This code implements a **Recurrent Neural Network (RNN)** using **Long Short-Term Memory (LSTM)** units to predict stock prices based on historical data. Here's a step-by-step breakdown:

**1. Importing Libraries**

The necessary libraries are imported:

* numpy for numerical computations.
* matplotlib.pyplot for plotting the results.
* pandas for handling and preprocessing data.
* MinMaxScaler from sklearn.preprocessing for scaling data.
* Sequential, LSTM, Dropout, and Dense from Keras to build the RNN model.

**2. Loading and Preprocessing the Training Data**

* **Dataset Load**: The training data is loaded from the provided URL.
* url = 'https://raw.githubusercontent.com/mwitiderrick/stockprice/master/NSE-TATAGLOBAL.csv'
* dataset\_train = pd.read\_csv(url)
* **Extracting Training Set**: Only the 'Open' column is used (column index 1:2).
* training\_set = dataset\_train.iloc[:, 1:2].values
* **Feature Scaling**: The stock prices are normalized using MinMaxScaler to scale values between 0 and 1. Normalization helps the neural network converge faster.
* sc = MinMaxScaler(feature\_range=(0,1))
* training\_set\_scaled = sc.fit\_transform(training\_set)

**3. Creating Data for the RNN**

The model uses past 60 days of stock prices to predict the 61st day's price:

* **Inputs (X\_train)**: Sequences of 60 stock prices.
* **Outputs (y\_train)**: The stock price following each sequence.

This is achieved using a loop:

X\_train = []

y\_train = []

for i in range(60, 2035):

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 Input Data**: The X\_train array is reshaped to match the 3D input format expected by LSTM (samples, time-steps, features):
* X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

**4. Building the RNN Model**

An LSTM-based model is created with:

* **Input LSTM Layer**: First LSTM layer with 50 units and return\_sequences=True (to allow further LSTM stacking).
* **Dropout Layers**: Added after each LSTM layer to reduce overfitting by randomly setting 20% of inputs to 0.
* **Final LSTM Layer**: Outputs predictions without return\_sequences.
* **Dense Layer**: Fully connected layer to output the final prediction.

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(units=50))

model.add(Dropout(0.2))

model.add(Dense(units=1))

* **Compile the Model**: The model is compiled with the **Adam optimizer** (efficient for training deep models) and the **mean squared error** loss function.
* model.compile(optimizer='adam', loss='mean\_squared\_error')

**5. Training the RNN**

The model is trained on the training data for 100 epochs with a batch size of 32:

model.fit(X\_train, y\_train, epochs=100, batch\_size=32)

**6. Preparing Test Data**

The test dataset is loaded similarly to the training dataset:

url = 'https://raw.githubusercontent.com/mwitiderrick/stockprice/master/tatatest.csv'

dataset\_test = pd.read\_csv(url)

real\_stock\_price = dataset\_test.iloc[:, 1:2].values

* **Combining Train and Test Data**: The last 60 training data points and the entire test dataset are combined to ensure continuity in sequence generation.
* dataset\_total = pd.concat((dataset\_train['Open'], dataset\_test['Open']), axis=0)
* inputs = dataset\_total[len(dataset\_total) - len(dataset\_test) - 60:].values
* inputs = inputs.reshape(-1, 1)
* inputs = sc.transform(inputs)
* **Creating Test Inputs**: Sequences of 60 previous stock prices are generated for test predictions.
* X\_test = []
* for i in range(60, 76):
* 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))

**7. Making Predictions**

* **Predicting Stock Prices**: The model makes predictions on the test set, and the predictions are scaled back to their original range using the inverse transform of the scaler.
* predicted\_stock\_price = model.predict(X\_test)
* predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

**8. Visualization**

The predicted stock prices are plotted alongside the real stock prices:

plt.plot(real\_stock\_price, color='black', label='TATA Stock Price')

plt.plot(predicted\_stock\_price, color='green', label='Predicted TATA Stock Price')

plt.title('TATA Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('TATA Stock Price')

plt.legend()

plt.show()

**How It Works**

1. The model learns patterns from 60-day sequences of stock prices to predict the next day's price.
2. During testing, it uses sequences of known prices (from the test data) to make predictions.
3. The results are visualized to show how well the model predicts future stock prices.

**Potential Issues & Improvements**

1. **Overfitting**: The model might overfit the training data. Consider using techniques like dropout, regularization, or early stopping.
2. **Data Leakage**: Ensure there’s no information from the future in training data during real-world deployment.
3. **More Features**: Incorporate additional data like trading volume, news sentiment, or technical indicators to improve predictions.