Customer segmentation Using Deep Learning Model (A005 , A014)

Importing Libraries The purpose of this step is to load all the necessary Python libraries and tools required for data processing, model building, and evaluation. These libraries include:

Pandas: For data manipulation and analysis (e.g., loading CSV files, cleaning data).

NumPy: For numerical operations and handling arrays.

TensorFlow/Keras: For building and training deep learning models.

Scikit-learn: For preprocessing (e.g., encoding labels) and evaluation (e.g., confusion matrix, classification report).

Matplotlib/Seaborn: For visualizing data and results (e.g., plotting confusion matrices).

OS/Requests/Zipfile: For downloading and extracting external resources like GloVe embeddings.

This step ensures that all dependencies are available for the subsequent steps in the pipeline.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM,
Dense, Dropout, GlobalMaxPooling1D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import os
import requests
import zipfile
```

The purpose of this step is to load the transaction dataset into a Pandas DataFrame for further processing. The dataset contains transaction descriptions, amounts, and customer IDs, which are essential for classifying transactions into industries and segmenting customers.

Input: A CSV file (synthetic_transactions_meaningful.csv) containing transaction data.

Output: A Pandas DataFrame (df) with columns like Description, Amount, and Customer ID.

This step is critical because the entire pipeline depends on the availability and quality of the dataset.

```
# Step 1: Load your dataset
df = pd.read_csv('formatted_transactions.csv')  # Replace with your
dataset path


df.head()

                         Transaction ID        Date   Amount  \
0  834a7bcb-fcb7-4746-8a8e-396631d84281  15-02-2025  6325.11
1  e28ac142-b5b6-41ab-b05a-2c51b02e07d5  16-09-2024   468.38
2  15c536f4-29b2-4ad1-a541-362f20f3b56e  02-11-2024  3632.53
3  e417b507-6256-49d9-88b8-a968dc86d842  12-02-2025  8696.02
4  fc5ade61-a07b-4f5c-8910-533af0a583e6  09-11-2024  9492.04

                   Transaction Description
0   Credit Card - Ola - Order##fihhg-ICICI
1      UPI - TataCliq - Order##nioDr-ICICI
2   Cash - RedBus - Order##tWjlg-Axis Bank
3      UPI - Zee5 - Order##yqGBV-Axis Bank
4  Net Banking - Rapido - Order##Llaos-SBI
```

Data Preprocessing The purpose of this step is to clean and preprocess the transaction descriptions to extract meaningful company names. This is done using a custom function extract_company, which:

Splits the transaction description on " - " to isolate the company name.

If no " - " is found, the entire description is treated as the company name.

Input: Raw transaction descriptions (e.g., "Payment - Netflix").

Output: A new column Company in the DataFrame containing cleaned company names (e.g., "Netflix").

This step ensures that the company names are standardized, making it easier to classify them into industries.

```
# Step 2: Data Preprocessing
def extract_company(desc):
    """
    Extracts the company name from the transaction description.
    Example: "Payment - Netflix" → "Netflix"
    """
    if ' - ' in desc:  # Check if the description contains " - "
        return desc.split(' - ')[1]  # Split the string and return the
second part
```

```
    return desc  # If no " - " is found, return the whole description

# Apply the extract_company function to the Transaction Description
column
df['Company'] = df['Transaction Description'].apply(extract_company)
df['Customer ID'] = np.random.randint(1, 501, size=len(df))  # Random
IDs between 1 and 500
```

Industry Classification Setup The purpose of this step is to define a mapping of industries to companies and create a training dataset for the deep learning model. This involves:

Defining a dictionary (INDUSTRY_MAPPING) where each key is an industry (e.g., "streaming") and the value is a list of companies in that industry.

Flattening this dictionary into a list of (company, industry) pairs and converting it into a DataFrame (train_df).

Input: Industry-to-company mappings.

Output: A training dataset (train_df) with columns Company and Industry.

This step provides labeled data for training the deep learning model.

```
# Step 3: Industry Classification Setup
INDUSTRY_MAPPING = {
    'streaming': [
        'Netflix', 'Hotstar', 'Zee5', 'SonyLiv', 'Prime Video',
'Disney+', 'Hulu', 'HBO Max', 'YouTube Premium',
        'Apple TV+', 'Voot', 'AltBalaji', 'MX Player', 'JioCinema',
'Viu', 'Discovery+', 'Lionsgate Play',
        'Sun NXT', 'Aha', 'Hoichoi', 'Eros Now', 'ShemarooMe',
'Hungama Play', 'DocuBay', 'Mubi', 'Crunchyroll',
        'Funimation', 'Paramount+', 'Peacock', 'Tubi'
    ],# List of companies in the streaming industry
    'food_delivery': [
        'Zomato', 'Swiggy', 'Uber Eats', 'DoorDash', 'Grubhub',
'Postmates', 'Deliveroo', 'Just Eat', 'Foodpanda',
        'Glovo', 'Seamless', 'Caviar', 'SkipTheDishes', 'Menulog',
'Delivery Hero', 'Talabat', 'iFood', 'Rappi',
        'Gojek', 'GrabFood', 'Dunzo', 'Faasos', 'Box8', 'FreshMenu',
'Domino\'s', 'Pizza Hut', 'McDonald\'s',
        'Burger King', 'KFC', 'Subway'
    ], # List of companies in the food delivery industry
    'ecommerce': [
        'Amazon', 'Flipkart', 'Myntra', 'Snapdeal', 'eBay', 'Walmart',
'Alibaba', 'Etsy', 'Shopify', 'Target',
        'Best Buy', 'Costco', 'JD.com', 'Rakuten', 'ShopClues', 'Paytm
Mall', 'Tata Cliq', 'Nykaa', 'Meesho',
        'Ajio', 'FirstCry', 'BigBasket', 'Grofers', 'Reliance
Digital', 'Croma', 'Vijay Sales', 'Lenskart',
```

```
        'Urban Ladder', 'Pepperfry', 'Zivame'
    ],
    'transportation': [
        'Ola', 'Uber', 'Rapido', 'Lyft', 'BlaBlaCar', 'Grab', 'Bolt',
'Didi', 'Gojek', 'Gett', 'Yandex.Taxi',
        'Careem', 'Curb', 'Via', 'Jugnoo', 'Meru Cabs', 'Easy Cabs',
'Savaari', 'Quick Ride', 'Shuttl', 'Zoomcar',
        'Revv', 'Drivezy', 'Avis', 'Hertz', 'Enterprise', 'Sixt',
'Budget', 'Zipcar', 'Turo'
    ],
    'health_fitness': [
        'Cult.fit', 'HealthifyMe', 'Fitbit', 'MyFitnessPal',
'Peloton', 'ClassPass', 'Gymshark', 'Zwift', 'Strava',
        'Headspace', 'Calm', 'Noom', 'Freeletics', 'Daily Burn',
'8fit', 'Aaptiv', 'FitOn', 'Glo', 'Obé Fitness',
        'Beachbody', 'Fitness Blender', 'YogaGlo', 'Down Dog', 'Alo
Moves', 'Centr', 'Fiit', 'Les Mills On Demand',
        'Nike Training Club', 'Adidas Training'
    ],
    'travel': [
        'MakeMyTrip', 'Yatra', 'ClearTrip', 'Booking.com', 'Expedia',
'Airbnb', 'TripAdvisor', 'Kayak', 'Agoda',
        'Skyscanner', 'Goibibo', 'Trivago', 'CheapOair', 'Priceline',
'Hotwire', 'Travelocity', 'Orbitz', 'Vrbo',
        'Hostelworld', 'Thomas Cook', 'SOTC', 'Cox & Kings', 'Kesari
Tours', 'Club Mahindra', 'OYO', 'Treebo',
        'FabHotels', 'Lemon Tree Hotels', 'Taj Hotels',
'Marriott','RedBus'
    ],
    'entertainment': [
        'Spotify', 'BookMyShow', 'Netflix', 'Disney+', 'Hulu',
'YouTube', 'Twitch', 'TikTok', 'Instagram',
        'Facebook', 'Snapchat', 'Pinterest', 'Reddit', 'Tumblr',
'SoundCloud', 'Gaana', 'JioSaavn', 'Wynk Music',
        'Hungama', 'Spotify', 'Apple Music', 'Amazon Music',
'Pandora', 'Deezer', 'Tidal', 'Bandcamp', 'Mixcloud',
        'iHeartRadio', 'Audible', 'Podbean'
    ],
    'banking': [
        'ICICI', 'SBI', 'HDFC', 'Axis Bank', 'Kotak Mahindra',
'Citibank', 'HSBC', 'Standard Chartered',
        'Bank of America', 'Chase', 'Wells Fargo', 'Barclays',
'Deutsche Bank', 'BNP Paribas', 'Santander',
        'RBS', 'UBS', 'Credit Suisse', 'DBS', 'PNB', 'Canara Bank',
'Bank of Baroda', 'IDBI Bank', 'Yes Bank',
        'IndusInd Bank', 'RBL Bank', 'Federal Bank', 'Karur Vysya
Bank', 'South Indian Bank', 'Union Bank'
    ],
    'telecom': [
```

```
        'Airtel', 'Jio', 'Vodafone', 'Verizon', 'AT&T', 'T-Mobile',
'Sprint', 'MTN', 'Orange', 'Telstra',
        'Deutsche Telekom', 'China Mobile', 'NTT', 'SoftBank',
'Telefonica', 'BT', 'Comcast', 'Reliance Jio',
        'BSNL', 'Idea', 'Tata Docomo', 'Aircel', 'MTS', 'Uninor',
'Videocon', 'Loop Mobile', 'Telenor',
        'Docomo Pacific', 'Digicel', 'Globe Telecom'
    ],
    'automotive': [
        'Tesla', 'Toyota', 'Ford', 'BMW', 'Mercedes-Benz', 'Honda',
'Hyundai', 'Maruti Suzuki', 'Tata Motors',
        'Volkswagen', 'Audi', 'Nissan', 'Chevrolet', 'Kia', 'Renault',
'Porsche', 'Jaguar', 'Land Rover',
        'Volvo', 'Fiat', 'Mazda', 'Subaru', 'Mitsubishi', 'Lexus',
'Skoda', 'Mahindra', 'Force Motors',
        'Ashok Leyland', 'Eicher Motors', 'Bajaj Auto'
    ],
    'technology': [
        'Apple', 'Microsoft', 'Google', 'Samsung', 'Intel', 'IBM',
'Oracle', 'Adobe', 'Sony', 'Dell', 'HP',
        'Lenovo', 'Asus', 'Acer', 'Toshiba', 'Canon', 'Nikon',
'Panasonic', 'LG', 'Huawei', 'Xiaomi', 'OnePlus',
        'Qualcomm', 'NVIDIA', 'AMD', 'Broadcom', 'Cisco', 'Ericsson',
'SAP', 'Infosys'
    ],
    'social_media': [
        'Facebook', 'Instagram', 'Twitter', 'LinkedIn', 'Snapchat',
'Pinterest', 'Reddit', 'TikTok', 'WhatsApp',
        'Telegram', 'Signal', 'WeChat', 'Line', 'Viber', 'Discord',
'Clubhouse', 'Tumblr', 'Quora', 'Medium',
        'Nextdoor', 'Meetup', 'Flickr', 'VK', 'Weibo', 'QQ',
'MySpace', 'Tagged', 'Badoo', 'Hi5', 'Friendster'
    ],
    'retail': [
        'Walmart', 'Target', 'Costco', 'Home Depot', 'IKEA', 'Best
Buy', 'Tesco', 'Carrefour', 'Aldi', '7-Eleven',
        'Reliance Retail', 'Future Group', 'D-Mart', 'Spencer\'s',
'Big Bazaar', 'More Retail', 'Shoppers Stop',
        'Lifestyle', 'Pantaloons', 'Westside', 'Landmark Group',
'SPAR', 'HyperCity', 'Star Bazaar', 'Vishal Mega Mart',
        'Easyday', 'Nilgiris', 'Foodworld', 'Reliance Fresh'
    ],
    'education': [
        'Byju\'s', 'Unacademy', 'Coursera', 'Udemy', 'Khan Academy',
'Duolingo', 'edX', 'Vedantu', 'Toppr',
        'WhiteHat Jr', 'Simplilearn', 'UpGrad', 'Great Learning',
'Eruditus', 'Emeritus', 'Skillshare', 'Pluralsight',
        'LinkedIn Learning', 'MasterClass', 'Udacity', 'Codecademy',
'DataCamp', 'Coding Ninjas', 'Scaler Academy',
```

```
        'Camp K12', 'Extramarks', 'TopperLearning', 'Meritnation',
'Embibe', 'Adda247'
    ],
    'gaming': [
        'Steam', 'Epic Games', 'PlayStation', 'Xbox', 'Nintendo', 'EA
Sports', 'Ubisoft', 'Blizzard', 'Riot Games',
        'Zynga', 'Activision', 'Take-Two Interactive', 'Square Enix',
'Capcom', 'Bandai Namco', 'Sega', 'Konami',
        'NetEase', 'Tencent Games', 'Garena', 'Gameloft', 'Supercell',
'King', 'Niantic', 'Roblox', 'Minecraft',
        'Fortnite', 'PUBG', 'Valorant', 'Call of Duty'
    ],
    'food_beverage': [
        'McDonald\'s', 'Starbucks', 'KFC', 'Domino\'s', 'Pizza Hut',
'Subway', 'Burger King', 'Coca-Cola',
        'Pepsi', 'Nestle', 'Unilever', 'P&G', 'Kraft Heinz', 'General
Mills', 'Kellogg\'s', 'Mondelez', 'Danone',
        'Tyson Foods', 'Hershey\'s', 'Campbell Soup', 'Conagra
Brands', 'JBS', 'Cargill', 'ADM', 'Tata Consumer',
        'Britannia', 'Amul', 'Parle', 'ITC Foods', 'Haldiram\'s'
    ],
    'pharmaceuticals': [
        'Pfizer', 'Johnson & Johnson', 'Novartis', 'Roche', 'Merck',
'GSK', 'Sanofi', 'Abbott', 'AstraZeneca',
        'Bayer', 'Novo Nordisk', 'Bristol-Myers Squibb', 'Eli Lilly',
'Amgen', 'Gilead Sciences', 'Biogen',
        'Teva', 'Takeda', 'Aurobindo Pharma', 'Sun Pharma', 'Dr.
Reddy\'s', 'Cipla', 'Lupin', 'Zydus Cadila',
        'Torrent Pharma', 'Glenmark', 'Biocon', 'Divis Labs', 'Alkem
Labs', 'Ipca Labs'
    ],
    'insurance': [
        'LIC', 'ICICI Prudential', 'HDFC Life', 'SBI Life', 'Max
Life', 'Bajaj Allianz', 'New India Assurance',
        'United India', 'National Insurance', 'Reliance General',
'Tata AIG', 'Chola MS', 'Star Health', 'Care Health',
        'ManipalCigna', 'Aditya Birla Health', 'Kotak Life', 'Aviva
Life', 'Exide Life', 'Bharti AXA', 'Future Generali',
        'Royal Sundaram', 'Shriram Life', 'Canara HSBC', 'Pramerica
Life', 'IDBI Federal', 'Edelweiss Tokio',
        'Aegon Life', 'IndiaFirst Life', 'PNB MetLife'
    ],
    'logistics': [
        'FedEx', 'UPS', 'DHL', 'Blue Dart', 'DTDC', 'Amazon
Logistics', 'Delhivery', 'Ecom Express', 'XpressBees',
        'Shadowfax', 'Gati', 'SafeExpress', 'Aramex', 'TNT Express',
'SF Express', 'YTO Express', 'ZTO Express',
        'STO Express', 'Best Express', 'Yunda Express', 'Kerry
Logistics', 'Nippon Express', 'DB Schenker',
```

```
        'Kuehne + Nagel', 'CEVA Logistics', 'Agility', 'XPO
Logistics', 'DSV', 'Panalpina', 'Geodis'
    ],
    'real_estate': [
        'PropTiger', 'MagicBricks', '99acres', 'Housing.com', 'Square
Yards', 'CommonFloor', 'Makaan.com', 'Zillow',
        'Realtor.com', 'Redfin', 'Trulia', 'Opendoor', 'Compass',
'WeWork', 'Knight Frank', 'CBRE', 'JLL', 'Colliers',
        'Savills', 'Cushman & Wakefield', 'RE/MAX', 'Century 21',
'Coldwell Banker', 'ERA Real Estate', 'Sotheby\'s',
        'Christie\'s International', 'Engel & Völkers', 'Luxury
Portfolio', 'Douglas Elliman', 'Corcoran Group'
    ],
    'energy': [
        'Reliance Industries', 'BP', 'Shell', 'ExxonMobil', 'Chevron',
'Total', 'Indian Oil', 'HPCL', 'BPCL',
        'NTPC', 'ONGC', 'GAIL', 'Adani Power', 'Tata Power', 'NLC
India', 'NHPC', 'SJVN', 'Power Grid Corporation',
        'Siemens', 'GE Power', 'ABB', 'Schneider Electric', 'Vestas',
'Suzlon', 'First Solar', 'SunPower',
        'Canadian Solar', 'Trina Solar', 'JinkoSolar', 'Enphase
Energy'
    ],
    'aviation': [
        'Indigo', 'Air India', 'SpiceJet', 'Vistara', 'Emirates',
'Qatar Airways', 'Singapore Airlines', 'Delta',
        'American Airlines', 'Lufthansa', 'British Airways', 'Air
France', 'KLM', 'Cathay Pacific', 'Qantas',
        'Turkish Airlines', 'Etihad Airways', 'ANA', 'Japan Airlines',
'Korean Air', 'Thai Airways', 'Malaysia Airlines',
        'AirAsia', 'Ryanair', 'EasyJet', 'Southwest Airlines',
'JetBlue', 'Alaska Airlines', 'United Airlines',
        'Air Canada'
    ],
    'hospitality': [
        'Marriott', 'Hilton', 'Hyatt', 'Taj Hotels', 'Oberoi',
'Airbnb', 'Booking.com', 'Expedia', 'Trivago',
        'Agoda', 'InterContinental', 'Accor', 'Wyndham', 'Choice
Hotels', 'Radisson', 'Shangri-La', 'Four Seasons',
        'Mandarin Oriental', 'Rosewood', 'Fairmont', 'Ritz-Carlton',
'St. Regis', 'W Hotels', 'Westin', 'Sheraton',
        'Holiday Inn', 'Crowne Plaza', 'Novotel', 'ibis', 'Best
Western'
    ],
    'fashion': [
        'Zara', 'H&M', 'Nike', 'Adidas', 'Puma', 'Levi\'s', 'Gucci',
'Louis Vuitton', 'Prada', 'Uniqlo',
        'Forever 21', 'Gap', 'Tommy Hilfiger', 'Calvin Klein', 'Ralph
Lauren', 'Burberry', 'Chanel', 'Dior',
```

```
        'Versace', 'Armani', 'Dolce & Gabbana', 'Fendi', 'Balenciaga',
'Givenchy', 'Yves Saint Laurent',
        'Michael Kors', 'Coach', 'Kate Spade', 'Tory Burch', 'Jimmy
Choo'
    ],
    'media_entertainment': [
        'Disney', 'Warner Bros', 'Sony Pictures', 'Netflix', 'Amazon
Prime Video', 'HBO', 'BBC', 'CNN', 'Fox',
        'ViacomCBS', 'Comcast', 'NBCUniversal', 'Paramount',
'Discovery', 'AMC Networks', 'Lionsgate', 'MGM',
        'A24', 'STX Entertainment', 'Legendary Entertainment',
'DreamWorks', 'Pixar', 'Marvel Studios',
        'DC Films', 'Universal Pictures', '20th Century Studios',
'Miramax', 'Focus Features', 'A&E Networks',
        'Showtime'
    ]
}

# Create a training dataset for the deep learning model
all_companies = [(company, industry)
                 for industry, companies in INDUSTRY_MAPPING.items()
                 for company in companies]
train_df = pd.DataFrame(all_companies, columns=['Company',
'Industry'])
```

Downloading and Loading GloVe Embeddings The purpose of this step is to download and load pre-trained GloVe word embeddings, which are used to represent company names as dense vectors. This involves:

Downloading the GloVe embeddings from Stanford's website if they are not already available locally.

Extracting the embeddings and loading them into a dictionary (embeddings_index), where each word maps to its corresponding vector.

Input: GloVe embeddings file (glove.6B.100d.txt).

Output: A dictionary (embeddings_index) containing word vectors.

This step is crucial because word embeddings capture semantic relationships between words, which improves the model's ability to understand and classify company names.

```
# Step 4: Download and load GloVe embeddings
def download_glove_embeddings():
    glove_url = "http://nlp.stanford.edu/data/glove.6B.zip"
    glove_dir = "glove"
    os.makedirs(glove_dir, exist_ok=True)
    glove_zip_path = os.path.join(glove_dir, "glove.6B.zip")
```

```
    # Download GloVe embeddings if not already downloaded
    if not os.path.exists(glove_zip_path):
        print("Downloading GloVe embeddings...")
        response = requests.get(glove_url, stream=True)
        with open(glove_zip_path, "wb") as f:
            for chunk in response.iter_content(chunk_size=128):
                f.write(chunk)
        print("Download complete.")

    # Extract GloVe embeddings
    glove_extracted_path = os.path.join(glove_dir,
"glove.6B.100d.txt")
    if not os.path.exists(glove_extracted_path):
        print("Extracting GloVe embeddings...")
        with zipfile.ZipFile(glove_zip_path, 'r') as zip_ref:
            zip_ref.extractall(glove_dir)
        print("Extraction complete.")

    return glove_extracted_path

# Load GloVe embeddings
glove_path = download_glove_embeddings()
embeddings_index = {}
with open(glove_path, encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
```

Preparing the Embedding Matrix The purpose of this step is to create an embedding matrix that maps each word in the vocabulary to its corresponding GloVe vector. This involves:

Tokenizing the company names using Keras' Tokenizer.

Converting the tokenized sequences into padded sequences of fixed length (max_length).

Creating an embedding matrix where each row corresponds to a word in the vocabulary.

Input: Tokenized company names and GloVe embeddings.

Output: An embedding matrix (embedding_matrix) and padded sequences (padded_sequences).

This step ensures that the input data is in a format suitable for training the deep learning model.

```
# Step 5: Prepare embedding matrix
vocab_size = 10000  # Use the top 10,000 words
embedding_dim = 100  # GloVe embedding dimension
max_length = 50  # Maximum length of input sequences
```

```
# Tokenize company names
tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
tokenizer.fit_on_texts(train_df['Company'])
sequences = tokenizer.texts_to_sequences(train_df['Company'])
padded_sequences =
tf.keras.preprocessing.sequence.pad_sequences(sequences,
maxlen=max_length, padding='post', truncating='post')

# Create embedding matrix
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    if i < vocab_size:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector


# Step 6: Encode industry labels
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(train_df['Industry'])
```

Building the Model The purpose of this step is to define the architecture of the deep learning model. The model consists of:

An Embedding layer that uses the pre-trained GloVe embeddings.

A Bidirectional LSTM layer to capture sequential patterns in the company names.

A Global Max Pooling layer to reduce the sequence to a single vector.

A Dense layer with ReLU activation for feature extraction.

A Dropout layer to prevent overfitting.

A Softmax output layer for multi-class classification.

Input: Padded sequences of company names.

Output: A compiled Keras model ready for training.

This step defines the core of the classification pipeline, enabling the model to learn from the data.

```
# Step 7: Build the Bidirectional LSTM model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=max_length, weights=[embedding_matrix], trainable=False),
    Bidirectional(LSTM(128, return_sequences=True)),  # Bidirectional
LSTM layer
    GlobalMaxPooling1D(),  # Global max pooling to reduce sequence to
a single vector
```

```
    Dense(64, activation='relu'),  # Fully connected layer
    Dropout(0.5),  # Dropout to prevent overfitting
    Dense(len(label_encoder.classes_), activation='softmax')  # Output
layer
])

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=Adam(learning_rate=0.001),  # Use Adam
optimizer with a lower learning rate
              metrics=['accuracy'])
```

```
C:\Users\arpan\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
# Step 8: Define callbacks
#early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)  # Increase patience
#reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=3, min_lr=1e-6)  # Learning rate scheduler
```

Training the Model The purpose of this step is to train the deep learning model on the labeled dataset. This involves:

Splitting the data into training and validation sets.

Training the model for a fixed number of epochs (epochs=50).

Monitoring the training and validation loss/accuracy to ensure the model is learning effectively.

Input: Padded sequences and corresponding industry labels.

Output: A trained model (model) and training history (history).

This step is where the model learns to classify company names into industries based on the provided data.

```
# Step 9: Train the model
history = model.fit(padded_sequences, labels, epochs=50,
validation_split=0.2, verbose=1)
'''
history = model.fit(
    padded_sequences,
    labels,
    epochs=50,
```

```
    validation_split=0.2,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)
'''
```

```
Epoch 1/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 18s 155ms/step - accuracy: 0.0510 - loss:
3.2001 - val_accuracy: 0.0000e+00 - val_loss: 3.4228
Epoch 2/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 0.1544 - loss:
3.0387 - val_accuracy: 0.0000e+00 - val_loss: 3.9137
Epoch 3/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 78ms/step - accuracy: 0.2234 - loss:
2.8459 - val_accuracy: 0.0000e+00 - val_loss: 4.4303
Epoch 4/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 83ms/step - accuracy: 0.3218 - loss:
2.5589 - val_accuracy: 0.0000e+00 - val_loss: 5.3728
Epoch 5/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 88ms/step - accuracy: 0.3572 - loss:
2.2513 - val_accuracy: 0.0000e+00 - val_loss: 5.6456
Epoch 6/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - accuracy: 0.3740 - loss:
2.1216 - val_accuracy: 0.0000e+00 - val_loss: 6.8418
Epoch 7/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 93ms/step - accuracy: 0.4307 - loss:
1.9163 - val_accuracy: 0.0000e+00 - val_loss: 7.3291
Epoch 8/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 116ms/step - accuracy: 0.4932 - loss:
1.6852 - val_accuracy: 0.0000e+00 - val_loss: 7.7280
Epoch 9/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 88ms/step - accuracy: 0.5114 - loss:
1.6328 - val_accuracy: 0.0000e+00 - val_loss: 8.3733
Epoch 10/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 96ms/step - accuracy: 0.5255 - loss:
1.5242 - val_accuracy: 0.0000e+00 - val_loss: 9.0812
Epoch 11/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 109ms/step - accuracy: 0.5511 - loss:
1.4176 - val_accuracy: 0.0000e+00 - val_loss: 9.5014
Epoch 12/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 116ms/step - accuracy: 0.5862 - loss:
1.3354 - val_accuracy: 0.0000e+00 - val_loss: 9.4217
Epoch 13/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 118ms/step - accuracy: 0.6225 - loss:
1.2040 - val_accuracy: 0.0000e+00 - val_loss: 10.1236
Epoch 14/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 130ms/step - accuracy: 0.5877 - loss:
1.2613 - val_accuracy: 0.0000e+00 - val_loss: 9.9552
Epoch 15/50
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 125ms/step - accuracy: 0.5732 - loss:
1.3269 - val_accuracy: 0.0000e+00 - val_loss: 10.4743
Epoch 16/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 101ms/step - accuracy: 0.6483 - loss:
1.1570 - val_accuracy: 0.0000e+00 - val_loss: 11.5599
Epoch 17/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.6352 - loss:
1.1379 - val_accuracy: 0.0000e+00 - val_loss: 11.2226
Epoch 18/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 112ms/step - accuracy: 0.6579 - loss:
1.0486 - val_accuracy: 0.0000e+00 - val_loss: 12.0239
Epoch 19/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.6796 - loss:
1.0639 - val_accuracy: 0.0000e+00 - val_loss: 12.1038
Epoch 20/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 111ms/step - accuracy: 0.6991 - loss:
1.0244 - val_accuracy: 0.0000e+00 - val_loss: 11.8242
Epoch 21/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - accuracy: 0.6686 - loss:
1.0395 - val_accuracy: 0.0000e+00 - val_loss: 12.9972
Epoch 22/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 92ms/step - accuracy: 0.6945 - loss:
0.9590 - val_accuracy: 0.0000e+00 - val_loss: 13.2367
Epoch 23/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 111ms/step - accuracy: 0.6962 - loss:
0.9746 - val_accuracy: 0.0000e+00 - val_loss: 13.0192
Epoch 24/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 86ms/step - accuracy: 0.7377 - loss:
0.8781 - val_accuracy: 0.0000e+00 - val_loss: 13.2686
Epoch 25/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.7292 - loss:
0.9329 - val_accuracy: 0.0000e+00 - val_loss: 13.6748
Epoch 26/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step - accuracy: 0.7251 - loss:
0.8567 - val_accuracy: 0.0000e+00 - val_loss: 14.0973
Epoch 27/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 88ms/step - accuracy: 0.7360 - loss:
0.8872 - val_accuracy: 0.0000e+00 - val_loss: 14.5430
Epoch 28/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - accuracy: 0.7584 - loss:
0.8428 - val_accuracy: 0.0000e+00 - val_loss: 14.7211
Epoch 29/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 128ms/step - accuracy: 0.7164 - loss:
0.9292 - val_accuracy: 0.0000e+00 - val_loss: 15.4968
Epoch 30/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 116ms/step - accuracy: 0.7380 - loss:
0.8199 - val_accuracy: 0.0000e+00 - val_loss: 15.6155
Epoch 31/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 84ms/step - accuracy: 0.7500 - loss:
```

```
0.8343 - val_accuracy: 0.0000e+00 - val_loss: 15.4597
Epoch 32/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 134ms/step - accuracy: 0.7426 - loss:
0.8034 - val_accuracy: 0.0000e+00 - val_loss: 15.9790
Epoch 33/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 130ms/step - accuracy: 0.7390 - loss:
0.8616 - val_accuracy: 0.0000e+00 - val_loss: 16.2048
Epoch 34/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 127ms/step - accuracy: 0.7496 - loss:
0.7924 - val_accuracy: 0.0000e+00 - val_loss: 16.0007
Epoch 35/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 110ms/step - accuracy: 0.7645 - loss:
0.8350 - val_accuracy: 0.0000e+00 - val_loss: 15.8972
Epoch 36/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 118ms/step - accuracy: 0.7435 - loss:
0.8368 - val_accuracy: 0.0000e+00 - val_loss: 16.7576
Epoch 37/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 119ms/step - accuracy: 0.7691 - loss:
0.7856 - val_accuracy: 0.0000e+00 - val_loss: 15.9285
Epoch 38/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 135ms/step - accuracy: 0.7391 - loss:
0.7595 - val_accuracy: 0.0000e+00 - val_loss: 17.0396
Epoch 39/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step - accuracy: 0.7358 - loss:
0.7933 - val_accuracy: 0.0000e+00 - val_loss: 16.1910
Epoch 40/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - accuracy: 0.7457 - loss:
0.8218 - val_accuracy: 0.0000e+00 - val_loss: 15.9765
Epoch 41/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.7422 - loss:
0.8101 - val_accuracy: 0.0000e+00 - val_loss: 15.7825
Epoch 42/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.7427 - loss:
0.7832 - val_accuracy: 0.0000e+00 - val_loss: 16.1614
Epoch 43/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 55ms/step - accuracy: 0.7461 - loss:
0.7603 - val_accuracy: 0.0000e+00 - val_loss: 16.4001
Epoch 44/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 93ms/step - accuracy: 0.7402 - loss:
0.7678 - val_accuracy: 0.0000e+00 - val_loss: 17.5091
Epoch 45/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 89ms/step - accuracy: 0.7544 - loss:
0.7663 - val_accuracy: 0.0000e+00 - val_loss: 17.6217
Epoch 46/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - accuracy: 0.7727 - loss:
0.7445 - val_accuracy: 0.0000e+00 - val_loss: 17.7602
Epoch 47/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - accuracy: 0.7289 - loss:
0.7870 - val_accuracy: 0.0000e+00 - val_loss: 18.0925
```

```
Epoch 48/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 79ms/step - accuracy: 0.7706 - loss:
0.7060 - val_accuracy: 0.0000e+00 - val_loss: 18.6677
Epoch 49/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 90ms/step - accuracy: 0.7877 - loss:
0.6822 - val_accuracy: 0.0000e+00 - val_loss: 18.4620
Epoch 50/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 85ms/step - accuracy: 0.7799 - loss:
0.6729 - val_accuracy: 0.0000e+00 - val_loss: 18.6656

'\nhistory = model.fit(\n    padded_sequences, \n    labels, \n
epochs=50, \n    validation_split=0.2, \n
callbacks=[early_stopping, reduce_lr], \n    verbose=1\n)\n'
```

Evaluating the Model The purpose of this step is to assess the performance of the trained model. This involves:

Generating predictions on the training data.

Creating a confusion matrix to visualize the model's performance.

Generating a classification report with metrics like precision, recall, and F1-score.

Input: Trained model and padded sequences.

Output: Confusion matrix and classification report.

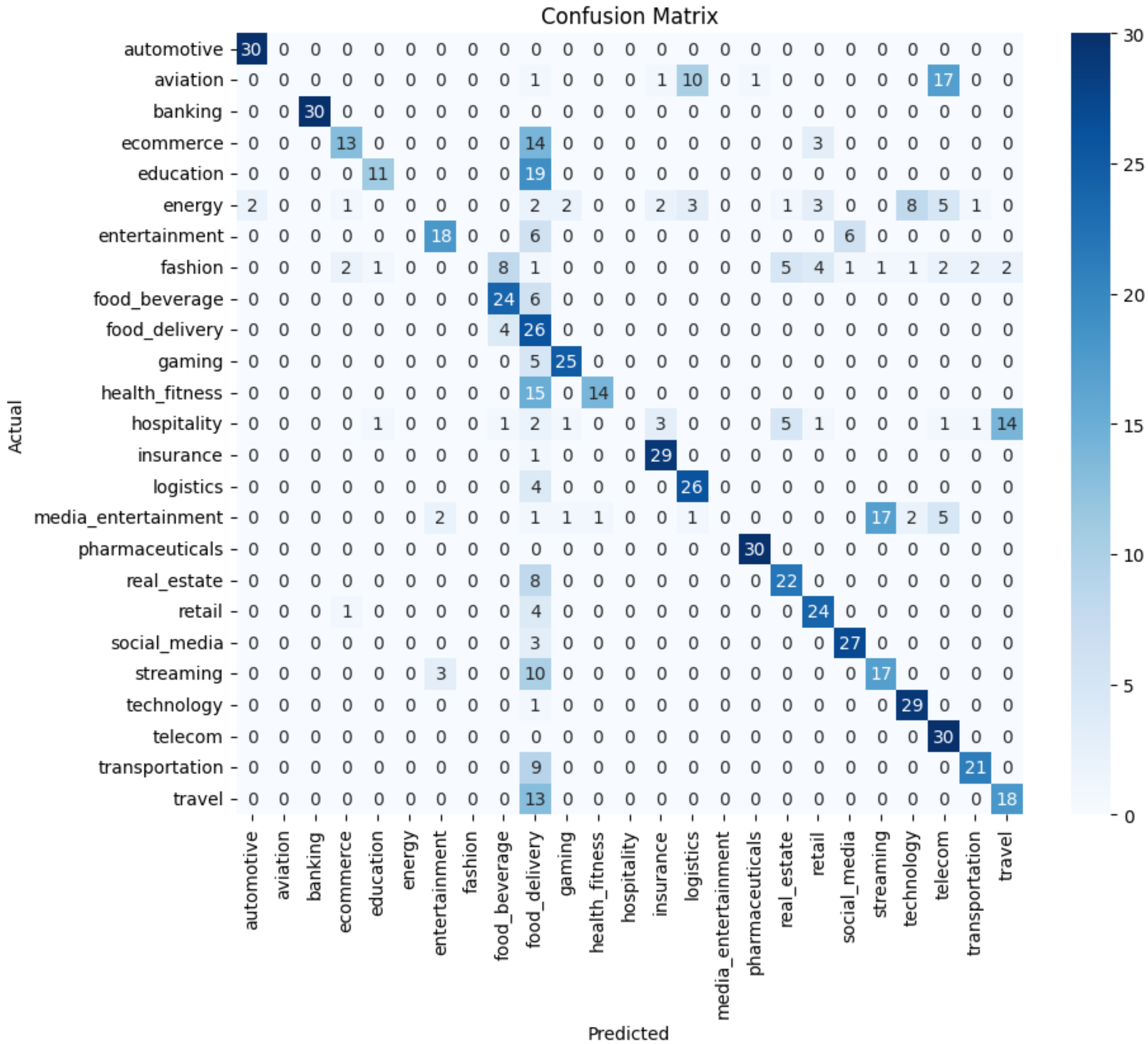
This step provides insights into how well the model is performing and identifies areas for improvement.

```python
# Step 10: Evaluate the model
y_pred = model.predict(padded_sequences)
y_pred_classes = np.argmax(y_pred, axis=1)

# Confusion Matrix
conf_matrix = confusion_matrix(labels, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.savefig('confusion_matrix_lstm.png')
plt.show()

# Classification Report
print("Classification Report:")
print(classification_report(labels, y_pred_classes,
target_names=label_encoder.classes_))
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step
```



Classification Report:

	precision	recall	f1-score	support
automotive	0.94	1.00	0.97	30
aviation	0.00	0.00	0.00	30
banking	1.00	1.00	1.00	30
ecommerce	0.76	0.43	0.55	30
education	0.85	0.37	0.51	30
energy	0.00	0.00	0.00	30
entertainment	0.78	0.60	0.68	30
fashion	0.00	0.00	0.00	30
food_beverage	0.65	0.80	0.72	30
food_delivery	0.17	0.87	0.29	30

```
               gaming       0.86      0.83      0.85        30
       health_fitness       0.93      0.48      0.64        29
          hospitality       0.00      0.00      0.00        30
            insurance       0.83      0.97      0.89        30
            logistics       0.65      0.87      0.74        30
  media_entertainment       0.00      0.00      0.00        30
      pharmaceuticals       0.97      1.00      0.98        30
          real_estate       0.67      0.73      0.70        30
               retail       0.69      0.83      0.75        29
         social_media       0.79      0.90      0.84        30
            streaming       0.49      0.57      0.52        30
           technology       0.72      0.97      0.83        30
              telecom       0.50      1.00      0.67        30
       transportation       0.84      0.70      0.76        30
               travel       0.53      0.58      0.55        31

             accuracy                           0.62       749
            macro avg       0.58      0.62      0.58       749
         weighted avg       0.58      0.62      0.58       749


C:\Users\arpan\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\metrics\_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\arpan\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\metrics\_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\arpan\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\metrics\_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

Classifying Companies The purpose of this step is to classify companies in the dataset into industries using a combination of deep learning predictions and rule-based classification. This involves:

Defining a CompanyClassifier class that:

Uses the trained model for predictions if the confidence is high.

Falls back to keyword-based rules if the confidence is low.

Applying the classifier to the Company column in the dataset.

Input: Company names in the dataset.

Output: A new column Industry in the DataFrame containing predicted industries.

This step ensures that all companies are classified, even if the model's confidence is low.

```
# Step 11: Classify companies in the dataset
class CompanyClassifier:
    def __init__(self):
        self.model = model
        self.tokenizer = tokenizer
        self.label_encoder = label_encoder
        self.keyword_rules = {
    'streaming': [
        'stream', 'flix', 'tv', 'video', 'movie', 'series', 'show',
'watch', 'entertainment', 'on demand',
        'premium', 'subscription', 'channel', 'episode', 'season',
'documentary', 'anime', 'cartoon',
        'live', 'broadcast', 'vod', 'ott', 'platform', 'library',
'catalog', 'originals', 'replay', 'binge'
    ],
    'food_delivery': [
        'food', 'eats', 'restaurant', 'meal', 'delivery', 'takeout',
'takeaway', 'cuisine', 'dine', 'dish',
        'menu', 'order', 'hungry', 'snack', 'beverage', 'drink',
'catering', 'groceries', 'kitchen', 'chef',
        'recipe', 'fast food', 'pizza', 'burger', 'sushi', 'dessert',
'bakery', 'coffee', 'tea', 'juice'
    ],
    'ecommerce': [
        'shop', 'store', 'buy', 'sell', 'market', 'mall', 'retail',
'online', 'deal', 'offer', 'discount',
        'sale', 'product', 'item', 'cart', 'checkout', 'payment',
'shipping', 'delivery', 'return', 'refund',
        'fashion', 'electronics', 'appliance', 'grocery', 'furniture',
'beauty', 'accessory', 'gadget', 'brand'
    ],
    'transportation': [
        'ride', 'taxi', 'cab', 'car', 'bike', 'scooter', 'auto',
'share', 'pool', 'commute', 'travel', 'move',
        'transit', 'route', 'fare', 'book', 'driver', 'passenger',
'pickup', 'drop', 'distance', 'mileage',
        'fuel', 'trip', 'journey', 'express', 'local', 'outstation',
'rental', 'lease'
    ],
    'health_fitness': [
        'health', 'fitness', 'gym', 'workout', 'exercise', 'yoga',
```

```
'pilates', 'meditation', 'wellness', 'diet',
        'nutrition', 'weight', 'loss', 'gain', 'muscle', 'strength',
'cardio', 'trainer', 'coach', 'class',
        'session', 'studio', 'club', 'active', 'fit', 'shape',
'recovery', 'therapy', 'massage', 'sport'
    ],
    'travel': [
        'travel', 'trip', 'tour', 'vacation', 'holiday', 'flight',
'hotel', 'resort', 'stay', 'booking',
        'ticket', 'destination', 'itinerary', 'sightseeing',
'adventure', 'explore', 'journey', 'cruise',
        'package', 'tourist', 'visa', 'passport', 'airport',
'airline', 'luggage', 'backpack', 'safari', 'camp'
    ],
    'entertainment': [
        'entertainment', 'music', 'movie', 'concert', 'theater',
'show', 'performance', 'festival', 'event',
        'ticket', 'booking', 'live', 'stream', 'comedy', 'drama',
'art', 'culture', 'gaming', 'sports',
        'amusement', 'park', 'arcade', 'casino', 'club', 'party',
'dance', 'karaoke', 'standup', 'magic'
    ],
    'banking': [
        'bank', 'financial', 'credit', 'loan', 'mortgage', 'savings',
'account', 'deposit', 'withdrawal',
        'transfer', 'investment', 'wealth', 'insurance', 'policy',
'premium', 'claim', 'card', 'debit',
        'credit', 'atm', 'branch', 'interest', 'rate', 'fund',
'stock', 'bond', 'mutual', 'portfolio', 'advisor'
    ],
    'telecom': [
        'telecom', 'mobile', 'phone', 'sim', 'data', 'internet',
'broadband', 'wifi', 'network', 'signal',
        'call', 'sms', 'message', 'roaming', 'plan', 'package',
'recharge', 'balance', 'tariff', 'connection',
        'service', 'provider', 'operator', '5g', '4g', 'lte', 'voip',
'fiber', 'satellite', 'communication'
    ],
    'automotive': [
        'auto', 'car', 'bike', 'vehicle', 'motor', 'engine', 'tire',
'fuel', 'petrol', 'diesel', 'electric',
        'hybrid', 'charging', 'service', 'repair', 'spare', 'part',
'accessory', 'dealership', 'showroom',
        'insurance', 'finance', 'lease', 'rental', 'test drive',
'mileage', 'performance', 'safety', 'luxury', 'suv'
    ],
    'technology': [
        'tech', 'software', 'hardware', 'computer', 'laptop',
'desktop', 'server', 'cloud', 'data', 'ai',
```

```
        'machine learning', 'iot', 'blockchain', 'cyber', 'security',
'network', 'coding', 'programming',
        'developer', 'engineer', 'app', 'website', 'platform',
'solution', 'service', 'support', 'maintenance',
        'upgrade', 'innovation', 'digital'
    ],
    'social_media': [
        'social', 'media', 'network', 'connect', 'share', 'post',
'like', 'comment', 'follow', 'friend',
        'message', 'chat', 'group', 'community', 'profile', 'feed',
'story', 'reel', 'trend', 'viral',
        'influencer', 'creator', 'content', 'platform', 'app',
'engagement', 'follower', 'hashtag', 'mention', 'tag'
    ],
    'retail': [
        'retail', 'shop', 'store', 'mall', 'outlet', 'supermarket',
'hypermarket', 'grocery', 'fashion',
        'electronics', 'appliance', 'furniture', 'beauty', 'cosmetic',
'accessory', 'jewelry', 'watch',
        'footwear', 'clothing', 'brand', 'discount', 'sale', 'offer',
'deal', 'promotion', 'loyalty', 'reward',
        'membership', 'customer'
    ],
    'education': [
        'education', 'learn', 'study', 'course', 'class', 'school',
'college', 'university', 'institute',
        'academy', 'training', 'coaching', 'tutor', 'teacher',
'student', 'exam', 'test', 'certificate',
        'degree', 'diploma', 'online', 'offline', 'skill',
'knowledge', 'workshop', 'seminar', 'webinar',
        'lecture', 'assignment', 'project'
    ],
    'gaming': [
        'game', 'gaming', 'play', 'console', 'pc', 'mobile', 'online',
'offline', 'multiplayer', 'singleplayer',
        'arcade', 'action', 'adventure', 'puzzle', 'strategy', 'rpg',
'simulation', 'sports', 'racing', 'shooter',
        'vr', 'ar', 'esports', 'tournament', 'leaderboard', 'score',
'level', 'character', 'quest', 'loot'
    ],
    'food_beverage': [
        'food', 'beverage', 'drink', 'snack', 'meal', 'restaurant',
'cafe', 'bakery', 'bar', 'pub', 'brewery',
        'coffee', 'tea', 'juice', 'soda', 'alcohol', 'wine', 'beer',
'spirit', 'cocktail', 'mocktail', 'dessert',
        'sweet', 'chocolate', 'ice cream', 'pastry', 'cake', 'bread',
'pizza', 'burger'
    ],
    'pharmaceuticals': [
```

```
        'pharma', 'medicine', 'drug', 'pill', 'tablet', 'capsule',
'syrup', 'injection', 'vaccine', 'health',
        'care', 'treatment', 'therapy', 'clinic', 'hospital',
'doctor', 'nurse', 'patient', 'prescription',
        'generic', 'branded', 'otc', 'rx', 'research', 'lab',
'clinical', 'trial', 'biotech', 'life science'
    ],
    'insurance': [
        'insurance', 'policy', 'premium', 'claim', 'cover',
'protection', 'health', 'life', 'vehicle', 'home',
        'travel', 'accident', 'medical', 'term', 'endowment', 'ulip',
'pension', 'annuity', 'renewal', 'agent',
        'broker', 'advisor', 'risk', 'benefit', 'compensation',
'liability', 'indemnity', 'assurance', 'coverage', 'plan'
    ],
    'logistics': [
        'logistics', 'delivery', 'courier', 'parcel', 'package',
'shipment', 'cargo', 'freight', 'transport',
        'warehouse', 'storage', 'supply', 'chain', 'distribution',
'fulfillment', 'order', 'tracking', 'dispatch',
        'pickup', 'drop', 'express', 'standard', 'international',
'domestic', 'customs', 'clearance', 'inventory',
        'stock', 'route'
    ],
    'real_estate': [
        'real estate', 'property', 'house', 'apartment', 'flat',
'villa', 'land', 'plot', 'commercial',
        'residential', 'rent', 'lease', 'buy', 'sell', 'invest',
'broker', 'agent', 'developer', 'builder',
        'construction', 'project', 'location', 'amenity', 'furnished',
'unfurnished', 'sale', 'purchase',
        'mortgage', 'loan', 'valuation'
    ],
    'energy': [
        'energy', 'power', 'electricity', 'solar', 'wind', 'hydro',
'thermal', 'nuclear', 'renewable',
        'non-renewable', 'oil', 'gas', 'petroleum', 'coal',
'refinery', 'generation', 'transmission',
        'distribution', 'grid', 'utility', 'meter', 'billing',
'tariff', 'subsidy', 'sustainability',
        'green', 'carbon', 'emission', 'climate'
    ],
    'aviation': [
        'aviation', 'airline', 'flight', 'airport', 'pilot', 'cabin',
'crew', 'ticket', 'booking', 'boarding',
        'pass', 'luggage', 'baggage', 'check-in', 'security',
'immigration', 'customs', 'terminal', 'gate',
        'runway', 'hangar', 'cargo', 'freight', 'charter', 'private',
'jet', 'turbine', 'propeller', 'aircraft'
```

```
    ],
    'hospitality': [
        'hospitality', 'hotel', 'resort', 'stay', 'accommodation',
'room', 'suite', 'lobby', 'reception',
        'check-in', 'check-out', 'amenity', 'spa', 'pool', 'gym',
'restaurant', 'bar', 'cafe', 'buffet',
        'breakfast', 'lunch', 'dinner', 'banquet', 'event',
'conference', 'meeting', 'wedding', 'party', 'service'
    ],
    'fashion': [
        'fashion', 'clothing', 'apparel', 'wear', 'outfit', 'dress',
'shirt', 'pant', 'jean', 'skirt', 'jacket',
        'coat', 'suit', 'tie', 'accessory', 'jewelry', 'watch', 'bag',
'shoe', 'footwear', 'designer', 'brand',
        'luxury', 'trend', 'style', 'collection', 'season', 'launch',
'sale', 'discount'
    ],
    'media_entertainment': [
        'media', 'entertainment', 'news', 'tv', 'radio', 'print',
'digital', 'publishing', 'advertising',
        'marketing', 'pr', 'public relations', 'journalism',
'reporting', 'anchor', 'reporter', 'editor',
        'content', 'article', 'blog', 'vlog', 'podcast', 'stream',
'broadcast', 'live', 'event', 'festival',
        'award', 'celebrity'
    ]
}
        self.mapping = INDUSTRY_MAPPING

    def classify_company(self, company):
        # Try deep learning prediction first
        sequence = self.tokenizer.texts_to_matrix([company],
mode='tfidf')
        pred = self.model.predict(sequence, verbose=0)
        confidence = np.max(pred)

        if confidence > 0.7:  # If confidence is high, use the model's
prediction
            return
self.label_encoder.inverse_transform([np.argmax(pred)])[0]
        else:  # If confidence is low, use rule-based classification
            return self._rule_based_classification(company)

    def _rule_based_classification(self, company):
        # Rule-based classification using keywords
        for industry, companies in self.mapping.items():
            if company in companies:
                return industry

        lower_company = company.lower()  # Convert company name to
```

```
lowercase
        for industry, keywords in self.keyword_rules.items():
            if any(kw in lower_company for kw in keywords):
                return industry

        return 'Other'  # If no match, classify as "Other"

# Classify companies in the dataset
classifier = CompanyClassifier()
df['Industry'] = df['Company'].apply(classifier.classify_company)
```

Customer Segmentation The purpose of this step is to segment customers based on their spending patterns across industries. This involves:

Grouping transactions by Customer ID and Industry.

Calculating the total spending and percentage of spending in each industry.

Identifying the primary industry segment for each customer.

Input: Processed transaction data with industry classifications.

Output: A DataFrame (customer_spending) containing customer segments and spending details.

This step helps businesses understand customer behavior and tailor marketing strategies accordingly.

```
# Step 12: Customer Segmentation
# Group customer spending by industry
customer_spending = df.groupby(['Customer ID', 'Industry'])
['Amount'].sum().unstack(fill_value=0)
spending_percentages =
customer_spending.div(customer_spending.sum(axis=1), axis=0)
customer_spending['Primary Segment'] =
spending_percentages.idxmax(axis=1)
customer_spending['Segment Confidence'] =
spending_percentages.max(axis=1)
customer_spending['Total Spending'] =
customer_spending.select_dtypes(include=[np.number]).sum(axis=1)
customer_spending['Transaction Count'] = df.groupby('Customer
ID').size()
```

Saving Results The purpose of this step is to save the processed data and customer segments for further analysis or reporting. This involves:

Saving the processed transaction data to a CSV file (processed_transactions123.csv).

Saving the customer segments to another CSV file (customer_segments123.csv).

Input: Processed DataFrame and customer segments.

Output: CSV files containing the final results.

This step ensures that the results are persisted and can be shared or analyzed later.

```
# Step 13: Save results
df.to_csv('/Desktop/processed_transactions123.csv', index=False)
customer_spending.to_csv('/Desktop/customer_segments123.csv')

print("Processing complete! Check saved files and visualizations.")
```