_test = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))

print('The RMSE error on the test dataset', rmse_test)
```



The RMSE error on the test dataset 148.07942819424466