NLP Assignment

Let's break down the provided notebook into comprehensive notes, explaining each section, step-by-step. We'll include detailed explanations for the code, what it does, and why it's being used. I'll divide this into multiple sections to ensure clarity and organization.

📘 Comprehensive Notes on the Given Code

📝 Section 1: Data Loading and Preprocessing

🔍 Objective

- Load the data from an Excel file.
- Ensure columns are correctly named.
- Create dictionaries for translation and sentiment scoring.
- Prepare data for visualization and machine learning.

Code Breakdown

```
import pandas as pd

# Load the dataset
file_path = '~/Downloads/data_tshikama_xls.xlsx'  # Replace with your file path
df = pd.read_excel(file_path)

# Ensure columns are named correctly
df.columns = ['ciluba', 'french', 'score', 'sentiment', 'nature']
```

Explanation

1. **Data Loading**:
    - `pd.read_excel()` is used to load the Excel file.
    - Ensure the dataset is correctly imported by renaming columns to `['ciluba', 'french', 'score', 'sentiment', 'nature']`.

2. **Why This Step?**
    - Proper column naming is essential for clear identification and easier data manipulation.

📝 Section 2: Creating Dictionaries for Translation and Sentiment Scoring

Objective

- Create translation and scoring dictionaries to convert French text into Ciluba and assign sentiment scores to words.

Code Breakdown

```
# Create translation and scoring dictionaries from the dataset
translation_lexique = dict(zip(df['french'].str.lower(), df['ciluba']))
lexique = dict(zip(df['ciluba'].str.lower(), df['score']))

def translate_text_using_lexicon(text, translation_lexique):
    words = text.lower().split()
    translated_words = [translation_lexique.get(word, word) for word in words]
    translated_text = ' '.join(translated_words)
    return translated_text

def analyse_sentiment(text):
    words = text.lower().split()
    word_scores = {word: lexique.get(word, 0) for word in words}
    score = sum(word_scores.values())
    if score > 0.05:
        sentiment = "Positif"
    elif score < -0.05:
        sentiment = "Négatif"
    else:
        sentiment = "Neutre"
    return score, sentiment, word_scores
```

Explanation

1. **Creating Dictionaries**:
    - `translation_lexique`: Translates French words into Ciluba using a dictionary.

- `lexique`: Assigns sentiment scores to Ciluba words.

2. **Translation Function**:
   - `translate_text_using_lexicon()`: Converts French text into Ciluba using the dictionary.
3. **Sentiment Analysis Function**:
   - `analyse_sentiment()` calculates the sentiment score based on the provided lexique dictionary.
   - The sentiment is classified as:
     - **Positif** if the score > 0.05.
     - **Négatif** if the score < -0.05.
     - **Neutre** otherwise.

Testing Translation and Sentiment Analysis

```
# French text to translate
french_text = "Arrange Seulement"

# Translate the text using the lexicon
translated_text = translate_text_using_lexicon(french_text,
translation_lexique)

# Analyse the sentiment of the translated text
total_score, sentiment, word_scores = analyse_sentiment(translated_text)

# Display results
print("Translated Text (Ciluba):", translated_text)
print("Total Score:", total_score)
print("Sentiment:", sentiment)
print("Word Scores:", word_scores)
```

Explanation

- This section tests the translation and sentiment analysis functions.
- Converts a French sentence to Ciluba and calculates the sentiment score.

📝 Section 3: Data Visualization

Objective

- Visualize the distribution of categorical data to understand its composition.

Code Breakdown

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the count of each unique value for each column
plt.figure(figsize=(15, 10))

for i, column in enumerate(df.columns):
    plt.subplot(2, 3, i + 1)
    sns.countplot(data=df, x=column, palette='viridis')
    plt.title(f'Count of {column}')
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

Explanation

1. **Visualization of Categorical Data**:
   - `sns.countplot()` is used to show the frequency of each unique value in each column.
   - Useful for understanding the distribution of data across categories.
2. **Why This Step?**
   - Helps identify potential class imbalances and understand the dataset's structure.

📝 Section 4: Data Encoding and Feature Scaling for Machine Learning

Objective

- Convert categorical data into numerical format using Label Encoding.
- Standardize numerical features for better model performance.

Code Breakdown

```
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Encode the 'sentiment' and 'nature' columns into numerical labels
label_encoder_sentiment = LabelEncoder()
df['sentiment_encoded'] =
label_encoder_sentiment.fit_transform(df['sentiment'])
label_encoder_nature = LabelEncoder()
df['nature_encoded'] = label_encoder_nature.fit_transform(df['nature'])

# Prepare features (X) and target (y)
X = df[['score', 'sentiment_encoded']]
y = df['nature_encoded']

# Convert 'score' to numeric, if it's not already
X['score'] = pd.to_numeric(X['score'], errors='coerce').fillna(0)

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Explanation

1. **Label Encoding**:
   - Converts categorical variables (`sentiment` and `nature`) into numerical labels.
2. **Feature Scaling**:
   - `StandardScaler()` standardizes features to have a mean of 0 and standard deviation of 1.
   - This improves model performance, especially for algorithms like k-NN and SVM.

📝 Section 5: Building a Machine Learning Model

Objective

- Train a Random Forest model to classify the `nature` of text based on features.

Code Breakdown

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Initialize and train Random Forest classifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Predict on test data
y_pred = rf_model.predict(X_test)
```

Explanation

1. **Train-Test Split**:
   - Splits the dataset into training (80%) and testing (20%) sets.
2. **Model Training**:
   - Trains a **Random Forest Classifier** to predict the nature of text based on sentiment and score.

📝 Section 6: Model Evaluation using Confusion Matrix

Objective

- Evaluate the performance of the model using a Confusion Matrix.

Code Breakdown

```
# Decode the numerical labels for y_test and y_pred
y_test_decoded = label_encoder_nature.inverse_transform(y_test)
y_pred_decoded = label_encoder_nature.inverse_transform(y_pred)

# Compute confusion matrix
cm_rf = confusion_matrix(y_test_decoded, y_pred_decoded,
labels=label_encoder_nature.classes_)

# Plot confusion matrix
plt.figure(figsize=(13, 10))
ConfusionMatrixDisplay(confusion_matrix=cm_rf,
display_labels=label_encoder_nature.classes_).plot(cmap='Blues',
```

```
values_format='d')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```

Explanation

1. **Confusion Matrix**:
   - Displays the performance of the model in terms of True Positives, True Negatives, False Positives, and False Negatives.
2. **Why This Step?**
   - Useful for evaluating classification performance and identifying areas where the model may be misclassifying.

🔍 Conclusion & Insights

- This notebook demonstrates a comprehensive workflow for loading data, translating text, analyzing sentiment, visualizing distributions, encoding categorical variables, training a machine learning model, and evaluating it.
- The techniques used here (label encoding, feature scaling, and Random Forest) are fundamental for **text classification tasks**.

Let me know if you need any additional explanations or if you'd like me to expand on specific sections!