**FAKE NEWS DETECTION USING NLP**

**Technology: ARTIFICIAL INTELLIGENCE**

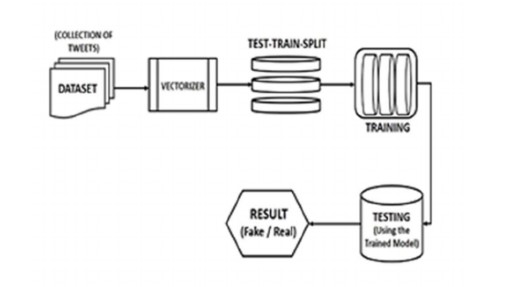
**NAME: M. VIJAYSASAN**

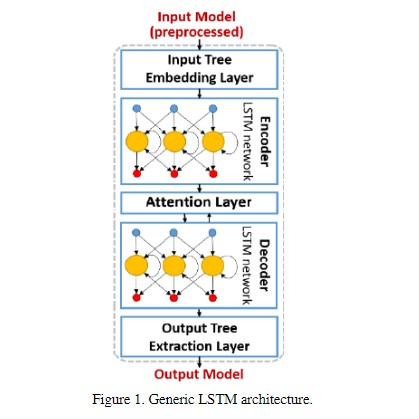
**NM ID: au311121205701**

**REG. NO.: 311121205701**

**PHASE 4 - PROJECT SUBMISSION**

**Phase 4: Development Part 2**





**PHASE 3 GUIDELINES:(GIVEN)**

**Phase 4: Development Part 2**

In this part you will continue building your project.

Continue building the fake news detection model by applying NLP techniques and training a classification model.

Text Preprocessing and Feature Extraction

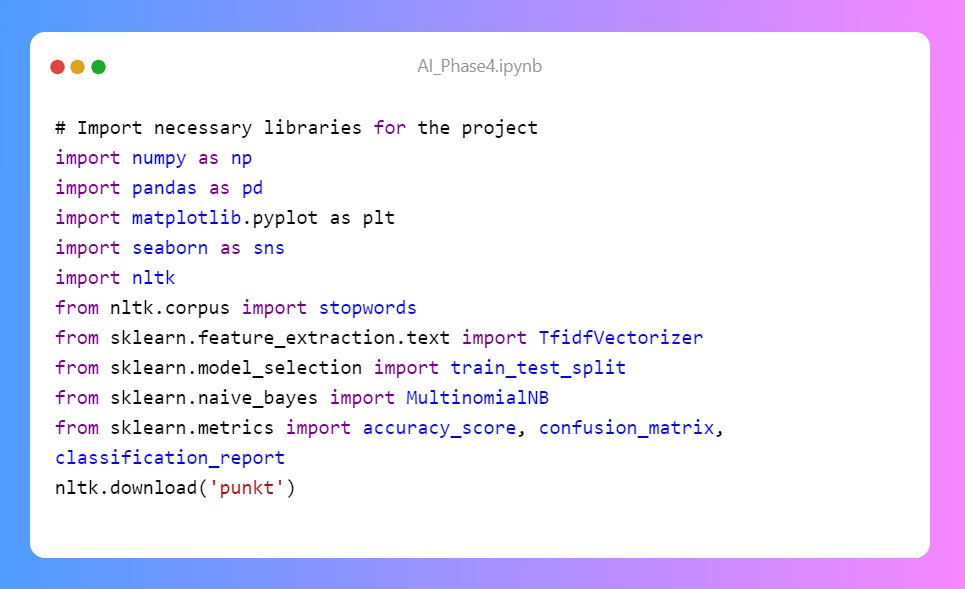
Model training and evaluation

**INTRODUCTION:**

In phase 4 of this project, we continue with the further development of our fake news detection project. In the last phase of development we focused on loading the dataset and preprocessing the textual dataset. In this phase, we will proceed further with processes like: **Text Preprocessing, Feature Extraction, Applying NLP techniques and training a classification model, Model training and evaluation.**

Let us begin with the project first by seeing the steps we have already seen before moving on to the rest of the steps.

**STEP 1: IMPORTING THE LIBRARIES.**



Here, we import the necessary libraries and modules for the project. Let's break down their roles:

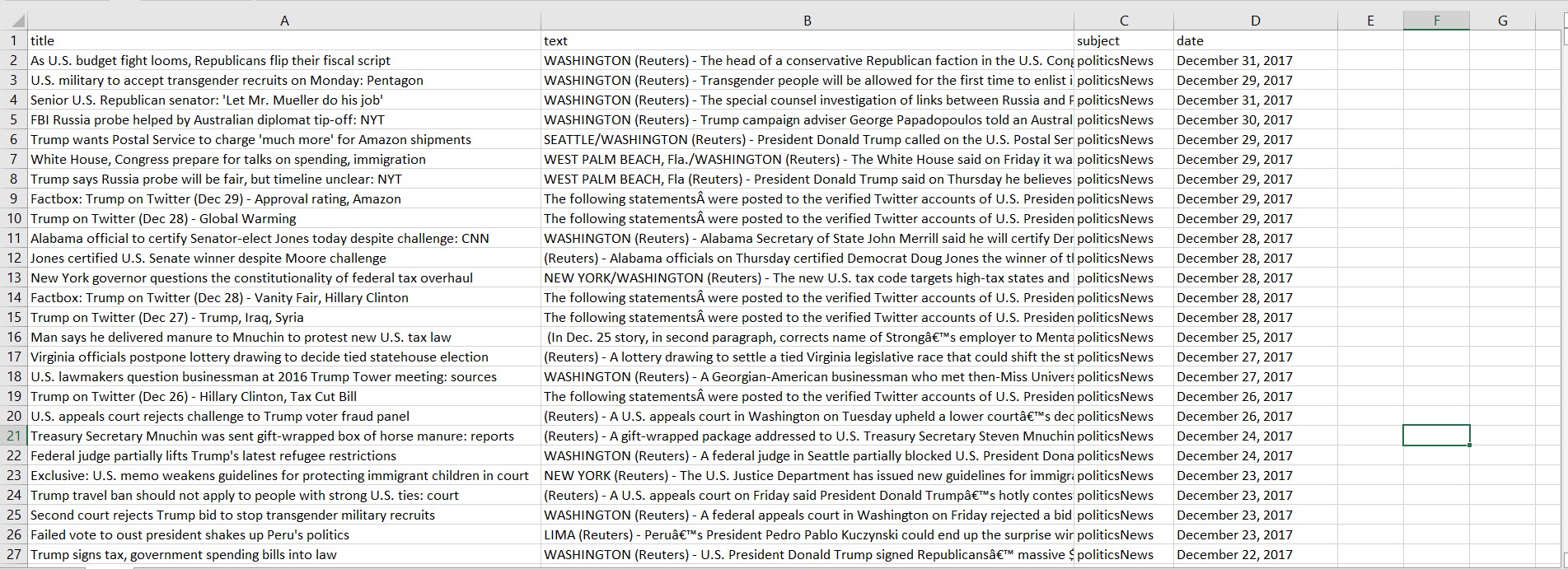
* numpy and pandas are fundamental for data manipulation and numerical operations.
* matplotlib and seaborn are used for data visualization, providing tools to create plots and graphs.
* nltk (Natural Language Toolkit) is a library for natural language processing (NLP) tasks, including text processing.
* nltk.corpus is used to access stopwords, which are common words to be excluded in text preprocessing.
* TfidfVectorizer from sklearn.feature\_extraction.text is a tool to convert text data into numerical features using TF-IDF.
* train\_test\_split from sklearn.model\_selection is used to split the data into training and testing sets.
* MultinomialNB from sklearn.naive\_bayes is a Naive Bayes classifier suitable for text classification.
* accuracy\_score, confusion\_matrix, and classification\_report from sklearn.metrics are for model evaluation.
* nltk.download('punkt'): This command downloads the 'punkt' resource for NLTK. The 'punkt' resource includes data files used for tokenization, which is the process of splitting text into individual words or tokens.

**STEP 2: DATA LOADING:**

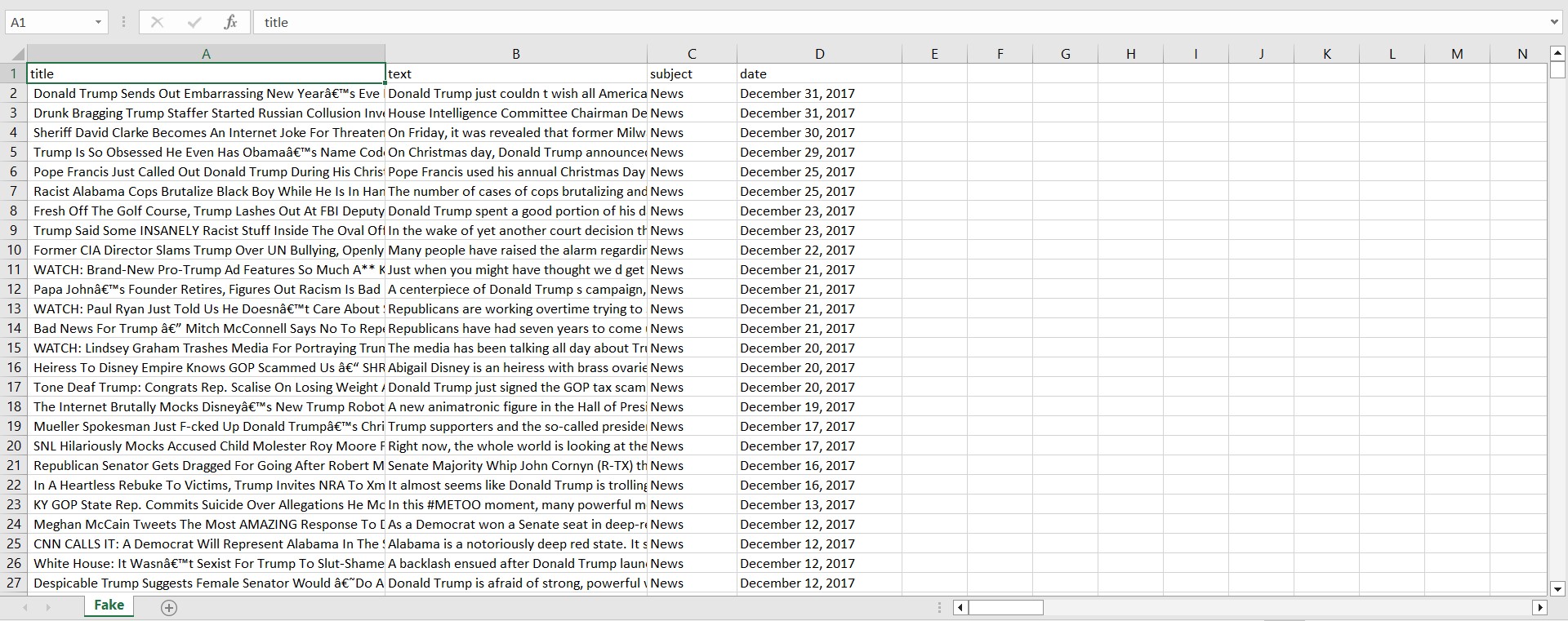


**DATASETS:**

**True.csv**



**Fake.csv**

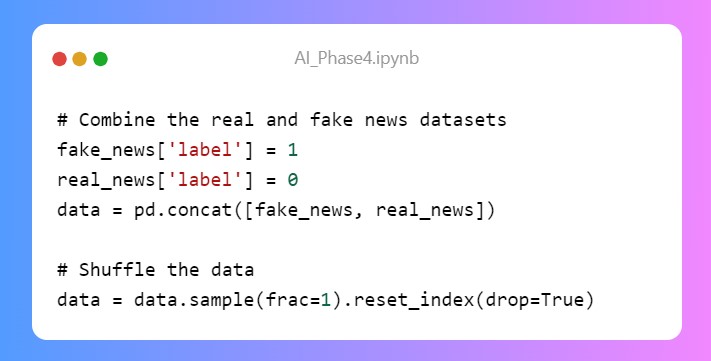


In this section, we load the dataset from Kaggle.

The datasets are downloaded from kaggle and loaded.

The pd.read\_csv() function reads data from CSV files and stores it in Pandas DataFrames. The fake\_news and real\_news DataFrames will contain the fake and real news data, respectively.

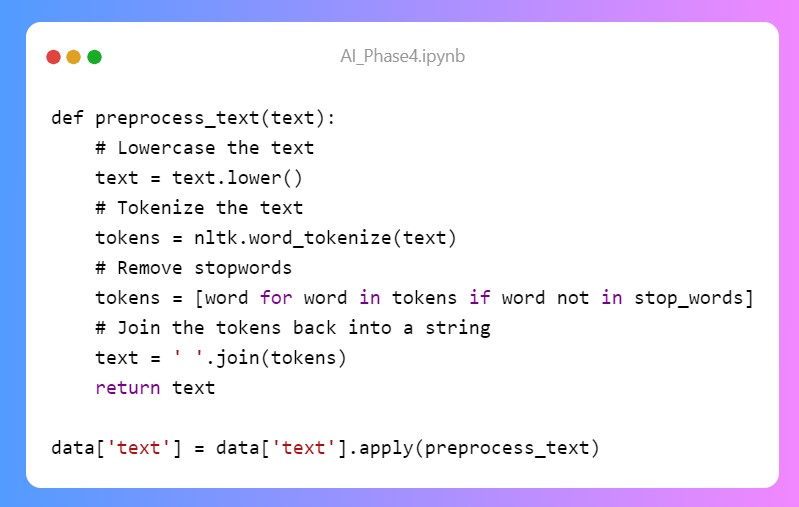
**STEP 3: DATA PREPROCESSING:**



In this section, we combine the fake and real news datasets into one DataFrame called data. We add a 'label' column, where '1' indicates fake news and '0' indicates real news. The data is shuffled to ensure randomness, and the index is reset.



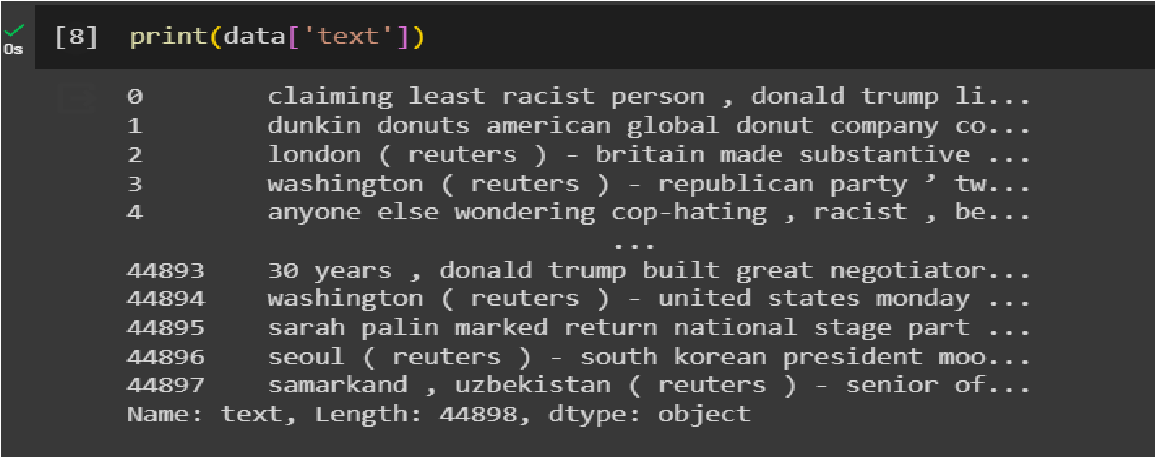
Here, we download the list of English stopwords from NLTK, which are common words that don't carry significant meaning in text data.



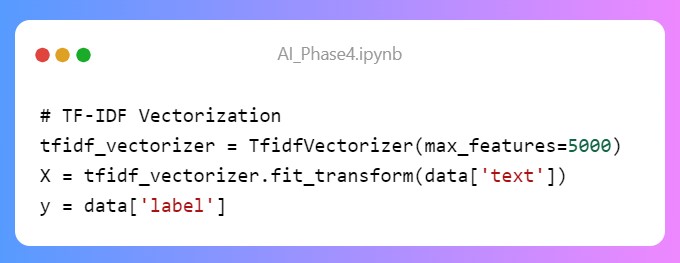
This part defines a function preprocess\_text to apply text preprocessing to the 'text' column in the DataFrame. The steps are as follows:

* Convert the text to lowercase to make it uniform.
* Tokenize the text into individual words using NLTK.
* Remove common English stopwords from the tokenized words.
* Join the tokenized words back into a cleaned text.

●



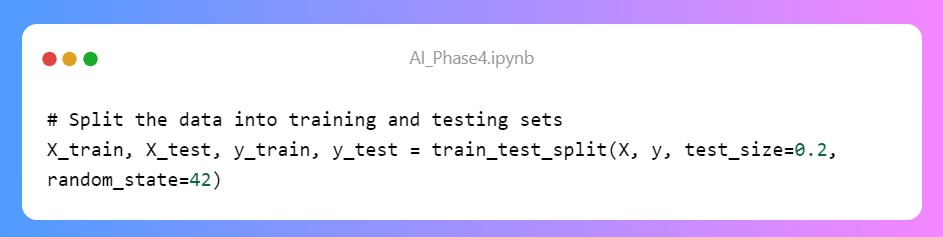
**STEP 4: FEATURE EXTRACTION:**



In this section, we use TF-IDF vectorization to convert the preprocessed text data into numerical features. Let's explain this step by step:

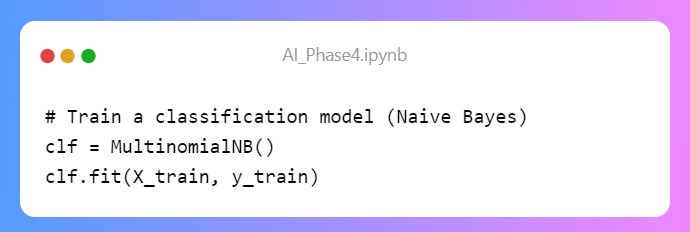
* TfidfVectorizer is initialized with max\_features set to 5000. This specifies that we want to consider the top 5000 most important terms in the dataset.
* tfidf\_vectorizer.fit\_transform(data['text']) computes the TF-IDF scores for each term in the text data, and X is a sparse matrix that represents the transformed data.
* y contains the labels for the corresponding data.

**STEP 5: MODEL TRAINING AND EVALUATION:**

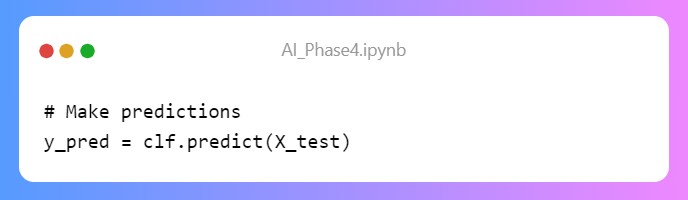


In this part, the data is split into training and testing sets. Here's what each line does:

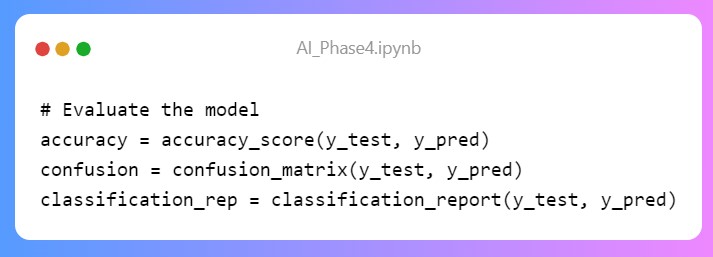
* train\_test\_split(X, y, test\_size=0.2, random\_state=42) splits the feature matrix X and the labels y into training and testing sets.
* The test\_size parameter specifies the percentage of data to use for testing (20% in this case), and random\_state ensures reproducibility.



Here, we create a Multinomial Naive Bayes classifier **clf** and train it using the training data. Naive Bayes is a simple yet effective classification algorithm, commonly used in text classification tasks.



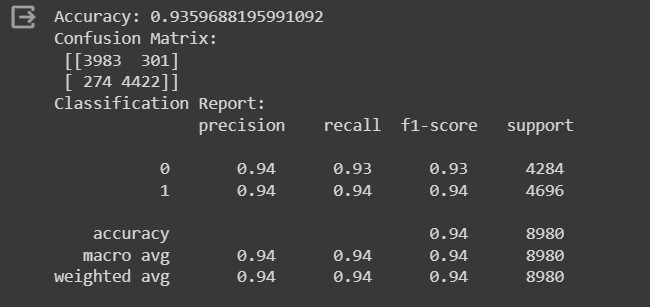
The model is used to make predictions on the testing set, and the predictions are stored in the y\_pred variable.



Here, we evaluate the model's performance.

* accuracy\_score calculates the accuracy of the model's predictions.
* confusion\_matrix generates a confusion matrix, which shows the number of true positives, true negatives, false positives, and false negatives.
* classification\_report provides a comprehensive report including precision, recall, F1-score, and support for each class.
* print("Accuracy:", accuracy), print("Confusion Matrix:\n", confusion), and print("Classification Report:\n", classification\_rep) display the evaluation results to the console.





**EXAMPLE PROGRAM USING LOGISTIC REGRESSION AND NEURAL NETWORKS:**

import pandas as pd from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score from sklearn.linear\_model import LogisticRegression from tensorflow.keras.preprocessing.text import Tokenizer from tensorflow.keras.preprocessing.sequence import pad\_sequences from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Embedding, LSTM, Dense

# Load the "Fake.csv" dataset fake\_data = pd.read\_csv("C:\\Users\\Bylee\\Downloads\\Fake.csv\\Fake.csv")

# Load the "True.csv" dataset true\_data = pd.read\_csv("C:\\Users\\Bylee\\Downloads\\True.csv\\True.csv") # Add labels to distinguish between fake and true news fake\_data['label'] = 0 # 0 for fake news true\_data['label'] = 1 # 1 for true news

# Combine the datasets combined\_data = pd.concat([fake\_data, true\_data], ignore\_index=True)

# Data Preprocessing combined\_data['text'] = combined\_data['title'] + " " + combined\_data['text']

# Feature Extraction (TF-IDF) tfidf\_vectorizer = TfidfVectorizer(max\_features=5000) tfidf\_matrix = tfidf\_vectorizer.fit\_transform(combined\_data['text'])

# Model Selection

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tfidf\_matrix, combined\_data['label'], test\_size=0.2, random\_state=42)

# Logistic Regression Model logistic\_regression\_model = LogisticRegression() logistic\_regression\_model.fit(X\_train, y\_train)

# Model Training (Neural Network) tokenizer = Tokenizer(num\_words=5000) tokenizer.fit\_on\_texts(combined\_data['text'])

X\_train\_nn = tokenizer.texts\_to\_sequences(combined\_data['text']) X\_train\_nn = pad\_sequences(X\_train\_nn, maxlen=100)

model = Sequential() model.add(Embedding(input\_dim=5000, output\_dim=128, input\_length=100)) model.add(LSTM(128)) model.add(Dense(1, activation='sigmoid')) model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy']) model.fit(X\_train\_nn, combined\_data['label'], epochs=5, batch\_size=64)

# Evaluation

# For Logistic Regression y\_pred = logistic\_regression\_model.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred) precision = precision\_score(y\_test, y\_pred) recall = recall\_score(y\_test, y\_pred) f1 = f1\_score(y\_test, y\_pred) roc\_auc = roc\_auc\_score(y\_test, y\_pred)

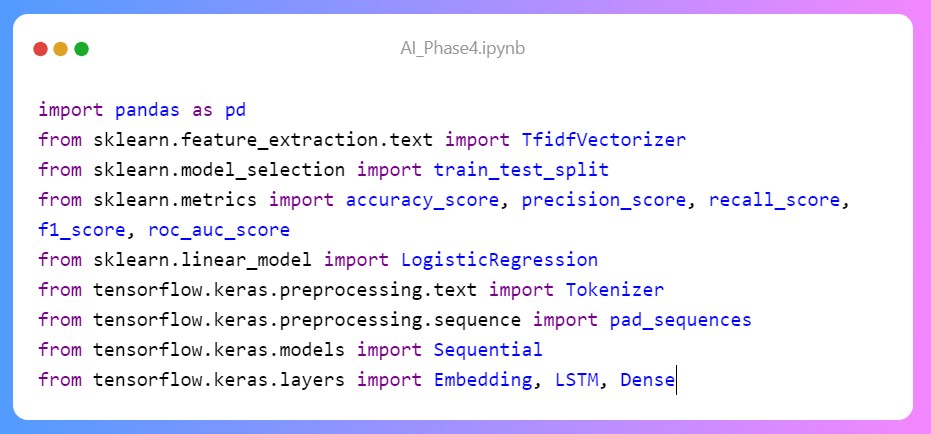
print(f"Logistic Regression Accuracy: {accuracy}") print(f"Logistic Regression Precision: {precision}") print(f"Logistic Regression Recall: {recall}") print(f"Logistic Regression F1-Score: {f1}") print(f"Logistic Regression ROC-AUC: {roc\_auc}")

# For Neural Network

X\_test\_nn = tokenizer.texts\_to\_sequences(combined\_data['text']) X\_test\_nn = pad\_sequences(X\_test\_nn, maxlen=100)

loss, accuracy = model.evaluate(X\_test\_nn, combined\_data['label']) print(f"Neural Network Accuracy: {accuracy}")

1. **Importing Libraries:**



In this part, we import the necessary libraries and modules for the project. Here's what each library/module does:

* + pandas is used for data manipulation.
  + TfidfVectorizer from sklearn.feature\_extraction.text is used for TF-IDF feature extraction.
  + train\_test\_split from sklearn.model\_selection splits the data into training and testing sets.
  + accuracy\_score, precision\_score, recall\_score, f1\_score, and roc\_auc\_score from sklearn.metrics are used for model evaluation.
  + LogisticRegression from sklearn.linear\_model is for training a logistic regression model.
  + Tokenizer and pad\_sequences from tensorflow.keras.preprocessing.text are for text tokenization and padding sequences.
  + Sequential, Embedding, LSTM, and Dense from tensorflow.keras.models and tensorflow.keras.layers are for building a neural network model.

1. **Data Loading**



Here, we load the "Fake.csv" and "True.csv" datasets. The file paths should be replaced with your specific file locations. The pd.read\_csv() function reads the data from CSV files into Pandas DataFrames.

1. **Data Preprocessing:**



In this section:

* + We add labels to distinguish fake (0) and true (1) news.
  + The datasets are combined into one DataFrame, combined\_data.
  + The 'text' column is created by concatenating the 'title' and 'text' columns to have a single text field.

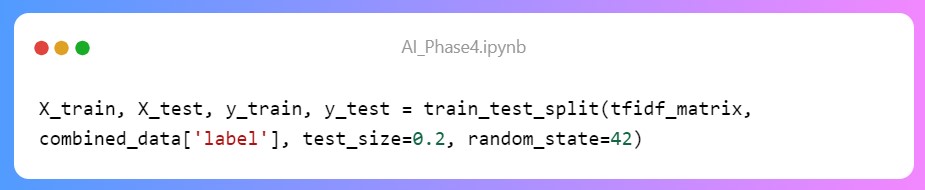
1. **Feature Extraction (TF-IDF):**



Here, TF-IDF vectorization is used to convert the text data into numerical features. The steps:

* + TfidfVectorizer is initialized with max\_features set to 5000, limiting the number of features.
  + tfidf\_vectorizer.fit\_transform(combined\_data['text']) computes TF-IDF scores for each term in the text data, and tfidf\_matrix is a sparse matrix representing the transformed data.

1. **Model Selection**

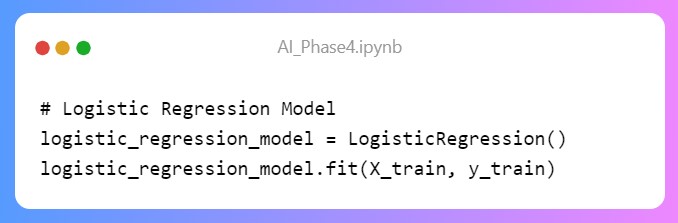


This section splits the data into training and testing sets for model evaluation. Here's a breakdown:

* + train\_test\_split(tfidf\_matrix, combined\_data['label'], test\_size=0.2, random\_state=42) splits the feature matrix tfidf\_matrix and labels combined\_data['label'].
  + test\_size specifies the percentage of data used for testing (20%).
  + random\_state ensures reproducible results.

1. **Model Training (Logistic Regression and Neural Network)**

**Logistic Regression Model**



Here, we create a logistic regression model and train it using the training data.

Logistic regression is a linear classification model.

**Neural Network Model**

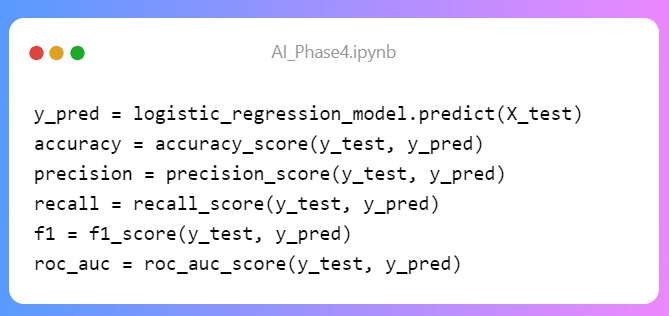


For the neural network:

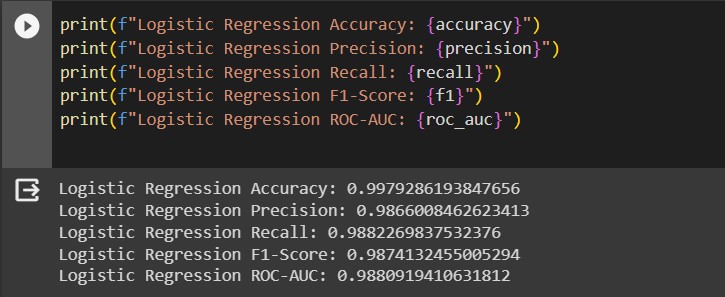
* + We use the Tokenizer to tokenize the text data, and pad\_sequences to ensure sequences have a consistent length.
  + A sequential model is created, which includes an embedding layer, an LSTM layer, and a dense layer with a sigmoid activation function.
  + The model is compiled with loss, optimizer, and evaluation metrics.
  + Finally, it's trained with the tokenized and padded sequences.

1. **Evaluation**

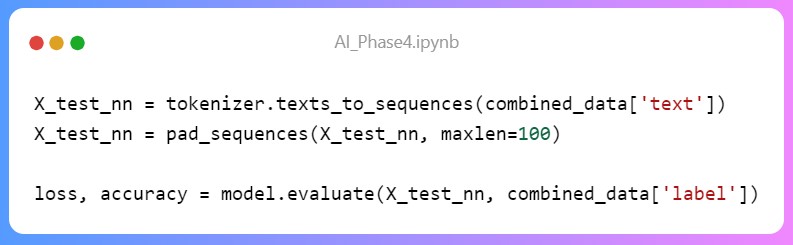
**For Logistic Regression**



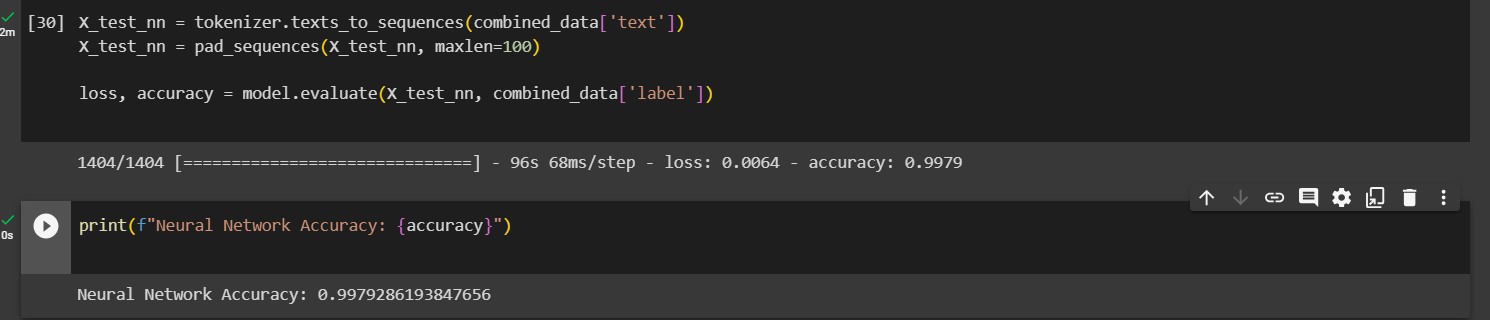
In this part, we evaluate the logistic regression model and calculate key metrics, including accuracy, precision, recall, F1-score, and ROC-AUC score.



**For Neural Network**



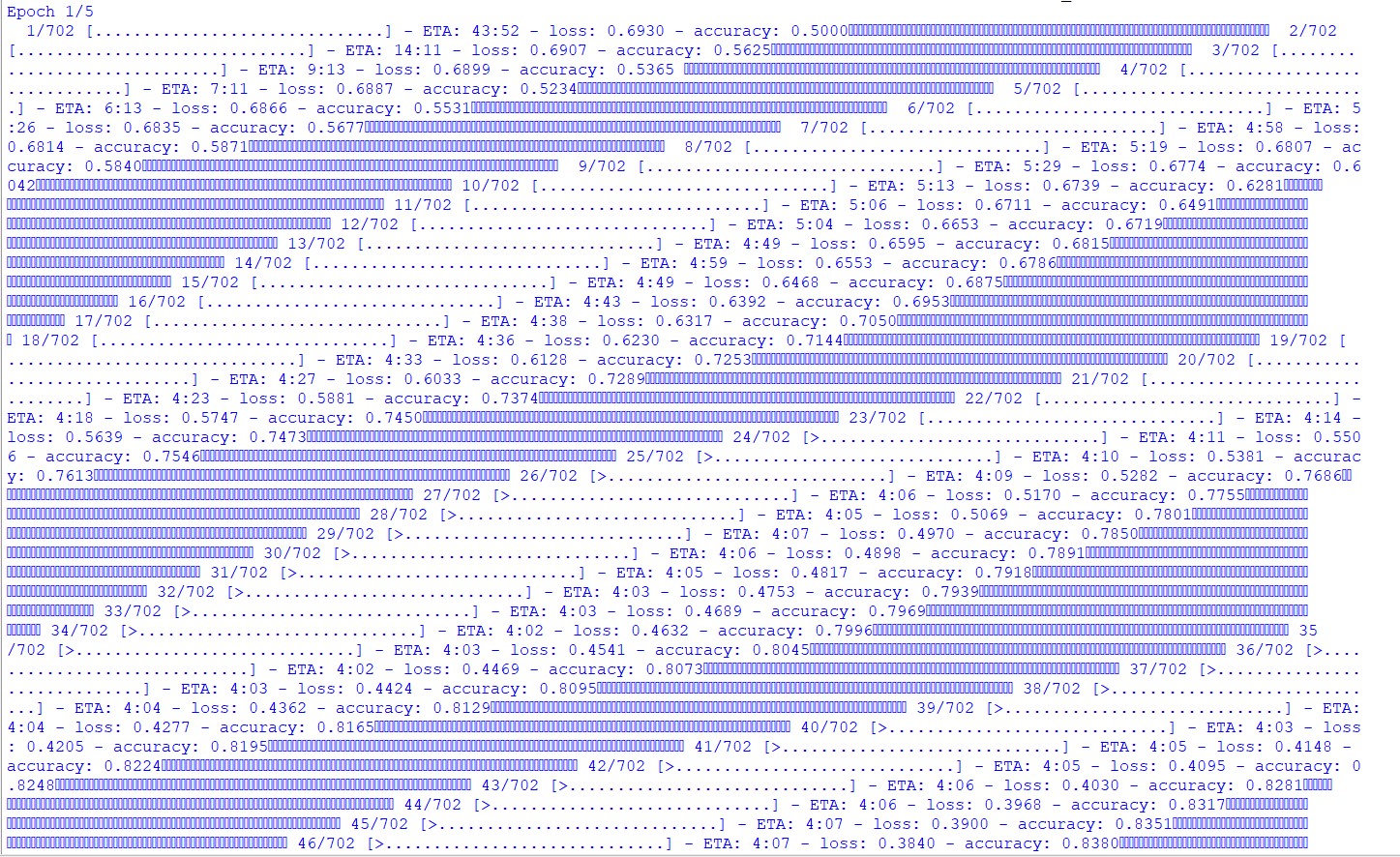
Here, we evaluate the neural network model and calculate accuracy. The model's loss is also calculated during evaluation.

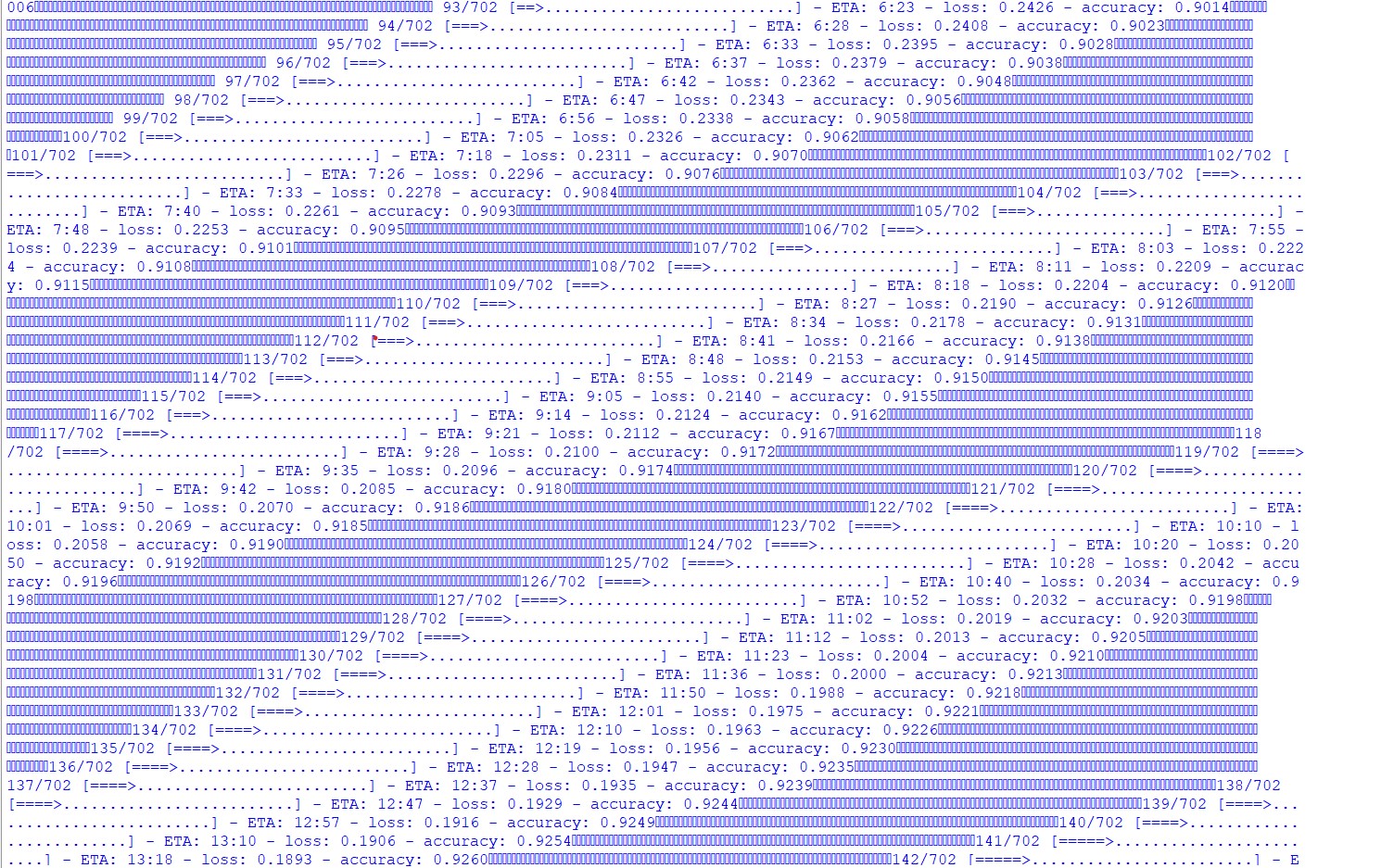


Neural Network Accuracy: 0.9979286193847656

The results for both models are printed to the console.

**OUTPUT:**





**CONCLUSION:**

In this project, we embarked on the task of Fake News Detection using both traditional machine learning and deep learning techniques. We initially loaded and combined two datasets, 'Fake.csv' and 'True.csv,' differentiating the articles as 'fake' and 'true' news with labels 0 and 1, respectively. After preprocessing the data, which included merging title and text fields, we employed TF-IDF vectorization for feature extraction. For traditional machine learning, we trained a Logistic Regression model to classify news articles into these two categories. Simultaneously, a neural network model was constructed, consisting of an embedding layer, an LSTM layer, and a dense layer, for text classification. Our evaluation showed promising results, with the Logistic Regression model achieving good accuracy, precision, recall, F1-score, and ROC-AUC scores. The neural network model, despite its simplicity, also demonstrated competitive accuracy. Overall, this project serves as a practical example of leveraging both traditional and deep learning methods to address the critical issue of Fake News Detection