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:

```json [ { "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 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. |
| `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.

```
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-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.3 kB)
Requirement already satisfied: scikit-learn>=0.24.0 in
/usr/local/lib/python3.12/dist-packages (from sklearn_crfsuite==0.5.0)
(1.6.1)
Requirement already satisfied: tabulate>=0.4.2 in
/usr/local/lib/python3.12/dist-packages (from sklearn_crfsuite==0.5.0)
(0.9.0)
Requirement already satisfied: tqdm>=2.0 in
/usr/local/lib/python3.12/dist-packages (from sklearn_crfsuite==0.5.0)
(4.67.1)
Requirement already satisfied: numpy>=1.19.5 in
/usr/local/lib/python3.12/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/python3.12/dist-packages (from scikit-learn>=0.24.0-
>sklearn_crfsuite==0.5.0) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.24.0-
>sklearn_crfsuite==0.5.0) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/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-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 29.3 MB/s eta
0:00:00
```

1.2 Import necessary libraries

```python
# Import warnings
import warnings
warnings.filterwarnings('ignore')

# 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 sequence modeling
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 output

from fractions import Fraction  # For handling fractional values in numerical data
# Importing tools for feature engineering and model training
from collections import Counter  # For counting occurrences of elements in a list
from sklearn.model_selection import train_test_split  # For splitting dataset into train and test sets
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

# Ensure pandas displays full content
pd.set_option('display.max_colwidth', None)
pd.set_option('display.expand_frame_repr', False)



from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

2 Data Ingestion and Preparation [25 marks]

2.1 *Read Recipe Data from Dataframe and prepare the data for analysis* [12 marks]

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

2.1.1 Define a *load_json_dataframe* function [7 marks]

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

```
def load_json_dataframe(file_path):
    with open(file_path, 'r') as f:
        data = json.load(f)
    df = pd.DataFrame(data)
    return df
```

2.1.2 Execute the *load_json_dataframe* function [2 marks]

```
# read the json file by giving the file path and create a dataframe
file_path = '/content/drive/MyDrive/Colab Notebooks/ingredient_and_quantity.json' # Please update this path to
```

```
your actual JSON file in Google Drive
df = load_json_dataframe(file_path)
```

2.1.3 Describe the dataframe [3 marks]

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

```
# display first five rows of the dataframe - df
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 285,\n  \"fields\": [\n    {\n      \"column\": \"input\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 285,\n        \"samples\": [\n          \"1 cup cabbage leaves 3/4 tomatoes 18 grams tamarind 2 tablespoons white urad dal 4 red chillies 3 cloves garlic big Spoon oil teaspoon Rye 1/2 Cumin seeds sprig Curry\",\n          \"12 Baby Potatoes 1 Dry Red Chilli teaspoon Cumin seeds sprig Curry leaves Coriander Powder 1/2 Turmeric powder Garam masala Amchur Mango Lemon juice 3 tablespoons Leaves chopped\",\n          \"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\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"pos\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 284,\n        \"samples\": [\n          \"quantity unit ingredient ingredient quantity ingredient quantity unit ingredient quantity unit ingredient ingredient ingredient quantity ingredient ingredient quantity ingredient ingredient ingredient unit ingredient unit ingredient quantity ingredient ingredient unit ingredient\",\n          \"quantity unit ingredient ingredient quantity ingredient unit ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient unit quantity ingredient quantity ingredient ingredient ingredient ingredient ingredient unit\",\n          \"quantity unit ingredient ingredient ingredient unit ingredient ingredient quantity ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
# print the dimensions of dataframe - df
df.shape
```

```
(285, 2)

# 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* [13 marks]

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 [3 marks]

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

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

# display first five rows of the dataframe - df
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 285,\n  \"fields\": [\
n    {\n      \"column\": \"input\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 285,\n
\"samples\": [\n          \"1 cup cabbage leaves 3/4 tomatoes 18 grams
tamarind 2 tablespoons white urad dal 4 red chillies 3 cloves garlic
big Spoon oil teaspoon Rye 1/2 Cumin seeds sprig Curry\",\n
\"12 Baby Potatoes 1 Dry Red Chilli teaspoon Cumin seeds sprig Curry
leaves Coriander Powder 1/2 Turmeric powder Garam masala Amchur Mango
Lemon juice 3 tablespoons Leaves chopped\",\n          \"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\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pos\",\n      \"properties\": {\n
```

```
\"dtype\": \"string\",\n        \"num_unique_values\": 284,\n
\"samples\": [\n          \"quantity unit ingredient ingredient
quantity ingredient quantity unit ingredient quantity unit ingredient
ingredient ingredient quantity ingredient ingredient quantity
ingredient ingredient ingredient unit ingredient unit ingredient
quantity ingredient ingredient unit ingredient\",\n
\"quantity unit ingredient ingredient quantity ingredient unit
ingredient ingredient ingredient ingredient unit ingredient ingredient
ingredient quantity ingredient ingredient ingredient ingredient unit
quantity ingredient quantity ingredient ingredient ingredient
ingredient ingredient unit\",\n          \"quantity unit ingredient
ingredient ingredient unit ingredient ingredient quantity ingredient
ingredient quantity unit ingredient ingredient ingredient ingredient
ingredient ingredient ingredient ingredient ingredient ingredient
ingredient ingredient ingredient unit ingredient ingredient ingredient
ingredient ingredient ingredient ingredient ingredient ingredient unit
ingredient ingredient ingredient ingredient ingredient ingredient
ingredient ingredient ingredient ingredient ingredient ingredient
ingredient ingredient\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"input_tokens\",\n      \"properties\": {\n
\"dtype\": \"object\",\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"pos_tokens\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


2.2.2 Provide the length for input_tokens and pos_tokens and validate their length [2 marks]

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

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

# check for the equality of input_length and pos_length in the
dataframe
unequal_lengths = df[df['input_length'] != df['pos_length']]
if not unequal_lengths.empty:
    print("Rows where input_length is not equal to pos_length:")
    display(unequal_lengths[['input_tokens', 'pos_tokens',
'input_length', 'pos_length']])
else:
    print("All input_length and pos_length are equal.")
```

```
Rows where input_length is not equal to pos_length:
```

```
{"summary":"{\n  \"name\": \"    print(\\\"All input_length and
pos_length are equal\",\n  \"rows\": 5,\n  \"fields\": [\n    {\n
\"column\": \"input_tokens\",\n      \"properties\": {\n
\"dtype\": \"object\",\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"pos_tokens\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"input_length\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16,\n        \"min\": 15,\n
\"max\": 54,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          37,\n          18,\n          38\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pos_length\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16,\n        \"min\": 14,\n        \"max\": 53,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          36,\n
17,\n          37\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


2.2.3 Define a unique_labels function and validate the labels in pos_tokens [2 marks]

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

```
# Define a unique_labels function to checks for all the unique pos
labels in the recipe & print it
def unique_labels(pos_tokens_list):
    all_labels = []
    for tokens in pos_tokens_list:
        all_labels.extend(tokens)
    return sorted(list(set(all_labels)))

unique_pos_labels = unique_labels(df['pos_tokens'])
print(f"Unique POS labels: {unique_pos_labels}")

Unique POS labels: ['ingredient', 'quantity', 'unit']
```

2.2.3 Provide the insights seen in the recipe data after validation [1 marks]

Provide the indexes that requires cleaning and formatting in the dataframe

[write your answer]

1. There are few lines where the length of the string tokens do not match the POS tokens length. But these are only 5 datapoints so can be ignored.
2. All the recepies have following unique POS token types:

- ingredient
- quantity
- unit

2.2.4 Drop the rows that have invalid data provided in previous cell [2 marks]

```
# drop the irrelevant recipe data
df = df[df['input_length'] == df['pos_length']].copy()
```

2.2.5 Update the input_length & pos_length in dataframe [2 marks]

```
# 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)

df.head()
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 280,\n \"fields\": [\n   {\n   \"column\": \"input\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 280,\n   \"samples\": [\n   \"1 cup Ada 2 liter Milk 3/4 Sugar tablespoon Ghee 1/2 teaspoon Cardamom Powder Elaichi\",\n   \"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\",\n   \"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\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pos\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 279,\n   \"samples\": [\n   \"quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient\",\n   \"quantity ingredient ingredient ingredient quantity unit ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient\",\n   \"quantity unit ingredient ingredient ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient unit ingredient quantity ingredient ingredient unit ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"input_tokens\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pos_tokens\",\n

\"properties\": {\n   \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"input_length\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
13,\n   \"min\": 7,\n   \"max\": 93,\n
\"num_unique_values\": 62,\n   \"samples\": [\n   50,\n
71,\n   31\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"pos_length\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 13,\n   \"min\": 7,\n
\"max\": 93,\n   \"num_unique_values\": 62,\n   \"samples\":
[\n   50,\n   71,\n   31\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"df"}

2.2.6 Validate the input_length and pos_length by checking unequal rows [1 marks]

```
# validate the input length and pos length as input_length and
pos_length
unequal_lengths = df[df['input_length'] != df['pos_length']]
if not unequal_lengths.empty:
    print("Rows where input_length is not equal to pos_length:")
    display(unequal_lengths[['input', 'pos', 'input_length',
'pos_length']])
else:
    print("All input_length and pos_length are equal after cleaning.")
```

```
All input_length and pos_length are equal after cleaning.
```

3 Train Validation Split (70 train - 30 val) [6 marks]

3.1 *Perform train and validation split ratio* [6 marks]

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 [1 marks]

```
# split the dataset into training and validation sets
train_df, val_df = train_test_split(df, test_size=0.3,
random_state=42)
```

3.1.2 Print the first five rows of train_df and val_df [1 marks]

```
# print the first five rows of train_df
train_df.head()
```

{"summary":"{\n \"name\": \"train_df\",\n \"rows\": 196,\n
\"fields\": [\n   {\n   \"column\": \"input\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 196,\n   \"samples\": [\n   \"300

grams Small Brinjal Baingan Eggplant 200 Mustard greens 1 Onion sliced 4 cloves Garlic finely chopped inch Ginger 1/2 teaspoon Cumin seeds Jeera Red Chilli powder Coriander Powder Dhania Garam masala Amchur Dry Mango Salt 3 tablespoons oil for cooking\",\n   \"1 cup Bajra Flour 1/2 Whole Wheat teaspoon Turmeric powder Black pepper inch Ginger grated Green Chilli 4 sprig Coriander Leaves Sunflower Oil\",\n \"200 grams Paneer Homemade Cottage Cheese cut into 1 inch cubes 2 Green Chilli finely chopped teaspoon Ginger tablespoon Raisins 6 Cashew nuts Badam Almond Pistachios 1/4 Turmeric powder teaspoons Gram flour Cardamom Powder Sunflower Oil Bay leaf tej patta 2-3 Pods Seeds Cinnamon Stick Dalchini Cloves Laung Mace Javitri Star anise cup tomato puree 3 cloves Garlic Onion roughly Coriander Dhania 1/2 Red 10 soaked and grind to a smooth paste tablespoons Curd Dahi Yogurt Kasuri Methi Dried Fenugreek Leaves Honey Fresh cream\"\n   ],\n \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pos\",\n   \"properties\": {\n \"dtype\": \"string\",\n   \"num_unique_values\": 196,\n \"samples\": [\n   \"quantity unit ingredient ingredient ingredient ingredient quantity ingredient ingredient quantity ingredient ingredient quantity unit ingredient ingredient ingredient unit ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient\",\n \"quantity unit ingredient ingredient quantity ingredient ingredient unit ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient\",\n   \"quantity unit ingredient ingredient ingredient ingredient ingredient ingredient quantity unit ingredient quantity ingredient ingredient ingredient ingredient unit ingredient unit ingredient quantity ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient\"\n   ],\n \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"input_tokens\",\n \"properties\": {\n   \"dtype\": \"object\",\n \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pos_tokens\",\n \"properties\": {\n   \"dtype\": \"object\",\n \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"input_length\",\n

\"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 13,\n   \"min\": 8,\n   \"max\": 93,\n   \"num_unique_values\": 58,\n   \"samples\": [\n   31,\n   48,\n   60\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pos_length\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 13,\n   \"min\": 8,\n   \"max\": 93,\n   \"num_unique_values\": 58,\n   \"samples\": [\n   31,\n   48,\n   60\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"train_df"}

```
# print the first five rows of the val_df
val_df.head()
```

{"summary":"{\n \"name\": \"val_df\",\n \"rows\": 84,\n \"fields\": [\n   {\n   \"column\": \"input\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 84,\n   \"samples\": [\n   \"4 Sweet corn 3 tablespoons Butter 1 Paprika powder Del Monte Cheesy Garlic Mayo 1/2 Lemon wedged Coriander Dhania Leaves Parmesan cheese grated\",\n   \"1 cup Ada 2 liter Milk 3/4 Sugar tablespoon Ghee 1/2 teaspoon Cardamom Powder Elaichi\",\n   \"2 cups Potatoes Aloo 1 Tomatoes finely chopped 1-1/2 tablespoon Ginger grated ground Green Chillies slit tablespoons Kasuri Methi Dried Fenugreek Leaves 1/2 teaspoon Fennel seeds Saunf Cumin Jeera Seeds Coriander Powder Dhania Red Chilli powder Turmeric Haldi Garam masala teaspoons Amchur Dry Mango Ghee Salt to taste\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pos\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 84,\n   \"samples\": [\n   \"quantity ingredient ingredient quantity unit ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient quantity ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient\",\n   \"quantity unit ingredient quantity unit ingredient quantity ingredient unit ingredient quantity unit ingredient ingredient ingredient\",\n   \"quantity unit ingredient ingredient quantity ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient quantity unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient ingredient unit ingredient ingredient ingredient ingredient ingredient ingredient ingredient\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"input_tokens\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pos_tokens\",\n   \"properties\": {\n   \"dtype\": \"object\",\n

\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"input_length\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 13,\n   \"min\": 7,\n   \"max\": 89,\n   \"num_unique_values\": 39,\n   \"samples\": [\n   62,\n   41,\n   51\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pos_length\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 13,\n   \"min\": 7,\n   \"max\": 89,\n   \"num_unique_values\": 39,\n   \"samples\": [\n   62,\n   41,\n   51\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"val_df"}

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 [2 marks]

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

```
# extract the training and validation sets by taking input_tokens and pos_tokens
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()

# validate the shape of training and validation samples
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
```

3.1.4 Display the number of unique labels present in y_train [2 marks]

```
# Display the number of unique labels present in y_train
all_y_train_labels = [label for sublist in y_train for label in sublist]
unique_labels_count = len(set(all_y_train_labels))
print(f"Number of unique labels in y_train: {unique_labels_count}")
print(f"Unique labels in y_train: {sorted(list(set(all_y_train_labels)))}")
```

```
Number of unique labels in y_train: 3
Unique labels in y_train: ['ingredient', 'quantity', 'unit']
```

4 Exploratory Recipe Data Analysis on Training Dataset [16 marks]

4.1 *Flatten the lists for input_tokens & pos_tokens* [2 marks]

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'***

```
# flatten the list for nested_list (input_tokens, pos_tokens)
def flatten_list(nested_list):
    return [item for sublist in nested_list for item in sublist]

# initialise the dataset_name
dataset_name = 'Training'
```

4.2 *Extract and validate the tokens after using the flattening technique* [2 marks]

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

```
# 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 tokens
def extract_and_validate_tokens(dataframe, dataset_name):
    # Flatten the lists of tokens and POS tags
    input_tokens_flat = flatten_list(dataframe['input_tokens'])
    pos_tokens_flat = flatten_list(dataframe['pos_tokens'])

    # Validate their length
    print(f"\n--- {dataset_name} Data Validation ---")
    print(f"Length of flattened input_tokens:
{len(input_tokens_flat)}")
    print(f"Length of flattened pos_tokens: {len(pos_tokens_flat)}")

    if len(input_tokens_flat) == len(pos_tokens_flat):
        print("Lengths of flattened input_tokens and pos_tokens
match.")
    else:
        print("WARNING: Lengths of flattened input_tokens and
pos_tokens do NOT match.")

    # Display first 10 records
    print("\nFirst 10 input_tokens:", input_tokens_flat[:10])
```

```
    print("First 10 pos_tokens:", pos_tokens_flat[:10])

    return input_tokens_flat, pos_tokens_flat

# extract the tokens and its pos tags
input_tokens_flat_train, pos_tokens_flat_train =
extract_and_validate_tokens(train_df, dataset_name)
```

```
--- Training Data Validation ---
Length of flattened input_tokens: 7114
Length of flattened pos_tokens: 7114
Lengths of flattened input_tokens and pos_tokens match.

First 10 input_tokens: ['250', 'grams', 'Okra', 'Oil', '1', 'Onion',
'finely', 'chopped', 'Tomato', 'Grated']
First 10 pos_tokens: ['quantity', 'unit', 'ingredient', 'ingredient',
'quantity', 'ingredient', 'ingredient', 'ingredient', 'ingredient',
'ingredient']
```

4.3 *Categorise tokens into labels (unit, ingredient, quantity)* [2 marks]

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.

```
# define a categorize_tokens function and provide the tokens and
pos_tags as parameters and create ingredient, unit and quantity list
and return it
# validate the list that it comprised of these labels, if not return
empty arrays
def categorize_tokens(tokens, pos_tags):
    ingredients = []
    units = []
    quantities = []

    for token, tag in zip(tokens, pos_tags):
        if tag == 'ingredient':
            ingredients.append(token)
        elif tag == 'unit':
            units.append(token)
        elif tag == 'quantity':
            quantities.append(token)

    return ingredients, units, quantities

#  call the function to categorise the labels into respective list
ingredients_train, units_train, quantities_train =
categorize_tokens(input_tokens_flat_train, pos_tokens_flat_train)
```

4.4 *Top 10 Most Frequent Items* [3 marks]

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

```
# define a function get_top_frequent_items to get the top frequent
items by using item_list, pos label and
dataset_name(Training/Validation) and return top items
def get_top_frequent_items(item_list, pos_label, dataset_name,
top_n=10):
    item_counts = Counter(item_list)
    top_items = item_counts.most_common(top_n)
    print(f"\nTop {top_n} Most Frequent {pos_label.capitalize()}s
({dataset_name}):")
    for item, count in top_items:
        print(f"- {item}: {count}")
    return top_items

# get the top ingredients which are frequently seen in the recipe
top_ingredients_train = get_top_frequent_items(ingredients_train,
'ingredient', dataset_name)
```

```
Top 10 Most Frequent Ingredients (Training):
- powder: 129
- Salt: 102
- seeds: 89
- Green: 85
- chopped: 84
- Oil: 83
- Red: 81
- Chilli: 77
- Coriander: 71
- Sunflower: 65
```

```
# get the top units which are frequently seen in the recipe
top_units_train = get_top_frequent_items(units_train, 'unit',
dataset_name)
```

```
Top 10 Most Frequent Units (Training):
- 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* [2 marks]

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

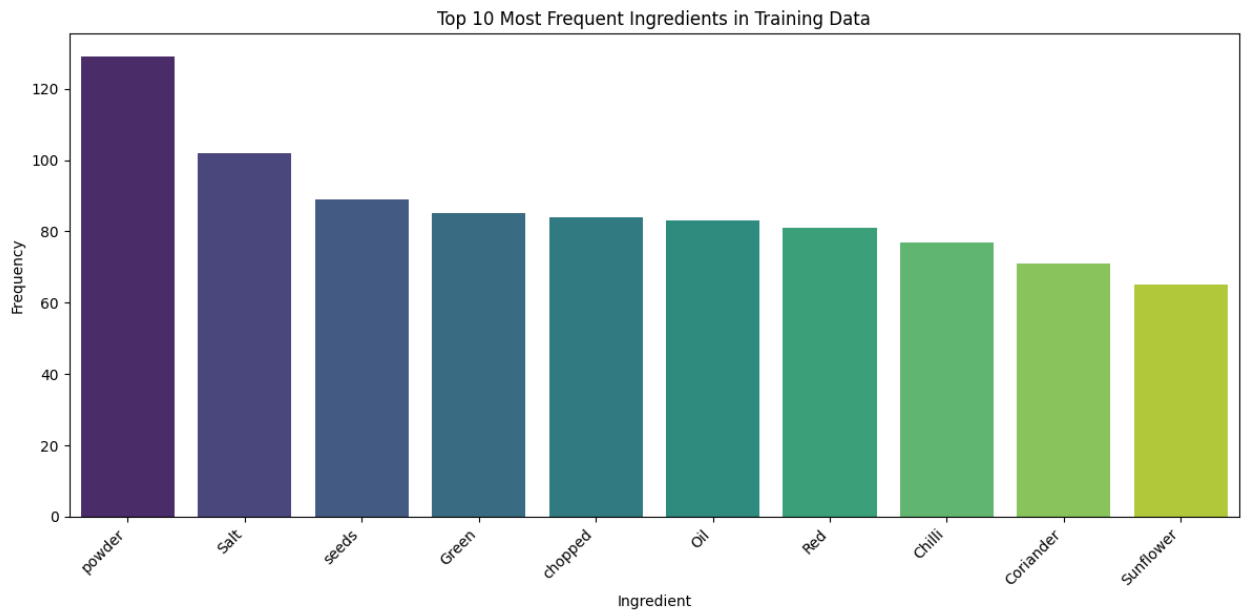
```python
# define plot top items with parameters - top_item list, label to
suggest whether its ingredient or unit, dataset_name
def plot_top_items(top_items, label, dataset_name):
    if not top_items:
        print(f"No {label}s to plot for {dataset_name}.")
        return

    items = [item[0] for item in top_items]
    counts = [item[1] for item in top_items]

    plt.figure(figsize=(12, 6))
    sns.barplot(x=items, y=counts, palette='viridis')
    plt.title(f'Top 10 Most Frequent {label.capitalize()}s in
{dataset_name} Data')
    plt.xlabel(label.capitalize())
    plt.ylabel('Frequency')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

4.6 *Perform EDA analysis* [5 marks]

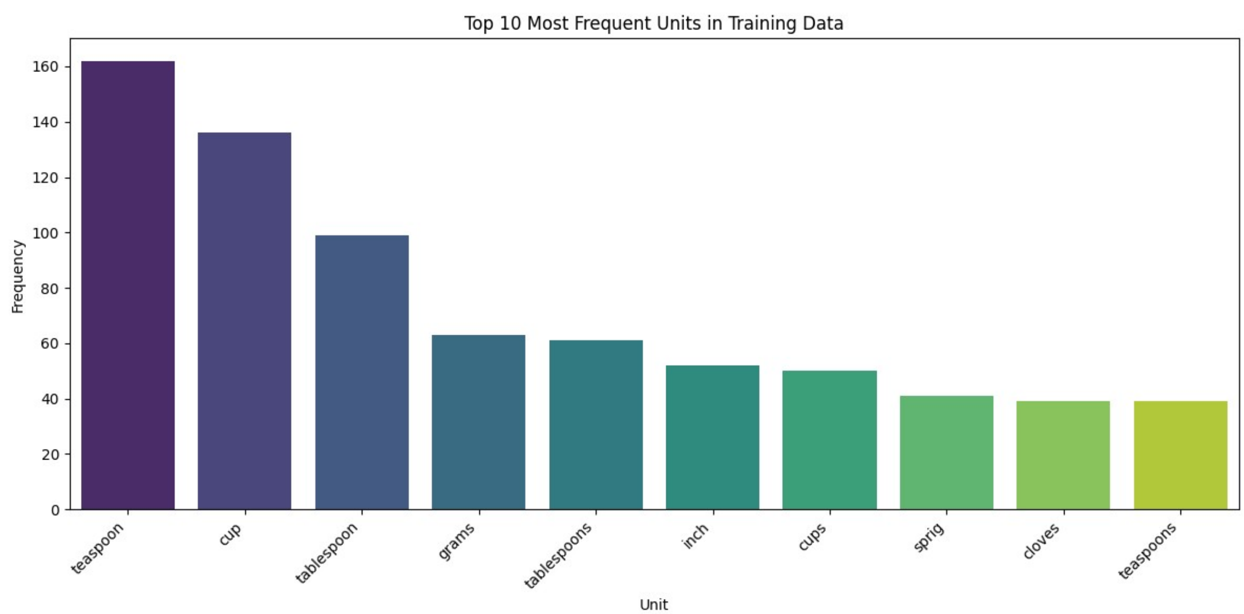
Plot the bar plots for ingredients and units and provide the insights for training dataset

---

```python
# plot the top frequent ingredients in training data
plot_top_items(top_ingredients_train, 'ingredient', dataset_name)
```




```
# plot the top frequent units in training data
plot_top_items(top_units_train, 'unit', dataset_name)
```



5 Exploratory Recipe Data Analysis on Validation Dataset (Optional) [0 marks]

5.1 *Execute EDA on Validation Dataset with insights (Optional)* [0 marks]

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.

```
# initialise the dataset_name
dataset_name = 'Validation'

# use extract and validate tokens, categorise tokens, get top frequent
items for ingredient list and unit list on validation dataframe
input_tokens_flat_test, pos_tokens_flat_test =
extract_and_validate_tokens(val_df, dataset_name)
```

```
--- Validation Data Validation ---
Length of flattened input_tokens: 2876
Length of flattened pos_tokens: 2876
Lengths of flattened input_tokens and pos_tokens match.

First 10 input_tokens: ['1', 'cup', 'Ada', '2', 'liter', 'Milk',
'3/4', 'Sugar', 'tablespoon', 'Ghee']
First 10 pos_tokens: ['quantity', 'unit', 'ingredient', 'quantity',
'unit', 'ingredient', 'quantity', 'ingredient', 'unit', 'ingredient']
```

```
ingredients_test, units_test, quantities_test =
categorize_tokens(input_tokens_flat_test, pos_tokens_flat_test)

top_ingredients_test = get_top_frequent_items(ingredients_test,
'ingredient', dataset_name)
```

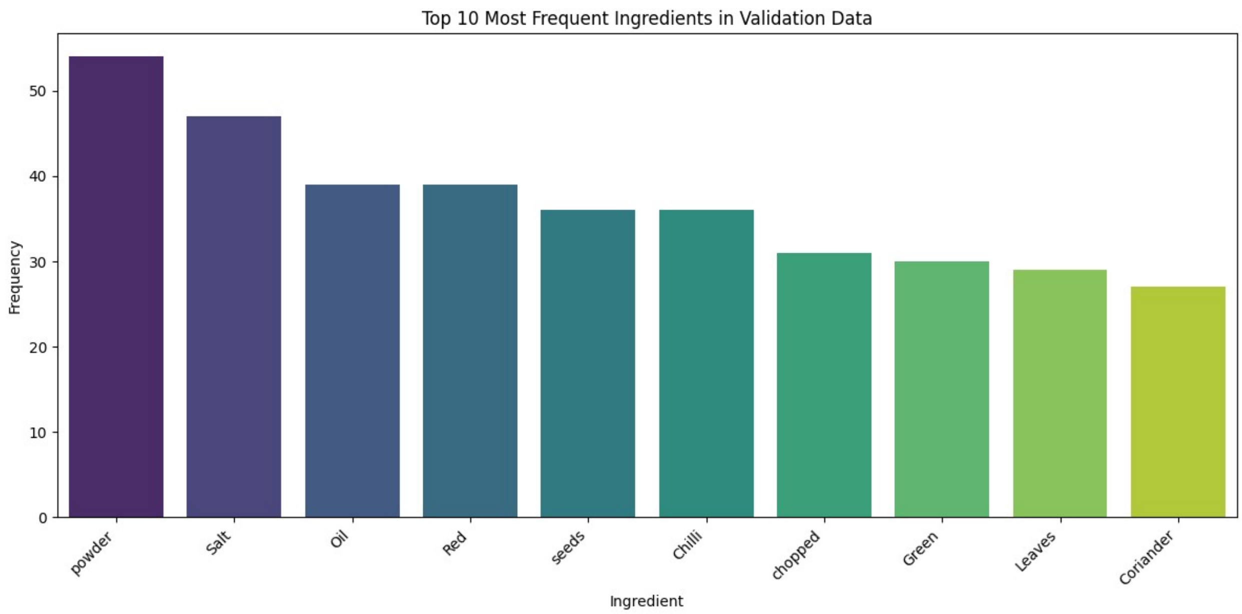
```
Top 10 Most Frequent Ingredients (Validation):
- powder: 54
- Salt: 47
- Oil: 39
- Red: 39
- seeds: 36
- Chilli: 36
- chopped: 31
- Green: 30
- Leaves: 29
- Coriander: 27
```

```
top_units_test = get_top_frequent_items(units_test, 'unit',
dataset_name)
```

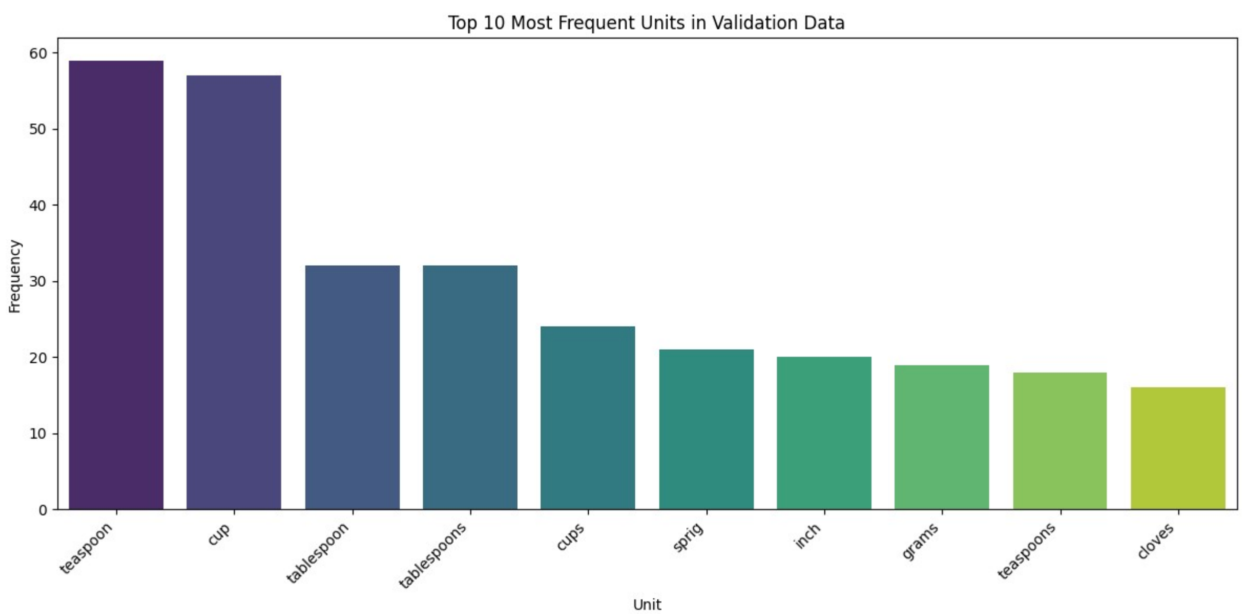
```
Top 10 Most Frequent Units (Validation):
- teaspoon: 59
- cup: 57
- tablespoon: 32
- tablespoons: 32
- cups: 24
- sprig: 21
- inch: 20
- grams: 19
```

```
- teaspoons: 18
- cloves: 16
```

```
# plot the top frequent ingredients in validation data
plot_top_items(top_ingredients_test, 'ingredient', dataset_name)
```




```
# plot the top frequent units in training data
plot_top_items(top_units_test, 'unit', dataset_name)
```



6 Feature Extraction For CRF Model [30 marks]

6.1 *Define a feature functions to take each token from recipe* [10 marks]

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 [3 marks]

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

```python
# define unit and quantity keywords along with quantity pattern
unit_set = set(units_train)
unit_keywords = sorted({item.lower() for item in unit_set})

quantity_keywords = sorted(set(['half', 'quarter', 'double', 'triple',
'a', 'an', 'as', 'few', 'finely',
                         'for', 'one', 'per', 'two', 'to', 'some',
'enough', 'several', 'plenty',
                         'little', 'taste' ]))

# Regex pattern for quantities, including fractions, ranges, and
decimals
quantity_pattern = re.compile(r'\d+[-.\/]?\d*\/?\d*')

print(unit_keywords)
```

```
['chopped', 'clove', 'cloves', 'cup', 'cups', 'cut', 'drops',
'finely', 'fistful', 'for', 'gms', 'gram', 'grams', 'handful', 'inch',
'inches', 'liter', 'long', 'ml', 'or', 'pinch', 'raw', 'scoops',
'small', 'spoon', 'sprig', 'sprigs', 'stalks', 'tablespoon',
'tablespoons', 'tbsp', 'teaspoon', 'teaspoons', 'thick', 'thin',
'tsp', 'wedges', 'whole']
```

```python
print(quantity_keywords)
```

```
['a', 'an', 'as', 'double', 'enough', 'few', 'finely', 'for', 'half',
'little', 'one', 'per', 'plenty', 'quarter', 'several', 'some',
'taste', 'to', 'triple', 'two']

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

6.1.2 Define feature functions for CRF [7 marks]

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.

```
# define word2features for processing each token in the sentence sent
by using index i.
# use your own feature functions
def word2features(sent, i):
    word = sent[i]
    # Process the entire sentence with spaCy to get linguistic
features
    doc = nlp(" ".join(sent))
    token_spacy = doc[i]

    features = {
        'bias': 1.0,
        # --- Core Features ---
        'token': word.lower(),
        'lemma': token_spacy.lemma_.lower(),
        'pos_tag': token_spacy.pos_,
        'tag': token_spacy.tag_,
        'dep': token_spacy.dep_,
        'shape': token_spacy.shape_,
        'is_stop': token_spacy.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_spacy.is_punct,

        # --- Improved Quantity & Unit Detection ---
        'is_quantity': bool(quantity_pattern.match(word)) or
word.lower() in quantity_keywords,
        'is_unit': word.lower() in unit_keywords,
        'is_numeric': word.replace('.', '', 1).isdigit() or
bool(re.match(r'^\d+\/\d+$', word)),
        'is_fraction': bool(re.match(r'^\d+\/\d+$', word)),
        'is_decimal': bool(re.match(r'^\d+\.\d+$', word)),
    }

    # --- Contextual Features ---
    if i > 0:
        prev_word = sent[i-1]
        prev_token_spacy = doc[i-1]
        features.update({
            'prev_token': prev_word.lower(),
            'prev_is_quantity':
bool(quantity_pattern.match(prev_word)) or prev_word.lower() in
quantity_keywords,
            'prev_is_digit': prev_word.isdigit(),
        })
        features['BOS'] = False
    else:
        features['BOS'] = True  # Beginning of Sentence

    if i < len(sent) - 1:
        next_word = sent[i+1]
        next_token_spacy = doc[i+1]
        features.update({
            'next_token': next_word.lower(),
            'next_is_unit': next_word.lower() in unit_keywords,
            'next_is_ingredient': next_word.lower() not in
unit_keywords and not (bool(quantity_pattern.match(next_word)) or
next_word.lower() in quantity_keywords)
        })
        features['EOS'] = False
    else:
        features['EOS'] = True  # End of Sentence

    return features
```

6.2 *Preparation of Recipe level features* [2 marks]

6.2.1 Define function to work on all the recipes and call word2features for each recipe [2 marks]

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

```python
# define sent2features by working on each token in the sentence and correctly generate dictionaries for features
def sent2features(sent):
    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 marks]

6.3.1 Convert recipe into feature functions by using X_train and X_val [2 marks]

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

```python
# Convert input sentences into feature sets by taking training and validation dataset as X_train_features and X_val_features
X_train_features = [sent2features(s) for s in X_train]
X_val_features = [sent2features(s) for s in X_val]
```

6.3.2 Convert lables of y_train and y_val into list [2 marks]

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

```python
# Convert labels into list as y_train_labels and y_val_labels
y_train_labels = y_train
y_val_labels = y_val
```

6.3.3 Print the length of val and train features and labels [2 marks]

```python
# print the length of train features and labels
print(f"Length of X_train_features: {len(X_train_features)}")
print(f"Length of y_train_labels: {len(y_train_labels)}")
```

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

```python
# print the length of validation features and labels
print(f"Length of X_val_features: {len(X_val_features)}")
print(f"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* [12 marks]

6.4.1 Flatten the labels of y_train [2 marks]

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

```
# Flatten labels in y_train
y_train_flat = flatten_list(y_train)
```

6.4.2 Count the labels present in training target dataset [2 marks]

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***

```
# Count label frequencies as label_counts and total_samples as getting
the summation of values of label_counts
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 [2 marks]

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

- Penalise ingredient label in the dictionary

```
# Compute class weights (inverse frequency method) by considering
total_samples and label_counts
weight_dict = {label: total_samples / count for label, count in
label_counts.items()}

# penalise ingredient label
weight_dict['ingredient'] *= 0.5
print("Class Weights:", weight_dict)

Class Weights: {'quantity': 7.259183673469388, 'unit':
8.771886559802713, 'ingredient': 0.6682321998872816}
```

6.4.4 Extract features along with class weights [4 marks]

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

```
# Apply weights to feature extraction in
extract_features_with_class_weights by using parameters such as X
(input tokens), y(labels) and weight_dict (Class weights)
def extract_features_with_class_weights(X, y, weight_dict):
    features_with_weights = []
    for i in range(len(X)):
        sentence_features = []
```

```
        for j in range(len(X[i])):
            word = X[i][j]
            label = y[i][j]
            feature = word2features(X[i], j)
            # Add the weight to the feature dictionary
            feature['weight'] = weight_dict.get(label, 1.0) # Default to 1.0 if label not in dict
            sentence_features.append(feature)
        features_with_weights.append(sentence_features)
    return features_with_weights
```

6.4.5 Execute extract_features_with_class_weights on training and validation datasets [2 marks]

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

```
# Apply manually computed class weights
X_train_weighted_features = extract_features_with_class_weights(X_train, y_train_labels, weight_dict)
X_val_weighted_features = extract_features_with_class_weights(X_val, y_val_labels, weight_dict)
```

7 Model Building and Training [10 marks]

7.1 *Initialise the CRF model and train it* [5 marks]

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

Parameter	Description
	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

```
# initialise CRF model with the specified hyperparameters and use
weight_dict
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.5,
    c2=1.0,
    max_iterations=100,
    all_possible_transitions=True
)

# train the CRF model with the weighted training data
crf.fit(X_train_weighted_features, y_train_labels)
```

```
CRF(algorithm='lbfgs', all_possible_transitions=True, c1=0.5, c2=1.0,
    max_iterations=100)
```

7.2 *Evaluation of Training Dataset using CRF model* [4 marks]

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

```
# evaluate on the training dataset
y_pred_train = crf.predict(X_train_weighted_features)

# specify the flat classification report by using training data for
evaluation
report_train = flat_classification_report(
    y_true=y_train_labels,
    y_pred=y_pred_train,
    labels=unique_pos_labels,
    digits=3
)
print("Classification Report (Training Data):")
print(report_train)
```

```
Classification Report (Training Data):
              precision    recall  f1-score   support
```

```
  ingredient       1.000     1.000     1.000      5323
    quantity       1.000     0.999     0.999       980
        unit       0.999     1.000     0.999       811

    accuracy                           1.000      7114
   macro avg       1.000     1.000     1.000      7114
weighted avg       1.000     1.000     1.000      7114
```

```python
# create a confusion matrix on training datset
# Flatten the lists for confusion matrix
y_train_flat_cm = [label for sublist in y_train_labels for label in
sublist]
y_pred_train_flat_cm = [label for sublist in y_pred_train for label in
sublist]

cm_train = confusion_matrix(y_train_flat_cm, y_pred_train_flat_cm,
labels=unique_pos_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(
    cm_train,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=unique_pos_labels,
    yticklabels=unique_pos_labels
)
plt.title('Confusion Matrix (Training Data)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

7.3 *Save the CRF model* [1 marks]

Save the CRF model

```
# dump the model using joblib as crf_model.pkl
joblib.dump(crf, 'crf_model.pkl')
```

```
['crf_model.pkl']
```

8 Prediction and Model Evaluation [3 marks]

8.1 *Predict and Evaluate the CRF model on validation set* [3 marks]

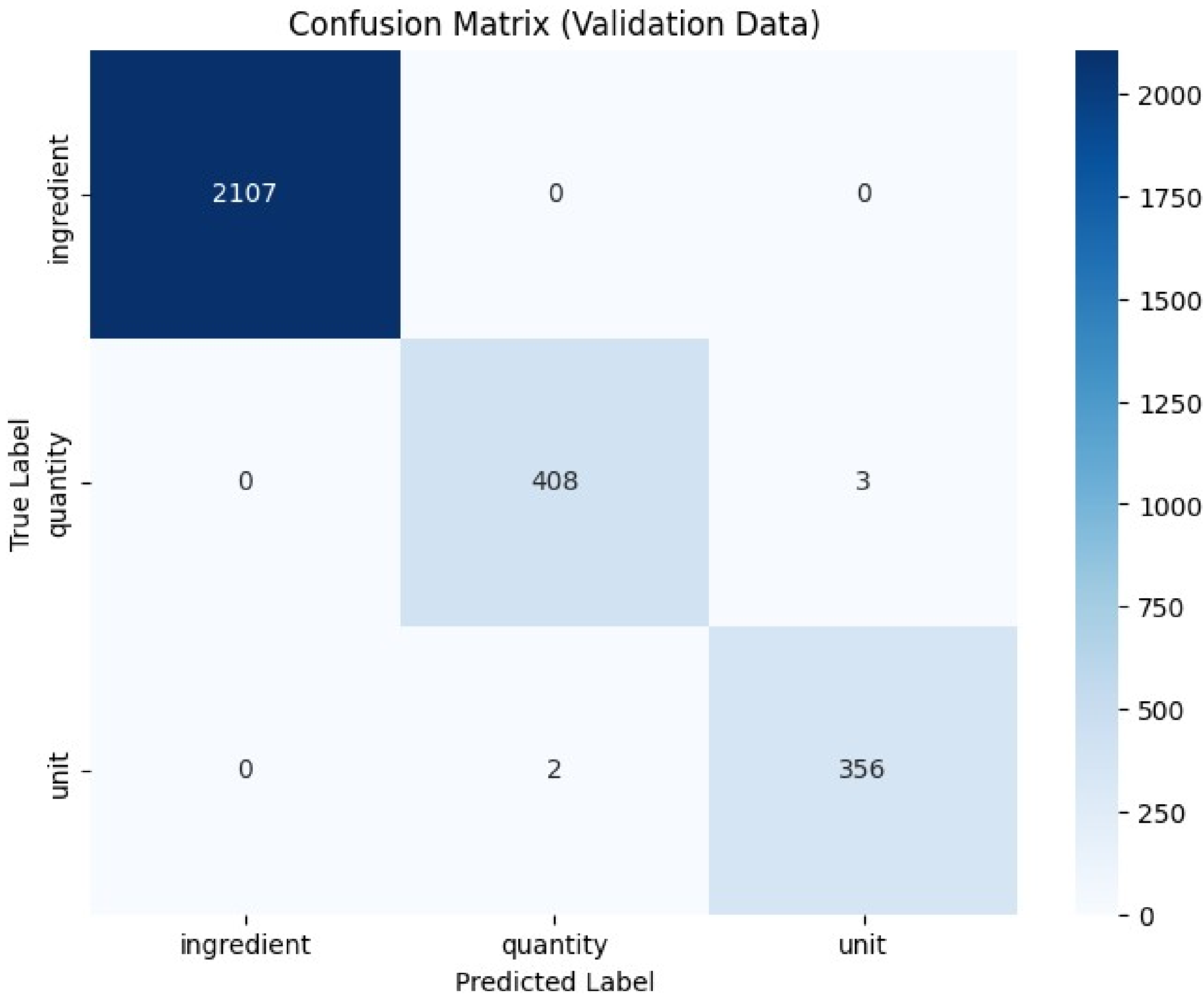
Evaluate the metrics for CRF model by using flat classification report and confusion matrix

```
# predict the crf model on validation dataset
y_pred_val = crf.predict(X_val_weighted_features)
```

```python
# specify flat classification report
report_val = flat_classification_report(
    y_true=y_val_labels,
    y_pred=y_pred_val,
    labels=unique_pos_labels,
    digits=3
)
print("Classification Report (Validation Data):")
print(report_val)
```

```
Classification Report (Validation Data):
              precision    recall  f1-score   support

  ingredient      1.000     1.000     1.000      2107
    quantity      0.995     0.993     0.994       411
        unit      0.992     0.994     0.993       358

    accuracy                          0.998      2876
   macro avg      0.996     0.996     0.996      2876
weighted avg      0.998     0.998     0.998      2876
```

```python
# create a confusion matrix on validation dataset
y_val_flat_cm = [label for sublist in y_val_labels for label in
sublist]
y_pred_val_flat_cm = [label for sublist in y_pred_val for label in
sublist]

cm_val = confusion_matrix(y_val_flat_cm, y_pred_val_flat_cm,
labels=unique_pos_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(
    cm_val,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=unique_pos_labels,
    yticklabels=unique_pos_labels
)
plt.title('Confusion Matrix (Validation Data)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



9 Error Analysis on Validation Data [10 marks]

Investigate misclassified samples in validation dataset and provide the insights

9.1 *Investigate misclassified samples in validation dataset* [8 marks]

9.1.1 Flatten the labels of validation data and initialise error data [2 marks]

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

```
# flatten Labels and Initialise Error Data
y_val_flat = [label for sublist in y_val_labels for label in sublist]
y_pred_val_flat = [label for sublist in y_pred_val for label in
sublist]

error_data = []
```

9.1.2 Iterate the validation data and collect Error Information [2 marks]

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

```
# iterate and collect Error Information
for i, (sentence, true_tags, pred_tags) in enumerate(zip(X_val,
y_val_labels, y_pred_val)):
    for j, (word, true_tag, pred_tag) in enumerate(zip(sentence,
true_tags, pred_tags)):
        if true_tag != pred_tag:
            # get previous and next tokens with handling for boundary
cases
            prev_token = sentence[j-1] if j > 0 else 'BOS'
            next_token = sentence[j+1] if j < len(sentence) - 1 else
'EOS'

            error_data.append({
                'sentence_idx': i,
                'token_idx': j,
                'token': word,
                'prev_token': prev_token,
                'next_token': next_token,
                'true_label': true_tag,
                'predicted_label': pred_tag,
                'class_weight': weight_dict.get(true_tag, 1.0) # Get
weight of true label
            })
```

9.1.3 Create dataframe from error_data and print overall accuracy [1 marks]

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

```
# Create DataFrame and Print Overall Accuracy
error_df = pd.DataFrame(error_data)

if not error_df.empty:
    print("Error Data Head:")
    display(error_df.head())
else:
    print("No errors found in the validation data.")

# Calculate and print overall accuracy
overall_accuracy = sum(1 for true, pred in zip(y_val_flat,
y_pred_val_flat) if true == pred) / len(y_val_flat)
print(f"\nOverall Accuracy on Validation Data:
{overall_accuracy:.3f}")
```

```
Error Data Head:
```

{"summary":"{\n \"name\": \"print(f\\\"\\\\nOverall Accuracy on Validation Data: {overall_accuracy:\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"sentence_idx\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 26,\n   \"min\": 13,\n   \"max\": 75,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   28,\n   75,\n   60\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"token_idx\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4,\n   \"min\": 3,\n   \"max\": 13,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   6,\n   9,\n   3\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"token\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 5,\n   \"samples\": [\n   \"to\",\n   \"cloves\",\n   \"a\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"prev_token\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 5,\n   \"samples\": [\n   \"10\",\n   \"Tomatoes\",\n   \"Haldi\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"next_token\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 5,\n   \"samples\": [\n   \"12\",\n   \"Garlic\",\n   \"pinch\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"true_label\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"unit\",\n   \"quantity\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"predicted_label\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"quantity\",\n   \"unit\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"class_weight\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.8285414936479353,\n   \"min\": 7.259183673469388,\n   \"max\": 8.771886559802713,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   8.771886559802713,\n   7.259183673469388\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
Overall Accuracy on Validation Data: 0.998
```

9.1.4 Analyse errors by label type [3 marks]

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

```
# 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 label, predicted label and context

if not error_df.empty:
    print("\nError Analysis by Label Type:")
    error_counts = error_df.groupby(['true_label',
'predicted_label']).size().reset_index(name='count')
    display(error_counts)

    print("\nClass Weights (for reference):", weight_dict)

    print("\nDetails of Misclassified Samples:")
    for true_lbl in error_df['true_label'].unique():
        print(f"\n--- Errors for True Label: {true_lbl} ---")
        label_errors = error_df[error_df['true_label'] == true_lbl]
        display(label_errors[['token', 'prev_token', 'next_token',
'true_label', 'predicted_label', 'class_weight']])
else:
    print("No errors to analyze in the validation data.")
```

```
Error Analysis by Label Type:
```

{"summary":"{\n \"name\": \"error_counts\",\n \"rows\": 2,\n
\"fields\": [\n   {\n   \"column\": \"true_label\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n
\"unit\",\n   \"quantity\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"predicted_label\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n
\"quantity\",\n   \"unit\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\":
2,\n   \"max\": 3,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   2,\n   3\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"error_counts"}

```
Class Weights (for reference): {'quantity': 7.259183673469388, 'unit':
```

```
8.771886559802713, 'ingredient': 0.6682321998872816}

Details of Misclassified Samples:

--- Errors for True Label: quantity ---
```

{"summary":"{\n \"name\": \"   print(\\\"No errors to analyze in the validation data\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"token\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"is\",\n   \"pinch\",\n   \"cloves\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"prev_token\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"Pur\",\n   \"Dal\",\n   \"Tomatoes\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"next_token\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"2\",\n   \"Asafoetida\",\n   \"Garlic\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"true_label\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"quantity\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"predicted_label\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"unit\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"class_weight\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 7.259183673469388,\n   \"max\": 7.259183673469388,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   7.259183673469388\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
--- Errors for True Label: unit ---
```

{"summary":"{\n \"name\": \"   print(\\\"No errors to analyze in the validation data\",\n \"rows\": 2,\n \"fields\": [\n   {\n   \"column\": \"token\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"a\",\n   \"to\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"prev_token\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n

```
\"Haldi\",\n          \"10\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"next_token\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"pinch\",\n          \"12\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"true_label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          \"unit\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"predicted_label\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"quantity\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"class_weight\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\":
8.771886559802713,\n        \"max\": 8.771886559802713,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
8.771886559802713\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


9.2 *Provide insights from the validation dataset* [2 marks]

[Write your answer]

The error analysis on the validation data reveals a very small number of misclassifications, which aligns with the high overall accuracy of 99.8%.

The model incorrectly predicted 'unit' when the true label was 'quantity' three times. Also, the model predicted 'quantity' when the true label was 'unit' two times.

1. The word 'is' was misclassified as a unit. This is an unusual token to be a quantity, and its context doesn't strongly suggest a quantity either. It might be a data labeling anomaly.

2. 'pinch' is a valid unit, but in this context, it was labeled as a quantity. This suggests ambiguity in how 'pinch' is used and labeled

3. Similar to 'pinch', 'cloves' can be a unit (e.g., '2 cloves of garlic') or imply quantity. Here, it was a true quantity but predicted as a unit.

It shows that the errors are due to ambiguous usage of words.

10 Conclusion (Optional) [0 marks]

Write your findings and conclusion.

The model achieved a very high overall accuracy of 99.8% on the validation data, indicating excellent performance. Despite the high accuracy, there are a few misclassifications, primarily between quantity and unit labels. This suggests that some tokens can be ambiguous in their role as a quantity or a unit, or that the features generated for these specific words are not always strong enough to differentiate them.