Identifying Key Entities in Recipe Data

Business Objective: The goal of this assignment is to train a Named Entity Recognition (NER) model using Conditional Random Fields (CRF) to extract key entities from recipe data. The model will classify words into predefined categories such as ingredients, quantities and units, enabling the creation of a structured database of recipes and ingredients that can be used to power advanced features in recipe management systems, dietary tracking apps, or e-commerce platforms.

Data Description

The given data is in JSON format, representing a **structured recipe ingredient list** with **Named Entity Recognition (NER) labels**. Below is a breakdown of the data fields:


```
[
    {
        "input": "6 Karela Bitter Gourd Pavakkai Salt 1 Onion 3 tablespoon Gram flour besan 2 teaspoons Turmeric powder Haldi Red Chilli Cumin seeds Jeera Coriander Powder Dhania Amchur Dry Mango Sunflower Oil",
        "pos": "quantity ingredient ingredient ingredient ingredient ingredient quantity ingredient quantity unit ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient"
    },
    {
      "input": "2-1/2 cups rice cooked 3 tomatoes teaspoons BC Belle Bhat powder 1 teaspoon chickpea lentils 1/2 cumin seeds white urad dal mustard green chilli dry red 2 cashew or peanuts 1-1/2 tablespoon oil asafoetida",
      "pos": "quantity unit ingredient ingredient quantity ingredient unit ingredient ingredient ingredient ingredient quantity unit ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient quantity unit ingredient ingredient"
    }
]
```


Key	Description
`input`	Contains a raw ingredient list from a recipe.

Key	Description
`pos`	Represents the corresponding part-of-speech (POS) tags or NER labels, identifying quantities, ingredients, and units.

1 Import libraries

1.1 Installation of sklearn-crfsuite

sklearn-crfsuite is a Python wrapper for CRFsuite, a fast and efficient implementation of Conditional Random Fields (CRFs). It is designed to integrate seamlessly with scikit-learn for structured prediction tasks such as Named Entity Recognition (NER), Part-of-Speech (POS) tagging, and chunking.

In [1]:
```
# installation of sklearn_crfsuite
!pip install sklearn_crfsuite==0.5.0
```

```
Collecting sklearn_crfsuite==0.5.0
  Downloading sklearn_crfsuite-0.5.0-py2.py3-none-any.whl.metadata (4.9 kB)
Collecting python-crfsuite>=0.9.7 (from sklearn_crfsuite==0.5.0)
  Downloading python_crfsuite-0.9.11-cp311-cp311-manylinux_2_17_x86_64.manylinu
x2014_x86_64.whl.metadata (4.3 kB)
Requirement already satisfied: scikit-learn>=0.24.0 in /usr/local/lib/python3.1
1/dist-packages (from sklearn_crfsuite==0.5.0) (1.6.1)
Requirement already satisfied: tabulate>=0.4.2 in /usr/local/lib/python3.11/dis
t-packages (from sklearn_crfsuite==0.5.0) (0.9.0)
Requirement already satisfied: tqdm>=2.0 in /usr/local/lib/python3.11/dist-pack
ages (from sklearn_crfsuite==0.5.0) (4.67.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn>=0.24.0->sklearn_crfsuite==0.5.0) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-p
ackages (from scikit-learn>=0.24.0->sklearn_crfsuite==0.5.0) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn>=0.24.0->sklearn_crfsuite==0.5.0) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.1
1/dist-packages (from scikit-learn>=0.24.0->sklearn_crfsuite==0.5.0) (3.6.0)
Downloading sklearn_crfsuite-0.5.0-py2.py3-none-any.whl (10 kB)
Downloading python_crfsuite-0.9.11-cp311-cp311-manylinux_2_17_x86_64.manylinux2
014_x86_64.whl (1.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 17.2 MB/s eta 0:00:00
Installing collected packages: python-crfsuite, sklearn_crfsuite
Successfully installed python-crfsuite-0.9.11 sklearn_crfsuite-0.5.0
```

1.2 Import necessary libraries

In [2]:
```
# Import warnings
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Import necessary libraries
```

```
import json  # For handling JSON data
import pandas as pd  # For data manipulation and analysis
import re  # For regular expressions (useful for text preprocessing)
import matplotlib.pyplot as plt  # For visualisation
import seaborn as sns  # For advanced data visualisation
import sklearn_crfsuite  # CRF (Conditional Random Fields) implementation for
import numpy as np  # For numerical computations
# Saving and loading machine learning models
import joblib
import random
import spacy
from IPython.display import display, Markdown # For displaying well-formatted

from fractions import Fraction  # For handling fractional values in numerical
# Importing tools for feature engineering and model training
from collections import Counter  # For counting occurrences of elements in a l
from sklearn.model_selection import train_test_split  # For splitting dataset
from sklearn_crfsuite import metrics  # For evaluating CRF models
from sklearn_crfsuite.metrics import flat_classification_report
from sklearn.utils.class_weight import compute_class_weight
from collections import Counter
from sklearn.metrics import confusion_matrix
```

In [4]:
```
# Ensure pandas displays full content
pd.set_option('display.max_colwidth', None)
pd.set_option('display.expand_frame_repr', False)
```

2 Data Ingestion and Preparation

2.1 *Read Recipe Data from Dataframe and prepare the data for analysis*

Read the data from JSON file, print first five rows and describe the dataframe

2.1.1 Define a *load_json_dataframe* function

Define a function that takes path of the ingredient_and_quantity.json file and reads it, convert it into dataframe - df and return it.

In [5]:
```
# define a function to load json file to a dataframe
def load_json_dataframe(json_path):
    """
    Loads a JSON file containing recipe data and returns a pandas DataFrame.

    Args:
        json_path (str): Path to the JSON file.

    Returns:
        pd.DataFrame: DataFrame containing the recipe data.
    """
```

```
    with open(json_path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    df = pd.DataFrame(data)
    return df
```

2.1.2 Execute the *load_json_dataframe* function

In [6]:
```
# read the json file by giving the file path and create a dataframe
json_path = "ingredient_and_quantity.json"  # update with your actual file pat
df = load_json_dataframe(json_path)
```

2.1.3 Describe the dataframe

Print first five rows of dataframe along with dimensions. Display the information of dataframe

In [7]:
```
# display first five rows of the dataframe - df
df.head()
```

Out[7]:

	input	pos
0	6 Karela Bitter Gourd Pavakkai Salt 1 Onion 3 tablespoon Gram flour besan 2 teaspoons Turmeric powder Haldi Red Chilli Cumin seeds Jeera Coriander Powder Dhania Amchur Dry Mango Sunflower Oil	quantity ingredient ingredient ingredient ingredient ingredient quantity ingredient quantity unit ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient
1	2-1/2 cups rice cooked 3 tomatoes teaspoons BC Belle Bhat powder 1 teaspoon chickpea lentils 1/2 cumin seeds white urad dal mustard green chilli dry red 2 cashew or peanuts 1-1/2 tablespoon oil asafoetida	quantity unit ingredient ingredient quantity ingredient unit ingredient ingredient ingredient ingredient quantity unit ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient quantity unit ingredient ingredient
2	1-1/2 cups Rice Vermicelli Noodles Thin 1 Onion sliced 1/2 cup Carrots Gajjar chopped 1/3 Green peas Matar 2 Chillies 1/4 teaspoon Asafoetida hing Mustard seeds White Urad Dal Split Ghee sprig Curry leaves Salt Lemon juice	quantity unit ingredient ingredient ingredient ingredient quantity ingredient ingredient quantity unit ingredient ingredient ingredient quantity ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient
3	500 grams Chicken 2 Onion chopped 1 Tomato 4 Green Chillies slit inch Ginger finely 6 cloves Garlic 1/2 teaspoon Turmeric powder Haldi Garam masala tablespoon Sesame Gingelly Oil 1/4 Methi Seeds Fenugreek Coriander Dhania Dry Red Fennel seeds Saunf cups Sorrel Leaves Gongura picked and	quantity unit ingredient quantity ingredient ingredient quantity ingredient quantity ingredient ingredient ingredient unit ingredient ingredient quantity unit ingredient quantity unit ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient
4	1 tablespoon chana dal white urad 2 red chillies coriander seeds 3 inches ginger onion tomato Teaspoon mustard asafoetida sprig curry	quantity unit ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient unit ingredient ingredient unit ingredient

In [8]:
```
# print the dimensions of dataframe - df
print(df.shape)
```

```
(285, 2)
```

In [9]:
```
# print the information of the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 285 entries, 0 to 284
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   input   285 non-null    object
 1   pos     285 non-null    object
dtypes: object(2)
memory usage: 4.6+ KB
```

2.2 *Recipe Data Manipulation*

Create derived metrics in dataframe and provide insights of the dataframe

2.2.1 Create input_tokens and pos_tokens columns by splitting the input and pos from the dataframe

Split the input and pos into input_tokens and pos_tokens in the dataframe and display it in the dataframe

In [10]:
```
# split the input and pos into input_tokens and pos_tokens in the dataframe

# Tokenize input
df['input_tokens'] = df['input'].apply(lambda x: x.split())
# Tokenize POS
df['pos_tokens'] = df['pos'].apply(lambda x: x.split())
```

In [11]:
```
# display first five rows of the dataframe - df
df.head()
```

Out[11]:

	input	pos	input_tokens	pos_tokens
0	6 Karela Bitter Gourd Pavakkai Salt 1 Onion 3 tablespoon Gram flour besan 2 teaspoons Turmeric powder Haldi Red Chilli Cumin seeds Jeera Coriander Powder Dhania Amchur Dry Mango Sunflower Oil	quantity ingredient ingredient ingredient ingredient ingredient quantity ingredient quantity unit ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient	[6, Karela, Bitter, Gourd, Pavakkai, Salt, 1, Onion, 3, tablespoon, Gram, flour, besan, 2, teaspoons, Turmeric, powder, Haldi, Red, Chilli, Cumin, seeds, Jeera, Coriander, Powder, Dhania, Amchur, Dry, Mango, Sunflower, Oil]	[quantity, ingredient, ingredient, ingredient, ingredient, ingredient, quantity, ingredient, quantity, unit, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient]
1	2-1/2 cups rice cooked 3 tomatoes teaspoons BC Belle Bhat powder 1 teaspoon chickpea lentils 1/2 cumin seeds white urad dal mustard green chilli dry red 2 cashew or peanuts 1-1/2 tablespoon oil asafoetida	quantity unit ingredient ingredient quantity ingredient unit ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient quantity unit ingredient ingredient	[2-1/2, cups, rice, cooked, 3, tomatoes, teaspoons, BC, Belle, Bhat, powder, 1, teaspoon, chickpea, lentils, 1/2, cumin, seeds, white, urad, dal, mustard, green, chilli, dry, red, 2, cashew, or, peanuts, 1-1/2, tablespoon, oil, asafoetida]	[quantity, unit, ingredient, ingredient, quantity, ingredient, unit, ingredient, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient]
2	1-1/2 cups Rice Vermicelli Noodles Thin 1 Onion sliced 1/2 cup Carrots Gajjar chopped 1/3 Green peas Matar 2 Chillies 1/4 teaspoon Asafoetida hing Mustard seeds White Urad Dal Split Ghee sprig Curry leaves Salt Lemon juice	quantity unit ingredient ingredient ingredient ingredient quantity ingredient ingredient quantity unit ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient quantity unit ingredient ingredient	[1-1/2, cups, Rice, Vermicelli, Noodles, Thin, 1, Onion, sliced, 1/2, cup, Carrots, Gajjar, chopped, 1/3, Green, peas, Matar, 2, Chillies, 1/4, teaspoon, Asafoetida, hing, Mustard, seeds, White, Urad, Dal, Split, Ghee, sprig, Curry, leaves, Salt, Lemon, juice]	[quantity, unit, ingredient, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, quantity, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient]

	input	pos	input_tokens	pos_tokens
3	500 grams Chicken 2 Onion chopped 1 Tomato 4 Green Chillies slit inch Ginger finely 6 cloves Garlic 1/2 teaspoon Turmeric powder Haldi Garam masala tablespoon Sesame Gingelly Oil 1/4 Methi Seeds Fenugreek Coriander Dhania Dry Red Fennel seeds Saunf cups Sorrel Leaves Gongura picked and	quantity unit ingredient quantity ingredient ingredient quantity ingredient quantity ingredient ingredient ingredient unit ingredient ingredient quantity unit ingredient quantity unit ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient	[500, grams, Chicken, 2, Onion, chopped, 1, Tomato, 4, Green, Chillies, slit, inch, Ginger, finely, 6, cloves, Garlic, 1/2, teaspoon, Turmeric, powder, Haldi, Garam, masala, tablespoon, Sesame, Gingelly, Oil, 1/4, Methi, Seeds, Fenugreek, Coriander, Dhania, Dry, Red, Fennel, seeds, Saunf, cups, Sorrel, Leaves, Gongura, picked, and]	[quantity, unit, ingredient, quantity, ingredient, ingredient, quantity, ingredient, quantity, ingredient, ingredient, unit, ingredient, ingredient, quantity, unit, ingredient, quantity, unit, ingredient, ingredient, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, ingredient]
4	1 tablespoon chana dal white urad 2 red chillies coriander seeds 3 inches ginger onion tomato Teaspoon mustard asafoetida sprig curry	quantity unit ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient quantity unit ingredient ingredient ingredient unit ingredient ingredient unit ingredient	[1, tablespoon, chana, dal, white, urad, 2, red, chillies, coriander, seeds, 3, inches, ginger, onion, tomato, Teaspoon, mustard, asafoetida, sprig, curry]	[quantity, unit, ingredient, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, ingredient]

2.2.2 Provide the length for input_tokens and pos_tokens and validate their length

Create input_length and pos_length columns in the dataframe and validate both the lengths. Check for the rows that are unequal in input and pos length

In [12]:
```
# create input_length and pos_length columns for the input_tokens and pos-toke
df['input_length'] = df['input_tokens'].apply(len)
df['pos_length'] = df['pos_tokens'].apply(len)
```

In [13]:
```
# check for the equality of input_length and pos_length in the dataframe
unequal_rows = df[df['input_length'] != df['pos_length']]
print(f"Number of rows with unequal input and pos lengths: {len(unequal_rows)}
```

```
if not unequal_rows.empty:
    display(unequal_rows[['input', 'pos', 'input_length', 'pos_length']])
```

```
Number of rows with unequal input and pos lengths: 5
```

	input	pos	input_length	pos_length
17	2 cups curd 1 cup gourd cucumber green cor coriander 1/2 teaspoon cumin powder salt	quantity unit ingredient quantity unit ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient	15	14
27	1 Baguette sliced 1 1/2 tablespoon Butter 1/2 Garlic minced cup Spinach Leaves Palak Red Bell pepper Capsicum Tomato finely chopped Onion Black powder Italian seasoning teaspoon Fresh cream Cheddar cheese grated Salt Roasted tomato pasta sauce	quantity ingredient ingredient quantity unit ingredient quantity ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient	37	36
79	1/2 cup Poha Flattened rice 2 tablespoons Rice flour 2 1/2 liter Milk 1 Nolen Gur or brown sugar Cardamom Elaichi Pods/Seeds 8-10 Mixed nuts almonds/cashews tablespoon Raisins pinch Saffron strands and a little more for garnish Salt	quantity unit ingredient ingredient ingredient quantity unit ingredient ingredient quantity unit ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient unit ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient	38	37
164	1/2 cup All Purpose Flour Maida Whole Wheat 1/4 Hung Curd Greek Yogurt 250 grams Chicken minced 1 Spinach Leaves Palak finely chopped Onion 4 cloves Garlic Tomatoes tablespoon Cumin powder Jeera Coriander Powder Dhania 1 1/2 teaspoon Paprika Black pepper 3 sprig Mint Pudina 10 Spring Bulb & Greens 100 Feta	quantity unit ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient quantity unit ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient quantity ingredient ingredient	54	53

	input	pos	input_length	pos_length
	Cheese crumbled			
207	1 cup Cashew nuts Badam Almond 1 1/4 cups Sugar 1/2 Water teaspoon Cardamom Powder Ghee for greasing	quantity unit ingredient ingredient ingredient ingredient quantity unit ingredient quantity ingredient unit ingredient ingredient ingredient unit ingredient	18	17

2.2.3 Define a unique_labels function and validate the labels in pos_tokens

Define a unique_labels function which checks for all the unique pos labels in the recipe & execute it.

In [14]:
```
# Define a unique_labels function to checks for all the unique pos labels in t
def unique_labels(df):
    unique = set(label for tokens in df['pos_tokens'] for label in tokens)
    print("Unique POS labels in the recipe data:", unique)
    return unique

# Execute the function
unique_labels(df)
```

```
Unique POS labels in the recipe data: {'ingredient', 'quantity', 'unit'}
```

Out[14]:
```
{'ingredient', 'quantity', 'unit'}
```

2.2.3 Provide the insights seen in the recipe data after validation

Provide the indexes that requires cleaning and formatting in the dataframe

2.2.4 Drop the rows that have invalid data provided in previous cell

In [15]:
```
# drop the irrelevant recipe data
df = df[df['input_length'] == df['pos_length']].reset_index(drop=True)
```

2.2.5 Update the input_length & pos_length in dataframe

In [16]:
```
# update the input and pos length in input_length and pos_length
df['input_length'] = df['input_tokens'].apply(len)
df['pos_length'] = df['pos_tokens'].apply(len)
```

2.2.6 Validate the input_length and pos_length by checking unequal rows

In [17]:
```
# validate the input length and pos length as input_length and pos_length
unequal_rows = df[df['input_length'] != df['pos_length']]
print(f"Number of rows with unequal input and pos lengths after cleaning: {len
```

```
if not unequal_rows.empty:
    display(unequal_rows[['input', 'pos', 'input_length', 'pos_length']])
else:
    print("All rows have matching input_length and pos_length.")
```

```
Number of rows with unequal input and pos lengths after cleaning: 0
All rows have matching input_length and pos_length.
```

3 Train Validation Split (70 train - 30 val)

3.1 *Perform train and validation split ratio*

Split the dataset with the help of input_tokens and pos_tokens and make a ratio of 70:30 split for training and validation datasets.

3.1.1 Split the dataset into train_df and val_df into 70:30 ratio

In [18]:
```
# split the dataset into training and validation sets
from sklearn.model_selection import train_test_split

train_df, val_df = train_test_split(
    df,
    test_size=0.3,
    random_state=42,
    shuffle=True
)

print(f"Training set size: {len(train_df)}")
print(f"Validation set size: {len(val_df)}")
```

```
Training set size: 196
Validation set size: 84
```

3.1.2 Print the first five rows of train_df and val_df

In [19]:
```
# print the first five rows of train_df
train_df.head()
```

Out[19]:

	input	pos	input_tokens	pos_tokens	input_length	pos_length
175	250 grams Okra Oil 1 Onion finely chopped Tomato Grated teaspoon Ginger 2 Garlic Finely 1/2 Cumin seeds 1/4 Teaspoon asafoetida cup cottage cheese pinched coriander powder mango red chilli turmeric	quantity unit ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient unit ingredient quantity ingredient ingredient quantity ingredient ingredient quantity unit ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient	[250, grams, Okra, Oil, 1, Onion, finely, chopped, Tomato, Grated, teaspoon, Ginger, 2, Garlic, Finely, 1/2, Cumin, seeds, 1/4, Teaspoon, asafoetida, cup, cottage, cheese, pinched, coriander, powder, mango, red, chilli, turmeric]	[quantity, unit, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, ingredient, unit, ingredient, quantity, ingredient, ingredient, quantity, ingredient, ingredient, quantity, unit, ingredient, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient]	31	31
55	200 grams Paneer Homemade Cottage Cheese 2 Potato Aloo Bay leaf tej patta Dry Red Chilli 1 tablespoon Panch Phoran Masala roasted and powdered Tomato big sized teaspoon Turmeric powder Haldi Cumin seeds Jeera Ginger	quantity unit ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient	[200, grams, Paneer, Homemade, Cottage, Cheese, 2, Potato, Aloo, Bay, leaf, tej, patta, Dry, Red, Chilli, 1, tablespoon, Panch, Phoran, Masala, roasted, and, powdered, Tomato, big, sized, teaspoon, Turmeric, powder, Haldi, Cumin, seeds, Jeera, Ginger, grated, Salt, 1/2, Sugar, Sunflower,	[quantity, unit, ingredient, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient,	41	41

	input	pos	input_tokens	pos_tokens	input_length	pos_length
	grated Salt 1/2 Sugar Sunflower Oil	ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient	Oil]	ingredient, ingredient, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, ingredient]		
109	500 grams Cabbage Patta Gobi Muttaikose 1 teaspoon Mustard seeds 1-1/2 White Urad Dal Split sprig Curry leaves Green Chilli 1/4 cup Fresh coconut Salt	quantity unit ingredient ingredient ingredient ingredient quantity unit ingredient ingredient quantity ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient	[500, grams, Cabbage, Patta, Gobi, Muttaikose, 1, teaspoon, Mustard, seeds, 1-1/2, White, Urad, Dal, Split, sprig, Curry, leaves, Green, Chilli, 1/4, cup, Fresh, coconut, Salt]	[quantity, unit, ingredient, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient]	25	25
213	500 grams Fresh Figs 1/4 cup Lemon juice 1 teaspoon zest 2 Red Chilli flakes 1/2 Honey Brown Sugar (Demerara Sugar)	quantity unit ingredient ingredient quantity unit ingredient ingredient quantity unit ingredient quantity ingredient ingredient	[500, grams, Fresh, Figs, 1/4, cup, Lemon, juice, 1, teaspoon, zest, 2, Red, Chilli, flakes, 1/2, Honey, Brown, Sugar, (Demerara, Sugar)]	[quantity, unit, ingredient, ingredient, quantity, unit, ingredient, ingredient, quantity, unit, ingredient, quantity, ingredient, ingredient,	21	21

	input	pos	input_tokens	pos_tokens	input_length	pos_length
		ingredient quantity ingredient ingredient ingredient ingredient ingredient		ingredient, quantity, ingredient, ingredient, ingredient, ingredient, ingredient]		
38	2 cups Water 1 teaspoon Tea leaves 1/4 Milk 10 Saffron strands	quantity unit ingredient quantity unit ingredient ingredient quantity ingredient quantity ingredient ingredient	[2, cups, Water, 1, teaspoon, Tea, leaves, 1/4, Milk, 10, Saffron, strands]	[quantity, unit, ingredient, quantity, unit, ingredient, ingredient, quantity, ingredient, quantity, ingredient, ingredient]	12	12

In [20]:
```
# print the first five rows of the val_df
val_df.head()
```

Out[20]:

	input	pos	input_tokens	pos_tokens	input_length	pos_length
33	1 cup Ada 2 liter Milk 3/4 Sugar tablespoon Ghee 1/2 teaspoon Cardamom Powder Elaichi	quantity unit ingredient quantity unit ingredient quantity ingredient unit ingredient quantity unit ingredient ingredient ingredient	[1, cup, Ada, 2, liter, Milk, 3/4, Sugar, tablespoon, Ghee, 1/2, teaspoon, Cardamom, Powder, Elaichi]	[quantity, unit, ingredient, quantity, unit, ingredient, quantity, ingredient, unit, ingredient, quantity, unit, ingredient, ingredient, ingredient]	15	15
108	1 Carrot Gajjar chopped 7 Potatoes Aloo 2 cups Cauliflower gobi cut to small florets Onion tablespoon Ginger Garlic Paste Salt teaspoons Sunflower Oil 1/2 cup Fresh coconut grated teaspoon Whole Black Peppercorns Green Chillies Fennel seeds Saunf Poppy 6 Cashew nuts inch Cinnamon Stick Dalchini Star anise 3 Cloves Laung Cardamom Elaichi Pods/Seeds Cumin Jeera	quantity ingredient ingredient ingredient quantity ingredient ingredient quantity unit ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient	[1, Carrot, Gajjar, chopped, 7, Potatoes, Aloo, 2, cups, Cauliflower, gobi, cut, to, small, florets, Onion, tablespoon, Ginger, Garlic, Paste, Salt, teaspoons, Sunflower, Oil, 1/2, cup, Fresh, coconut, grated, teaspoon, Whole, Black, Peppercorns, Green, Chillies, Fennel, seeds, Saunf, Poppy, 6, Cashew, nuts, inch, Cinnamon, Stick, Dalchini, Star, anise, 3, Cloves, Laung, Cardamom, Elaichi, Pods/Seeds, Cumin, Jeera]	[quantity, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, quantity, ingredient,	56	56

	input	pos	input_tokens	pos_tokens	input_length	pos_length
		ingredient unit ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient		ingredient, unit, ingredient, ingredient, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient]		
240	1 tablespoon Sunflower Oil 3 Potato Aloo Ginger paste Green Chilli chopped 1-1/12 tablespoons Sesame seeds Til teaspoon Red powder Cumin Jeera Coriander Powder Dhania 1/2 Garam masala 2 Sweet Chutney Date Tamarind Leaves few	quantity unit ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient	[1, tablespoon, Sunflower, Oil, 3, Potato, Aloo, Ginger, paste, Green, Chilli, chopped, 1-1/ 12, tablespoons, Sesame, seeds, Til, teaspoon, Red, powder, Cumin, Jeera, Coriander, Powder, Dhania, 1/2, Garam, masala, 2, Sweet, Chutney, Date, Tamarind, Leaves, few]	[quantity, unit, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, ingredient, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient]	35	35
259	1 cup green peas gram flour 1/2 cheese tsp ginger 2	quantity unit ingredient ingredient ingredient	[1, cup, green, peas, gram, flour, 1/ 2, cheese, tsp, ginger, 2,	[quantity, unit, ingredient, ingredient, ingredient,	18	18

	input	pos	input_tokens	pos_tokens	input_length	pos_length
	chillies turmeric powder cumin teaspoon salt oil	ingredient quantity ingredient unit ingredient quantity ingredient ingredient ingredient ingredient unit ingredient ingredient	chillies, turmeric, powder, cumin, teaspoon, salt, oil]	ingredient, quantity, ingredient, unit, ingredient, quantity, ingredient, ingredient, ingredient, ingredient, unit, ingredient, ingredient]		
154	2 cups Brown Rice cooked tablespoons Garlic chopped 1 Green Chilli 1/2 cup Carrots (Gajjar) beans (French Beans) Bell Pepper (Capsicum) Onion Cabbage (Patta Gobi/ Muttaikose) tablespoon Roasted tomato pasta sauce - or store bought Red teaspoon Soy Ginger freshly grated Spring Greens Salt Vinegar Extra Virgin Olive Oil as required	quantity unit ingredient ingredient ingredient unit ingredient quantity ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient	[2, cups, Brown, Rice, cooked, tablespoons, Garlic, chopped, 1, Green, Chilli, 1/2, cup, Carrots, (Gajjar), beans, (French, Beans), Bell, Pepper, (Capsicum), Onion, Cabbage, (Patta, Gobi/, Muttaikose), tablespoon, Roasted, tomato, pasta, sauce, -, or, store, bought, Red, teaspoon, Soy, Ginger, freshly, grated, Spring, Greens, Salt, Vinegar, Extra, Virgin, Olive, Oil, as, required]	[quantity, unit, ingredient, ingredient, ingredient, unit, ingredient, quantity, ingredient, ingredient, quantity, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, unit, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient	51	51

input	pos	input_tokens	pos_tokens	input_length	pos_length
	ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient		ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient, ingredient]		

3.1.3 Extract the dataset into train_df and val_df into X_train, X_val, y_train and y_val and display their length

Extract X_train, X_val, y_train and y_val by extracting the list of input_tokens and pos_tokens from train_df and val_df and also display their length

In [21]:
```
# extract the training and validation sets by taking input_tokens and pos_toke
X_train = train_df['input_tokens'].tolist()
y_train = train_df['pos_tokens'].tolist()
X_val = val_df['input_tokens'].tolist()
y_val = val_df['pos_tokens'].tolist()

print(f"Length of X_train: {len(X_train)}")
print(f"Length of y_train: {len(y_train)}")
print(f"Length of X_val: {len(X_val)}")
print(f"Length of y_val: {len(y_val)}")
```

```
Length of X_train: 196
Length of y_train: 196
Length of X_val: 84
Length of y_val: 84
```

In [22]:
```
# validate the shape of training and validation samples
for i in range(5):
    print(f"Train sample {i}: input_tokens={len(X_train[i])}, pos_tokens={len(
for i in range(5):
    print(f"Validation sample {i}: input_tokens={len(X_val[i])}, pos_tokens={l

# Check if all samples have matching input and pos token lengths
train_mismatch = [i for i in range(len(X_train)) if len(X_train[i]) != len(y_t
val_mismatch = [i for i in range(len(X_val)) if len(X_val[i]) != len(y_val[i])
print(f"Number of mismatched train samples: {len(train_mismatch)}")
print(f"Number of mismatched validation samples: {len(val_mismatch)}")
```

```
Train sample 0: input_tokens=31, pos_tokens=31
Train sample 1: input_tokens=41, pos_tokens=41
Train sample 2: input_tokens=25, pos_tokens=25
Train sample 3: input_tokens=21, pos_tokens=21
Train sample 4: input_tokens=12, pos_tokens=12
Validation sample 0: input_tokens=15, pos_tokens=15
Validation sample 1: input_tokens=56, pos_tokens=56
Validation sample 2: input_tokens=35, pos_tokens=35
Validation sample 3: input_tokens=18, pos_tokens=18
Validation sample 4: input_tokens=51, pos_tokens=51
Number of mismatched train samples: 0
Number of mismatched validation samples: 0
```

3.1.4 Display the number of unique labels present in y_train

In [25]:
```python
# Display the number of unique labels present in y_train
unique_labels = set(label for sample in y_train for label in sample)
print(f"Number of unique labels in y_train: {len(unique_labels)}")
print(f"Unique labels: {unique_labels}")
```

```
Number of unique labels in y_train: 3
Unique labels: {'ingredient', 'quantity', 'unit'}
```

4 Exploratory Recipe Data Analysis on Training Dataset

4.1 *Flatten the lists for input_tokens & pos_tokens*

Define a function **flatten_list** for flattening the structure for input_tokens and pos_tokens. The input parameter passed to this function is a nested list.

Initialise the dataset_name with a value '***Training***'

In [26]:
```python
# flatten the list for nested_list (input_tokens, pos_tokens)
def flatten_list(nested_list):
    """
    Flattens a nested list (list of lists) into a single list.
    """
    return [item for sublist in nested_list for item in sublist]
```

In [27]:
```python
# initialise the dataset_name
dataset_name = 'Training'
```

4.2 *Extract and validate the tokens after using the flattening technique*

Define a function named ***extract_and_validate_tokens*** with parameters dataframe and dataset_name (Training/Validation), validate the length of input_tokens and pos_tokens from dataframe and display first 10 records for both

the input_tokens and pos_tokens. Execute this function

In [28]:
```
# define a extract_and_validate_tokens with parameters (df, dataset_name)
# call the flatten_list and apply it on input_tokens and pos_tokens
# validate their length and display first 10 records having input and pos toke
def extract_and_validate_tokens(df, dataset_name):
    """
    Flattens input_tokens and pos_tokens columns, validates their lengths,
    and displays the first 10 records for both.
    """
    flat_input_tokens = flatten_list(df['input_tokens'].tolist())
    flat_pos_tokens = flatten_list(df['pos_tokens'].tolist())
    print(f"{dataset_name} - Total input tokens: {len(flat_input_tokens)}")
    print(f"{dataset_name} - Total pos tokens: {len(flat_pos_tokens)}")
    if len(flat_input_tokens) == len(flat_pos_tokens):
        print(f"Lengths match for {dataset_name} dataset.")
    else:
        print(f"Lengths do NOT match for {dataset_name} dataset!")
    print("\nFirst 10 input tokens:", flat_input_tokens[:10])
    print("First 10 pos tokens:", flat_pos_tokens[:10])
```

In [29]:
```
# extract the tokens and its pos tags
extract_and_validate_tokens(train_df, 'Training')
extract_and_validate_tokens(val_df, 'Validation')
```

```
Training - Total input tokens: 7114
Training - Total pos tokens: 7114
Lengths match for Training dataset.

First 10 input tokens: ['250', 'grams', 'Okra', 'Oil', '1', 'Onion', 'finely',
'chopped', 'Tomato', 'Grated']
First 10 pos tokens: ['quantity', 'unit', 'ingredient', 'ingredient', 'quantit
y', 'ingredient', 'ingredient', 'ingredient', 'ingredient', 'ingredient']
Validation - Total input tokens: 2876
Validation - Total pos tokens: 2876
Lengths match for Validation dataset.

First 10 input tokens: ['1', 'cup', 'Ada', '2', 'liter', 'Milk', '3/4', 'Suga
r', 'tablespoon', 'Ghee']
First 10 pos tokens: ['quantity', 'unit', 'ingredient', 'quantity', 'unit', 'in
gredient', 'quantity', 'ingredient', 'unit', 'ingredient']
```

4.3 *Categorise tokens into labels (unit, ingredient, quantity)*

Define a function ***categorize_tokens*** to categorise tokens into ingredients, units and quantities by using extracted tokens in the previous code and return a list of ingredients, units and quantities. Execute this function to get the list.

In [30]:
```
# define a categorize_tokens function and provide the tokens and pos_tags as p
# validate the list that it comprised of these labels, if not return empty arr

def categorize_tokens(tokens, pos_tags):
```

```
    """
    Categorizes tokens into ingredients, units, and quantities based on their
    Returns three lists: ingredients, units, quantities.
    If pos_tags contain labels other than 'ingredient', 'unit', or 'quantity',
    """
    valid_labels = {'ingredient', 'unit', 'quantity'}
    if not set(pos_tags).issubset(valid_labels):
        # If there are unexpected labels, return empty arrays
        return [], [], []
    ingredients = [token for token, tag in zip(tokens, pos_tags) if tag == 'in
    units = [token for token, tag in zip(tokens, pos_tags) if tag == 'unit']
    quantities = [token for token, tag in zip(tokens, pos_tags) if tag == 'qua
    return ingredients, units, quantities
```

In [31]:
```
#  call the function to categorise the labels into respective list
flat_input_tokens = flatten_list(train_df['input_tokens'].tolist())
flat_pos_tokens = flatten_list(train_df['pos_tokens'].tolist())
ingredients, units, quantities = categorize_tokens(flat_input_tokens, flat_pos

print("Sample ingredients:", ingredients[:10])
print("Sample units:", units[:10])
print("Sample quantities:", quantities[:10])
```

```
Sample ingredients: ['Okra', 'Oil', 'Onion', 'finely', 'chopped', 'Tomato', 'Gr
ated', 'Ginger', 'Garlic', 'Finely']
Sample units: ['grams', 'teaspoon', 'Teaspoon', 'cup', 'grams', 'tablespoon',
'teaspoon', 'grams', 'teaspoon', 'sprig']
Sample quantities: ['250', '1', '2', '1/2', '1/4', '200', '2', '1', '1/2', '50
0']
```

4.4 *Top 10 Most Frequent Items*

Define a function ***get_top_frequent_items*** to display top 10 most frequent items

Here, item_list is used as a general parameter where you will call this function for ingredient and unit list

Execute this function separately for top 10 most units and ingredients

In [32]:
```
# define a function get_top_frequent_items to get the top frequent items by us
from collections import Counter

def get_top_frequent_items(item_list, label, dataset_name):
    """
    Displays the top 10 most frequent items in the given item_list.
    """
    counter = Counter(item_list)
    top_items = counter.most_common(10)
    print(f"\nTop 10 most frequent {label}s in {dataset_name} dataset:")
    for item, count in top_items:
        print(f"{item}: {count}")
    return top_items
```

In [33]:
```
# get the top ingredients which are frequently seen in the recipe
top_ingredients = get_top_frequent_items(ingredients, 'ingredient', 'Training'
```

```
Top 10 most frequent ingredients in Training dataset:
powder: 129
Salt: 102
seeds: 89
Green: 85
chopped: 84
Oil: 83
Red: 81
Chilli: 77
Coriander: 71
Sunflower: 65
```

In [34]:
```
# get the top units which are frequently seen in the recipe
top_units = get_top_frequent_items(units, 'unit', 'Training')
```

```
Top 10 most frequent units in Training dataset:
teaspoon: 162
cup: 136
tablespoon: 99
grams: 63
tablespoons: 61
inch: 52
cups: 50
sprig: 41
cloves: 39
teaspoons: 39
```

4.5 *Plot Top 10 most frequent items*

Define a function ***plot_top_items*** to plot a bar graph on top 10 most frequent items for units and ingredients

Here, item_list is used as a general parameter where you will call this function for ingredient and unit list

In [35]:
```
# define plot top items with parameters - top_item list, label to suggest whet
import matplotlib.pyplot as plt

def plot_top_items(top_items, label, dataset_name):
    """
    Plots a bar graph for the top 10 most frequent items (ingredients or units

    Parameters:
        top_items (list of tuples): List of (item, count) pairs.
        label (str): 'ingredient' or 'unit'.
        dataset_name (str): Name of the dataset (e.g., 'Training').
    """
    items, counts = zip(*top_items)
    plt.figure(figsize=(10, 5))
    sns.barplot(x=list(items), y=list(counts), palette="viridis")
```

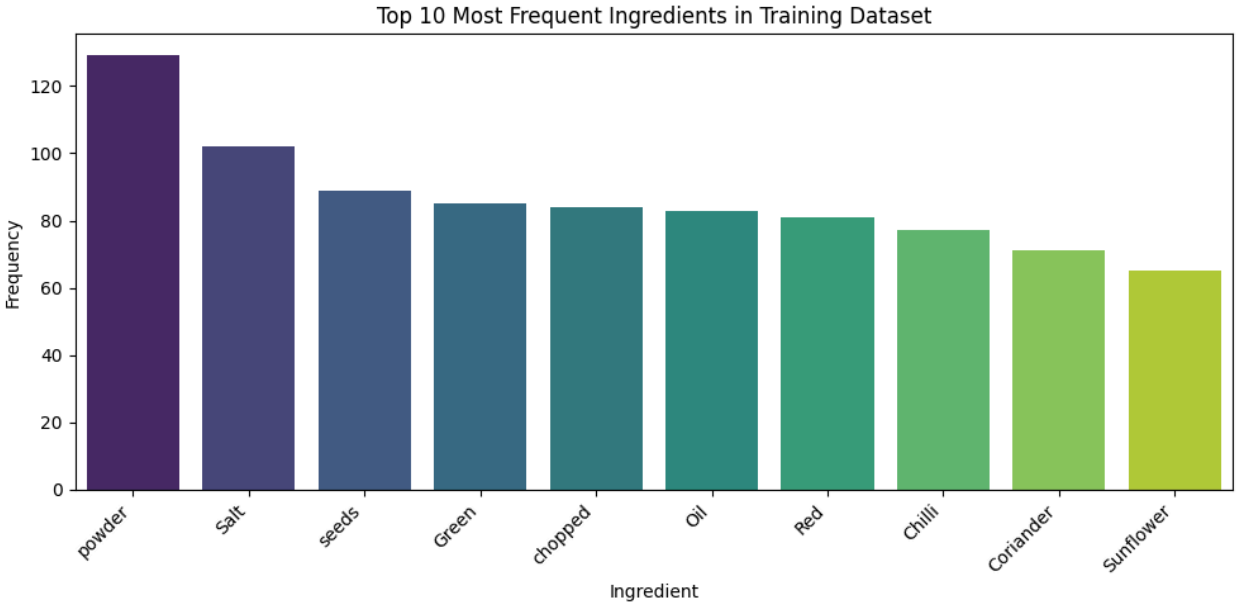
```
    plt.title(f"Top 10 Most Frequent {label.title()}s in {dataset_name} Datase
    plt.xlabel(label.title())
    plt.ylabel("Frequency")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

4.6 *Perform EDA analysis*

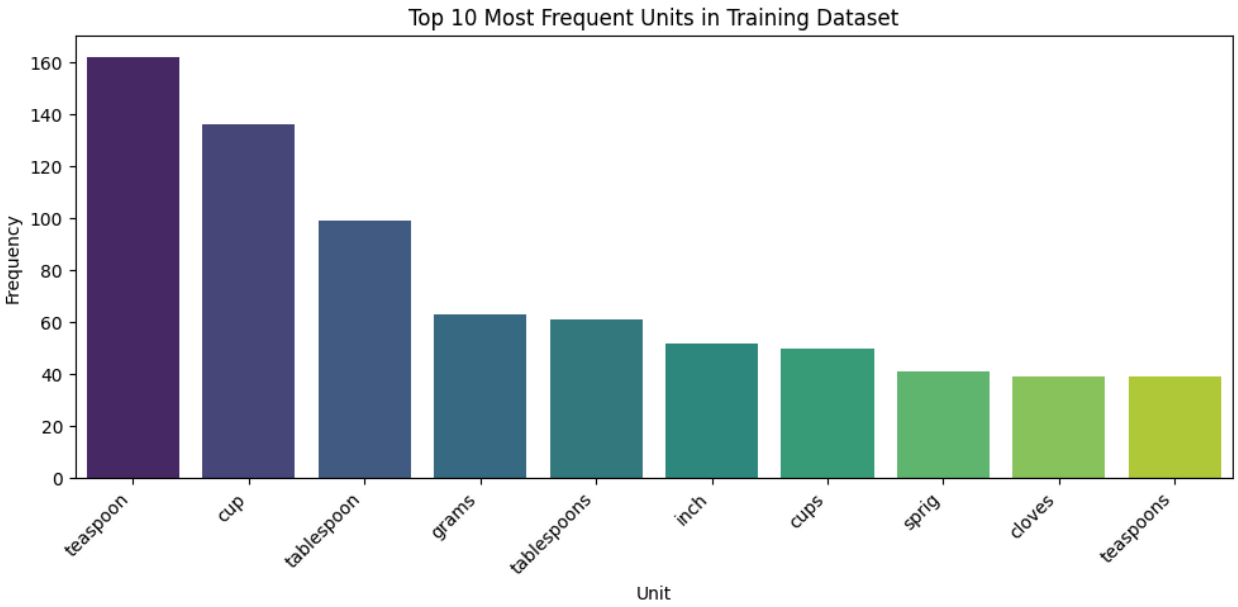
Plot the bar plots for ingredients and units and provide the insights for training dataset

---

In [36]:
```
# plot the top frequent ingredients in training data
plot_top_items(top_ingredients, 'ingredient', 'Training')
```



In [37]:
```
# plot the top frequent units in training data
plot_top_items(top_units, 'unit', 'Training')
```



5 Exploratory Recipe Data Analysis on Validation Dataset (Optional)

5.1 *Execute EDA on Validation Dataset with insights (Optional)*

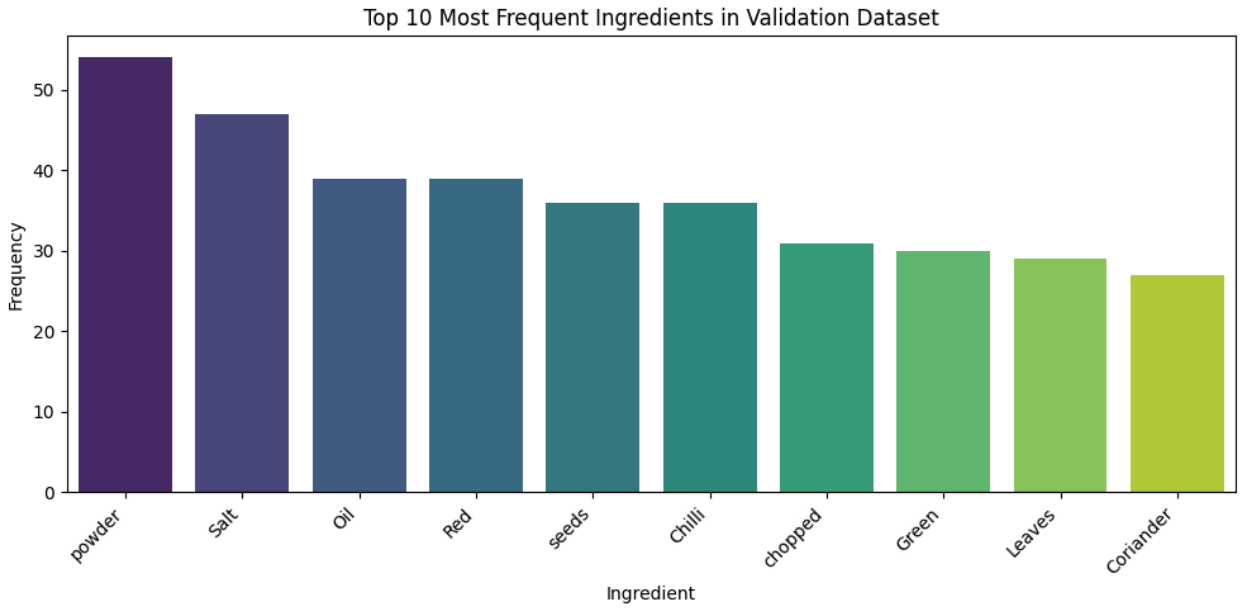
Initialise the dataset_name as ***Validation*** and call the ***plot_top_items*** for top 10 ingredients and units in the recipe data Provide the insights for the same.

In [38]:
```
# initialise the dataset_name
dataset_name = 'Validation'
```

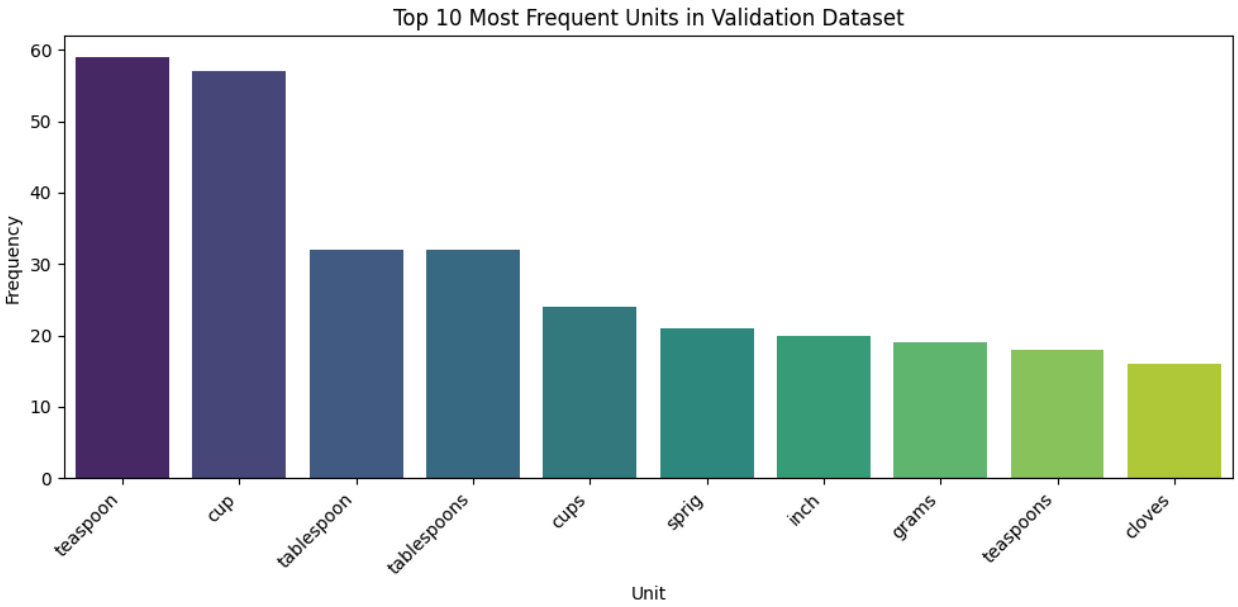
In [39]:
```
# use extract and validate tokens, categorise tokens, get top frequent items f

# Flatten and extract tokens and pos tags for validation set
flat_input_tokens_val = flatten_list(val_df['input_tokens'].tolist())
flat_pos_tokens_val = flatten_list(val_df['pos_tokens'].tolist())

# Categorise tokens for validation set
val_ingredients, val_units, val_quantities = categorize_tokens(flat_input_toke

# Get top 10 most frequent ingredients and units in validation set
top_ingredients_val = get_top_frequent_items(val_ingredients, 'ingredient', da
top_units_val = get_top_frequent_items(val_units, 'unit', dataset_name)
```

```
Top 10 most frequent ingredients in Validation dataset:
powder: 54
Salt: 47
Oil: 39
Red: 39
seeds: 36
Chilli: 36
chopped: 31
Green: 30
Leaves: 29
Coriander: 27

Top 10 most frequent units in Validation dataset:
teaspoon: 59
cup: 57
tablespoon: 32
tablespoons: 32
cups: 24
sprig: 21
inch: 20
grams: 19
teaspoons: 18
cloves: 16
```

In [40]:
```
# plot the top frequent ingredients in validation data
plot_top_items(top_ingredients_val, 'ingredient', dataset_name)
```



In [41]:
```
# plot the top frequent units in training data
plot_top_items(top_units_val, 'unit', dataset_name)
```



6 Feature Extraction For CRF Model

6.1 *Define a feature functions to take each token from recipe*

Define a function as ***word2features*** which takes a particular recipe and its index to work with all recipe input tokens and include custom key-value pairs.

Also, use feature key-value pairs to mark the beginning and end of the sequence and to also check whether the word belongs to unit, quantity etc. Use keyword sets for unit and quantity for differentiating feature functions well. Also make use of relevant regex patterns on fractions, whole numbers etc.

6.1.1 Define keywords for unit and quantity and create a quantity pattern to work on fractions, numbers and decimals

Create sets for **unit_keywords** and ***quantity_keywords*** and include all the words relevant for measuring the ingredients such as cup, tbsp, tsp etc. and in quantity keywords, include words such as half, quarter etc.

Also suggested to use regex pattern as ***quantity_pattern*** to work with quantity in any format such as fractions, numbers and decimals.

Then, load the spacy model and process the entire sentence

In [42]:
```
# define unit and quantity keywords along with quantity pattern

# Set of common unit keywords (add more as needed)
unit_keywords = {
```

```
    'cup', 'cups', 'tablespoon', 'tablespoons', 'tbsp', 'teaspoon', 'teaspoons
    'gram', 'grams', 'g', 'kg', 'kilogram', 'kilograms', 'ml', 'milliliter', '
    'l', 'liter', 'liters', 'ounce', 'ounces', 'oz', 'pound', 'pounds', 'lb',
    'pinch', 'pinches', 'dash', 'dashes', 'clove', 'cloves', 'slice', 'slices'
    'packet', 'packets', 'can', 'cans', 'stick', 'sticks', 'drop', 'drops'
}

# Set of common quantity keywords (add more as needed)
quantity_keywords = {
    'half', 'quarter', 'third', 'fourth', 'fifth', 'sixth', 'eighth', 'few', '
}

# Regex pattern to match quantities: integers, decimals, fractions (e.g., 1, 2
quantity_pattern = re.compile(r'^(\d+(\.\d+)?|\d+/\d+|\d+-\d+/\d+)$')
```

In [43]:
```
# load spaCy model
nlp = spacy.load("en_core_web_sm")
```

6.1.2 Define feature functions for CRF

Define ***word2features*** function and use the parameters such as sentence and its indexing as ***sent*** and ***i*** for extracting token level features for CRF Training. Build ***features*** dictionary, also mark the beginning and end of the sequence and use the ***unit_keywords***, ***quantity_keywords*** and ***quantity_pattern*** for knowing the presence of quantity or unit in the tokens

While building ***features*** dictionary, include

- ***Core Features*** - The core features of a token should capture its lexical and grammatical properties. Include attributes like the raw token, its lemma, part-of-speech tag, dependency relation, and shape, as well as indicators for whether it's a stop word, digit, or punctuation. The details of the features are given below:
    - `bias` - Constant feature with a fixed value of 1.0 to aid model learning.
    - `token` - The lowercase form of the current token.
    - `lemma` - The lowercase lemma (base form) of the token.
    - `pos_tag` - Part-of-speech (POS) tag of the token.
    - `tag` - Detailed POS tag of the token.
    - `dep` - Dependency relation of the token in the sentence.
    - `shape` - Shape of the token (e.g., "Xxx" for "Milk").
    - `is_stop` - Boolean indicating if the token is a stopword.
    - `is_digit` - Boolean indicating if the token consists of only digits.
    - `has_digit` - Boolean indicating if the token contains at

least one digit.
        - `has_alpha` - Boolean indicating if the token contains at least one alphabetic character.
        - `hyphenated` - Boolean indicating if the token contains a hyphen (-).
        - `slash_present` - Boolean indicating if the token contains a slash (/).
        - `is_title` - Boolean indicating if the token starts with an uppercase letter.
        - `is_upper` - Boolean indicating if the token is fully uppercase.
        - `is_punct` - Boolean indicating if the token is a punctuation mark.
- ***Improved Quantity and Unit Detection*** - Use key-value pairs to mark the presence of quantities and units in the features dictionary. Utilise the unit_keywords, quantity_keywords, and quantity_pattern to identify and flag these elements. The details of the features are given below:

    - `is_quantity` - Boolean indicating if the token matches a quantity pattern or keyword.
    - `is_unit` - Boolean indicating if the token is a known measurement unit.
    - `is_numeric` - Boolean indicating if the token matches a numeric pattern.
    - `is_fraction` - Boolean indicating if the token represents a fraction (e.g., 1/2).
    - `is_decimal` - Boolean indicating if the token represents a decimal number (e.g., 3.14).
    - `preceding_word` - The previous token in the sentence, if available.
    - `following_word` - The next token in the sentence, if available.
- ***Contextual Features*** - Incorporate contextual information by adding features for the preceding and following tokens. Include indicators like BOS and EOS to mark the beginning and end of the sequence, and utilise unit_keywords, quantity_keywords, and quantity_pattern to identify the types of neighboring tokens. The features are given below:

    - `prev_token` - The lowercase form of the previous token.
    - `prev_is_quantity` - Boolean indicating if the previous

token is a quantity.

- `prev_is_digit` - Boolean indicating if the previous token is a digit.
- `BOS` - Boolean indicating if the token is at the beginning of the sentence.
- `next_token` - The lowercase form of the next token.
- `next_is_unit` - Boolean indicating if the next token is a unit.
- `next_is_ingredient` - Boolean indicating if the next token is not a unit or quantity.
- `EOS` - Boolean indicating if the token is at the end of the sentence.

In [44]:

```python
# define word2features for processing each token in the sentence sent by using
# use your own feature functions

def word2features(sent, i):
    """
    Extracts features for the token at position i in the sentence sent.
    sent: list of tokens (strings)
    i: index of the token in the sentence
    Returns: dict of features for the token
    """
    # Process the sentence with spaCy
    doc = nlp(" ".join(sent))
    token = doc[i]
    word = token.text
    word_lower = word.lower()

    # --- Core Features ---
    features = {
        'bias': 1.0,
        'token': word_lower,
        'lemma': token.lemma_.lower(),
        'pos_tag': token.pos_,
        'tag': token.tag_,
        'dep': token.dep_,
        'shape': token.shape_,
        'is_stop': token.is_stop,
        'is_digit': word.isdigit(),
        'has_digit': any(char.isdigit() for char in word),
        'has_alpha': any(char.isalpha() for char in word),
        'hyphenated': '-' in word,
        'slash_present': '/' in word,
        'is_title': word.istitle(),
        'is_upper': word.isupper(),
        'is_punct': token.is_punct,
    }
```

```
    # --- Improved Quantity & Unit Detection ---
    features['is_quantity'] = (
        bool(quantity_pattern.match(word)) or word_lower in quantity_keywords
    )
    features['is_unit'] = word_lower in unit_keywords
    features['is_numeric'] = word.replace('.', '', 1).isdigit()
    features['is_fraction'] = bool(re.match(r'^\d+/\d+$', word))
    features['is_decimal'] = bool(re.match(r'^\d+\.\d+$', word))
    features['preceding_word'] = sent[i-1].lower() if i > 0 else ''
    features['following_word'] = sent[i+1].lower() if i < len(sent)-1 else ''

    # --- Contextual Features ---
    if i > 0:
        prev_word = sent[i-1].lower()
        features['prev_token'] = prev_word
        features['prev_is_quantity'] = (
            bool(quantity_pattern.match(prev_word)) or prev_word in quantity_k
        )
        features['prev_is_digit'] = prev_word.isdigit()
    else:
        features['BOS'] = True  # Beginning of sentence

    if i < len(sent) - 1:
        next_word = sent[i+1].lower()
        features['next_token'] = next_word
        features['next_is_unit'] = next_word in unit_keywords
        features['next_is_ingredient'] = (
            next_word not in unit_keywords and next_word not in quantity_keywo
        )
    else:
        features['EOS'] = True  # End of sentence

    return features
```

6.2 *Preparation of Recipe level features*

6.2.1 Define function to work on all the recipes and call word2features for each recipe

Define ***sent2features*** function and inputs ***sent*** as a parameter and correctly generate feature functions for each token present in the sentence

In [45]:
```
# define sent2features by working on each token in the sentence and correctly
def sent2features(sent):
    """
    Generates a list of feature dictionaries for each token in the sentence.
    Args:
        sent (list): List of tokens (strings) in the sentence.
    Returns:
        List of feature dictionaries, one per token.
    """
    return [word2features(sent, i) for i in range(len(sent))]
```

6.3 *Convert X_train, X_val, y_train and y_val into train and validation feature sets and labels*

6.3.1 Convert recipe into feature functions by using X_train and X_val

Create ***X_train_features*** and ***X_val_features*** as list to include the feature functions for each recipe present in training and validation sets

In [46]:
```
# Convert input sentences into feature sets by taking training and validation
X_train_features = [sent2features(sent) for sent in X_train]
X_val_features = [sent2features(sent) for sent in X_val]
```

6.3.2 Convert lables of y_train and y_val into list

Create ***y_train_labels*** and ***y_val_labels*** by using the list of y_train and y_val

In [47]:
```
# Convert labels into list as y_train_labels and y_val_labels
y_train_labels = [list(labels) for labels in y_train]
y_val_labels = [list(labels) for labels in y_val]
```

6.3.3 Print the length of val and train features and labels

In [48]:
```
# print the length of train features and labels
print("Length of X_train_features:", len(X_train_features))
print("Length of y_train_labels:", len(y_train_labels))
```

```
Length of X_train_features: 196
Length of y_train_labels: 196
```

In [49]:
```
# print the length of validation features and labels
print("Length of X_val_features:", len(X_val_features))
print("Length of y_val_labels:", len(y_val_labels))
```

```
Length of X_val_features: 84
Length of y_val_labels: 84
```

6.4 *Applying weights to feature sets*

6.4.1 Flatten the labels of y_train

Create ***y_train_flat*** to flatten the structure of nested y_train

In [50]:
```
# Flatten labels in y_train
y_train_flat = [label for sample in y_train for label in sample]
```

6.4.2 Count the labels present in training target dataset

Create ***label_counts*** to count the frequencies of labels present in y_train_flat and retrieve the total samples by using the values of label_counts as ***total_samples***

In [51]:
```
# Count label frequencies as label_counts and total_samples as getting the sum
label_counts = Counter(y_train_flat)
total_samples = sum(label_counts.values())
print("Label counts:", label_counts)
print("Total samples:", total_samples)
```

```
Label counts: Counter({'ingredient': 5323, 'quantity': 980, 'unit': 811})
Total samples: 7114
```

6.4.3 Compute weight_dict by using inverse frequency method for label weights

- Create ***weight_dict*** as dictionary with label and its inverse frequency count in ***label_counts***
- Penalise ingredient label in the dictionary

In [52]:
```
# Compute class weights (inverse frequency method) by considering total_sample
weight_dict = {label: total_samples / (len(label_counts) * count) for label, c
```

In [53]:
```
# penalise ingredient label
if 'ingredient' in weight_dict:
    weight_dict['ingredient'] = weight_dict['ingredient'] * 0.5

print("Class weights:", weight_dict)
```

```
Class weights: {'quantity': 2.419727891156463, 'unit': 2.923962186600904, 'ingr
edient': 0.22274406662909388}
```

6.4.4 Extract features along with class weights

Define a function ***extract_features_with_class_weights*** to work with training and validation datasets and extract features by applying class weights

In [54]:
```
# Apply weights to feature extraction in extract_features_with_class_weights b
def extract_features_with_class_weights(X, y, weight_dict):
    """
    For each sentence in X and its corresponding labels in y,
    attach the class weight to each token's feature dict based on its label.
    Returns a list of feature dicts with 'class_weight' key added.
    """
    weighted_features = []
    for sent_features, sent_labels in zip([sent2features(sent) for sent in X],
        sent_weighted = []
        for feat, label in zip(sent_features, sent_labels):
            feat_with_weight = feat.copy()
            feat_with_weight['class_weight'] = weight_dict.get(label, 1.0)
            sent_weighted.append(feat_with_weight)
        weighted_features.append(sent_weighted)
    return weighted_features
```

6.4.5 Execute extract_features_with_class_weights on training and validation datasets

Create ***X_train_weighted_features*** and ***X_val_weighted_features*** for extracting training and validation features along with their weights by calling ***extract_features_with_class_weights*** on the datasets

In [55]:
```
# Apply manually computed class weights
X_train_weighted_features = extract_features_with_class_weights(X_train, y_tra
X_val_weighted_features = extract_features_with_class_weights(X_val, y_val, we
```

7 Model Building and Training

7.1 *Initialise the CRF model and train it*

Train the CRF model with the specified hyperparameters such as

CRF Model Hyperparameters Explanation

Parameter	Description
algorithm='lbfgs'	Optimisation algorithm used for training. `lbfgs` (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) is a quasi-Newton optimisation method.
c1=0.5	L1 regularisation term to control sparsity in feature weights. Helps in feature selection.
c2=1.0	L2 regularisation term to prevent overfitting by penalising large weights.
max_iterations=100	Maximum number of iterations for model training. Higher values allow more convergence but increase computation time.
all_possible_transitions=True	Ensures that all possible state transitions are considered in training, making the model more robust.

Use weight_dict for training CRF

In [63]:
```
from sklearn_crfsuite import CRF
# initialise CRF model with the specified hyperparameters and use weight_dict
crf = CRF(
    algorithm='lbfgs',
    max_iterations=100,
    all_possible_transitions=True
    )

# Train the CRF model with the weighted training data
```

```
crf.fit(X_train_features, y_train_labels)
```

Out[63]:

CRF

```
CRF(algorithm='lbfgs', all_possible_transitions=True, max_iteration
s=100)
```

7.2 *Evaluation of Training Dataset using CRF model*

Evaluate on training dataset using CRF by using flat classification report and confusion matrix

In [64]:
```
# evaluate on the training dataset

# Predict labels for training data
y_pred_train = crf.predict(X_train_features)
```

In [65]:
```
# specify the flat classification report by using training data for evaluation

print(flat_classification_report(y_train_labels, y_pred_train))
```

```
              precision    recall  f1-score   support

  ingredient       0.97      0.99      0.98      5323
    quantity       0.95      0.90      0.92       980
        unit       0.96      0.88      0.92       811

    accuracy                           0.96      7114
   macro avg       0.96      0.92      0.94      7114
weighted avg       0.96      0.96      0.96      7114
```

In [66]:
```
from sklearn.metrics import confusion_matrix
# create a confusion matrix on training datset

y_train_true_flat = [label for sent in y_train_labels for label in sent]
y_train_pred_flat = [label for sent in y_pred_train for label in sent]

labels = sorted(list(set(y_train_true_flat)))
cm = confusion_matrix(y_train_true_flat, y_train_pred_flat, labels=labels)
print("Confusion Matrix (Training Data):")
print(pd.DataFrame(cm, index=labels, columns=labels))
```

```
Confusion Matrix (Training Data):
            ingredient  quantity  unit
ingredient        5261        36    26
quantity            94       880     6
unit                87         7   717
```

7.3 *Save the CRF model*

Save the CRF model

In [67]:
```
# dump the model using joblib as crf_model.pkl
joblib.dump(crf, 'crf_model.pkl')
```

Out[67]:
```
['crf_model.pkl']
```

8 Prediction and Model Evaluation

8.1 *Predict and Evaluate the CRF model on validation set*

Evaluate the metrics for CRF model by using flat classification report and confusion matrix

In [68]:
```
# predict the crf model on validation dataset
y_pred_val = crf.predict(X_val_features)
```

In [69]:
```
# specify flat classification report
print(flat_classification_report(y_val_labels, y_pred_val))
```

```
              precision    recall  f1-score   support

  ingredient       0.94      0.97      0.96      2107
    quantity       0.92      0.87      0.89       411
        unit       0.90      0.79      0.84       358

    accuracy                           0.93      2876
   macro avg       0.92      0.88      0.90      2876
weighted avg       0.93      0.93      0.93      2876
```

In [70]:
```
# create a confusion matrix on validation dataset
from sklearn.metrics import confusion_matrix

y_val_true_flat = [label for sent in y_val_labels for label in sent]
y_val_pred_flat = [label for sent in y_pred_val for label in sent]

labels = sorted(list(set(y_val_true_flat)))
cm_val = confusion_matrix(y_val_true_flat, y_val_pred_flat, labels=labels)
print("Confusion Matrix (Validation Data):")
print(pd.DataFrame(cm_val, index=labels, columns=labels))
```

```
Confusion Matrix (Validation Data):
            ingredient  quantity  unit
ingredient        2048        28    31
quantity            53       356     2
unit                73         3   282
```

9 Error Analysis on Validation Data

Investigate misclassified samples in validation dataset and provide the insights

9.1 *Investigate misclassified samples in validation dataset*

9.1.1 Flatten the labels of validation data and initialise error data

Flatten the true and predicted labels and initialise the error data as ***error_data***

In [71]:
```
# flatten Labels and Initialise Error Data

# Flatten the true and predicted labels for validation set
y_val_true_flat = [label for sent in y_val_labels for label in sent]
y_val_pred_flat = [label for sent in y_pred_val for label in sent]

# Initialise error data list to store misclassified samples
error_data = []
```

9.1.2 Iterate the validation data and collect Error Information

Iterate through validation data (X_val, y_val_labels, y_pred_val) and compare true vs. predicted labels. Collect error details, including surrounding context, previous/ next tokens, and class weights, then store them in error_data

In [72]:
```
# iterate and collect Error Information
for sent_idx, (tokens, true_labels, pred_labels) in enumerate(zip(X_val, y_val
    for i, (token, true_label, pred_label) in enumerate(zip(tokens, true_label
        if true_label != pred_label:
            prev_token = tokens[i-1] if i > 0 else ""
            next_token = tokens[i+1] if i < len(tokens)-1 else ""
            # get previous and next tokens with handling for boundary cases
            context = " ".join(tokens[max(0, i-2):min(len(tokens), i+3)])
            error_data.append({
                "sentence_index": sent_idx,
                "token": token,
                "prev_token": prev_token,
                "next_token": next_token,
                "true_label": true_label,
                "pred_label": pred_label,
                "context": context
            })
```

9.1.3 Create dataframe from error_data and print overall accuracy

Change error_data into dataframe and then use it to illustrate the overall accuracy of validation data

In [73]:
```
# Create DataFrame and Print Overall Accuracy

# Convert error_data to DataFrame
error_df = pd.DataFrame(error_data)

# Calculate overall accuracy on validation data
total_tokens = len(y_val_true_flat)
num_correct = sum([t == p for t, p in zip(y_val_true_flat, y_val_pred_flat)])
accuracy = num_correct / total_tokens if total_tokens > 0 else 0

print(f"Validation Accuracy: {accuracy:.4f}")
print(f"Total tokens: {total_tokens}")
print(f"Misclassified tokens: {len(error_df)}")

# Display first few rows of error_df for inspection
display(error_df.head())
```

```
Validation Accuracy: 0.9339
Total tokens: 2876
Misclassified tokens: 190
```

	sentence_index	token	prev_token	next_token	true_label	pred_label	cont
0	2	teaspoon	Til	Red	unit	ingredient	seed teasp pow
1	2	powder	Red	Cumin	ingredient	unit	teasp pow Cu Je
2	2	1/2	Dhania	Garam	quantity	ingredient	Pow Dha Ga ma
3	2	2	masala	Sweet	quantity	ingredient	Ga masa Sw Chut
4	2	Chutney	Sweet	Date	ingredient	quantity	2 Sw Chut D Tama

9.1.4 Analyse errors by label type

Analyse errors found in the validation data by each label and display their class weights along with accuracy and also display the error dataframe with token, previous token, next token, true label, predicted label and context

In [74]:
```python
# Analyse errors found in the validation data by each label
# and display their class weights along with accuracy
# and display the error dataframe with token, previous token, next token, true

import numpy as np

# Count errors by true label
error_counts = error_df['true_label'].value_counts()
print("Misclassification count by true label:")
print(error_counts)

# Calculate per-label accuracy
labels = sorted(set(y_val_true_flat))
label_accuracy = {}
for label in labels:
    total = np.sum(np.array(y_val_true_flat) == label)
    correct = np.sum((np.array(y_val_true_flat) == label) & (np.array(y_val_pr
    acc = correct / total if total > 0 else 0
    label_accuracy[label] = acc

print("\nPer-label accuracy:")
for label, acc in label_accuracy.items():
    print(f"{label}: {acc:.4f} (class weight: {weight_dict.get(label, 1.0):.3f

# Display error dataframe with relevant columns
display(error_df[['token', 'prev_token', 'next_token', 'true_label', 'pred_lab
```

```
Misclassification count by true label:
true_label
unit          76
ingredient    59
quantity      55
Name: count, dtype: int64

Per-label accuracy:
ingredient: 0.9720 (class weight: 0.223)
quantity: 0.8662 (class weight: 2.420)
unit: 0.7877 (class weight: 2.924)
```

	token	prev_token	next_token	true_label	pred_label	context
0	teaspoon	Til	Red	unit	ingredient	seeds Til teaspoon Red powder
1	powder	Red	Cumin	ingredient	unit	teaspoon Red powder Cumin Jeera
2	1/2	Dhania	Garam	quantity	ingredient	Powder Dhania 1/2 Garam masala
3	2	masala	Sweet	quantity	ingredient	Garam masala 2 Sweet Chutney
4	Chutney	Sweet	Date	ingredient	quantity	2 Sweet Chutney Date Tamarind
5	chillies	2	turmeric	ingredient	unit	ginger 2 chillies turmeric powder
6	teaspoon	cumin	salt	unit	ingredient	powder cumin teaspoon salt oil
7	tablespoon	Muttaikose)	Roasted	unit	ingredient	Gobi/ Muttaikose) tablespoon Roasted tomato
8	bought	store	Red	ingredient	unit	or store bought Red teaspoon
9	teaspoon	Red	Soy	unit	ingredient	bought Red teaspoon Soy Ginger

10 Conclusion (Optional)