Neural Networks for stocks

Import Historical Prices

- Apple (AAPL) - In the tech sector, an S&P 500 member
- BioNTech (BNTX) - In the pharma sector, **not** an S&P 500 member
- Disney (DIS) - In the entertainment sector, an S&P 500 member
- Royal Bank of Canada (RY) - In the banking sector, **not** an S&P 500 Member

Importing the historical prices from yfinance built in Google Colab

```
In [ ]: import yfinance as yf
```

```
In [ ]: apple = yf.download("AAPL", start="2018-06-02", end="2025-06-02")
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  1 of 1 completed
```

```
In [ ]: BioNTech = yf.download("BNTX", start="2018-06-02", end="2025-06-02")
```

```
[*********************100%***********************]  1 of 1 completed
```

```
In [ ]: Disney = yf.download("DIS", start="2018-06-02", end="2025-06-02")
```

```
[*********************100%***********************]  1 of 1 completed
```

```
In [ ]: Bank_of_Canada = yf.download("RY", start="2018-06-02", end="2025-06-02")
```

```
[*********************100%***********************]  1 of 1 completed
```

Creating a Code Which Accurately Predicts the Historical Stock Prices for AAPL

```
In [ ]: 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
```

```
In [ ]: data = apple[['Close']].copy()
```

```
In [ ]: data.sample()
```

Out[ ]:

Price	Close
Ticker	AAPL
Date	
2023-05-17	170.978119

In [ ]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)
```

In [ ]:

```
X = []
y = []

sequence_length = 60

for i in range(sequence_length, len(scaled_data)):
    X.append(scaled_data[i-sequence_length:i, 0])
    y.append(scaled_data[i, 0])

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

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

In [ ]:

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X.shape[1], 1)))
model.add(LSTM(50))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

In [ ]:

```
model.fit(X, y, epochs=10, batch_size=32)
```

```
Epoch 1/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 8s 59ms/step - loss: 0.0656
Epoch 2/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 6s 78ms/step - loss: 9.3593e-04
Epoch 3/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - loss: 9.2555e-04
Epoch 4/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 6s 79ms/step - loss: 8.7510e-04
Epoch 5/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 5s 82ms/step - loss: 7.8854e-04
Epoch 6/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 57ms/step - loss: 8.1374e-04
Epoch 7/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - loss: 8.1234e-04
Epoch 8/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 85ms/step - loss: 9.1753e-04
Epoch 9/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - loss: 7.1906e-04
Epoch 10/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 5s 60ms/step - loss: 6.9848e-04
```

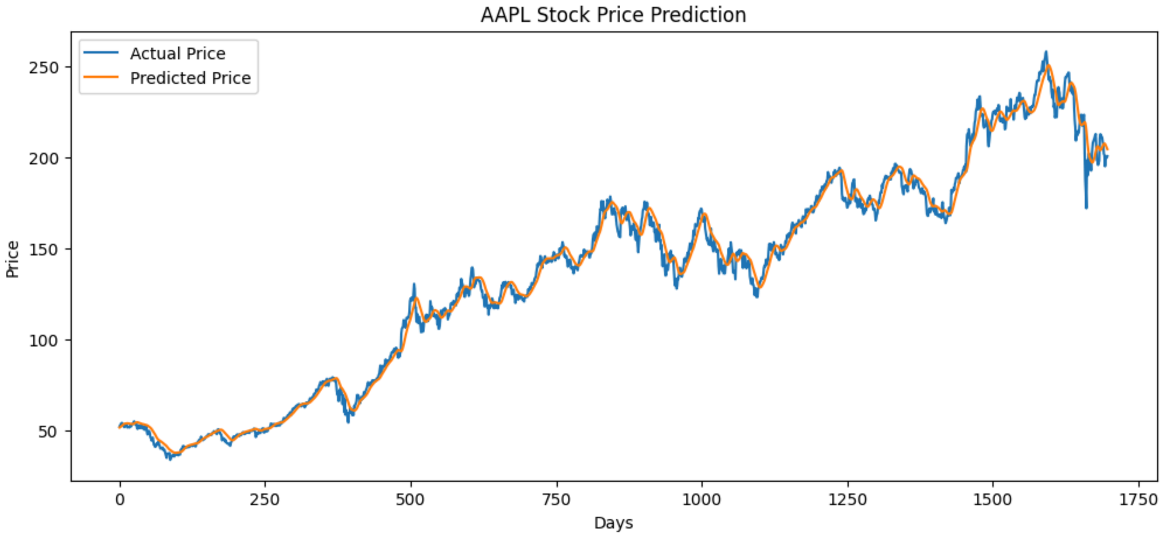
Out[ ]:

```
<keras.src.callbacks.history.History at 0x7c479e2dc950>
```

In [ ]:

```
predictions = model.predict(X)
predictions = scaler.inverse_transform(predictions)
real_prices = scaler.inverse_transform(y.reshape(-1, 1))

plt.figure(figsize=(12, 5))
plt.plot(real_prices, label='Actual Price')
plt.plot(predictions, label='Predicted Price')
plt.title('AAPL Stock Price Prediction')
plt.xlabel('Days')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step
```



The figure shows that the model accurately predicts the daily stock prices for Apple (AAPL). It struggles to capture harsh changes in the prices of the stock when it changes significantly in a short period.

Forecasting Future Prices Based on Historical Prices for AAPL

In [ ]:

```
data = apple[['Close']].copy()
```

In [ ]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)
```

In [ ]:

```
def create_sequences_multi_output(data, seq_len=1260, forecast_horizon=25
    X, y = [], []
    for i in range(seq_len, len(data) - forecast_horizon + 1):
        X.append(data[i-seq_len:i, 0])
        y.append(data[i:i+forecast_horizon, 0])
    return np.array(X), np.array(y)

X_all, y_all = create_sequences_multi_output(scaled_data, 1260, 252)

X_all = X_all.reshape((X_all.shape[0], X_all.shape[1], 1))
```

In [ ]:

```
split = int(len(X_all) * 0.9)
X_train, X_test = X_all[:split], X_all[split:]
y_train, y_test = y_all[:split], y_all[split:]
```

In [ ]:

```
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(X_train.shape[1],
model.add(LSTM(100))
model.add(Dense(252))  # Output: next 252 days
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

In [40]:

```
model.fit(X_train, y_train, epochs=40, batch_size=64)
```

```
Epoch 1/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 3s/step - loss: 0.0073
Epoch 2/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0072
Epoch 3/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0071
Epoch 4/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0071
Epoch 5/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 3s/step - loss: 0.0072
Epoch 6/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 4s/step - loss: 0.0072
Epoch 7/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 4s/step - loss: 0.0070
Epoch 8/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0073
Epoch 9/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0072
Epoch 10/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0071
Epoch 11/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 22s 4s/step - loss: 0.0071
Epoch 12/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0069
Epoch 13/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 3s/step - loss: 0.0072
Epoch 14/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 16s 3s/step - loss: 0.0072
Epoch 15/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0071
Epoch 16/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0072
Epoch 17/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 3s/step - loss: 0.0071
Epoch 18/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 4s/step - loss: 0.0070
Epoch 19/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 3s/step - loss: 0.0072
Epoch 20/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0071
Epoch 21/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 22s 4s/step - loss: 0.0071
Epoch 22/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 19s 4s/step - loss: 0.0072
Epoch 23/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0071
Epoch 24/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 16s 3s/step - loss: 0.0072
Epoch 25/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0072
Epoch 26/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 4s/step - loss: 0.0075
Epoch 27/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 19s 4s/step - loss: 0.0071
Epoch 28/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 4s/step - loss: 0.0072
Epoch 29/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0072
Epoch 30/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 22s 4s/step - loss: 0.0070
```

```
Epoch 31/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0071
Epoch 32/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0071
Epoch 33/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 3s/step - loss: 0.0073
Epoch 34/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0072
Epoch 35/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0072
Epoch 36/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 4s/step - loss: 0.0072
Epoch 37/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0071
Epoch 38/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 22s 4s/step - loss: 0.0073
Epoch 39/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - loss: 0.0070
Epoch 40/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - loss: 0.0070
```

```
Out[40]: <keras.src.callbacks.history.History at 0x7c479a3a7490>
```

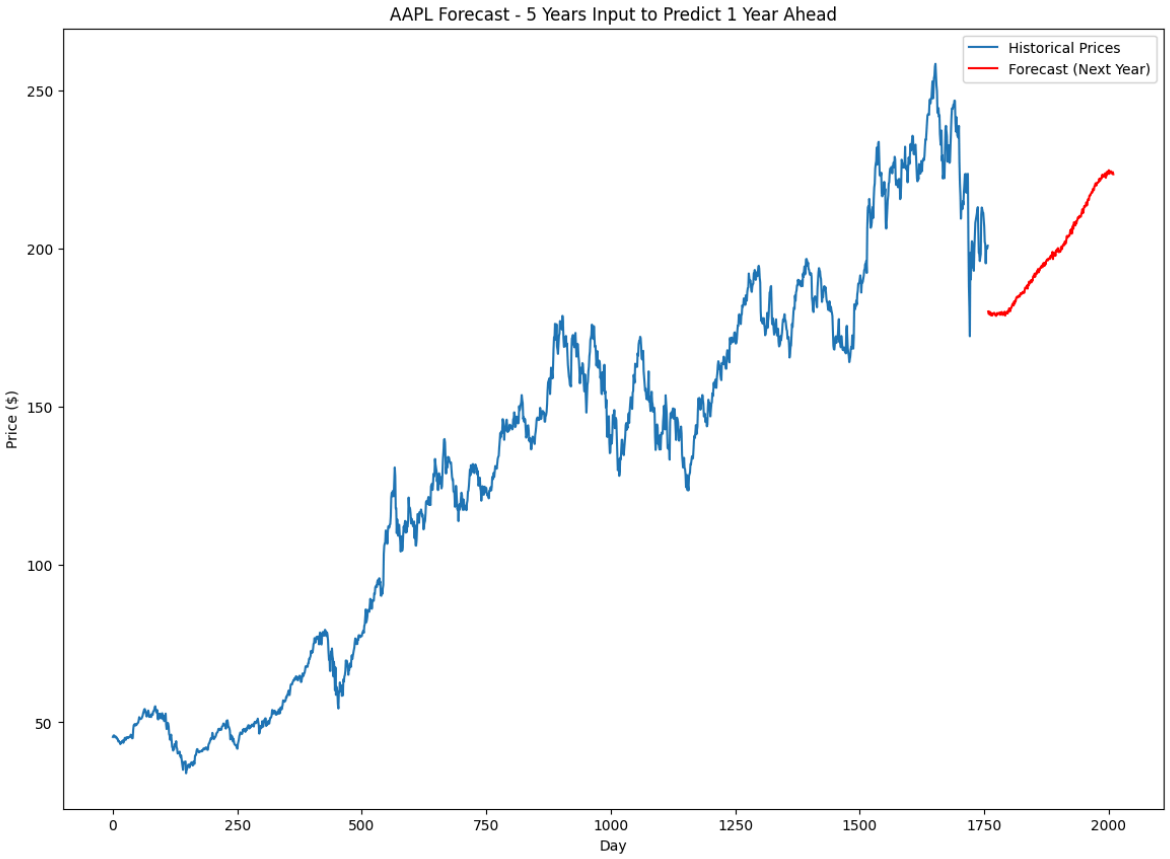
In [41]:

```
last_input = scaled_data[-1260:].reshape(1, 1260, 1)

forecast_scaled = model.predict(last_input)
forecast_prices = scaler.inverse_transform(forecast_scaled.reshape(-1, 1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 367ms/step
```

In [42]:

```
full_actual = data['Close'].values
days = np.arange(len(full_actual) + 252)

plt.figure(figsize=(14, 10))
plt.plot(days[:len(full_actual)], full_actual, label='Historical Prices')
plt.plot(days[len(full_actual):], forecast_prices, label='Forecast (Next
plt.title('AAPL Forecast - 5 Years Input to Predict 1 Year Ahead')
plt.xlabel('Day')
plt.ylabel('Price ($)')
plt.legend()
plt.show()
```



In [ ]:

```
model.save("stock_forecast_lstm.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
or `keras.saving.save_model(model)`. This file format is considered legac
y. We recommend using instead the native Keras format, e.g. `model.save('m
y_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```