STOCK PRICE PREDICTION

Stock price prediction is a complex task that involves analyzing historical data and making forecasts about future price movements. Two common approaches for stock price prediction are Linear Regression and Long Short-Term Memory (LSTM) neural networks. A brief introduction to both methods and how they can be used for stock price prediction is described below:

Linear Regression:

Linear regression is a simple and interpretable machine learning technique used for predicting a continuous variable, such as stock prices. It assumes a linear relationship between the input features and the target variable (stock price in this case).

Data Preparation: Use the historical stock price data and relevant features provided (e.g., trading volume, technical indicators, economic factors).

Feature Engineering: Extract and preprocess features, including lagged values of stock prices, moving averages, and other relevant data.

Model Training: Fitting a linear regression model to the training data, where the input features are used to predict the future stock price.

Evaluation: Using metrics like Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) to evaluate the model's performance on a validation dataset.

Prediction: To Make predictions on unseen data to forecast future stock prices.

Linear regression has limitations in capturing complex patterns in stock price data, which is where LSTM comes into play.

Long Short-Term Memory (LSTM):

LSTM is a type of recurrent neural network (RNN) designed to handle sequential data and capture long-term dependencies. It is particularly well-suited for time series forecasting tasks like stock price prediction.

Data Preparation: Similar to Linear Regression, we use the historical stock price data and relevant features.

Data Preprocessing: Normalize or scale the data to ensure that it falls within a specific range.

Sequence Creation: Converting the time series data into sequences of fixed length. Each sequence contains historical stock prices and corresponding features.

Model Architecture: Build an LSTM neural network architecture with input layers, LSTM layers, and output layers. The LSTM layers can capture temporal dependencies in the data.

Model Training: Training the LSTM model using the dataset, optimizing it to make accurate predictions.

Evaluation: Use appropriate evaluation metrics (e.g., Mean Absolute Error, Mean Absolute Percentage Error) to assess the model's performance on a validation dataset.

Use the trained LSTM model to make predictions about future stock prices based on input sequences.

LSTM models are generally better at capturing complex patterns and long-term dependencies in stock price data compared to linear regression. However, they are also more complex to train and require more data.

Both linear regression and LSTM models can serve as useful tools, but they may have certain limitations and risks associated with stock trading decisions.

BASIC PYTHON CODE FOR LINEAR REG

The first step in implementing linear regression with a dataset involves data preparation, which includes the following key steps:

Collecting the Data:

Obtain the dataset that contains the relevant information for the linear regression analysis. In the context of stock price prediction, this might include historical stock prices and relevant features.

Exploratory Data Analysis (EDA):

Performing an initial exploration of the dataset to understand its structure, features, and any missing values. We use tools like Python's Pandas library or Google colab to load and explore the data. EDA helps you gain insights into the dataset and decide which features to use for regression model.

Data Preprocessing:

Data preprocessing is crucial for preparing the dataset for linear regression. Common preprocessing steps include:

Handling missing data: Decide how to handle missing values (e.g., imputing with the mean, median, or remove rows with missing data).

Feature selection: Choosing relevant features (independent variables) for regression model.

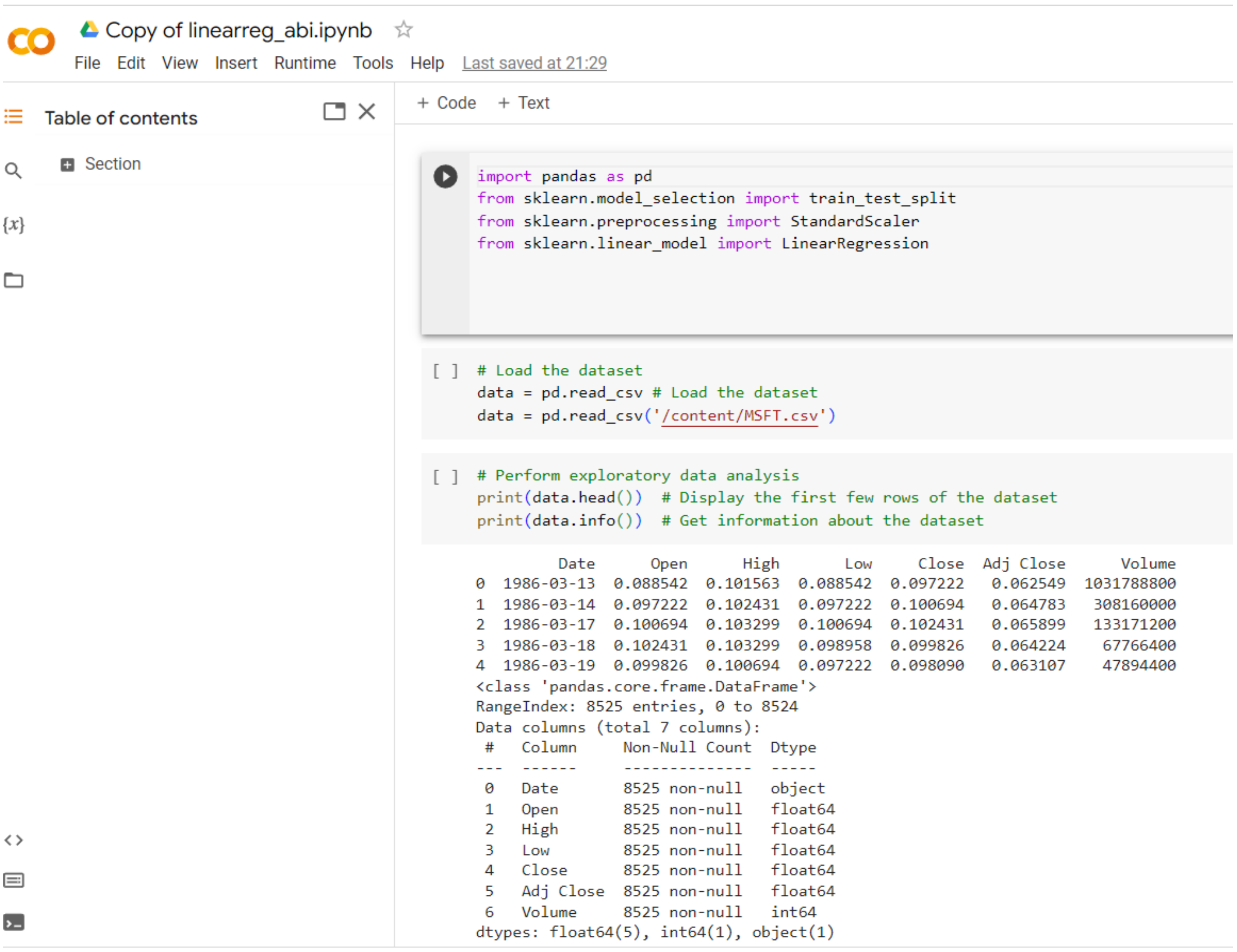
Data scaling: Normalize or standardize numerical features to ensure they have a similar scale (e.g., using Min-Max scaling).

Categorical variables: In case of categorical variables, we need to encode them (e.g., one-hot encoding) to make them suitable for regression.

Splitting the Data:

Split the dataset into training and testing sets. The training set is used to train the linear regression model, while the testing set is used to evaluate its performance. A common split ratio is 80% for training and 20% for testing.

Here is a Python code demonstrating the steps using the popular libraries Pandas and Scikit-Learn using our dataset:


Copy of linearreg_abi.ipynb
File Edit View Insert Runtime Tools Help Last saved at 21:29
Table of contents
Section
+ Code + Text

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
```

```python
# Load the dataset
data = pd.read_csv # Load the dataset
data = pd.read_csv('/content/MSFT.csv')
```

```python
# Perform exploratory data analysis
print(data.head())  # Display the first few rows of the dataset
print(data.info())  # Get information about the dataset
```

```
         Date      Open      High       Low     Close  Adj Close      Volume
0  1986-03-13  0.088542  0.101563  0.088542  0.097222   0.062549  1031788800
1  1986-03-14  0.097222  0.102431  0.097222  0.100694   0.064783   308160000
2  1986-03-17  0.100694  0.103299  0.100694  0.102431   0.065899   133171200
3  1986-03-18  0.102431  0.103299  0.098958  0.099826   0.064224    67766400
4  1986-03-19  0.099826  0.100694  0.097222  0.098090   0.063107    47894400
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8525 entries, 0 to 8524
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       8525 non-null   object 
 1   Open       8525 non-null   float64
 2   High       8525 non-null   float64
 3   Low        8525 non-null   float64
 4   Close      8525 non-null   float64
 5   Adj Close  8525 non-null   float64
 6   Volume     8525 non-null   int64  
dtypes: float64(5), int64(1), object(1)
```

Step 1: Import Libraries and Load dataset

numpy: A library for numerical operations in Python.

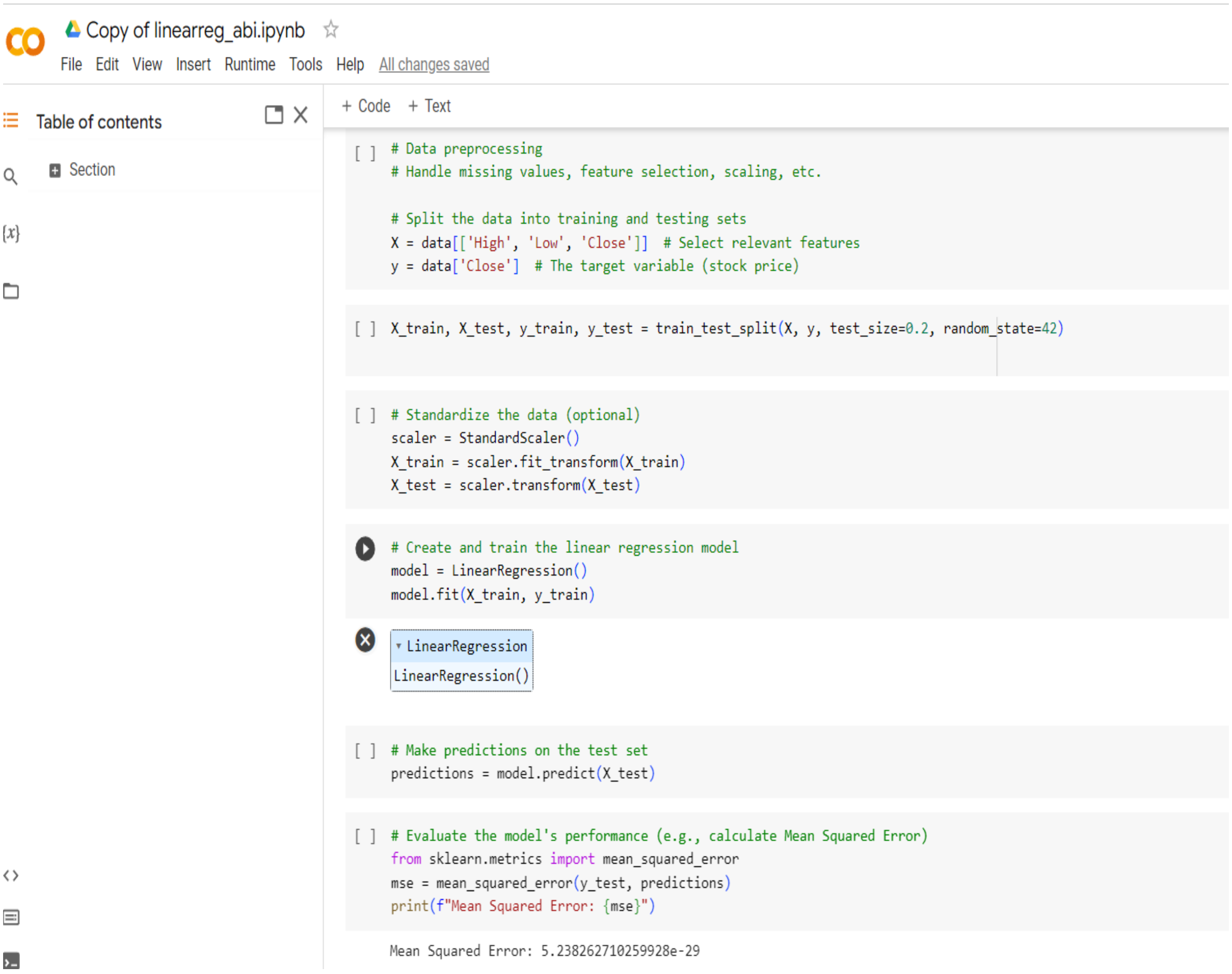
matplotlib. pyplot: Used for data visualization.

Linear Regression from sklearn.linear_model: This class provides the linear regression model that we'll use.

Import pandas and use one of its functions like **read_csv()** to load data from a CSV file

Step 2: Prepare Data

We have a dataset with two arrays: one representing the independent variable (usually denoted as 'X') and another representing the dependent variable (usually denoted as 'y').


Copy of linearreg_abi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Table of contents
Section
+ Code + Text

```
# Data preprocessing
# Handle missing values, feature selection, scaling, etc.

# Split the data into training and testing sets
X = data[['High', 'Low', 'Close']]  # Select relevant features
y = data['Close']  # The target variable (stock price)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Standardize the data (optional)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Make predictions on the test set
predictions = model.predict(X_test)
```

```
# Evaluate the model's performance (e.g., calculate Mean Squared Error)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error: {mse}")
```

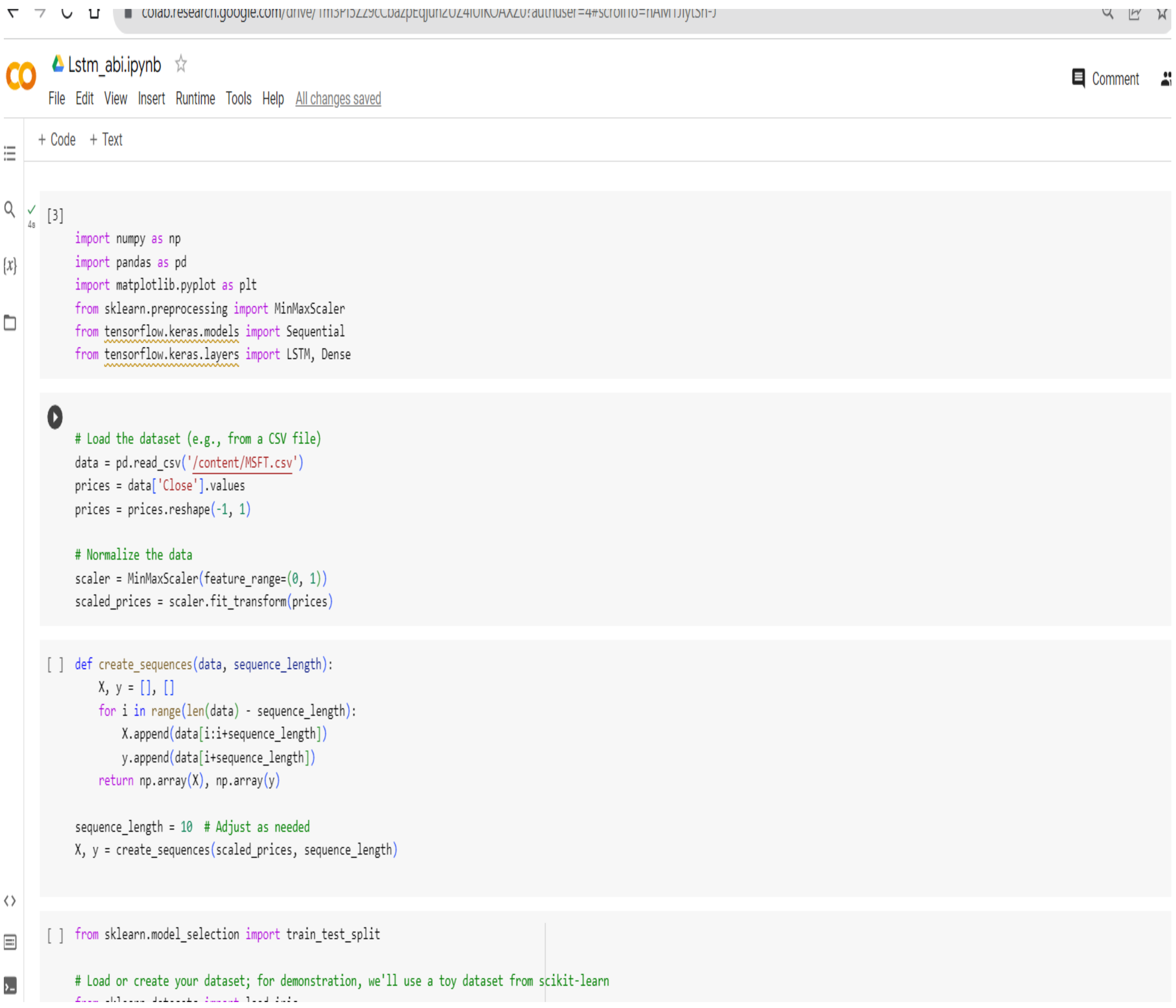
```
Mean Squared Error: 5.238262710259928e-29
```

Step 3: Create and Train the Linear Regression Model

Now, let's create an instance of the LinearRegression model and fit it to our data.

Step 4: Make Predictions

Now that the model is trained, you can use it to make predictions. This method predicts the values of the dependent variable ('y') based on the independent variable ('X').


colab.research.google.com/drive/1m3P15Z29cCbazpEqjun2OZ4IOlKOAXZ0?authuser=4#scrollTo=hAMTJlyLSh-J
Lstm_abi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
+ Code + Text
[3]

```python
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
```

```python
# Load the dataset (e.g., from a CSV file)
data = pd.read_csv('/content/MSFT.csv')
prices = data['Close'].values
prices = prices.reshape(-1, 1)

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_prices = scaler.fit_transform(prices)
```

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

sequence_length = 10  # Adjust as needed
X, y = create_sequences(scaled_prices, sequence_length)
```

```python
from sklearn.model_selection import train_test_split

# Load or create your dataset; for demonstration, we'll use a toy dataset from scikit-learn
```

Step 1: Import Libraries

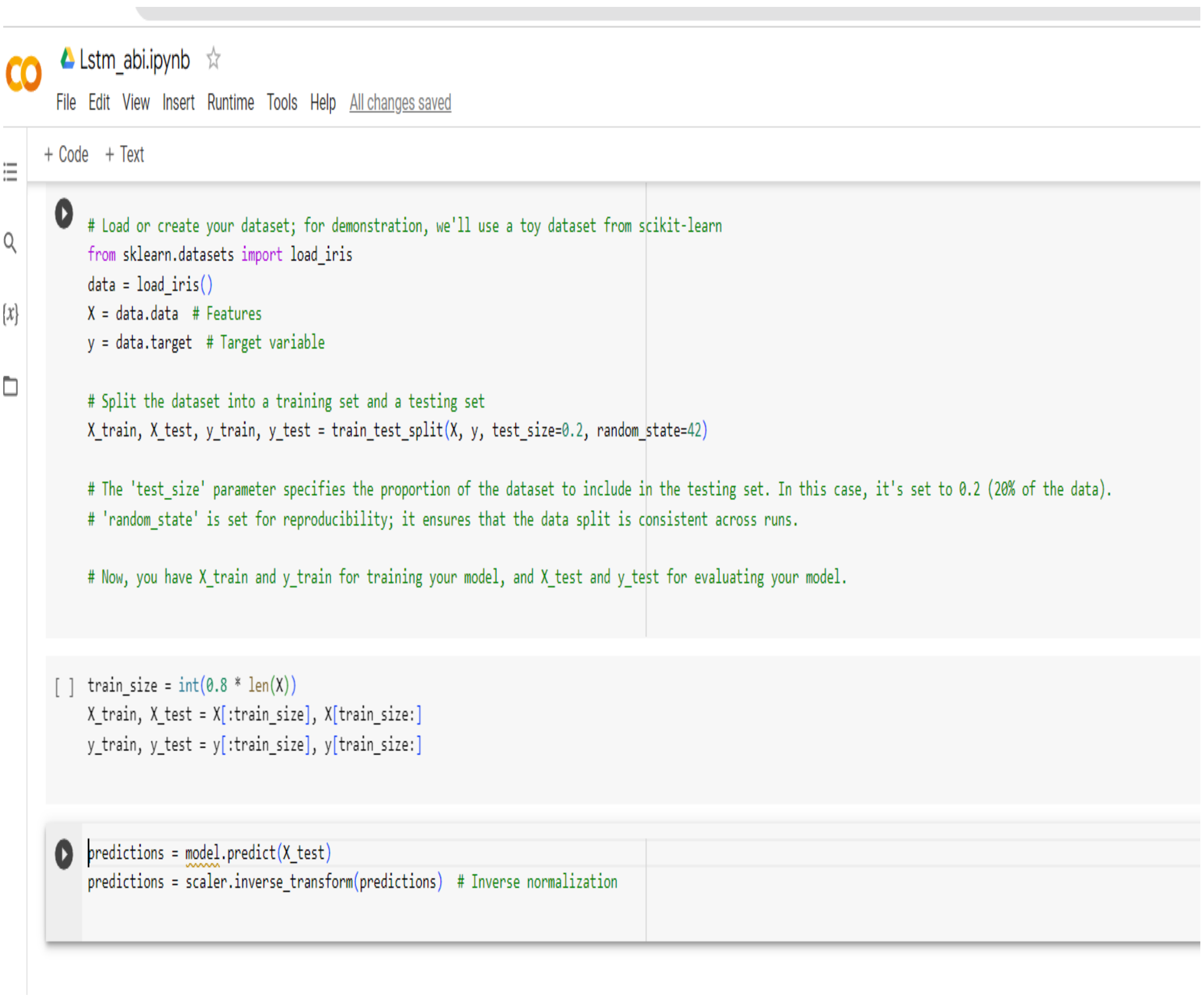
We'll need to import the necessary libraries, including TensorFlow/Keras and other data processing tools.

Step 2: Load and Preprocess Data

Load historical stock price data and preprocess it. We should have a dataset with at least two columns: one for the date and one for the stock prices.

Step 3: Prepare Sequences

To train the LSTM model, create sequences of dataset, where each input sequence contains a window of past stock prices and the corresponding output sequence contains the predicted stock price. This is referred to as sequence-to-sequence forecasting.



```
# Load or create your dataset; for demonstration, we'll use a toy dataset from scikit-learn
from sklearn.datasets import load_iris
data = load_iris()
X = data.data  # Features
y = data.target  # Target variable

# Split the dataset into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# The 'test_size' parameter specifies the proportion of the dataset to include in the testing set. In this case, it's set to 0.2 (20% of the data).
# 'random_state' is set for reproducibility; it ensures that the data split is consistent across runs.

# Now, you have X_train and y_train for training your model, and X_test and y_test for evaluating your model.
```

```
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

```
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)  # Inverse normalization
```

Step 4: Split Data into Training and Testing Sets

Split the data into training and testing sets to evaluate the model's performance.

Step 5: Train the Model

Train the LSTM model using the training data.

Step 7: Make Predictions

Use the trained model to make predictions on the test data.

We should fine-tune hyperparameters, optimize the model architecture, and consider other factors like feature engineering and using more sophisticated techniques for better stock price predictions.

Also, be aware that predicting stock prices is challenging and uncertain due to various external factors affecting financial markets.

The predicted stock prices are typically obtained by feeding a sequence of historical stock prices as input to your trained Long Short-Term Memory (LSTM) model.

The sequence used as input, often referred to as a "window" or "lookback window," is used to make predictions about the next stock price.

We can use visualization libraries like **matplotlib** to plot the actual and predicted stock prices to assess the model's performance.