Local Plan Data Collection Process

This script was developed to speed up the process of manually collecting local authority "local plans" data. The purpose of approaching this task programmatically was to create a more efficient method of collecting this data that can be scalable. This is due to the fact that the script does not need to be recreated and and can simply be invoked for future use. A good measure of how efficient something is compared to other methods is to observe the cost. This script took only 2 weeks to setup and complete collection, with the manual collection estimated to need 3 months. This means we can estimate that the first iteration of using the script was around £35,000 cheaper (by calculating the day rate of person collecting data), and estimated £38,500 for future use. This is due to the quicker set up time for running the collection script for future use.

Step 1: Manual Data Collection

We begin by visiting the following website:
UK Local Authorities

Process:

1. Click on each local authority listed on the page.
2. Note the **three-letter reference** for each local authority.
3. Follow the link to the local authority's website and search for their **local plans**.
4. Identify the page that contains information and documents relating to the local plan.
5. Record the following details in a spreadsheet:
    - **Start date**, **End date**, and **Adoption date** of the local plan.
    - The **URL** of the page containing this information.
    - The three-letter reference.

This manual process builds the initial dataset, which we will populate further using automated scripts.

Step 2: Automating Data Population

After manually collecting the local plan data, we use a Python script to automate the process of populating additional information in the spreadsheet.

What the script does:

1. **Load Data:** It loads two CSV files — one with the manually collected data and another with details about UK local authorities.
2. **Select Relevant Columns:** The script narrows down the second CSV to only include columns we need, such as the local authority code and official name.
3. **Prepare Data for Matching:** It extracts the three-letter codes from both datasets and ensures they are in the correct format for merging.

4. **Merge Data:** A left merge is performed using the three-letter codes to match the official names of the local authorities with the corresponding information in the manually collected data.
5. **Clean and Populate Columns:** The script fills in details like organisation names and generates unique references for each local plan based on the organisation and the start date of the local plan.
6. **Save Updated Data:** After processing, the updated dataset is saved as a new CSV file for further use.

Step 3: Extracting Document Links from Local Plan Pages

The next step involves extracting links to the local plan documents from the web pages we collected. Another Python script is used to:

1. **Fetch the Webpage:** The script downloads the content of the webpage corresponding to each local authority.
2. **Extract Document Links:** It looks for document links such as PDFs, Word files, and other relevant documents.
3. **Clean the Text:** The link text is cleaned and standardised to remove irrelevant characters (e.g., file sizes, extra spaces).
4. **Generate References:** Each document is assigned a reference number, combining the local authority reference and a counter.
5. **Fuzzy Matching:** The script attempts to match the document titles with official local plan names from a reference dataset, ensuring proper categorisation of the documents.
6. **Store the Results:** The extracted data, including document links and matched references, is saved into a new CSV file.

Error Handling:

During the extraction process, if the script encounters an error (e.g., the webpage cannot be accessed), the URL is added to a list of failed URLs. This list is saved in a separate CSV file for further manual review.

Step 4: Completing the Data

Once the automated process is complete, the final dataset contains a full list of document URLs for local plans from each local authority. Any missing or failed entries can be manually reviewed and updated to ensure completeness.

Imports

Note: Some of these libraries will need to be pip installed before running the code.

In [1]:
```
# Imports
import pandas as pd
import requests
from bs4 import BeautifulSoup
```

```
import re
from urllib.parse import urljoin
from fuzzywuzzy import process
import warnings
from slugify import slugify
```

Local Plan Sheet Completion Code

In [2]:

```
# Load the data
df0 = pd.read_csv("data/local_plan_manual_collection.csv")
df1 = pd.read_csv("documents/uk_local_authorities_future.csv")

# Only select relevant columns from df1
df1 = df1[["local-authority-code", "official-name"]]

# Ensure 'organisation' and 'local-authority-code' are treated as strings
df0['organisation'] = df0['organisation'].astype(str)
df1['local-authority-code'] = df1['local-authority-code'].astype(str)

# Extract the codes from both df0 and df1
df0['org_code'] = df0['organisation'].str.extract(r'([A-Z]{3,4})')
df1['org_code'] = df1['local-authority-code'].str.extract(r'([A-Z]{3,4})')

# Perform a left merge on the extracted 3-letter codes
df = pd.merge(df0, df1[['org_code', 'official-name']], on='org_code', how='left')

# Copy the 'official-name' column into 'organisation-name'
df['organisation-name'] = df['official-name']

# Ensure 'period-start-date' is numeric and convert it to an integer without decima
df['period-start-date'] = pd.to_numeric(df['period-start-date'], errors='coerce')
df['period-start-date'] = df['period-start-date'].fillna(0).astype(int)  # Replace

# Populating the 'reference' column
df['slug'] = df['organisation-name'].apply(lambda x: slugify(str(x)))
df['reference'] = df['slug'] + "-local-plan-" + df['period-start-date'].astype(str)
df.drop('slug', axis=1, inplace=True)

# Populating the 'name' column
df['name'] = df['official-name'] + " Local Plan " + df['period-start-date'].astype(

# Prepend 'local-authority:' to each entry in the 'organisation' column
df['organisation'] = "local-authority:" + df['organisation']

# Drop the 'org_code' and 'official-name' columns
df = df.drop(columns=['org_code', 'official-name'])

# Filter rows where 'documentation-url' is empty or missing
df_missing_url = df[df['documentation-url'].isna() | df['documentation-url'].eq('')

# Save the rows without 'documentation-url' to a separate CSV file
df_missing_url.to_csv('data/missing_documentation_url.csv', index=False)

# Drop rows where 'documentation-url' is empty or missing from the original datafra
df = df[~df['documentation-url'].isna() & ~df['documentation-url'].eq('')]

# Save the updated dataframe to a CSV file
df.to_csv('data/outputs/local_plan.csv', index=False)
```

Local Plans Documents Sheet Data Scrape and Completion Code

In [ ]:

```
def clean_text(text):
    """
    Cleans the provided text by replacing or removing unwanted characters.
    """
    text = re.sub(r'[^\x00-\x7F]+', "'", text)  # Replace all non-ASCII characters
    text = re.sub(r'\[\s*pdf\s*\]', '', text, flags=re.IGNORECASE)  # Remove [pdf]
    text = re.sub(r'\s+', ' ', text)  # Normalise any excessive spaces
    text = re.sub(r'\(\d+(,\d{3})?KB\)|\d+(,\d{3})?KB|\d+MB', '', text)  # Remove f

    # Split the text into words and remove apostrophes at the end of each word
    words = text.split()
    cleaned_words = [word.rstrip("'") for word in words]  # Remove apostrophe at th

    cleaned_text = ' '.join(cleaned_words)

    return cleaned_text.strip()

def extract_links_from_page(url, plan_prefix, reference_data):
    """
    Extracts all document links from a webpage, cleans the text associated with eac
    and matches the text with a reference from an external CSV file.

    Parameters:
    url (str): The URL of the webpage to scrape.
    plan_prefix (str): The prefix to use for naming references.
    reference_data (pd.DataFrame): The dataframe containing the reference data to m

    Returns:
    list: A list of lists, where each sublist contains the reference, plan prefix,
          full URL of the document, the input URL, and the matched reference from t
    """
    headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
    'Accept-Language': 'en-US,en;q=0.9',
    'Accept-Encoding': 'gzip, deflate, br',
}

    response = requests.get(url, verify=False)
    #response = requests.get(url, verify=certifi.where())
    soup = BeautifulSoup(response.content, 'html.parser')

    # Find all <a> tags that contain href attributes
    links = soup.find_all('a', href=True)

    link_data = []
    counter = 1

    # Process href links and check if they contain 'pdf', 'doc', 'document', or 'fi
    for link in links:
        href = link['href']
        full_url = urljoin(url, href)  # Handle relative URLs

        # Check if 'pdf', 'doc', 'document', or 'file' is in the URL
        if any(x in href.lower() for x in ['pdf', 'doc', 'document', 'file']):
            text = link.get_text(strip=True)  # Get the link text
            text = clean_text(text)  # Clean the text
            reference = f"{plan_prefix}-{counter}"  # Create the reference using th

            # Fuzzy match the text to the "name" column in reference_data
```

```
            match = process.extractOne(text, reference_data['name'])
            matched_reference = reference_data.loc[reference_data['name'] == match[

            link_data.append([reference, plan_prefix, text, full_url, url, matched_
            counter += 1  # Increment the counter for each link

    return link_data

# Main script logic
if __name__ == "__main__":
    # Load your main DataFrame (df) containing 'reference' and 'documentation-url'
    df = pd.read_csv('')

    # Load the reference data from the CSV file (assumed to be the same for all row
    reference_data = pd.read_csv('documents/development-plan-document-type.csv')

    # Create an empty list to hold all extracted link data across all iterations
    all_link_data = []

    # List to hold tuples of failed URLs and their associated plan_prefix
    failed_urls = []

    # Iterate over each row in the DataFrame
    for index, row in df.iterrows():
        ref = row['reference']  # Reference from the current row
        url = row['documentation-url']  # Documentation URL from the current row

        try:
            # Attempt to extract links from the current page
            link_data = extract_links_from_page(url, ref, reference_data)

            # Append the extracted data to the all_link_data list
            all_link_data.extend(link_data)
        except Exception as e:
            # If an error occurs, print the error and add the URL and plan_prefix t
            print(f"Error processing {url} with plan {ref}: {e}")
            failed_urls.append((ref, url))
            continue  # Move on to the next URL

    # Create a DataFrame from the combined list of link data
    final_df = pd.DataFrame(all_link_data, columns=['reference', 'plan', 'text', 'u

    # Populate blank rows with 'supplementary-planning-documents'
    final_df['matched_reference'].fillna('supplementary-planning-documents', inplac

    # Rename columns to specification
    final_df.rename(columns={'text':'name',
                             'url':'document-url',
                             'input_url':'documentation-url',
                             'matched_reference':'document-types'},
                    inplace=True
                    )

    # Save the final DataFrame as a CSV file
    output_path = ''
    final_df.to_csv(output_path, index=False)

    print(f"Data saved as {output_path}")

    # Save failed URLs along with their plan_prefix to a CSV file
    if failed_urls:
        # Create a DataFrame for failed URLs with columns 'reference' and 'document
        failed_df = pd.DataFrame(failed_urls, columns=['reference', 'documentation-
        failed_urls_path = 'data/failed_urls.csv'
```

```
        failed_df.to_csv(failed_urls_path, index=False)

        print(f"Failed URLs saved to {failed_urls_path}")

        # Print out the list of failed URLs
        print("\nThe following URLs failed during processing:")
        for ref, failed_url in failed_urls:
            print(f"Reference: {ref}, URL: {failed_url}")
```