# Gold Price Prediction

Build a time series model to predict the future price of gold, which can be highly beneficial for traders. To achieve this, we will utilize the historical gold price data spanning 10 years (from 2013 to 2023).

## Step 1: Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_percentage_error
import tensorflow as tf
from keras import Model
from keras.layers import Input, Dense, Dropout
from keras.layers import LSTM
```

## Step 2: Reading Dataset

In [2]:
```
df = pd.read_csv('/input/gold-price-10-years-20132023/Gold Price (2013-2023).csv')
```

## Step 3: Dataset Overview

In [3]:
```
df
```

Out[3]:

| | Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|---|
| **0** | 12/30/2022 | 1,826.20 | 1,821.80 | 1,832.40 | 1,819.80 | 107.50K | 0.01% |
| **1** | 12/29/2022 | 1,826.00 | 1,812.30 | 1,827.30 | 1,811.20 | 105.99K | 0.56% |
| **2** | 12/28/2022 | 1,815.80 | 1,822.40 | 1,822.80 | 1,804.20 | 118.08K | -0.40% |
| **3** | 12/27/2022 | 1,823.10 | 1,808.20 | 1,841.90 | 1,808.00 | 159.62K | 0.74% |
| **4** | 12/26/2022 | 1,809.70 | 1,805.80 | 1,811.95 | 1,805.55 | NaN | 0.30% |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **2578** | 01/08/2013 | 1,663.20 | 1,651.50 | 1,662.60 | 1,648.80 | 0.13K | 0.97% |
| **2579** | 01/07/2013 | 1,647.20 | 1,657.30 | 1,663.80 | 1,645.30 | 0.09K | -0.16% |
| **2580** | 01/04/2013 | 1,649.90 | 1,664.40 | 1,664.40 | 1,630.00 | 0.31K | -1.53% |
| **2581** | 01/03/2013 | 1,675.60 | 1,688.00 | 1,689.30 | 1,664.30 | 0.19K | -0.85% |
| **2582** | 01/02/2013 | 1,689.90 | 1,675.80 | 1,695.00 | 1,672.10 | 0.06K | 0.78% |

2583 rows × 7 columns

The dataset comprises daily gold price information, encompassing the daily Open, High, and Low prices, as well as the final price of each day (Price), alongside transaction volumes and daily price changes.

**Dataset Basic Information:**

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2583 entries, 0 to 2582
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Date      2583 non-null   object
 1   Price     2583 non-null   object
 2   Open      2583 non-null   object
 3   High      2583 non-null   object
 4   Low       2583 non-null   object
 5   Vol.      2578 non-null   object
 6   Change %  2583 non-null   object
dtypes: object(7)
memory usage: 141.4+ KB
```

**All variables are stored as object.**

# Step 4: Data Preparation

## Step 4.1: Feature Subset Selection

As we won't be utilizing the 'Vol.' and 'Change %' features to predict the 'Price,' we will eliminate these two features from the analysis.

In [5]:
```
df.drop(['Vol.', 'Change %'], axis=1, inplace=True)
```

## Step 4.2: Transforming Data

The 'Date' feature is currently stored as an object in the data frame. To improve the speed of calculations, we will convert its data type to datetime and subsequently sort this feature in ascending order.

In [6]:
```
df['Date'] = pd.to_datetime(df['Date'])
df.sort_values(by='Date', ascending=True, inplace=True)
df.reset_index(drop=True, inplace=True)
```

The ' , ' sign is redundant in the dataset. Initially, we will remove it from the entire dataset and subsequently convert the data type of the numerical variables to float.

In [7]:
```
NumCols = df.columns.drop(['Date'])
df[NumCols] = df[NumCols].replace({',': ''}, regex=True)
df[NumCols] = df[NumCols].astype('float64')
```

**Result:**

In [8]:
```
df.head()
```

Out[8]:

| | Date | Price | Open | High | Low |
|---|---|---|---|---|---|
| **0** | 2013-01-02 | 1689.9 | 1675.8 | 1695.0 | 1672.1 |
| **1** | 2013-01-03 | 1675.6 | 1688.0 | 1689.3 | 1664.3 |
| **2** | 2013-01-04 | 1649.9 | 1664.4 | 1664.4 | 1630.0 |
| **3** | 2013-01-07 | 1647.2 | 1657.3 | 1663.8 | 1645.3 |
| **4** | 2013-01-08 | 1663.2 | 1651.5 | 1662.6 | 1648.8 |

## Step 4.3: Checking Duplicates

**There are no duplicate samples in Date feature:**

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

## Step 4.4: Checking Missing Values

**There are no missing values in the dataset:**

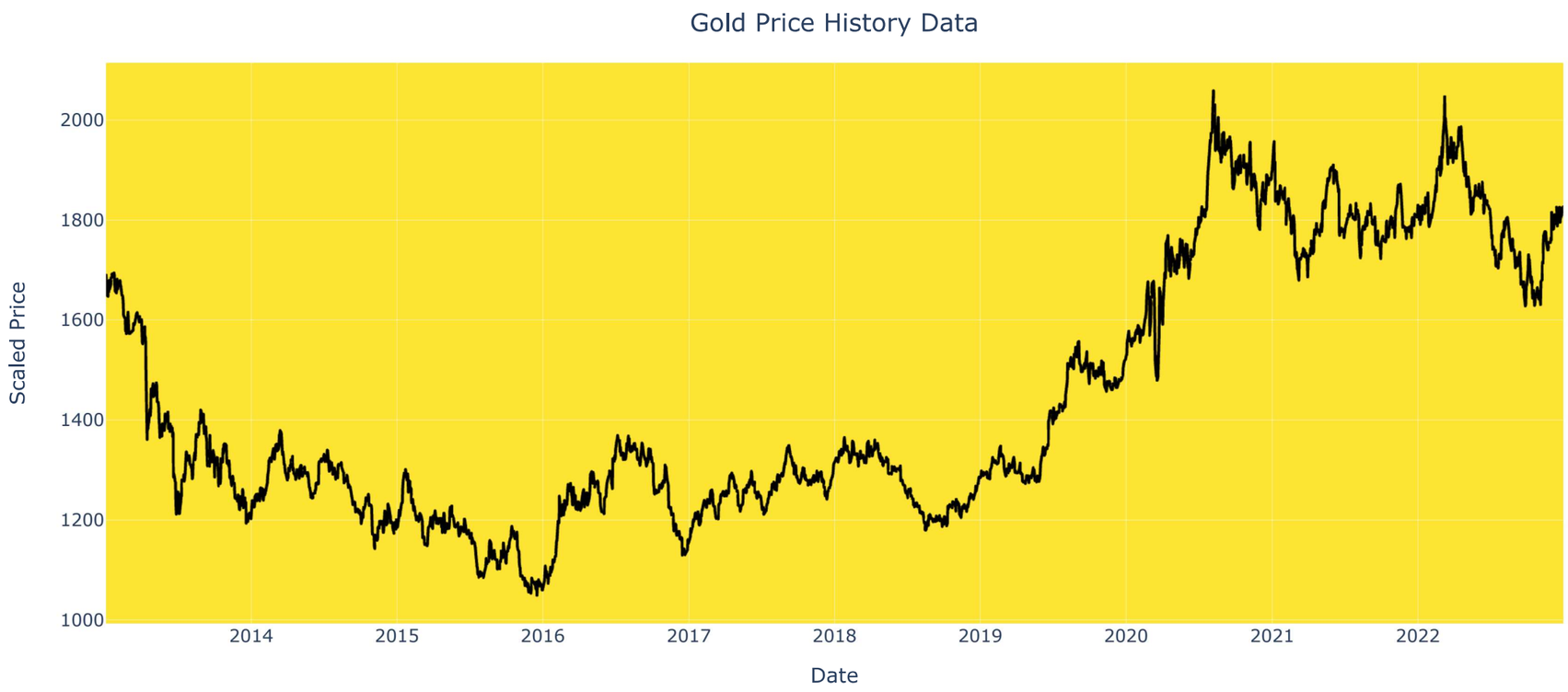
In [10]:
```
df.isnull().sum().sum()
```

Out[10]:
```
0
```

# Step 5: Visualizing Gold Price History Data

**Interactive Gold Price Chart:**

In [11]:
```
fig = px.line(y=df.Price, x=df.Date)
fig.update_traces(line_color='black')
fig.update_layout(xaxis_title="Date",
                  yaxis_title="Scaled Price",
                  title={'text': "Gold Price History Data", 'y':0.95, 'x':0.5, 'xanchor':'center', 'yanchor':'top'},
                  plot_bgcolor='rgba(255,223,0,0.8)')
```




# Step 6: Splitting Data to Training & Test Sets

Given the nature of time series data, training on future data is not feasible. Therefore, random division of time series data is inappropriate. In time series splitting, the testing set always occurs later than the training set. As a practice, we designate the last year for testing and allocate all preceding data for training.

In [12]:
```
test_size = df[df.Date.dt.year==2022].shape[0]
test_size
```

Out[12]:
```
260
```

**Gold Price Training and Test Sets Plot:**

In [13]:
```
plt.figure(figsize=(15, 6), dpi=150)
plt.rcParams['axes.facecolor'] = 'yellow'
plt.rc('axes',edgecolor='white')
plt.plot(df.Date[:-test_size], df.Price[:-test_size], color='black', lw=2)
plt.plot(df.Date[-test_size:], df.Price[-test_size:], color='blue', lw=2)
plt.title('Gold Price Training and Test Sets', fontsize=15)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price', fontsize=12)
plt.legend(['Training set', 'Test set'], loc='upper left', prop={'size': 15})
plt.grid(color='white')
plt.show()
```


## Step 7: Data Scaling

Since our objective is to predict the 'Price' solely based on its historical data, we utilize MinMaxScaler to scale the 'Price.' This helps in avoiding intensive computations.

In [14]:
```
scaler = MinMaxScaler()
scaler.fit(df.Price.values.reshape(-1,1))
```

Out[14]:
```
MinMaxScaler()
```

# Step 8: Restructure Data & Create Sliding Window

The process of utilizing preceding time steps to forecast subsequent time steps is referred to as the sliding window method. This technique transforms time series data into a supervised learning format. By employing past time steps as input variables and the following time step as the output variable, we create this framework. The count of previous time steps utilized is termed the 'window width.' In this context, we set the window width to 60. Consequently, X_train and X_test will consist of nested lists containing sequences of 60 timestamp prices. Similarly, y_train and y_test will represent lists of gold prices, with each corresponding to the subsequent day's gold price for the lists in X_train and X_test, respectively.

In [15]:
```
window_size = 60
```

**Training Set:**

In [16]:
```
train_data = df.Price[:-test_size]
train_data = scaler.transform(train_data.values.reshape(-1,1))
```

In [17]:
```
X_train = []
y_train = []

for i in range(window_size, len(train_data)):
    X_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
```

**Test Set:**

In [18]:
```
test_data = df.Price[-test_size-60:]
test_data = scaler.transform(test_data.values.reshape(-1,1))
```

In [19]:
```
X_test = []
y_test = []

for i in range(window_size, len(test_data)):
    X_test.append(test_data[i-60:i, 0])
    y_test.append(test_data[i, 0])
```

# Step 9: Converting Data to Numpy Arrays

Now, X_train and X_test exist as nested lists (two-dimensional lists), while y_train is a one-dimensional list. It is essential to convert them into NumPy arrays with a higher dimension, as this format is compatible with TensorFlow for training the neural network.

In [20]:
```
X_train = np.array(X_train)
X_test  = np.array(X_test)
y_train = np.array(y_train)
y_test  = np.array(y_test)
```

In [21]:
```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test  = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
y_train = np.reshape(y_train, (-1,1))
y_test  = np.reshape(y_test, (-1,1))
```

In [22]:
```
print('X_train Shape: ', X_train.shape)
print('y_train Shape: ', y_train.shape)
print('X_test Shape:  ', X_test.shape)
print('y_test Shape:  ', y_test.shape)
```

```
X_train Shape:  (2263, 60, 1)
y_train Shape:  (2263, 1)
X_test Shape:   (260, 60, 1)
y_test Shape:   (260, 1)
```

# Step 10: Creating an LSTM Network

We construct an LSTM network, a type of Recurrent Neural Network specifically designed to address the vanishing gradient problem.

**Model Definition:**

In [23]:
```
def define_model():
    input1 = Input(shape=(window_size,1))
    x = LSTM(units = 64, return_sequences=True)(input1)
    x = Dropout(0.2)(x)
    x = LSTM(units = 64, return_sequences=True)(x)
    x = Dropout(0.2)(x)
    x = LSTM(units = 64)(x)
    x = Dropout(0.2)(x)
    x = Dense(32, activation='softmax')(x)
    dnn_output = Dense(1)(x)

    model = Model(inputs=input1, outputs=[dnn_output])
    model.compile(loss='mean_squared_error', optimizer='Nadam')
    model.summary()

    return model
```

**Model Training:**

In [24]:
```
model = define_model()
history = model.fit(X_train, y_train, epochs=150, batch_size=32, validation_split=0.1, verbose=1)
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 60, 1)]           0
_________________________________________________________________
lstm (LSTM)                  (None, 60, 64)            16896
_________________________________________________________________
dropout (Dropout)            (None, 60, 64)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 60, 64)            33024
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 64)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                33024
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 32)                2080
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 85,057
Trainable params: 85,057
Non-trainable params: 0
_________________________________________________________________
Epoch 1/150
64/64 [==============================] - 8s 26ms/step - loss: 0.0391 - val_loss: 0.0755
Epoch 2/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0112 - val_loss: 0.0325
Epoch 3/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0068 - val_loss: 0.0131
Epoch 4/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0047 - val_loss: 0.0052
Epoch 5/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0033 - val_loss: 0.0024
Epoch 6/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0025 - val_loss: 0.0023
Epoch 7/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0020 - val_loss: 0.0032
Epoch 8/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0017 - val_loss: 0.0040
Epoch 9/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0016 - val_loss: 0.0032
Epoch 10/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0014 - val_loss: 0.0064
Epoch 11/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0014 - val_loss: 0.0088
Epoch 12/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0013 - val_loss: 0.0023
Epoch 13/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0012 - val_loss: 0.0053
Epoch 14/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0011 - val_loss: 0.0041
Epoch 15/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0011 - val_loss: 0.0028
Epoch 16/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0012 - val_loss: 0.0018
Epoch 17/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0010 - val_loss: 0.0025
```

```
Epoch 18/150
64/64 [==============================] - 1s 12ms/step - loss: 0.0010 - val_loss: 0.0015
Epoch 19/150
64/64 [==============================] - 1s 12ms/step - loss: 9.5075e-04 - val_loss: 0.0014
Epoch 20/150
64/64 [==============================] - 1s 12ms/step - loss: 9.3642e-04 - val_loss: 0.0018
Epoch 21/150
64/64 [==============================] - 1s 12ms/step - loss: 9.4137e-04 - val_loss: 0.0011
Epoch 22/150
64/64 [==============================] - 1s 12ms/step - loss: 8.3341e-04 - val_loss: 0.0023
Epoch 23/150
64/64 [==============================] - 1s 12ms/step - loss: 8.1240e-04 - val_loss: 0.0023
Epoch 24/150
64/64 [==============================] - 1s 13ms/step - loss: 8.7070e-04 - val_loss: 0.0021
Epoch 25/150
64/64 [==============================] - 1s 13ms/step - loss: 7.7523e-04 - val_loss: 0.0054
Epoch 26/150
64/64 [==============================] - 1s 12ms/step - loss: 7.1175e-04 - val_loss: 0.0024
Epoch 27/150
64/64 [==============================] - 1s 12ms/step - loss: 7.4981e-04 - val_loss: 0.0051
Epoch 28/150
64/64 [==============================] - 1s 12ms/step - loss: 7.3729e-04 - val_loss: 0.0035
Epoch 29/150
64/64 [==============================] - 1s 13ms/step - loss: 6.9453e-04 - val_loss: 0.0011
Epoch 30/150
64/64 [==============================] - 1s 12ms/step - loss: 6.7924e-04 - val_loss: 0.0037
Epoch 31/150
64/64 [==============================] - 1s 12ms/step - loss: 6.3584e-04 - val_loss: 0.0012
Epoch 32/150
64/64 [==============================] - 1s 12ms/step - loss: 6.1108e-04 - val_loss: 9.3361e-04
Epoch 33/150
64/64 [==============================] - 1s 12ms/step - loss: 6.2948e-04 - val_loss: 0.0011
Epoch 34/150
64/64 [==============================] - 1s 12ms/step - loss: 6.1625e-04 - val_loss: 0.0018
Epoch 35/150
64/64 [==============================] - 1s 12ms/step - loss: 6.5597e-04 - val_loss: 0.0033
Epoch 36/150
64/64 [==============================] - 1s 12ms/step - loss: 5.7656e-04 - val_loss: 0.0011
Epoch 37/150
64/64 [==============================] - 1s 12ms/step - loss: 6.3756e-04 - val_loss: 0.0015
Epoch 38/150
64/64 [==============================] - 1s 12ms/step - loss: 5.6003e-04 - val_loss: 0.0036
Epoch 39/150
64/64 [==============================] - 1s 12ms/step - loss: 5.7108e-04 - val_loss: 0.0015
Epoch 40/150
64/64 [==============================] - 1s 13ms/step - loss: 6.0612e-04 - val_loss: 0.0011
Epoch 41/150
64/64 [==============================] - 1s 12ms/step - loss: 5.4824e-04 - val_loss: 7.2340e-04
Epoch 42/150
64/64 [==============================] - 1s 12ms/step - loss: 5.6964e-04 - val_loss: 0.0010
Epoch 43/150
64/64 [==============================] - 1s 12ms/step - loss: 5.4637e-04 - val_loss: 8.6371e-04
Epoch 44/150
64/64 [==============================] - 1s 12ms/step - loss: 5.3382e-04 - val_loss: 0.0011
Epoch 45/150
64/64 [==============================] - 1s 12ms/step - loss: 5.3510e-04 - val_loss: 0.0049
Epoch 46/150
64/64 [==============================] - 1s 12ms/step - loss: 5.5819e-04 - val_loss: 7.0056e-04
Epoch 47/150
64/64 [==============================] - 1s 12ms/step - loss: 5.1448e-04 - val_loss: 8.7503e-04
```

```
Epoch 48/150
64/64 [==============================] - 1s 12ms/step - loss: 4.9898e-04 - val_loss: 0.0022
Epoch 49/150
64/64 [==============================] - 1s 12ms/step - loss: 4.8896e-04 - val_loss: 0.0012
Epoch 50/150
64/64 [==============================] - 1s 12ms/step - loss: 5.0690e-04 - val_loss: 9.3031e-04
Epoch 51/150
64/64 [==============================] - 1s 12ms/step - loss: 5.0496e-04 - val_loss: 0.0015
Epoch 52/150
64/64 [==============================] - 1s 12ms/step - loss: 4.7810e-04 - val_loss: 0.0047
Epoch 53/150
64/64 [==============================] - 1s 12ms/step - loss: 4.9425e-04 - val_loss: 9.3235e-04
Epoch 54/150
64/64 [==============================] - 1s 12ms/step - loss: 4.4064e-04 - val_loss: 0.0012
Epoch 55/150
64/64 [==============================] - 1s 12ms/step - loss: 4.7808e-04 - val_loss: 0.0012
Epoch 56/150
64/64 [==============================] - 1s 12ms/step - loss: 4.6378e-04 - val_loss: 0.0013
Epoch 57/150
64/64 [==============================] - 1s 12ms/step - loss: 4.7512e-04 - val_loss: 0.0015
Epoch 58/150
64/64 [==============================] - 1s 12ms/step - loss: 4.3685e-04 - val_loss: 0.0014
Epoch 59/150
64/64 [==============================] - 1s 12ms/step - loss: 4.3686e-04 - val_loss: 0.0021
Epoch 60/150
64/64 [==============================] - 1s 12ms/step - loss: 4.2073e-04 - val_loss: 9.9282e-04
Epoch 61/150
64/64 [==============================] - 1s 12ms/step - loss: 4.3980e-04 - val_loss: 0.0014
Epoch 62/150
64/64 [==============================] - 1s 12ms/step - loss: 4.1241e-04 - val_loss: 0.0018
Epoch 63/150
64/64 [==============================] - 1s 12ms/step - loss: 4.3011e-04 - val_loss: 9.1947e-04
Epoch 64/150
64/64 [==============================] - 1s 12ms/step - loss: 4.2963e-04 - val_loss: 0.0011
Epoch 65/150
64/64 [==============================] - 1s 13ms/step - loss: 3.9337e-04 - val_loss: 0.0014
Epoch 66/150
64/64 [==============================] - 1s 14ms/step - loss: 4.0252e-04 - val_loss: 0.0016
Epoch 67/150
64/64 [==============================] - 1s 12ms/step - loss: 4.0917e-04 - val_loss: 0.0047
Epoch 68/150
64/64 [==============================] - 1s 12ms/step - loss: 4.0192e-04 - val_loss: 0.0028
Epoch 69/150
64/64 [==============================] - 1s 12ms/step - loss: 3.8767e-04 - val_loss: 0.0020
Epoch 70/150
64/64 [==============================] - 1s 12ms/step - loss: 3.8085e-04 - val_loss: 0.0032
Epoch 71/150
64/64 [==============================] - 1s 12ms/step - loss: 4.0028e-04 - val_loss: 0.0024
Epoch 72/150
64/64 [==============================] - 1s 13ms/step - loss: 4.0167e-04 - val_loss: 0.0046
Epoch 73/150
64/64 [==============================] - 1s 12ms/step - loss: 3.8072e-04 - val_loss: 0.0031
Epoch 74/150
64/64 [==============================] - 1s 12ms/step - loss: 3.6165e-04 - val_loss: 0.0026
Epoch 75/150
64/64 [==============================] - 1s 12ms/step - loss: 3.9340e-04 - val_loss: 0.0037
Epoch 76/150
64/64 [==============================] - 1s 12ms/step - loss: 3.9610e-04 - val_loss: 0.0021
Epoch 77/150
64/64 [==============================] - 1s 12ms/step - loss: 3.9367e-04 - val_loss: 0.0029
```

```
Epoch 78/150
64/64 [==============================] - 1s 12ms/step - loss: 3.8513e-04 - val_loss: 0.0016
Epoch 79/150
64/64 [==============================] - 1s 13ms/step - loss: 3.7551e-04 - val_loss: 0.0019
Epoch 80/150
64/64 [==============================] - 1s 12ms/step - loss: 3.6345e-04 - val_loss: 0.0015
Epoch 81/150
64/64 [==============================] - 1s 12ms/step - loss: 3.4427e-04 - val_loss: 0.0013
Epoch 82/150
64/64 [==============================] - 1s 12ms/step - loss: 3.5126e-04 - val_loss: 0.0025
Epoch 83/150
64/64 [==============================] - 1s 12ms/step - loss: 3.6290e-04 - val_loss: 0.0045
Epoch 84/150
64/64 [==============================] - 1s 12ms/step - loss: 3.6494e-04 - val_loss: 0.0037
Epoch 85/150
64/64 [==============================] - 1s 12ms/step - loss: 3.9884e-04 - val_loss: 0.0035
Epoch 86/150
64/64 [==============================] - 1s 12ms/step - loss: 3.4125e-04 - val_loss: 0.0022
Epoch 87/150
64/64 [==============================] - 1s 12ms/step - loss: 3.8033e-04 - val_loss: 0.0022
Epoch 88/150
64/64 [==============================] - 1s 12ms/step - loss: 3.4742e-04 - val_loss: 0.0022
Epoch 89/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3082e-04 - val_loss: 0.0023
Epoch 90/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3499e-04 - val_loss: 0.0014
Epoch 91/150
64/64 [==============================] - 1s 12ms/step - loss: 3.5043e-04 - val_loss: 0.0023
Epoch 92/150
64/64 [==============================] - 1s 12ms/step - loss: 3.5247e-04 - val_loss: 0.0015
Epoch 93/150
64/64 [==============================] - 1s 12ms/step - loss: 3.4368e-04 - val_loss: 0.0021
Epoch 94/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2395e-04 - val_loss: 0.0036
Epoch 95/150
64/64 [==============================] - 1s 12ms/step - loss: 3.5211e-04 - val_loss: 0.0020
Epoch 96/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3650e-04 - val_loss: 0.0011
Epoch 97/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2978e-04 - val_loss: 0.0015
Epoch 98/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2550e-04 - val_loss: 0.0015
Epoch 99/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2612e-04 - val_loss: 0.0023
Epoch 100/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3683e-04 - val_loss: 0.0012
Epoch 101/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3698e-04 - val_loss: 0.0015
Epoch 102/150
64/64 [==============================] - 1s 12ms/step - loss: 3.4038e-04 - val_loss: 0.0011
Epoch 103/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3410e-04 - val_loss: 0.0013
Epoch 104/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3673e-04 - val_loss: 0.0023
Epoch 105/150
64/64 [==============================] - 1s 12ms/step - loss: 3.4207e-04 - val_loss: 0.0012
Epoch 106/150
64/64 [==============================] - 1s 13ms/step - loss: 3.2968e-04 - val_loss: 0.0016
Epoch 107/150
64/64 [==============================] - 1s 13ms/step - loss: 3.1465e-04 - val_loss: 0.0013
```

```
Epoch 108/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2096e-04 - val_loss: 0.0026
Epoch 109/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3033e-04 - val_loss: 0.0022
Epoch 110/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3751e-04 - val_loss: 0.0017
Epoch 111/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2052e-04 - val_loss: 0.0017
Epoch 112/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0302e-04 - val_loss: 0.0012
Epoch 113/150
64/64 [==============================] - 1s 12ms/step - loss: 3.1094e-04 - val_loss: 0.0017
Epoch 114/150
64/64 [==============================] - 1s 12ms/step - loss: 3.3542e-04 - val_loss: 0.0013
Epoch 115/150
64/64 [==============================] - 1s 12ms/step - loss: 3.4713e-04 - val_loss: 0.0020
Epoch 116/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0626e-04 - val_loss: 0.0020
Epoch 117/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2971e-04 - val_loss: 0.0015
Epoch 118/150
64/64 [==============================] - 1s 13ms/step - loss: 2.9782e-04 - val_loss: 0.0015
Epoch 119/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2105e-04 - val_loss: 0.0013
Epoch 120/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2170e-04 - val_loss: 0.0021
Epoch 121/150
64/64 [==============================] - 1s 12ms/step - loss: 3.1594e-04 - val_loss: 0.0024
Epoch 122/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0467e-04 - val_loss: 0.0021
Epoch 123/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2585e-04 - val_loss: 0.0014
Epoch 124/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2100e-04 - val_loss: 0.0023
Epoch 125/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0119e-04 - val_loss: 0.0031
Epoch 126/150
64/64 [==============================] - 1s 12ms/step - loss: 3.1427e-04 - val_loss: 0.0024
Epoch 127/150
64/64 [==============================] - 1s 12ms/step - loss: 3.1565e-04 - val_loss: 0.0017
Epoch 128/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0318e-04 - val_loss: 0.0022
Epoch 129/150
64/64 [==============================] - 1s 12ms/step - loss: 3.1343e-04 - val_loss: 0.0043
Epoch 130/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0644e-04 - val_loss: 0.0025
Epoch 131/150
64/64 [==============================] - 1s 12ms/step - loss: 3.1867e-04 - val_loss: 0.0017
Epoch 132/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0957e-04 - val_loss: 0.0028
Epoch 133/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2224e-04 - val_loss: 0.0028
Epoch 134/150
64/64 [==============================] - 1s 12ms/step - loss: 3.1426e-04 - val_loss: 0.0029
Epoch 135/150
64/64 [==============================] - 1s 12ms/step - loss: 2.9686e-04 - val_loss: 0.0015
Epoch 136/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2061e-04 - val_loss: 0.0019
Epoch 137/150
64/64 [==============================] - 1s 12ms/step - loss: 2.9586e-04 - val_loss: 0.0017
```

```
Epoch 138/150
64/64 [==============================] - 1s 12ms/step - loss: 2.8031e-04 - val_loss: 0.0020
Epoch 139/150
64/64 [==============================] - 1s 12ms/step - loss: 2.9590e-04 - val_loss: 0.0021
Epoch 140/150
64/64 [==============================] - 1s 12ms/step - loss: 3.2200e-04 - val_loss: 0.0040
Epoch 141/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0347e-04 - val_loss: 0.0019
Epoch 142/150
64/64 [==============================] - 1s 12ms/step - loss: 2.9579e-04 - val_loss: 0.0018
Epoch 143/150
64/64 [==============================] - 1s 12ms/step - loss: 2.9383e-04 - val_loss: 0.0019
Epoch 144/150
64/64 [==============================] - 1s 12ms/step - loss: 3.0219e-04 - val_loss: 0.0025
Epoch 145/150
64/64 [==============================] - 1s 12ms/step - loss: 2.9513e-04 - val_loss: 0.0026
Epoch 146/150
64/64 [==============================] - 1s 12ms/step - loss: 2.8991e-04 - val_loss: 0.0031
Epoch 147/150
64/64 [==============================] - 1s 13ms/step - loss: 2.8320e-04 - val_loss: 0.0023
Epoch 148/150
64/64 [==============================] - 1s 13ms/step - loss: 2.8063e-04 - val_loss: 0.0029
Epoch 149/150
64/64 [==============================] - 1s 12ms/step - loss: 2.9779e-04 - val_loss: 0.0024
Epoch 150/150
64/64 [==============================] - 1s 12ms/step - loss: 2.8182e-04 - val_loss: 0.0017
```

# Step 11: Model Evaluation

**Evaluate the time series forecast using MAPE (Mean Absolute Percentage Error) metric:**

In [25]:
```
result = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
```

```
9/9 [==============================] - 0s 4ms/step - loss: 0.0013
```

In [26]:
```
MAPE = mean_absolute_percentage_error(y_test, y_pred)
Accuracy = 1 - MAPE
```

In [27]:
```
print("Test Loss:", result)
print("Test MAPE:", MAPE)
print("Test Accuracy:", Accuracy)
```

```
Test Loss: 0.0013038208708167076
Test MAPE: 0.03934129384696483
Test Accuracy: 0.9606587061530352
```

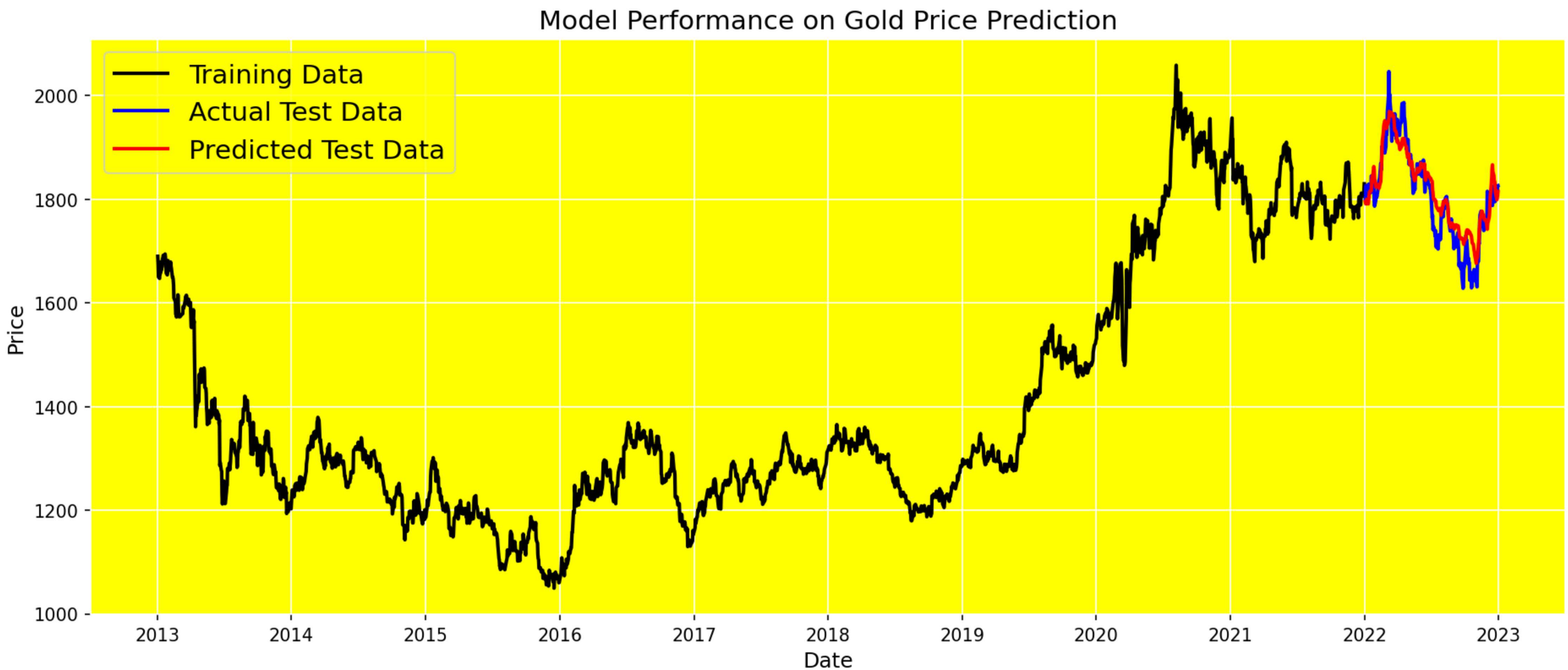
# Step 12: Visualizing Results

Rescaling the predicted and actual Price values to their original scale:

In [28]:
```
y_test_true = scaler.inverse_transform(y_test)
y_test_pred = scaler.inverse_transform(y_pred)
```

Evaluating the proximity of the model's predicted prices to the actual prices:

In [29]:
```
plt.figure(figsize=(15, 6), dpi=150)
plt.rcParams['axes.facecolor'] = 'yellow'
plt.rc('axes',edgecolor='white')
plt.plot(df['Date'].iloc[:-test_size], scaler.inverse_transform(train_data), color='black', lw=2)
plt.plot(df['Date'].iloc[-test_size:], y_test_true, color='blue', lw=2)
plt.plot(df['Date'].iloc[-test_size:], y_test_pred, color='red', lw=2)
plt.title('Model Performance on Gold Price Prediction', fontsize=15)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price', fontsize=12)
plt.legend(['Training Data', 'Actual Test Data', 'Predicted Test Data'], loc='upper left', prop={'size': 15})
plt.grid(color='white')
plt.show()
```




## Conclusion:

Upon observation, the price predicted by the LSTM model closely aligns with the actual prices. This alignment is further validated by the obtained metrics on the test data:

Loss: 0.001 Accuracy (1-MAPE): 96%