In [1]:

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

```
C:\Users\abhij\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarn
ing: A NumPy version >=1.18.5 and <1.25.0 is required for this version of
SciPy (detected version 1.26.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:

```
data = pd.read_csv('GOOGL.csv')
```

In [3]:

```
data_copy = data.copy()

data_copy.dropna(inplace=True)

selected_features = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
data_copy = data_copy[selected_features]

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data_copy)
```

In [4]:

```
df = pd.DataFrame(data)
df.head()
```

Out[4]:

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| **0** | 2004-08-19 | 49.813286 | 51.835709 | 47.800831 | 49.982655 | 49.982655 | 44871300 |
| **1** | 2004-08-20 | 50.316402 | 54.336334 | 50.062355 | 53.952770 | 53.952770 | 22942800 |
| **2** | 2004-08-23 | 55.168217 | 56.528118 | 54.321388 | 54.495735 | 54.495735 | 18342800 |
| **3** | 2004-08-24 | 55.412300 | 55.591629 | 51.591621 | 52.239193 | 52.239193 | 15319700 |
| **4** | 2004-08-25 | 52.284027 | 53.798351 | 51.746044 | 52.802086 | 52.802086 | 9232100 |

In [5]:

```
missing_values = df.isnull().sum()
print("Missing Values:\n", missing_values)
```

```
Missing Values:
 Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

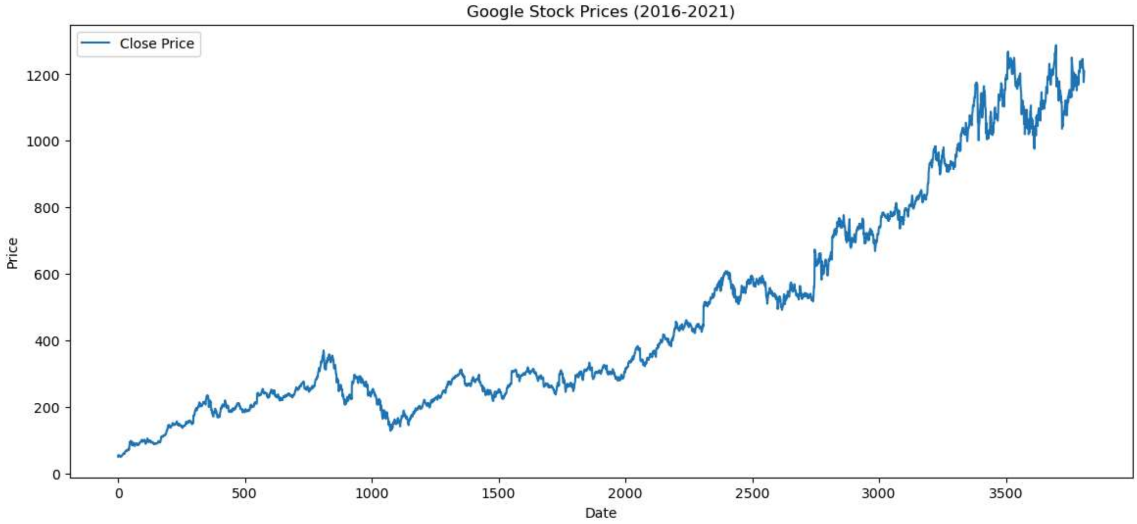
In [6]:

```
print("Dataset shape:", df.shape)
print("Columns:", df.columns)
print("Info:\n", df.info())
print("Summary statistics:\n", df.describe())
```

```
Dataset shape: (3809, 7)
Columns: Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volu
me'], dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3809 entries, 0 to 3808
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       3809 non-null   object 
 1   Open       3809 non-null   float64
 2   High       3809 non-null   float64
 3   Low        3809 non-null   float64
 4   Close      3809 non-null   float64
 5   Adj Close  3809 non-null   float64
 6   Volume     3809 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 208.4+ KB
Info:
 None
Summary statistics:
                Open         High           Low        Close    Adj Close  \
count  3809.000000  3809.000000  3809.000000  3809.000000  3809.000000   
mean    477.021219   481.312940   472.442959   476.979070   476.979070   
std     325.569981   328.160631   323.008258   325.744535   325.744535   
min      49.409801    50.680038    47.800831    49.818268    49.818268   
25%     235.616852   238.615616   233.484848   235.517227   235.517227   
50%     313.823700   316.558472   310.386597   313.290710   313.290710   
75%     703.619995   711.478027   695.719971   704.239990   704.239990   
max    1274.000000  1289.270020  1266.295044  1287.579956  1287.579956   

             Volume  
count  3.809000e+03  
mean   7.181291e+06  
std    8.108893e+06  
min    7.900000e+03  
25%    1.831000e+06  
50%    4.492500e+06  
75%    9.330100e+06  
max    8.254150e+07  
```

In [7]:

```
plt.figure(figsize=(14, 6))
plt.plot(data['Close'], label='Close Price')
plt.title('Google Stock Prices (2016-2021)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```



In [8]:

```
df['Date'] = pd.to_datetime(df['Date'], utc=True)
df.head()
```

Out[8]:

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| **0** | 2004-08-19 00:00:00+00:00 | 49.813286 | 51.835709 | 47.800831 | 49.982655 | 49.982655 | 44871300 |
| **1** | 2004-08-20 00:00:00+00:00 | 50.316402 | 54.336334 | 50.062355 | 53.952770 | 53.952770 | 22942800 |
| **2** | 2004-08-23 00:00:00+00:00 | 55.168217 | 56.528118 | 54.321388 | 54.495735 | 54.495735 | 18342800 |
| **3** | 2004-08-24 00:00:00+00:00 | 55.412300 | 55.591629 | 51.591621 | 52.239193 | 52.239193 | 15319700 |
| **4** | 2004-08-25 00:00:00+00:00 | 52.284027 | 53.798351 | 51.746044 | 52.802086 | 52.802086 | 9232100 |

In [9]:

```
def prepare_data(data, time_steps):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:(i + time_steps)])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)

time_steps = 60

X, y = prepare_data(scaled_data, time_steps)
```

In [10]:

```
split_ratio = 0.8  # Train-test split ratio
split_index = int(split_ratio * len(X))
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]
```

In [11]:

```
model = Sequential([
    LSTM(units=100, return_sequences=True, input_shape=(X_train.shape[1], X
    Dropout(0.2),
    LSTM(units=100, return_sequences=True),
    Dropout(0.2),
    LSTM(units=100),
    Dropout(0.2),
    Dense(units=len(selected_features))
])

# Compile model
model.compile(optimizer='adam', loss='mean_squared_error',metrics=['accurac

# Display model
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm (LSTM)                 (None, 60, 100)           42800     
                                                                 
 dropout (Dropout)           (None, 60, 100)           0         
                                                                 
 lstm_1 (LSTM)               (None, 60, 100)           80400     
                                                                 
 dropout_1 (Dropout)         (None, 60, 100)           0         
                                                                 
 lstm_2 (LSTM)               (None, 100)               80400     
                                                                 
 dropout_2 (Dropout)         (None, 100)               0         
                                                                 
 dense (Dense)               (None, 6)                 606       
                                                                 
=================================================================
Total params: 204206 (797.68 KB)
Trainable params: 204206 (797.68 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

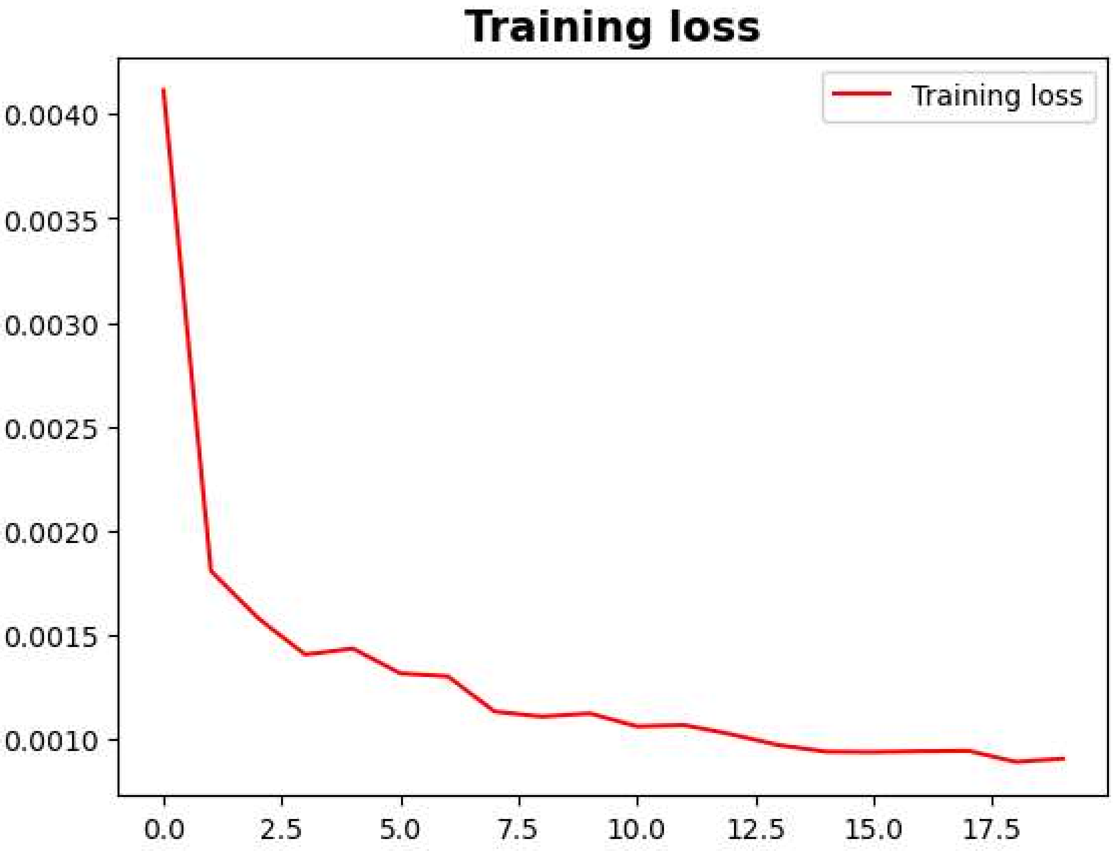
In [12]:
```
# Measure training time
start_time = time.time()

epochs = 20
history = model.fit(X_train, y_train, epochs=epochs, batch_size=32, verbose

training_time = time.time() - start_time
print("Training Time:", training_time, "seconds")
```

```
Epoch 1/20
94/94 [==============================] - 49s 367ms/step - loss: 0.0041 - a
ccuracy: 0.2911
Epoch 2/20
94/94 [==============================] - 32s 337ms/step - loss: 0.0018 - a
ccuracy: 0.3528
Epoch 3/20
94/94 [==============================] - 33s 346ms/step - loss: 0.0016 - a
ccuracy: 0.3545
Epoch 4/20
94/94 [==============================] - 33s 349ms/step - loss: 0.0014 - a
ccuracy: 0.3505
Epoch 5/20
94/94 [==============================] - 33s 347ms/step - loss: 0.0014 - a
ccuracy: 0.3715
Epoch 6/20
94/94 [==============================] - 33s 348ms/step - loss: 0.0013 - a
ccuracy: 0.3631
Epoch 7/20
94/94 [==============================] - 33s 346ms/step - loss: 0.0013 - a
ccuracy: 0.3918
Epoch 8/20
94/94 [==============================] - 32s 339ms/step - loss: 0.0011 - a
ccuracy: 0.3738
Epoch 9/20
94/94 [==============================] - 30s 319ms/step - loss: 0.0011 - a
ccuracy: 0.3778
Epoch 10/20
94/94 [==============================] - 33s 349ms/step - loss: 0.0011 - a
ccuracy: 0.3701
Epoch 11/20
94/94 [==============================] - 33s 350ms/step - loss: 0.0011 - a
ccuracy: 0.3748
Epoch 12/20
94/94 [==============================] - 33s 346ms/step - loss: 0.0011 - a
ccuracy: 0.3888
Epoch 13/20
94/94 [==============================] - 33s 348ms/step - loss: 0.0010 - a
ccuracy: 0.3731
Epoch 14/20
94/94 [==============================] - 33s 351ms/step - loss: 9.7154e-04
- accuracy: 0.3848
Epoch 15/20
94/94 [==============================] - 32s 345ms/step - loss: 9.4064e-04
- accuracy: 0.3948
Epoch 16/20
94/94 [==============================] - 33s 347ms/step - loss: 9.3903e-04
- accuracy: 0.3871
Epoch 17/20
94/94 [==============================] - 33s 347ms/step - loss: 9.4265e-04
- accuracy: 0.4011
Epoch 18/20
94/94 [==============================] - 29s 310ms/step - loss: 9.4495e-04
- accuracy: 0.3968
Epoch 19/20
94/94 [==============================] - 31s 335ms/step - loss: 8.9141e-04
- accuracy: 0.4011
Epoch 20/20
94/94 [==============================] - 32s 341ms/step - loss: 9.0711e-04
```

```
- accuracy: 0.4075
Training Time: 660.7366044521332 seconds
```

In [13]:

```
loss = history.history['loss']

epochs = range(len(loss))

plt.plot(epochs, loss, 'r', label='Training loss')

plt.title('Training loss', size=15, weight='bold')
plt.legend(loc=0)
plt.figure()

plt.show()
#model evaluasi
train_loss = model.evaluate(X_train, y_train, verbose=0)
test_loss = model.evaluate(X_test, y_test, verbose=0)

print(f"Train Loss: {train_loss}")
print(f"Test Loss: {test_loss}")
```

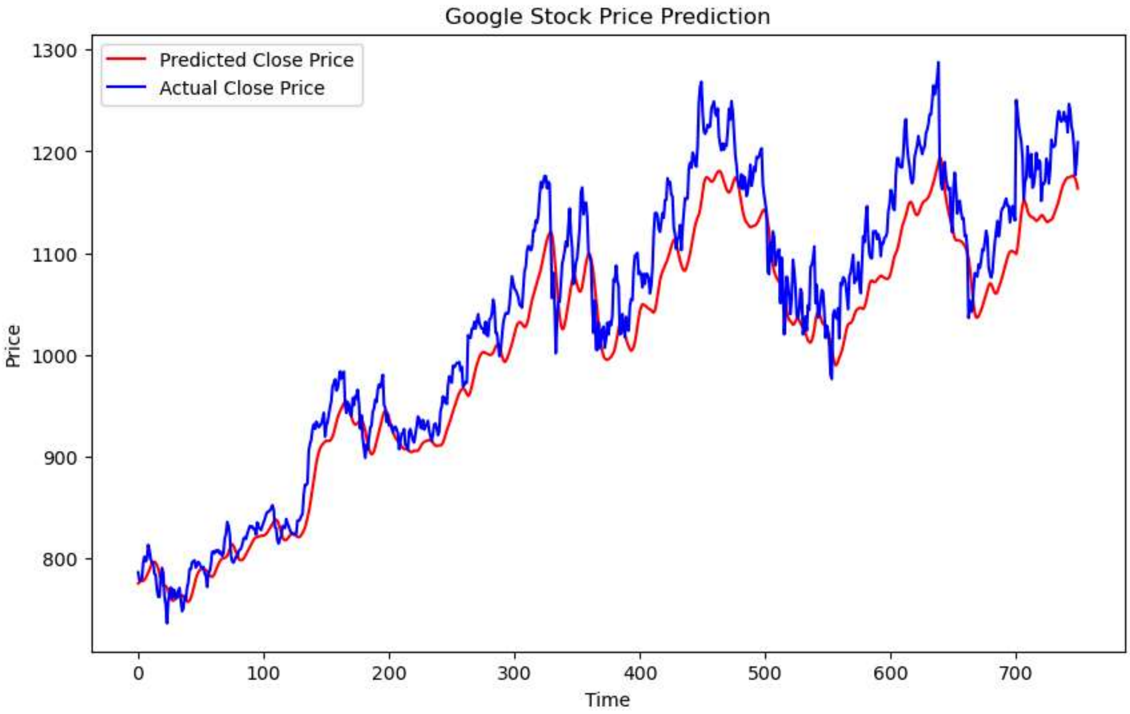


```
<Figure size 640x480 with 0 Axes>

Train Loss: [0.0007493224693462253, 0.47649216651916504]
Test Loss: [0.0009760549874044955, 0.41999998688697815]
```

In [14]:

```
# Measure prediction time
start_time = time.time()

predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)
y_test_inverse = scaler.inverse_transform(y_test)

prediction_time = time.time() - start_time
print("Prediction Time:", prediction_time, "seconds")

# predicted vs actual
plt.figure(figsize=(10, 6))
plt.plot(predictions[:,3], label='Predicted Close Price', color='r')
plt.plot(y_test_inverse[:,3], label='Actual Close Price', color='b')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
24/24 [==============================] - 5s 97ms/step
Prediction Time: 5.656156539916992 seconds
```



In [ ]: