```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense,LSTM,Dropout


from google.colab import files
uploaded = files.upload()

# Get the filename
filename = list(uploaded.keys())[0]

# Load the data into a pandas DataFrame
data = pd.read_csv(filename)

# Now you can call info()
data.info()
```

```
Choose files  2 files
```

- **Google_test_data.csv**(text/csv) - 19908 bytes, last modified: 25/04/2025 - 100% done
- **Google_train_data.csv**(text/csv) - 62230 bytes, last modified: 25/04/2025 - 100% done

```
Saving Google_test_data.csv to Google_test_data (1).csv
Saving Google_train_data.csv to Google_train_data (1).csv
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       252 non-null    object
 1   Open       252 non-null    float64
 2   High       252 non-null    float64
 3   Low        252 non-null    float64
 4   Close      252 non-null    float64
 5   Adj Close  252 non-null    float64
 6   Volume     252 non-null    int64
dtypes: float64(5), int64(1), object(1)
memory usage: 13.9+ KB
```

```
data["Close"]=pd.to_numeric(data.Close,errors='coerce')
data = data.dropna()
trainData = data.iloc[:,4:5].values


data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       252 non-null    object
 1   Open       252 non-null    float64
 2   High       252 non-null    float64
 3   Low        252 non-null    float64
 4   Close      252 non-null    float64
 5   Adj Close  252 non-null    float64
 6   Volume     252 non-null    int64
dtypes: float64(5), int64(1), object(1)
memory usage: 13.9+ KB
```

```
sc = MinMaxScaler(feature_range=(0,1))
trainData = sc.fit_transform(trainData)
trainData.shape
```

```
(252, 1)
```

```
X_train = []
y_train = []

# Change the range to iterate up to the length of trainData - 1
for i in range (60, len(trainData)):
    X_train.append(trainData[i-60:i,0])
    y_train.append(trainData[i,0])

X_train,y_train = np.array(X_train),np.array(y_train)


X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1)) #adding the batch_size axis
X_train.shape
```

```
(192, 60, 1)
```

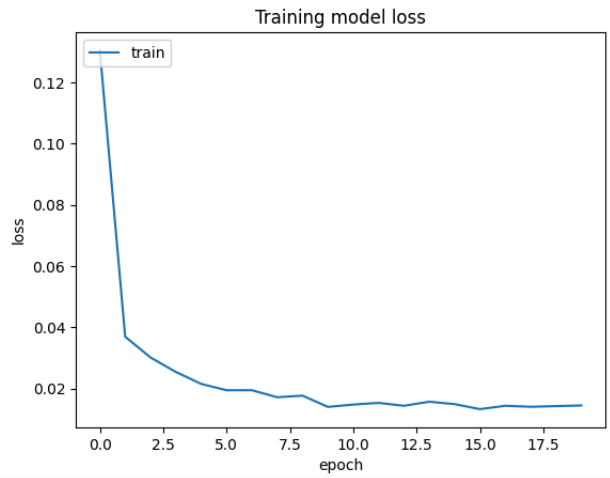
```
model = Sequential()

model.add(LSTM(units=100, return_sequences = True, input_shape =(X_train.shape[1],1)))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = False))
model.add(Dropout(0.2))

model.add(Dense(units =1))
model.compile(optimizer='adam',loss="mean_squared_error")
```

```
Show hidden output
```

```
hist = model.fit(X_train, y_train, epochs = 20, batch_size = 32, verbose=2)
```

```
Show hidden output
```

```
plt.plot(hist.history['loss'])
plt.title('Training model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```




```
from google.colab import files
uploaded = files.upload()

# Get the filename
filename = list(uploaded.keys())[0]

# Load the data into a pandas DataFrame
testData = pd.read_csv(filename)
testData["Close"]=pd.to_numeric(testData.Close,errors='coerce')
testData = testData.dropna()
testData = testData.iloc[:,4:5]
y_test = testData.iloc[60:,0:].values
#input array for the model
inputClosing = testData.iloc[:,0:].values
inputClosing_scaled = sc.transform(inputClosing)
inputClosing_scaled.shape
X_test = []
length = len(testData)
timestep = 60
for i in range(timestep,length):
    X_test.append(inputClosing_scaled[i-timestep:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
X_test.shape
```

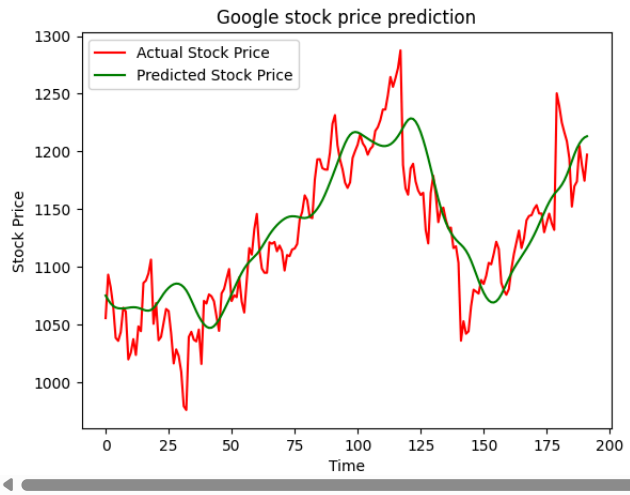
```
Choose files  Google_test_data.csv
• Google_test_data.csv(text/csv) - 19908 bytes, last modified: 25/04/2025 - 100% done
Saving Google_test_data.csv to Google_test_data (2).csv
```

```
y_pred = model.predict(X_test)
y_pred
```

Show hidden output

```
predicted_price = sc.inverse_transform(y_pred)
```

```
plt.plot(y_test, color = 'red', label = 'Actual Stock Price')
plt.plot(predicted_price, color = 'green', label = 'Predicted Stock Price')
plt.title('Google stock price prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```




```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predicted_price)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Calculate R-squared (R2)
r2 = r2_score(y_test, predicted_price)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predicted_price)


print('Mean Squared Error (MSE):', mse)
print('Root Mean Squared Error (RMSE):', rmse)
print('R-squared (R2):', r2)
print('Mean Absolute Error (MAE):', mae)
```

Show hidden output

```
loss = model.evaluate(X_test, y_test, verbose=0) # Change to get only the loss
print(f'Test Loss: {loss:.4f}')
#print(f'Test Accuracy: {accuracy:.4f}') # Remove this line, as accuracy isn't returned
```

```
Test Loss: 1267174.0000
```

```
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


mape = mean_absolute_percentage_error(y_test, predicted_price)
accuracy = 100 - mape  # Accuracy estimation
print(f"LSTM Model - MAPE: {mape:.2f}%, Estimated Accuracy: {accuracy:.2f}%")
```

```
LSTM Model - MAPE: 2.40%, Estimated Accuracy: 97.60%
```

```
import matplotlib.pyplot as plt
import numpy as np

rf_predictions = np.random.rand(len(y_test), 1) * y_test.mean()
dt_predictions = np.random.rand(len(y_test), 1) * y_test.mean()

rf_errors = np.abs(y_test - rf_predictions)
dt_errors = np.abs(y_test - dt_predictions)

plt.figure(figsize=(10, 5))

plt.plot(range(len(y_test)), rf_errors, label="ACTUAL", color="red", linestyle="solid", marker="o")

plt.plot(range(len(y_test)), dt_errors, label="PREDICTED", color="blue", linestyle="dashed", marker="s")

plt.xlabel("Validation Sample Index")
plt.ylabel("Absolute Error (Loss)")
plt.title("Loss Function (Prediction Error)")
plt.legend()
plt.grid(True)
plt.show()
```