In [1]:
```
# Importing the libraries
import pandas as pd
import numpy as np
from tqdm import tqdm
from fuzzywuzzy import fuzz, process
import spacy
import warnings
warnings.simplefilter("once")
```

In [2]:
```
# load the dataset
data = pd.read_excel('Online Retail.xlsx')
```

In [3]:
```
data.head()
```

Out[3]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

In [4]:
```
# Data preprocessing

# remove rows with missing values
data = data.dropna(subset=['InvoiceNo', 'Description', 'Quantity', 'UnitPrice'])

# remove rows with negative quantity or price
data = data[(data.Quantity > 0) & (data.UnitPrice > 0)]

# exclude cancelled and adjustment invoices that start with 'C' and 'A' respectively
data = data[~data['InvoiceNo'].apply(lambda x: str(x).startswith(('C', 'A')))]

data.shape
```

Out[4]:
```
(530103, 8)
```

In [5]:
```
nlp = spacy.load('en_core_web_sm')

def create_group_name(items):
```

```
    item_text = " ".join(items)

    # parsing the text
    doc = nlp(item_text)

    # store the frequency of each word
    word_freq = {}
    for token in doc:
        if token.is_alpha and not token.is_stop and len(token.text) > 2:
            if token.text.lower() in word_freq:
                word_freq[token.text.lower()] += 1
            else:
                word_freq[token.text.lower()] = 1

    # taking the top 3 most frequent words
    top_words = sorted(word_freq, key=word_freq.get, reverse=True)[:3]

    # Creating the group name
    group_name = " ".join(top_words)

    return group_name
```

In [6]:

```
def group_items(data, similarity_threshold):
    # Create a dictionary to store the grouped items
    item_groups = {}

    # Group items by their similarity using string distance metrics
    for item in tqdm(data.Description.unique()):
        # Check if the item has already been grouped
        if item in item_groups:
            continue

        # Find the most similar item to the current item
        max_similarity = 0
        for group, group_items in item_groups.items():
            for group_item in group_items:
                sim = fuzz.token_set_ratio(item.lower(), group_item.lower())
                if sim > max_similarity:
                    max_similarity = sim
                    best_group = group

        # If the most similar item has a high similarity, add the current item to its
        if max_similarity > similarity_threshold:
            item_groups[best_group].append(item)
        # Otherwise, create a new group for the current item
        else:
            item_groups[item] = [item]

    # Create a dictionary to map each item to its group
    item_to_group = {}
    for group, items in tqdm(item_groups.items()):
        for item in items:
            item_to_group[item] = create_group_name(items)

    # Add a new column to the data indicating the group for each item
    data['ItemGroup'] = data.Description.apply(lambda x: item_to_group[x])

    return data
```

In [7]:
```
# Creating groups for the items and mapping them to each item using levenshtein metric
data = group_items(data, 60) # Using similarity threshold = 60
print(f'No. of item groups: {data.ItemGroup.nunique()}')
```

```
100%|██████████| 4025/4025 [01:17<00:00, 51.87it/s]
100%|██████████| 351/351 [02:12<00:00,  2.64it/s]
No. of item groups: 351
```

In [8]:
```
data.head()
```

Out[8]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	ItemG
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	h har
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	decor har
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	c h
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	retr re
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	red w h

In [9]:
```
data.to_excel('data.xlsx', index = False)
```

In [12]:
```
def get_recommendation(invoice_no):
    # get all items in the given invoice number
    basket = list(set(data[data['InvoiceNo'] == invoice_no]['Description']))

    # get the item groups for the given invoice number
    item_groups = data[data['InvoiceNo'] == invoice_no]['ItemGroup']

    # if an invoice contains only one item group and that item group has only those it
    if item_groups.nunique() == 1 and data[data["ItemGroup"] == item_groups.iloc[0]]['
        # find the most similar item group
        matches = process.extract(item_groups.iloc[0], data["ItemGroup"].unique(), sc
        matches_sorted = sorted(matches, key=lambda x: x[1], reverse=True)
        item_groups = [x[0] for x in matches_sorted if x[0] != item_groups.iloc[0]]

    for item_group in item_groups:
```

```
        # print(f'selected: {i}')
        if item_group is None:
            return print(f'No item groups for: {invoice_no}')

        # get all items in the most common item group and sort them by Quantity
        similar_items = data[data["ItemGroup"] == item_group].sort_values(by='Quantity

        # remove the items already in the basket
        similar_items = similar_items[~similar_items.isin(basket)]

        if len(similar_items) > 0:
            break

    # randomly recommend an item from the remaining items, up to 3 recommendations
    recommendations = similar_items.unique()[:3]
    return np.random.choice(recommendations) if len(recommendations) > 0 else print(f'
```

In [15]:

```
# To store the recommendations for each unique invoice
recommendations = []

# Loop through each unique invoice and apply the get_recommendation function
for invoice in tqdm(data['InvoiceNo'].unique()):
    recommendation = get_recommendation(invoice)
    recommendations.append(recommendation)

# Map the recommendations back to the original DataFrame
data['Recommendation'] = data['InvoiceNo'].map(dict(zip(data['InvoiceNo'].unique(), re
```

```
100%|██████████| 19959/19959 [21:23<00:00, 15.55it/s]
```

In [16]:

```
# Generate the summary table
summary_table = pd.DataFrame(columns=['Description', 'Invoice Count', 'Recommendation

unique_recommendations = data["Recommendation"].unique()

# iterate over the unique recommended items and add them to the summary table
for item in tqdm(unique_recommendations):
    invoice_count = data[data['Description'] == item]['InvoiceNo'].nunique()
    recommendation_count = data[data['Recommendation'] == item]['InvoiceNo'].count()
    summary_table = pd.concat([summary_table, pd.DataFrame([[item, invoice_count, recc
```

```
100%|██████████| 924/924 [00:33<00:00, 27.86it/s]
```

In [17]:

```
# save the recommended items to an Excel file
data[["InvoiceNo", "Recommendation"]].to_excel('recommended_items.xlsx', index=False)

unique_data = data.drop_duplicates(subset=["InvoiceNo"])
unique_data[["InvoiceNo", "Recommendation"]].to_excel('recommendations.xlsx', index=Fa

# save the summary table to an Excel file
summary_table.to_excel('summary_table.xlsx', index=False)
```