Identifying Key Entities in Recipe Data Report:

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.

Step 1: Import the Necessary Libraries such as NLTK, Spacy, Numpy, Pandas and others.

Step 2: Data Ingestion and Preparation:

- Load the data in Panda Dataframe
- **df = load_json_dataframe('ingredient_and_quantity.json')**
- Create Derived Metrics:
  df['input_tokens'] = df['input'].str.split()
  df['pos_tokens'] = df['pos'].str.split()
- Check Unique Labels by creating a Function for it
- Unique Labels are **: {'ingredient', 'quantity', 'unit'}**
- Find the Index for Cleaning and Formatting , and those indexes are below:
  Indexes requiring cleaning and formatting: **[17, 27, 79, 164, 207]**

Step 3: Train Validation and Split (30-70)

- Split the Data set :
  train_df, val_df = train_test_split(df, test_size=0.3, random_state=42)
- Number of unique labels in y_train: 3

Step 4: EDA on Training Set

- 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.
- 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

- Output:

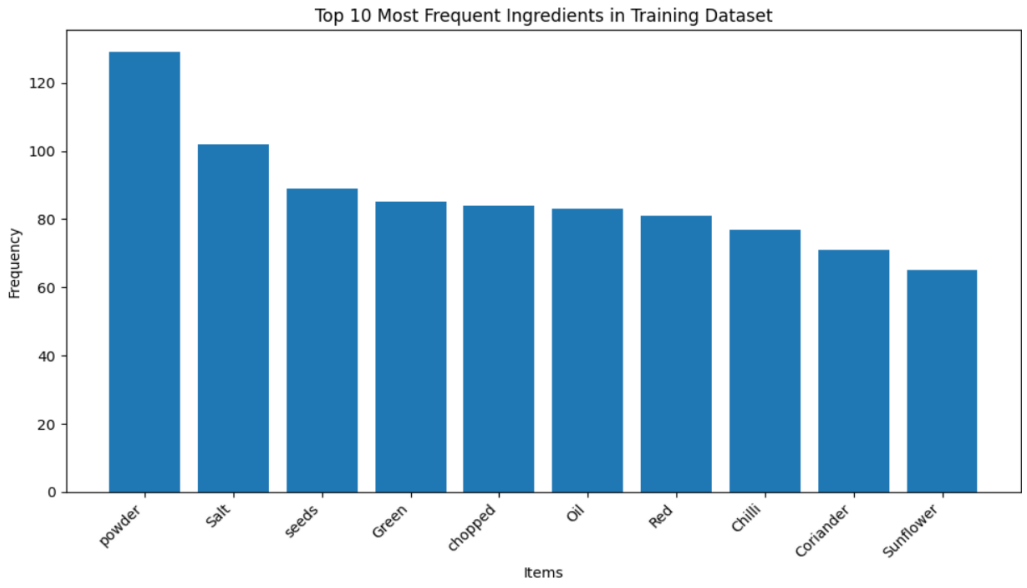
```
First 10 input tokens for Training dataset:
['250', 'grams', 'Okra', 'Oil', '1', 'Onion', 'finely', 'chopped', 'Toma
to', 'Grated']
```

```
First 10 pos tokens for Training dataset:
['quantity', 'unit', 'ingredient', 'ingredient', 'quantity', 'ingredient
', 'ingredient', 'ingredient', 'ingredient', 'ingredient']
```

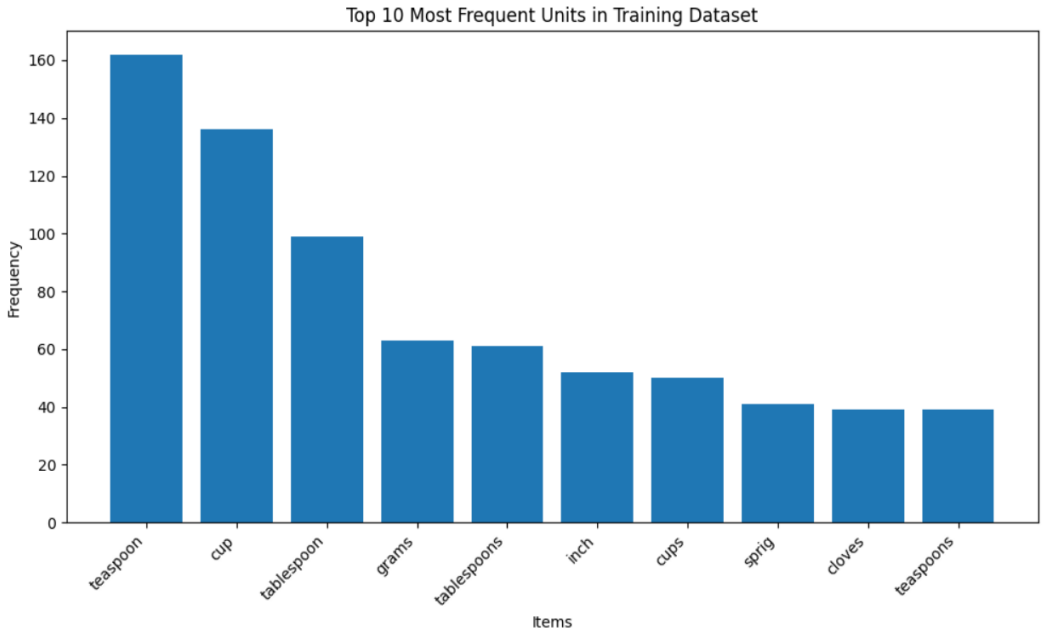
- Define a function ***categorize_tokens*** to categorise tokens into ingredients, units and quantities by using extracted tokens in the previous code and return a list of ingredients, units and quantities. Execute this function to get the list.

```
Output:
============

Ingredients: ['Okra', 'Oil', 'Onion', 'finely', 'chopped', 'Tomato', 'Gr
ated', 'Ginger', 'Garlic', 'Finely']

Units: ['grams', 'teaspoon', 'Teaspoon', 'cup', 'grams', 'tablespoon', '
teaspoon', 'grams', 'teaspoon', 'sprig']

Quantities: ['250', '1', '2', '1/2', '1/4', '200', '2', '1', '1/2', '500
']
```

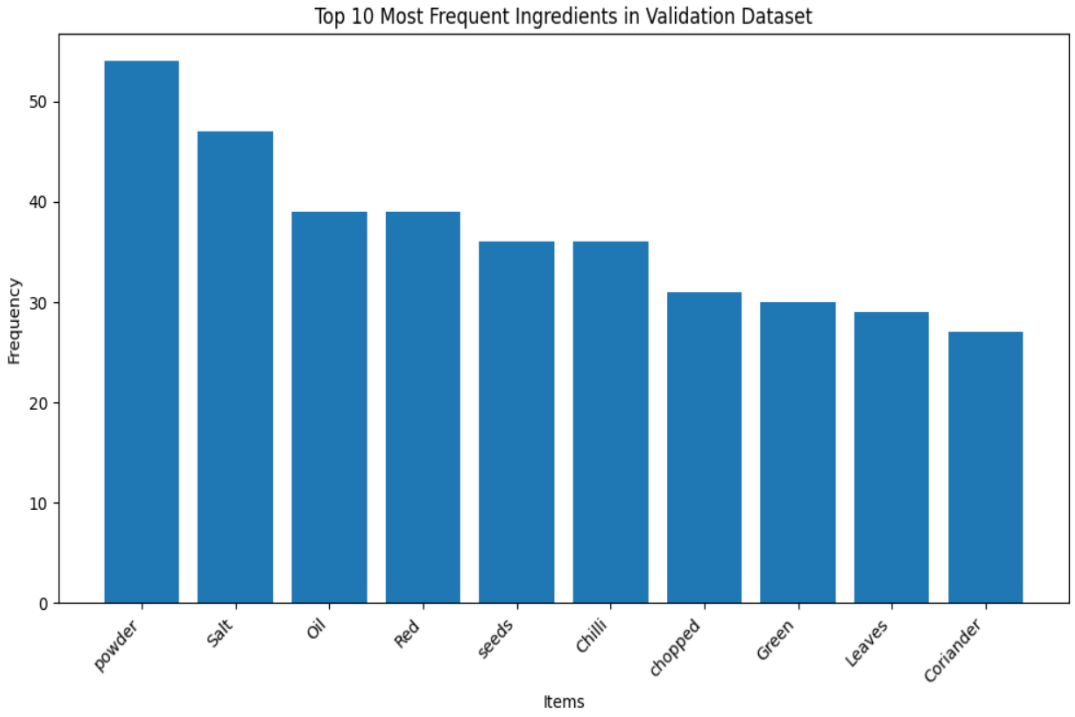
- Top 10 Most Frequent **Ingredients**

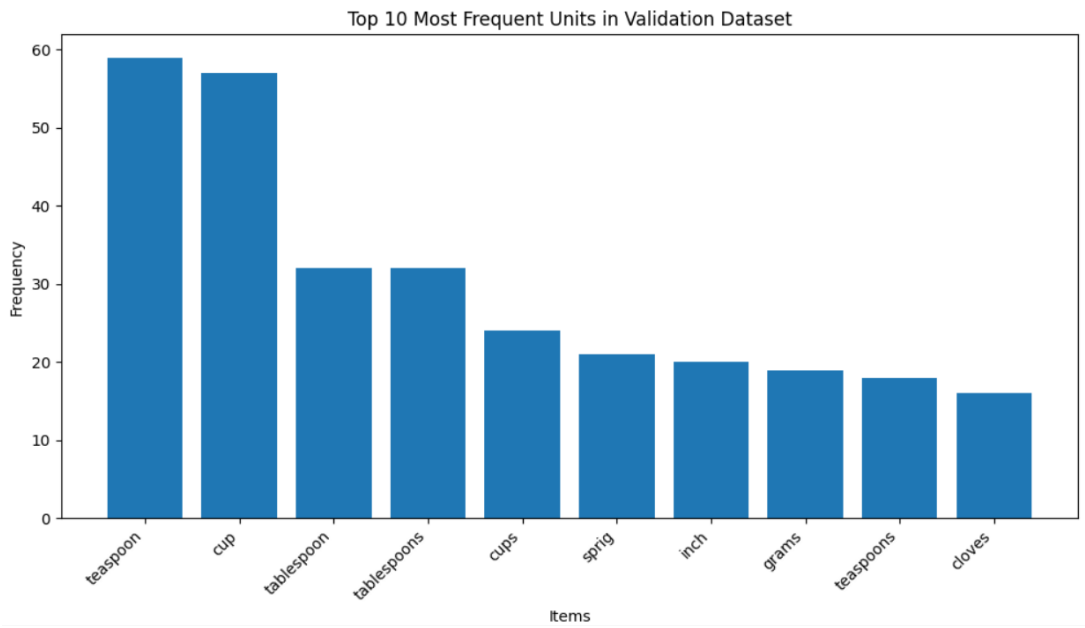


- Top 10 Most Frequent **Units**



Step 5: EDA on Validation Data Set





Step 6: Feature Extraction for CRF Model

- Define a feature function to take each token from recipe
- # Define keywords for units and quantities
- unit_keywords = {"cup", "cups", "teaspoon", "teaspoons", "tablespoon", "tablespoons", "tbsp", "tsp", "tbsps", "tsps", "ounce", "ounces", "oz", "pound", "pounds", "lb", "lbs", "gram", "grams", "g", "kilogram", "kilograms", "kg", "milliliter", "milliliters", "ml", "liter", "liters", "l", "pinch", "pinches"}
- quantity_keywords = {"half", "quarter", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten"}

- # Regular expression pattern for quantities (fractions, numbers, decimals)
- **quantity_pattern = r"^\d+(?:-\d+/\d+|\/\d+|\.\d+)?$"** # Matches fractions, decimals, and whole numbers
- 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.
- Convert X_train, X_val, y_train and y_val into train and validation feature sets and labels

- Apply weight to the feature set
  X_train_weighted_features, y_train_weighted_labels = extract_features_with_class_weights(X_train_features, y_train_labels, weight_dict)

  X_val_weighted_features, y_val_weighted_labels = extract_features_with_class_weights(X_val_features, y_val_labels, weight_dict)

Step 7: Model Building and Training

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

Evaluation of Training Data set:

```
              precision    recall  f1-score   support

  ingredient       1.00      1.00      1.00      5323
    quantity       0.99      0.98      0.98       980
        unit       0.98      0.99      0.98       811

    accuracy                           1.00      7114
   macro avg       0.99      0.99      0.99      7114
weighted avg       1.00      1.00      1.00      7114
```

```
# create a confusion matrix on training datset
print(confusion_matrix(y_train_flat, y_pred_train_flat, labels = list(unique_pos_labels)))
```

```
[[5323    0    0]
 [   0  960   20]
 [   0   10  801]]
```

Step 8: Prediction and Model Evaluation

```
              precision    recall  f1-score   support

  ingredient       1.00      1.00      1.00      2107
    quantity       0.98      0.98      0.98       411
        unit       0.97      0.98      0.98       358

    accuracy                           0.99      2876
   macro avg       0.99      0.99      0.99      2876
weighted avg       0.99      0.99      0.99      2876
```

```
# create a confusion matrix on validation dataset

print(confusion_matrix(y_val_weighted_labels_flat, y_pred_val_flat, labels=list(unique_pos_labels)))
```

```
[[2107    0    0]
 [   0  402    9]
 [   0    7  351]]
```

Step 9: Error Analysis on Validation Data

- Overall accuracy on validation data: **98.89%**

```
Label: ingredient
Number of errors: 0
Class Weight: 0.6682321998872816
Accuracy for ingredient: 100.00%
--------------------
Label: quantity
Number of errors: 18
Class Weight: 7.259183673469388
Accuracy for quantity: 98.16%
--------------------
Label: unit
Number of errors: 14
Class Weight: 8.771886559802713
Accuracy for unit: 98.27%
--------------------
```

END OF REPORT