```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, LeakyReLU, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
from tensorflow.keras.models import Sequential
```

```
!pip install --upgrade nltk
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('stopwords', download_dir='path_to_download_dir')
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.4)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package stopwords to path_to_download_dir...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
# Load data
data = pd.read_csv("/content/Data review bukalapak.csv")
```

```
# Preprocessing function
def preprocess_text(text):
    # Lowercase
    text = text.lower()
    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Tokenization
    tokens = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]
    # Join tokens
    text = ' '.join(tokens)
    return text
```

```
# Preprocess the 'Review' column
data['Review'] = data['Review'].apply(preprocess_text)
```

```
# Menampilkan nama-nama kolom yang ada dalam DataFrame
print(data.columns)
```

```
Index(['Nama akun pembeli', 'Review'], dtype='object')
```

```
# Find the unique words in the preprocessed text
unique_words = set(' '.join(data['Review']).split())
```

```
# Define constants
latent_dim = 100
input_dim = len(unique_words) # define the input dimension based on the preprocessing step
```

```
# Define the Generator model
def build_generator():
    model = Sequential([
        Dense(128, input_dim=latent_dim, activation='relu'),
        Dense(input_dim, activation='sigmoid')
    ])
    return model


# Define the Discriminator model
def build_discriminator():
    model = Sequential([
        Dense(256, input_dim=input_dim),
        LeakyReLU(alpha=0.2),
        Dropout(0.25),
        Dense(128),
        LeakyReLU(alpha=0.2),
        Dropout(0.25),
        Dense(1, activation='sigmoid')
    ])
    return model


# Define the GAN model
def build_gan(generator, discriminator):
    discriminator.trainable = False
    gan_input = tf.keras.Input(shape=(latent_dim,))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = tf.keras.Model(gan_input, gan_output)
    gan.compile(loss='binary_crossentropy', optimizer='adam')
    return gan


# Initialize models
generator = build_generator()
discriminator = build_discriminator()
gan = build_gan(generator, discriminator)
print(gan)
```

```
<keras.src.engine.functional.Functional object at 0x7d418312a1d0>
```

```
# Generate new data using the trained generator
generated_data = generator.predict(np.random.randn(len(data), latent_dim))
```

```
5/5 [==============================] - 0s 96ms/step
```

```
# Predict whether the generated data is fake or not using the discriminator model
predictions = discriminator.predict(generated_data)
print(predictions)
```

```
5/5 [==============================] - 1s 132ms/step
[[0.42562732]
 [0.42209017]
 [0.45168138]
 [0.43331918]
 [0.44047403]
 [0.43570492]
 [0.41654888]
 [0.41508403]
 [0.4312423 ]
 [0.41055572]
 [0.40570614]
 [0.37326247]
 [0.40959287]
 [0.40723768]
 [0.42138246]
 [0.39001286]
 [0.43284735]
 [0.43510497]
 [0.39616275]
 [0.44039392]
 [0.43254223]
 [0.40909404]
 [0.46117368]
 [0.4815804 ]
 [0.47249645]
 [0.43980923]
 [0.4204706 ]
 [0.41280973]
 [0.42398328]
 [0.42196417]
 [0.40689325]
 [0.44051164]
 [0.38750476]
 [0.4113773 ]
 [0.4101962 ]
 [0.40641475]
 [0.4191773 ]
 [0.41162422]
 [0.4357142 ]
 [0.4074462 ]
 [0.44413745]
 [0.42415917]
 [0.42026368]
 [0.40974772]
 [0.40102836]
 [0.4611908 ]
 [0.43296596]
 [0.44120422]
 [0.46217   ]
 [0.4416073 ]
 [0.42647028]
 [0.41388947]
 [0.42882234]
 [0.42031974]
 [0.4654425 ]
 [0.41634825]
```

ChatGPT

```
# Calculate the average prediction score
average_score = np.mean(predictions)
print("Average Prediction Score:", average_score)
```

```
Average Prediction Score: 0.42734727
```

```
# Convert the predictions to binary logic (1 for "not fake" and 0 for "fake")
binary_predictions = (predictions > average_score).astype(int)
```

```
# Map binary values to corresponding labels
labels = ["fake" if pred == 0 else "not fake" for pred in binary_predictions]
```

```
# Display the binary predictions
print("Binary Predictions:")
print(binary_predictions)
```

```
# Display the labels
print("Binary Predictions (0: fake, 1: not fake):", binary_predictions)
print("Labels:", labels)
```

```
Binary Predictions (0: fake, 1: not fake): [[0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [0]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
```