DATA CLEANING AND PREPROCESSING:

```
import time
import re
import pandas as pd
from nltk.tokenize import sent_tokenize
```

- This code imports necessary modules: `time` for measuring execution time, `re` for regular expressions, `pandas` for data manipulation, and `sent_tokenize` from NLTK for tokenizing sentences.

```
start = time.time()
```

- This line records the starting time of the program execution.

```
df = pd.read_csv("D://SATYAM//new_bot2//recipes.csv")
```

- This line reads a CSV file named "recipes.csv" and stores the data in a DataFrame called `df`.

```
print('The shape of the dataset:', df.shape)
print('Columns of the dataset:', df.columns)
print('The first five elements of the dataset:', df.head())
```

- These lines print out information about the dataset such as its shape (number of rows and columns), column names, and the first five rows of data.

```
df.drop_duplicates(inplace=True)
df.drop(columns=['RecipeYield', 'AggregatedRating', 'ReviewCount', 'RecipeServings'], inplace=Tr
ue)
df.drop(columns=['AuthorId', 'AuthorName', 'DatePublished', 'PrepTime', "TotalTime", 'Images',
'CookTime', 'Calories',
                 'FatContent', 'SaturatedFatContent', 'CholesterolContent', 'SodiumContent',
                 'CarbohydrateContent', 'FiberContent', 'SugarContent', 'ProteinContent'],
        inplace=True)
df.dropna(inplace=True)
```

- These lines perform data cleaning and preprocessing operations on the DataFrame `df`.
    - `drop_duplicates()` removes any duplicate rows.
    - `drop(columns=...)` removes specific columns from the DataFrame.
    - `dropna()` removes rows with missing values (NaN).

```
df['Keywords'] = df["Keywords"].str.replace(r'c\(|\"|\"|\)|FreeOf...|<,', '', regex=True)
df['RecipeCategory'] = df["RecipeCategory"].str.replace(r'c\(|\"|\"|\)|FreeOf...|<,', '', regex=True)
df['Keywords'] = df['Keywords'].str.split(',')
df['RecipeIngredientQuantities'] = df["RecipeIngredientQuantities"].str.replace(r'c\(|\"|\"|\)', '', regex=True)
df['RecipeIngredientParts'] = df["RecipeIngredientParts"].str.replace(r'c\(|\"|\"|\)', '', regex=True)
df['RecipeInstructions'] = df["RecipeInstructions"].str.replace(r'c\(|\"|\"|\)|\n|,', '', regex=True)
df['RecipeIngredientQuantities'] = df['RecipeIngredientQuantities'].str.split(',')
df['RecipeIngredientParts'] = df['RecipeIngredientParts'].str.split(',')
df['RecipeInstructions'] = df['RecipeInstructions'].apply(lambda x: (sent_tokenize(x)))
```

- These lines further clean and preprocess specific columns of the DataFrame:
    - `replace()` is used with regular expressions (`regex=True`) to remove unwanted characters or patterns from the strings in columns like 'Keywords', 'RecipeCategory', 'RecipeIngredientQuantities', 'RecipeIngredientParts', and 'RecipeInstructions'.
    - `split(',')` is used to split the strings into lists based on commas.
    - `sent_tokenize()` tokenizes the sentences in the 'RecipeInstructions' column using NLTK's `sent_tokenize` function.

```
def list_to_dict(lst):
    return {i + 1: val for i, val in enumerate(lst)}

df['RecipeInstructions'] = df['

RecipeInstructions'].apply(list_to_dict)
```

- This code defines a function `list_to_dict()` that converts a list into a dictionary by assigning index numbers as keys and list elements as values. It is applied to the 'RecipeInstructions' column using `apply()` .

```
df['Ingredients'] = df.apply(lambda row: dict(zip(row['RecipeIngredientParts'], row['RecipeIngredientQuantities'])),
                             axis=1)
```

- This line creates a new column 'Ingredients' in the DataFrame `df` . It combines the lists in 'RecipeIngredientParts' and 'RecipeIngredientQuantities' columns into a dictionary by zipping them together using `zip()` and then converting the result into a dictionary using `dict()` . This operation is applied row-wise using `apply()` .

```
def remove_whitespace(d):
    return {k: v.strip() if isinstance(v, str) else v for k, v in d.items()}

df['Ingredients'] = df['Ingredients'].apply(remove_whitespace)
```

- This code defines a function `remove_whitespace()` that removes leading and trailing whitespace from the values of a dictionary. It is applied to the 'Ingredients' column using `apply()`.

```
df['Keywords'] = df['Keywords'].apply(lambda x: [word.lstrip() for word in x])
```

- This line removes leading whitespace from each word in the 'Keywords' column by applying a lambda function with `lstrip()` to each element in the list.

```
df.drop(columns=['RecipeIngredientQuantities'], inplace=True)
```

- This line drops the 'RecipeIngredientQuantities' column from the DataFrame.

```
df = df.loc[:, ['Name', 'RecipeCategory',
                'Keywords', 'RecipeIngredientParts',
                'Ingredients', 'RecipeInstructions']]
```

- This line selects specific columns from the DataFrame `df` and reassigns the DataFrame to include only those columns.

```
df.rename(columns={'Name': 'title', 'RecipeIngredientParts': 'ingredients_names', 'Ingredients': 'ingredients',
                   'RecipeInstructions': 'directions'}, inplace=True)
```

- This line renames the columns in the DataFrame `df` to 'title', 'ingredients_names', 'ingredients', and 'directions' using `rename()`.

```
df = df[df['title'].str.match(r'^[a-zA-Z]')]
df = df[df['RecipeCategory'].str.match(r'^[a-zA-Z]')]
```

- These lines filter the DataFrame `df` to only include rows where the 'title' and 'RecipeCategory' columns begin with an alphabet character using regular expression matching.

```
no_ingredients = df['ingredients'] == {}
df = df.loc[~no_ingredients]
```

- These lines create a boolean mask `no_ingredients` that checks if the 'ingredients' column in `df` is an empty dictionary (`{}`). Rows with an empty 'ingredients' value are dropped from the DataFrame using `loc[~no_ingredients]` .

```
df['Features'] = df['Keywords'].apply(lambda x: [word for phrase in x for word in phrase.split(' ')]) \
                 + df['ingredients_names'].apply(lambda x: [word for phrase in x for word in phrase.split(' ')])
```

- This line creates a new column 'Features' in the DataFrame `df` . It combines the words from the 'Keywords' and 'ingredients_names' columns into a single list by splitting the phrases on spaces. The words are extracted using a list comprehension.

```
print('\n Features column created \n')
```

- This line prints a message indicating that the 'Features' column has been created.

```
df['ingredients_names'] = df['ingredients_names'].apply(lambda x: [word for phrase in x for word in phrase.split(' ')])
```

- This line further splits the phrases in the 'ingredients_names' column on spaces and creates a new list of individual words.

```
def remove_non_alpha(df, column):
    non_alpha_pattern = re.compile(r'[^a-zA-Z]')
    df[column] = df[column].apply(lambda x: [re.sub(non_alpha_pattern, '', item) for item in x])
    df[column] = df[column].apply(lambda x: [re.sub(r'http\S+', '', item) for item in x])
    return df

df = remove_non_alpha(df, 'Features')
df = remove_non_alpha(df, 'ingredients_names')
```

- These lines define a function `remove_non_alpha()` that removes non-alphabetical characters from each element in a given column of the DataFrame `df` . It applies two regular expression substitutions using `re.sub()` to remove non-alphabetical characters and URLs. The function is then applied to the 'Features' and 'ingredients_names' columns.

```
df.drop(columns=['Keywords'], inplace=True)
```

- This line drops the ‘Keywords’ column from the DataFrame.

```
df['ingredients_names'] = df['ingredients_names'].apply(lambda x: [item for item in x if item != ''])
```

- This line removes empty strings from each list in the ‘ingredients_names’ column.

```
df.to_csv('cleaned.csv')
```

- This line saves the cleaned DataFrame as a CSV file named ‘cleaned.csv’.

```
end = time.time()
execution_time = end - start
print('Total runtime:', execution_time)
```

- These lines record the end time of the program execution, calculate the total runtime by subtracting the start time from the end time, and print the total runtime.