****

**Batch: HO–DL 1 Roll No.: 16010422234 Experiment No. 08**

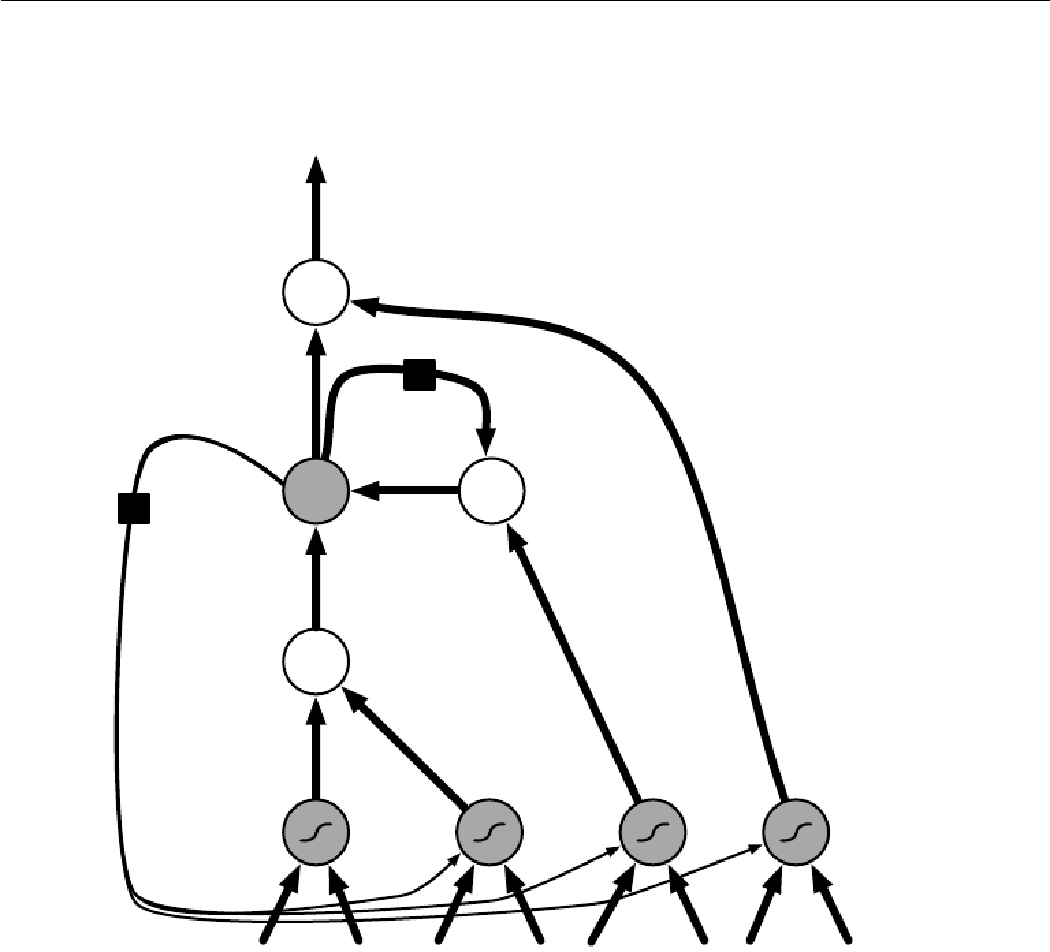
**Aim:** To implement LSTM network

**Resources needed: Python/Matlab**

**Theory:**

The most effective sequence models used in practical applications are called gated RNNs. These include the long short-term memory and networks based on the gated recurrent unit. Long Short Term Memory Networks is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network also known as RNN is used for persistent memory.

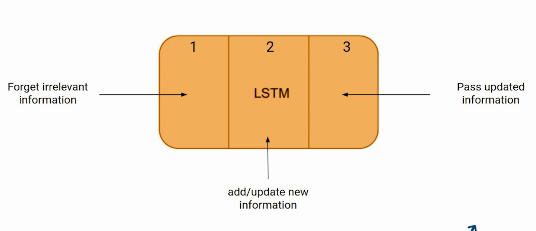
Let’s say while watching a video you remember the previous scene or while reading a book you know what happened in the earlier chapter. Similarly, RNNs work, they remember the previous information and use it for processing the current input. The shortcoming of RNN is, they cannot remember Long term dependencies due to vanishing gradients. LSTMs are explicitly designed to avoid long-term dependency problems.



**Fig-1: Block diagram of the LSTM recurrent network “cell.”**

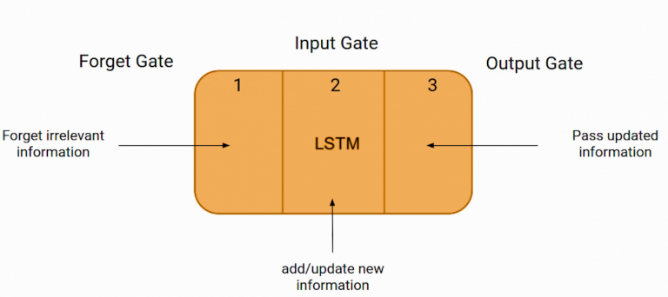
The clever idea of introducing self-loops to produce paths where the gradient can flow for long durations is a core contribution of the initial long short-term memory (LSTM) model (Hochreiter and Schmidhuber, 1997).

At a high-level LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. The LSTM consists of three parts, as shown in the image below and each part performs an individual function.



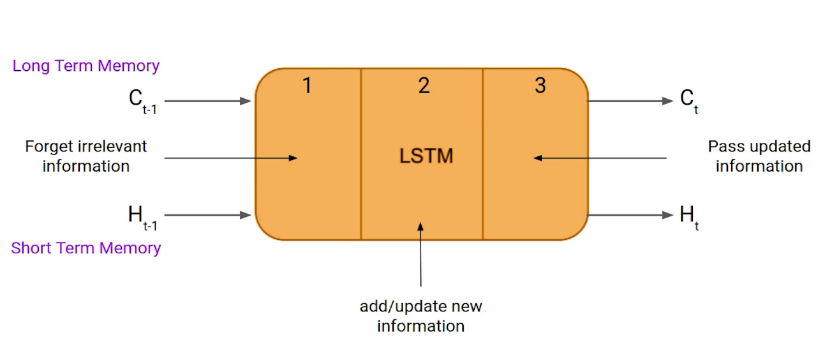
The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp.

These three parts of an LSTM cell are known as gates. The first part is called Forget gate, the second part is known as the Input gate and the last one is the Output gate.

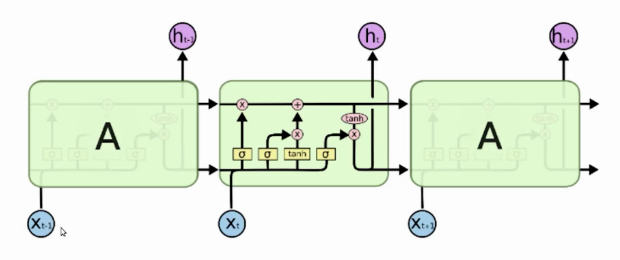


LSTM gates- Just like a simple RNN, an LSTM also has a hidden state where H(t-1) represents the hidden state of the previous timestamp and Ht is the hidden state of the current timestamp. In addition to that, LSTM also has a cell state represented by C(t-1) and C(t) for previous and current timestamps respectively.

Here the hidden state is known as Short term memory and the cell state is known as Long term memory. Refer to the following image.



This is the More intuitive diagram of the LSTM network.



The LSTM has been found extremely successful in many applications, such as unconstrained handwriting recognition (Graves et al., 2009), speech recognition (Graves et al., 2013; Graves and Jaitly, 2014), handwriting generation (Graves, 2013), machine translation (Sutskever et al., 2014), image captioning (Kiros et al., 2014b; Vinyals et al., 2014b; Xu et al., 2015) and parsing (Vinyals et al., 2014a).

**Activity:**

• Load any time series dataset.

• Pre-process and visualize the dataset.

• Form the Training and Testing Data.

• Develop and train LSTM model.

• Plot the predictions for training and testing data.

• Comment on the output.

**Program:**

**# Import necessary libraries**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.preprocessing import MinMaxScaler**

**from keras.models import Sequential**

**from keras.layers import LSTM, GRU, Dense, Dropout**

**from keras.optimizers import Adam**

**import pandas as pd**

**# Load the dataset**

**file\_path = '/content/Month\_Value\_1.csv'**

**data = pd.read\_csv(file\_path)**

**# Convert 'Period' to datetime**

**data['Period'] = pd.to\_datetime(data['Period'], format='%d.%m.%Y')**

**# Check for NaN values in the dataset**

**print("Checking for NaN values in the dataset:")**

**print(data.isna().sum()) # Check how many NaN values there are in each column**

**# Handle NaN values: Drop rows with NaN values**

**data = data.dropna()**

**# Replace zeros in numerical columns with small positive values to avoid issues during scaling**

**data[["Sales\_quantity", "Average\_cost", "The\_average\_annual\_payroll\_of\_the\_region"]] = data[["Sales\_quantity", "Average\_cost", "The\_average\_annual\_payroll\_of\_the\_region"]].replace(0, 0.0001)**

**# We will use 'Revenue' as the target and the other columns as features**

**features = ['Sales\_quantity', 'Average\_cost', 'The\_average\_annual\_payroll\_of\_the\_region']**

**target = 'Revenue'**

**# Scaling the data (MinMax scaling to range between 0 and 1)**

**scaler = MinMaxScaler(feature\_range=(0, 1))**

**scaled\_data = scaler.fit\_transform(data[features + [target]])**

**# Check for NaN or Inf values in the scaled data after scaling**

**print(f"Any NaN in scaled data: {np.any(np.isnan(scaled\_data))}")**

**print(f"Any Inf values in scaled data: {np.any(np.isinf(scaled\_data))}")**

**# Prepare the data for LSTM: create time series sequences**

**time\_step = 3 # Using 3 time steps as the window size**

**x\_data, y\_data = [], []**

**for i in range(time\_step, len(scaled\_data)):**

**x\_data.append(scaled\_data[i-time\_step:i, :-1]) # Features (excluding target)**

**y\_data.append(scaled\_data[i, -1]) # Target ('Revenue')**

**x\_data, y\_data = np.array(x\_data), np.array(y\_data)**

**# Split into training and testing sets**

**train\_size = int(len(x\_data) \* 0.8)**

**x\_train, x\_test = x\_data[:train\_size], x\_data[train\_size:]**

**y\_train, y\_test = y\_data[:train\_size], y\_data[train\_size:]**

**# Reshape data for LSTM (LSTM expects 3D input: [samples, time steps, features])**

**x\_train = x\_train.reshape(x\_train.shape[0], x\_train.shape[1], x\_train.shape[2])**

**x\_test = x\_test.reshape(x\_test.shape[0], x\_test.shape[1], x\_test.shape[2])**

**# Build the LSTM model with more units and layers for better learning**

**model = Sequential()**

**model.add(LSTM(units=100, return\_sequences=True, input\_shape=(x\_train.shape[1], x\_train.shape[2])))**

**model.add(Dropout(0.2)) # Add dropout for regularization**

**model.add(LSTM(units=100, return\_sequences=True))**

**model.add(Dropout(0.2)) # Add another dropout layer**

**model.add(GRU(units=50)) # Adding a GRU layer as an alternative**

**model.add(Dense(units=1)) # Output layer with 1 unit (for prediction of 'Revenue')**

**# Compile the model with a lower learning rate**

**model.compile(optimizer=Adam(learning\_rate=0.0001), loss='mean\_squared\_error')**

**# Train the model**

**model.fit(x\_train, y\_train, epochs=150, batch\_size=16) # Increased epochs**

**# Predict the results**

**train\_pred = model.predict(x\_train)**

**test\_pred = model.predict(x\_test)**

**# Inverse transform the predictions and actual values**

**train\_pred = scaler.inverse\_transform(np.concatenate((np.zeros((train\_pred.shape[0], 3)), train\_pred), axis=1))[:, -1]**

**y\_train\_actual = scaler.inverse\_transform(np.concatenate((np.zeros((y\_train.shape[0], 3)), y\_train.reshape(-1, 1)), axis=1))[:, -1]**

**test\_pred = scaler.inverse\_transform(np.concatenate((np.zeros((test\_pred.shape[0], 3)), test\_pred), axis=1))[:, -1]**

**y\_test\_actual = scaler.inverse\_transform(np.concatenate((np.zeros((y\_test.shape[0], 3)), y\_test.reshape(-1, 1)), axis=1))[:, -1]**

**# Plotting the predictions**

**plt.figure(figsize=(12, 6))**

**# Plot the actual training data (up to the actual training length)**

**plt.plot(data['Period'][:len(y\_train\_actual)], y\_train\_actual, color='blue', label='Actual Training Data')**

**# Plot the actual testing data (from train\_size + time\_step onward)**

**plt.plot(data['Period'][train\_size + time\_step:], y\_test\_actual, color='green', label='Actual Testing Data')**

**# Plot the predicted training data (up to the actual training length)**

**plt.plot(data['Period'][:len(train\_pred)], train\_pred, color='red', label='Predicted Training Data')**

**# Plot the predicted testing data (from train\_size + time\_step onward)**

**plt.plot(data['Period'][train\_size + time\_step:], test\_pred, color='orange', label='Predicted Testing Data')**

**plt.title('Revenue Prediction (LSTM)')**

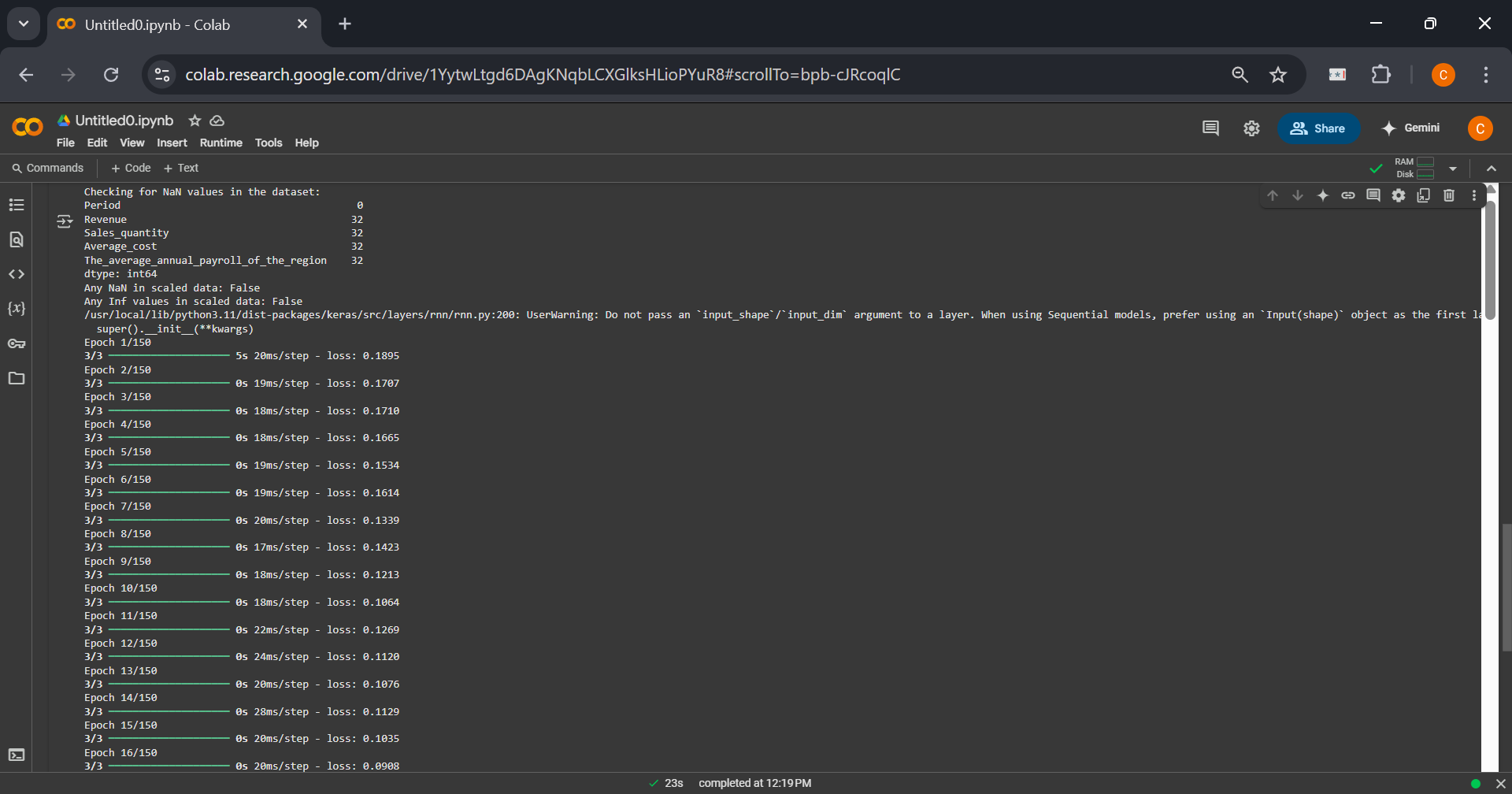
**plt.xlabel('Date')**

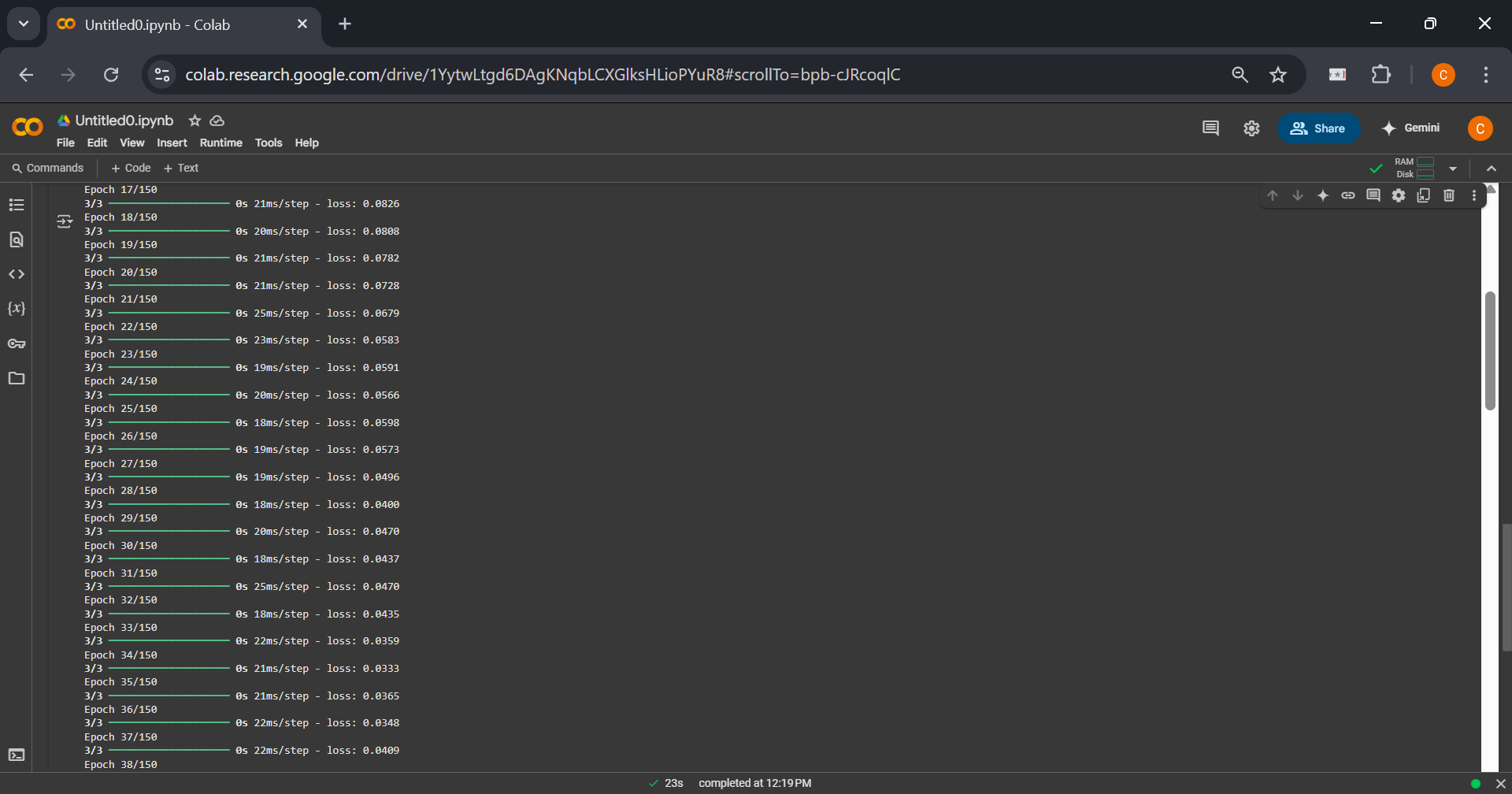
**plt.ylabel('Revenue')**

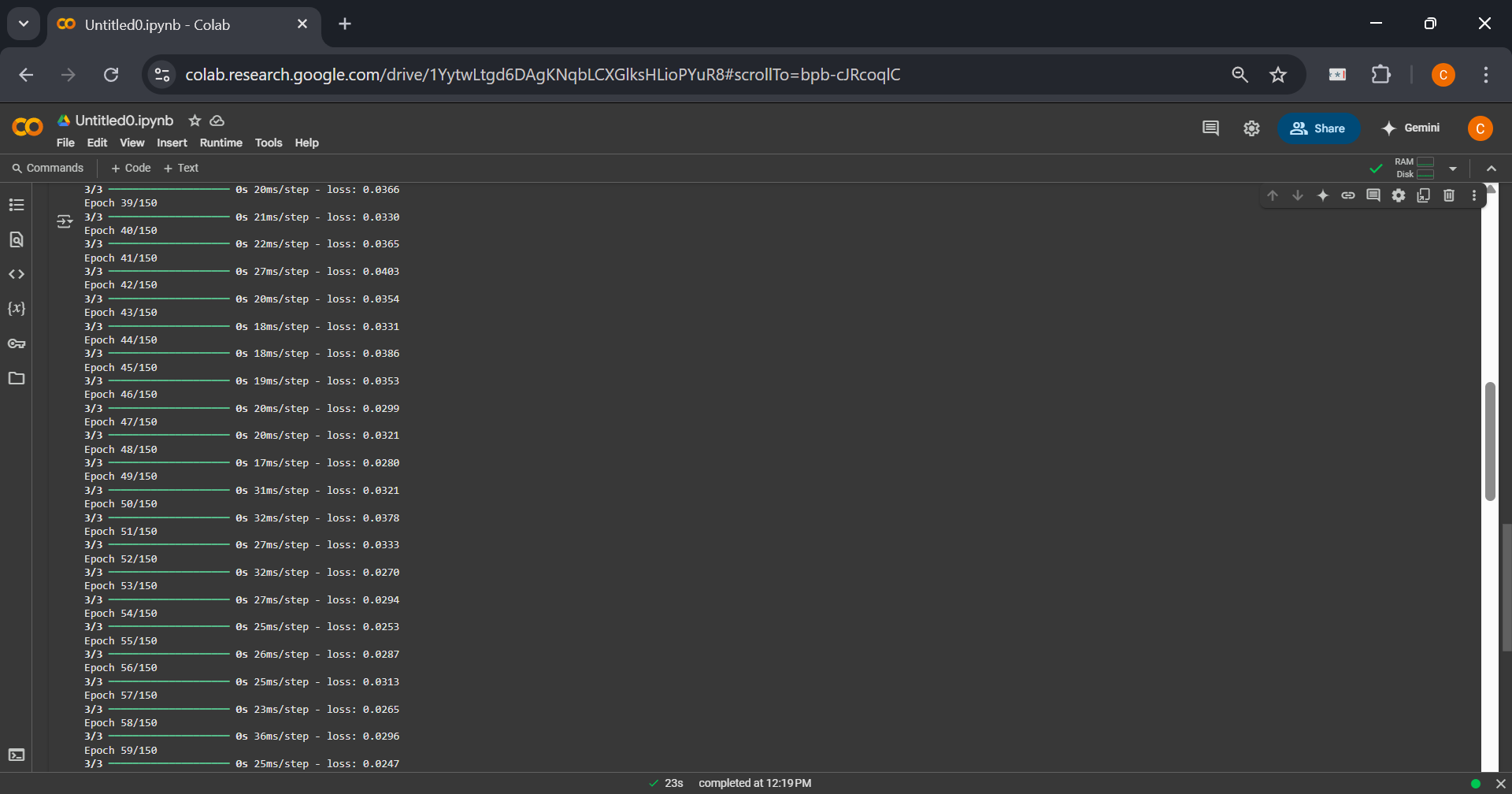
**plt.legend()**

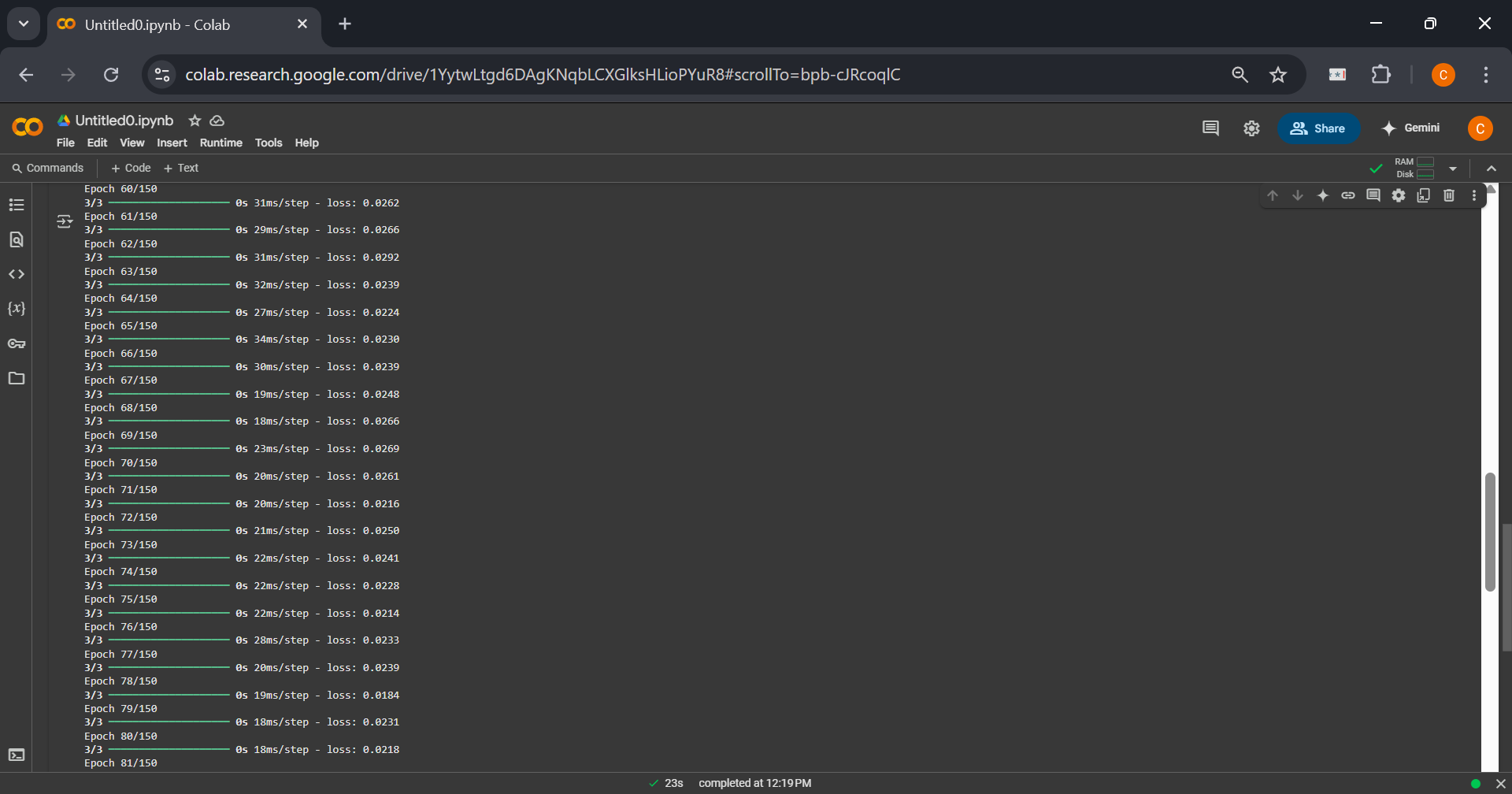
**plt.show()**

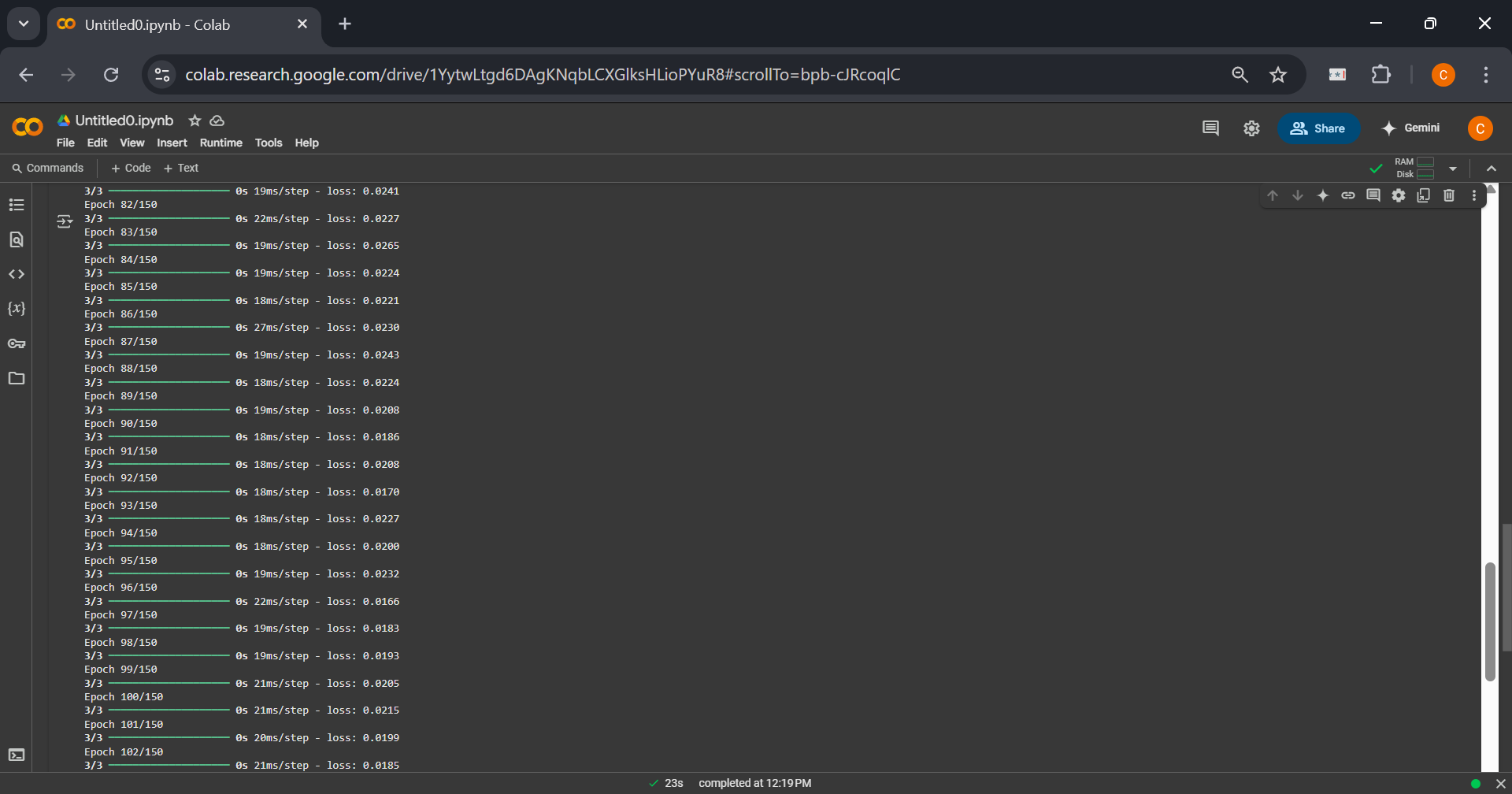
**Output:**

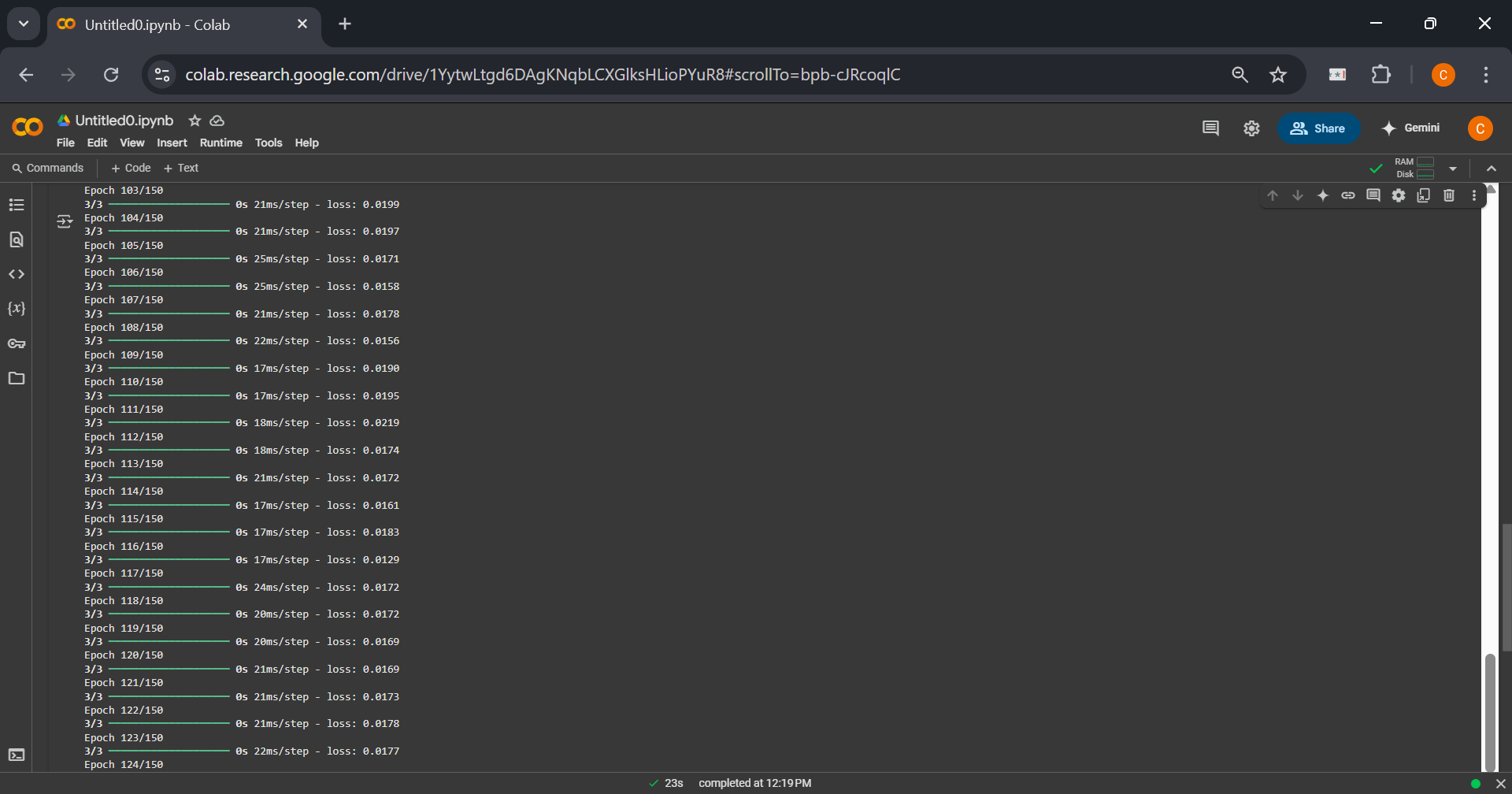
****

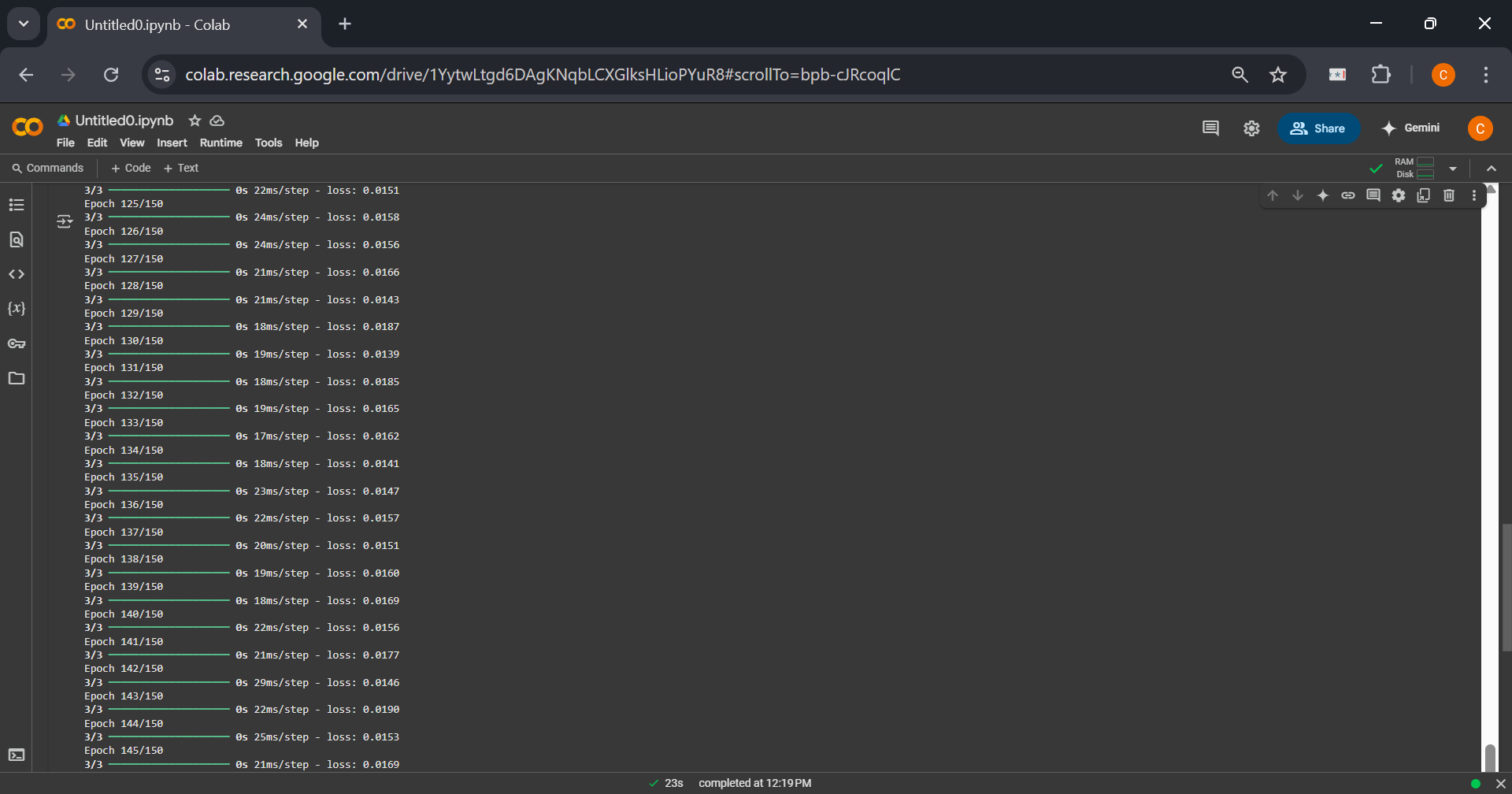
****

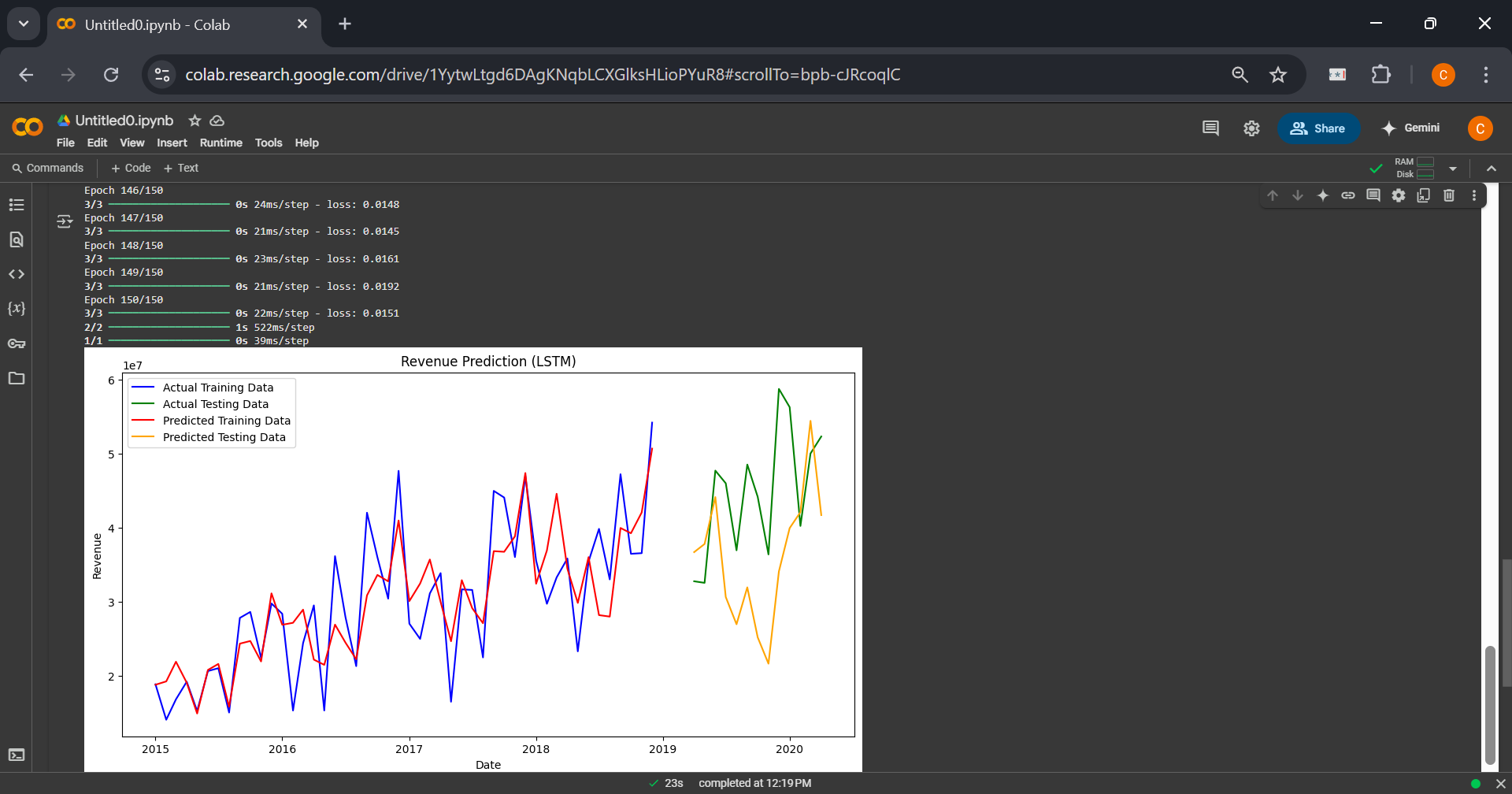
****

****

****

****

****

****

**Post Lab Question- Answers (If Any):**

**1. How does LSTM solve the vanishing gradient challenge?**

LSTM solves the vanishing gradient problem by introducing memory cells that can maintain information over long periods. The gradient can flow through these cells without decaying, as they have self-loops allowing the gradient to be passed through longer sequences.

**2. What are the practical applications of LSTM?**

LSTM networks are used in a variety of fields, including:

• Time series forecasting (e.g., stock market prediction)

• Speech recognition and synthesis

• Handwriting recognition and generation

• Language translation

• Image captioning

• Parsing

**CO – 4: Underhand the essentials of Recurrent and Recursive Nets.**

**Conclusion:**

LSTM networks are powerful tools for solving sequence-related problems by remembering long-term dependencies. The lab demonstrated how to build, train, and evaluate an LSTM model on time series data.

**Grade: AA / AB / BB / BC / CC / CD / DD**

**Signature of faculty in-charge with date**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**References:**

**Books/ Journals/ Websites:**

1. Josh Patterson and Adam Gibson, “Deep Learning A Practitioner’s Approach”, O’Reilly Media 2017
2. <https://www.ibm.com/cloud/learn/recurrent-neural-networks>
3. <https://searchenterpriseai.techtarget.com/definition/recurrent-neural-networks>
4. <https://www.sciencedirect.com/science/article/pii/B9780128161760000260>