Lab 02:
Web Scraping, Feature Engineering, and Exploratory Data Analysis (EDA)

Lab 02: Web Scraping, Feature Engineering, and Exploratory Data Analysis (EDA)

1. Introduction

In many real-world data science applications, structured datasets are not readily available[2]. This lab provides a foundational pipeline for natural language processing (NLP) and machine learning tasks by demonstrating the acquisition of unstructured data, its conversion into structured features, and initial data exploration[3]. The process involves:

1. **Web Scraping:** Extracting text-based content from blog-style web pages.
2. **Feature Engineering:** Creating numerical and categorical features from the raw text using NLP techniques.
3. **Exploratory Data Analysis (EDA):** Visualizing and analyzing the newly created features to uncover patterns and trends.

2. Objectives

By the end of this lab, students will be able to:

- Understand the process of **Web Scraping** using Python libraries.
- Acquire and structure data from multiple web URLs into a **Pandas DataFrame**.
- Perform **Feature Engineering** on text data using **spaCy** to extract linguistic metrics like token count, sentence count, and named entities.
- Calculate **Text Sentiment** and **Subjectivity** using the **TextBlob** library.
- Apply **Exploratory Data Analysis (EDA)** techniques, including distribution plots and TF-IDF analysis, to understand the data's characteristics.

3. Justification of Libraries and Tools

Library	Area	Justification
NumPy	Numerical Computing	Used for efficient, **vectorized** array operations, which are significantly faster than native Python loops for large datasets, as demonstrated in the initial efficiency test.
requests	Web Scraping	Essential for making HTTP requests to fetch the raw HTML content of the target URLs.
BeautifulSoup	Web Scraping	A high-level library used to parse the raw HTML obtained from requests and extract specific data (e.g., article titles and body text) based on HTML tags and classes.

Library	Area	Justification
pandas	Data Manipulation	The primary tool for storing the scraped data and extracted features in a tabular structure (**DataFrame**), enabling easy cleaning, transformation, and analysis.
spaCy	NLP / Feature Engineering	A powerful and efficient library for advanced NLP tasks. It is used to tokenize text, segment sentences, identify **Named Entities (NER)**, and determine the **Part-of-Speech (POS)** tags (like nouns) for feature extraction.
TextBlob	NLP / Sentiment Analysis	Provides a simple API for common NLP tasks, primarily used here to calculate the **Polarity** (positivity/negativity) and **Subjectivity** of the scraped text.
matplotlib & **seaborn**	Data Visualization	Standard Python plotting libraries. **Seaborn** builds on Matplotlib to provide a higher-level interface for drawing informative statistical graphics necessary for EDA, such as histograms and pair plots.
sklearn	Machine Learning	Specifically, `TfidfVectorizer` is used for **Term Frequency-Inverse Document Frequency** calculation, a key technique in NLP for converting text into a meaningful numerical representation for analysis.

4. Lab Procedure (Code Snippets and Explanation)

4.1 Setup and Initialization

Before starting, the required libraries must be installed. The notebook also includes a demonstration of Python's efficiency comparison using **NumPy**.

A. Installation and Setup

The following commands ensure all necessary packages and the **spaCy** English language model are installed:

Python

```
# Installation of required libraries
!pip install requests beautifulsoup4 lxml pandas spacy matplotlib seaborn textblob

# Download the small English language model for spaCy
!python -m spacy download en_core_web_sm
```

B. NumPy vs. Python Efficiency

This snippet demonstrates why libraries like **NumPy** are critical for performance in data science by comparing the time taken to sum a large array using a native Python loop versus NumPy's optimized, **vectorized** function.

Code Snippet	Explanation
`import numpy as np`	Imports NumPy for fast array processing.
`import time`	Imports the `time` module to benchmark execution speed.

Code Snippet	Explanation
data = np.random.rand(size)	Creates a large array (size=10,000,000) of random numbers.
np_sum = np.sum(data)	**NumPy method:** Uses a fast, compiled operation.
manual_sum = 0; for val in data: manual_sum += val	**Python method:** Uses a slower, interpreted loop.

Python

```python
import numpy as np
import time

size = 10000000
data = np.random.rand(size)

# NumPy vectorized sum
start_np = time.time()
np_sum = np.sum(data)
end_np = time.time()

# Manual Python loop sum
start_py = time.time()
manual_sum = 0
for val in data:
    manual_sum += val
end_py = time.time()

print(f"NumPy Sum: {np_sum:.2f} | Time: {end_np - start_np:.5f} sec")
print(f"Manual Sum: {manual_sum:.2f} | Time: {end_py - start_py:.5f} sec")
```

4.2 Step 1: Web Scraping and Data Acquisition

The goal is to scrape the article URL, Title, and body Text from a list of predefined blog links and store them in a DataFrame.

Code Snippet	Explanation
import requests, from bs4 import BeautifulSoup, import pandas as pd	Imports the core libraries for HTTP requests, HTML parsing, and data structuring.
headers = {...}	Defines a User-Agent header to mimic a web browser, preventing some sites from blocking the scraping request.

Code Snippet	**Explanation**
res = requests.get(url, ...)	Executes the HTTP GET request for the current URL.
soup = BeautifulSoup(res.content, 'lxml')	Parses the raw HTML content using the lxml parser (which is fast).
title = soup.find('h3', class_='post-title')	Uses BeautifulSoup to find the article title by its HTML tag (h3) and CSS class (post-title).
content_div = soup.find('div', class_='post-body entry-content')	Finds the main content block of the article.
text = ' '.join(p.get_text(strip=True) for p in paragraphs)	Extracts all text from all paragraph (<p>) tags within the content block and joins them into a single string.
df = pd.DataFrame(articles)	Creates the final DataFrame containing the scraped data.

Python
```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time

urls = [
    'https://techncruncher.blogspot.com/2025/01/top-10-ai-tools-that-will-transform.html',
    'https://techncruncher.blogspot.com/2023/12/limewire-ai-studio-review-2023-details.html',
    'https://techncruncher.blogspot.com/2023/01/top-10-ai-tools-in-2023-that-will-make.html',
    'https://techncruncher.blogspot.com/2022/11/top-10-ai-content-generator-writer.html',
    'https://techncruncher.blogspot.com/2022/09/cj-affiliate-ultimate-guide-to.html'
]

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
articles = []

for url in urls:
    try:
        # Fetch the page content
        res = requests.get(url, headers=headers, timeout=10)
        res.raise_for_status()
        soup = BeautifulSoup(res.content, 'lxml')

        # Extract Title
        title = soup.find('h3', class_='post-title')
        title = title.get_text(strip=True) if title else soup.title.get_text(strip=True)

        # Extract Body Text
```

```
            content_div = soup.find('div', class_='post-body entry-content')
            if not content_div:
                continue

            paragraphs = content_div.find_all('p')
            text = ' '.join(p.get_text(strip=True) for p in paragraphs)

            # Skip short articles for quality control
            if len(text) > 100:
                articles.append({'url': url, 'title': title, 'text': text})
            else:
                print(f"[SKIP] Too short: {url}")

            time.sleep(1) # Be polite to the server
        except Exception as e:
            print(f"[ERROR] {url} -> {e}")

    df = pd.DataFrame(articles)
    df.head()
```

4.3 Step 2: Feature Engineering (NLP)

Feature engineering is performed to quantify the qualitative text data by extracting meaningful metrics, making it suitable for analysis and downstream machine learning models.

A. Linguistic Feature Extraction using spaCy

The `extract_features` function uses **spaCy** to process the text and derive four key metrics:

1. **num_tokens**: The total count of words and punctuation marks in the text.
2. **num_sentences**: The number of complete sentences.
3. **num_entities**: The count of named entities (e.g., people, organizations, locations).
4. **num_nouns**: The count of words identified as common or proper nouns.

Code Snippet	Explanation
`nlp = spacy.load("en_core_web_sm")`	Loads the trained **spaCy** English model, which performs tokenization, POS tagging, and NER.
`doc = nlp(text)`	Processes the article text into a **Doc** object, which contains all linguistic annotations.
`len(doc)`	Gets the total number of tokens (words/punctuation).
`len(list(doc.sents))`	Gets the number of sentences via the sentence boundary detection.
`len(doc.ents)`	Gets the number of detected named entities.

Code Snippet	Explanation
len([t for t in doc if t.pos_ == 'NOUN'])	Uses list comprehension to count tokens whose Part-of-Speech tag is 'NOUN'.

Python
```
import spacy
import pandas as pd # Already imported, but listed for context

# Load the spaCy model
nlp = spacy.load("en_core_web_sm")

def extract_features(text):
    """Processes text with spaCy and extracts linguistic features."""
    doc = nlp(text)
    return pd.Series({
        'num_tokens': len(doc),
        'num_sentences': len(list(doc.sents)),
        'num_entities': len(doc.ents),
        'num_nouns': len([t for t in doc if t.pos_ == 'NOUN'])
    })

if not df.empty:
    # Apply the spaCy feature extraction
    features = df['text'].apply(extract_features)
    df = pd.concat([df, features], axis=1)

    # Calculate simple length feature
    df['title_length'] = df['title'].apply(len)

    print("Features extracted successfully.")
    print(df.head(2))
else:
    print("⚠ No articles found for feature extraction.")
```

B. Sentiment Feature Extraction using TextBlob

TextBlob provides a quick way to calculate the **polarity** (a measure from -1.0 to +1.0, where -1.0 is negative and +1.0 is positive) and **subjectivity** (a measure from 0.0 to 1.0, where 0.0 is objective and 1.0 is subjective) of the text.

Python
```
from textblob import TextBlob

def get_sentiment(text):
    """Calculates sentiment polarity and subjectivity using TextBlob."""
    blob = TextBlob(text)
    return pd.Series({
        'sentiment_polarity': blob.sentiment.polarity,
        'sentiment_subjectivity': blob.sentiment.subjectivity
    })

if not df.empty:
    # Apply TextBlob sentiment analysis
```

```
sentiment_features = df['text'].apply(get_sentiment)
df = pd.concat([df, sentiment_features], axis=1)

print("Sentiment features extracted successfully.")
print(df[['title', 'sentiment_polarity', 'sentiment_subjectivity']].head())
```

4.4 Step 3: Exploratory Data Analysis (EDA)

EDA uses visualization techniques to summarize the main characteristics of the dataset, helping to identify patterns, spot anomalies, and check assumptions.

A. Basic Feature Distributions

Histograms are used to visualize the distribution of two important derived features: Title Length and Sentiment Polarity.

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

plt.style.use('ggplot')

# 1. Title Length Distribution
plt.figure(figsize=(10, 5))
sns.histplot(df['title_length'], kde=True, bins=5, color='skyblue')
plt.title('Distribution of Article Title Lengths')
plt.xlabel('Title Length (Characters)')
plt.ylabel('Frequency')
plt.show()

# 2. Sentiment Polarity Distribution
plt.figure(figsize=(10, 5))
sns.histplot(df['sentiment_polarity'], kde=True, bins=5, color='lightcoral')
plt.title('Distribution of Article Sentiment Polarity')
plt.xlabel('Polarity Score (-1.0 to 1.0)')
plt.ylabel('Frequency')
plt.show()
```

B. Feature Correlation (Pair Plot)

A pair plot visualizes the pairwise relationships between the numerical features, allowing for quick inspection of potential correlations and data clusters.

Python
```python
# Select the numeric features for the pair plot
numeric_cols = [
    'title_length',
    'num_tokens',
    'num_sentences',
    'num_entities',
    'sentiment_polarity'
]
```

```python
sns.pairplot(df[numeric_cols])
plt.suptitle('Pair Plot of Derived Features', y=1.02)
plt.show()
```

C. Advanced Text Feature: TF-IDF Analysis

TF-IDF is a statistical measure that reflects how important a word is to a document relative to the entire corpus. The following code calculates the TF-IDF scores for the scraped articles and visualizes the top 20 most important terms.

Code Snippet	Explanation
from sklearn.feature_extraction.text import TfidfVectorizer	Imports the required transformer from sklearn.
vectorizer = TfidfVectorizer(...)	Initializes the vectorizer, setting stop_words='english' to ignore common words (like 'the', 'a'), and limiting to the top 20 features.
X = vectorizer.fit_transform(df['text'])	Calculates the TF-IDF scores for the article text column.
tfidf_df = pd.DataFrame(X.toarray(), ...)	Converts the sparse matrix output into a readable DataFrame with the vocabulary as column names.
tfidf_sums = tfidf_df.sum().sort_values(...)	Sums the TF-IDF scores across all documents to find the most globally important terms.
sns.barplot(...)	Creates a bar chart to visualize the top 20 terms.

Python

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=20)

# Fit and transform the text column
X = vectorizer.fit_transform(df['text'])
tfidf_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())

# Show top TF-IDF features across all documents
tfidf_sums = tfidf_df.sum().sort_values(ascending=False)

# Plot the top terms
plt.figure(figsize=(10, 5))
sns.barplot(x=tfidf_sums.index, y=tfidf_sums.values)
plt.title("Top 20 TF-IDF Terms Across Articles")
plt.ylabel("TF-IDF Score Sum")
plt.xlabel("Term")
plt.xticks(rotation=45, ha='right')
```

```
plt.grid(True)
plt.tight_layout()
plt.show()
```