Identifying Key Entities in Recipe Data

1. 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 managementsystems, dietary tracking apps, or e-commerce platforms.

2. Problem Statement

The goal of this assignment is to identify and classify key entities in cooking recipe data using Named Entity Recognition (NER). Specifically, we aim to extract and correctly label entities such as 'ingredient', 'quantity', and 'unit' from textual recipe instructions. This task is vital for structuring and digitizing recipe data to make it searchable and usable in digital applications.

- **Identify:** Recognize and classify words or phrases as ingredients, quantities, or units.
- **Structure:** Convert unstructured recipe ingredient lists into a structured database.
- **Enable Advanced Features:** Facilitate functionalities such as ingredient-based search, portion scaling, meal planning, and automated recipe recommendations, thereby reducing manual data entry and improving data consistency.

3. Assumptions

During the development of the Named Entity Recognition (NER) pipeline for identifying key entities in recipe data, several assumptions were made to simplify the modeling process and focus the scope of the problem. These assumptions influenced data preprocessing, feature extraction, and model design:

- **Entity Classes Are Mutually Exclusive:** Each token is assumed to belong to only one of the three entity classes: 'ingredient', 'quantity', or 'unit'. No overlapping or nested entities are considered.
- **Labels Are Assigned at the Token Level:** The model processes and assigns labels at the token level. Multi-word entities are expected to be captured by a sequence of token-level predictions.
- **Context Is Limited to Neighboring Tokens:** Feature engineering includes

context from a fixed number of preceding and following tokens, without considering the entire sentence or paragraph.

- **Class Weights Are Used to Address Imbalance:** To address imbalance in the dataset (e.g., more 'ingredient' tokens), class weights are computed and used in the training process.
- **No External Knowledge Base Is Used:** The model relies solely on patterns learned from the training data, without referencing external databases or lexicons for ingredients or units.
- **Evaluation Ignores Sentence-Level Entity Boundaries:** Evaluation is based on token-level accuracy and confusion matrices. Span-level or partial entity recognition is not accounted for.
- **Noise and Misspellings Are Minimal:** The code assumes a clean dataset without significant typos, OCR errors, or noisy data entries.

4. Methodology & Overall Approach

The approach involves preprocessing recipe texts, tokenizing them, and then applying NER to classify each token. Used traditional sequence labeling techniques along with feature engineering to build our models. The key stages in our methodology include:

1. **Importing necessary libraries:**
    - import sklearn_crfsuite # sklearn-crfsuite is a Py wrapper for CRFsuite (CRF implementation for sequence modeling)
    - import spacy # Library for advanced NLP tasks
    - from sklearn.model_selection import train_test_split
    - from sklearn_crfsuite import metrics # For evaluating CRF models
2. **Data Ingestion and Preparation:**
    - Read Recipe Data from Dataframe and prepare the data for analysis
    - df.shape and df.info() DataFrame

```python
# print the dimensions of dataframe - df
print(f"\nDataframe dimensions: {df.shape}")
print(f'Number of rows: {df.shape[0]}')
print(f'Number of columns: {df.shape[1]}')
```

```
Dataframe dimensions: (285, 2)
Number of rows: 285
Number of columns: 2
```

```python
# print the information of the dataframe
print("\nDataframe Information:")
print(df.info())
print(f"\nDataframe columns names:")
print(df.columns.tolist())
print(f"\nData Types")
print(df.dtypes)
print('\nNull value check:')
print(df.isnull().sum())
```

```
Dataframe Information:
<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
```

- Check for rows with unequal length of tokens in input and pos and remove them

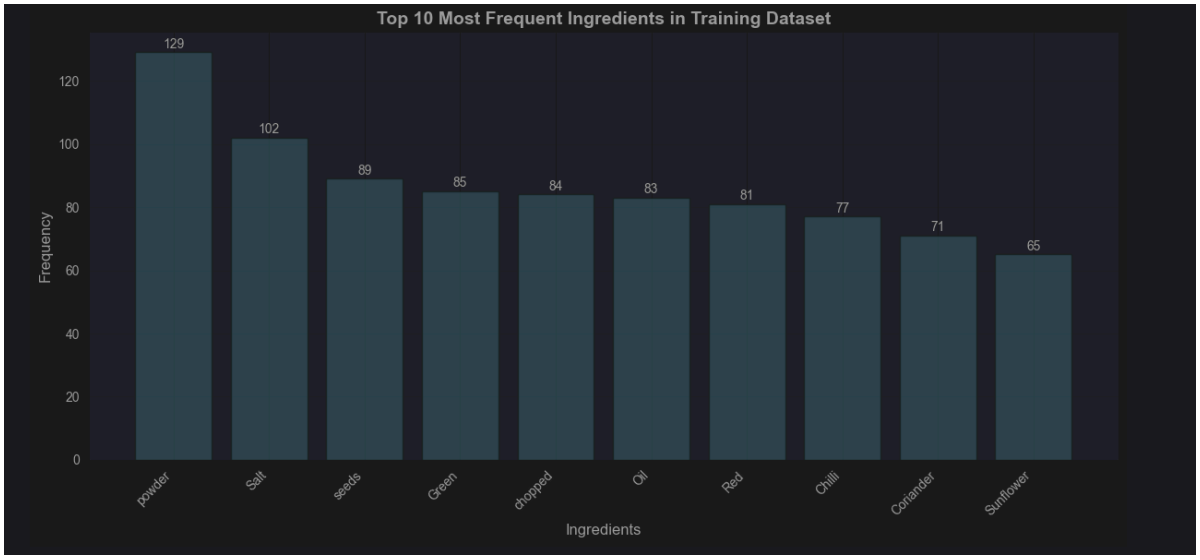
```python
# check for the equality of input_length and pos_length in the dataframe
length_mismatch = df[df['input_length'] != df['pos_length']]
print(f'Number of rows with length mismatch: {len(length_mismatch)}')
if len(length_mismatch) > 0:
    print('Rows with unequal input and pos lengths:')
    print(length_mismatch[['input_length', 'pos_length']].head(10))
    print(f'Indices of mismatched rows: {length_mismatch.index.tolist()}')
```

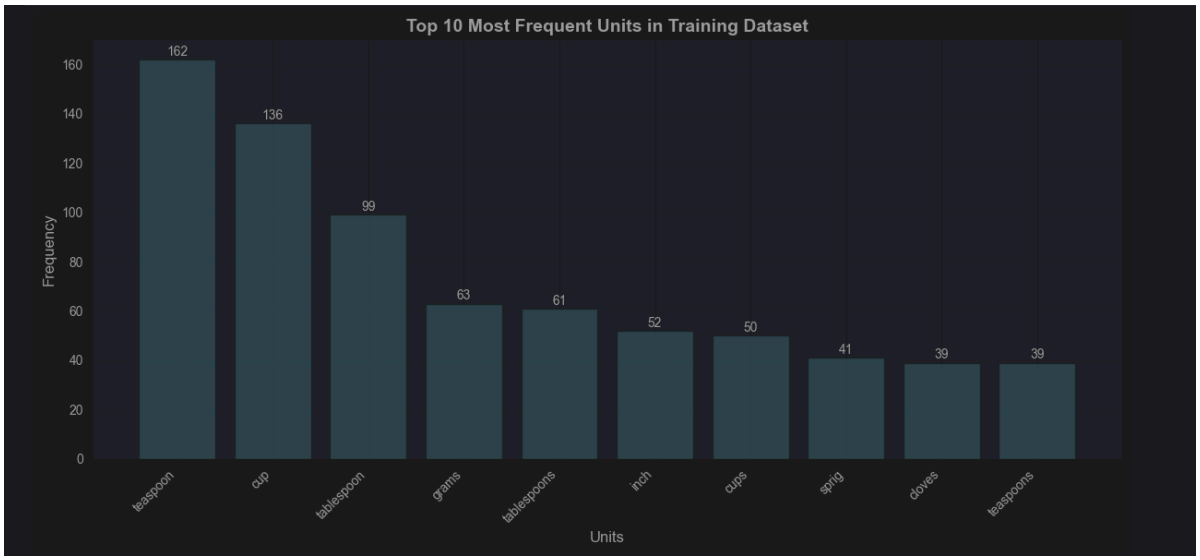
```
Number of rows with length mismatch: 5
Rows with unequal input and pos lengths:
     input_length  pos_length
17             15          14
27             37          36
79             38          37
164            54          53
207            18          17
Indices of mismatched rows: [17, 27, 79, 164, 207]
```

- We can observe that we have only 3 unique pos labels in the recipe: {'unit','quantity', 'ingredient'}

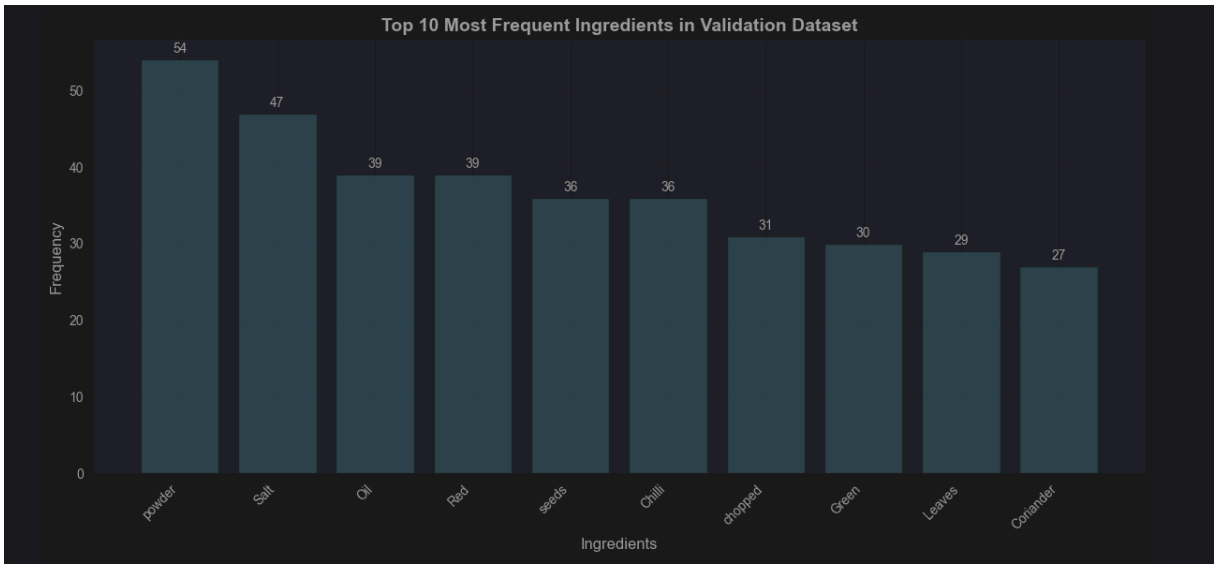
3. **Train-Validation Split:**
    - After splitting data into training (70%) and validation (30%) we have:
        - Training samples: 196 and Validation samples: 84

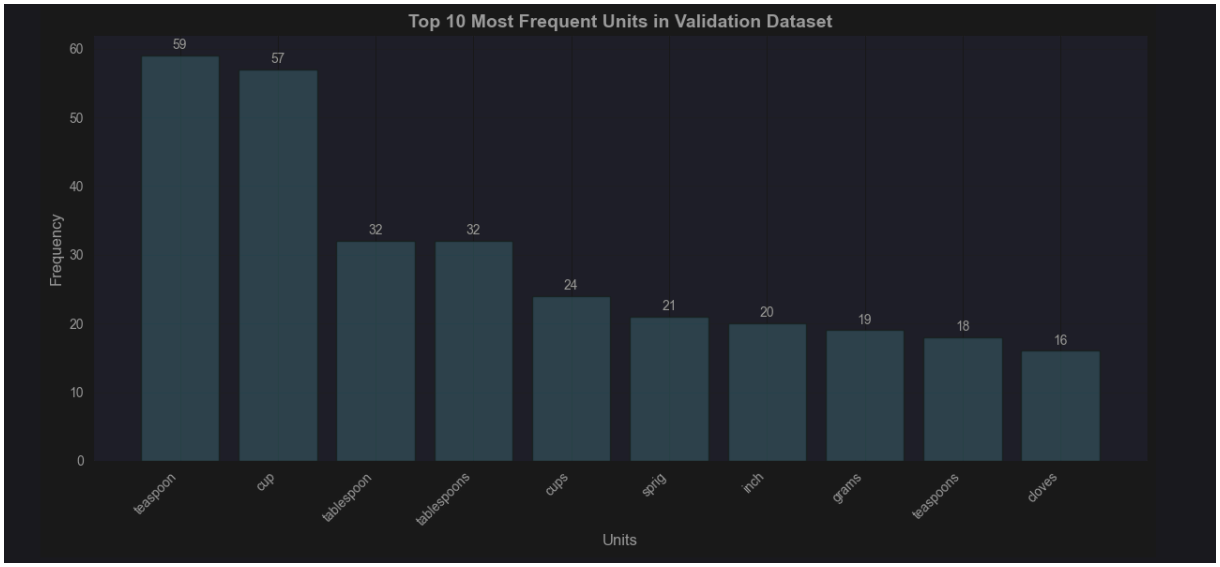
4. **Exploratory Data Analysis (EDA):**
    - **Token Distribution:** After flattening all tokens and labels from the training set, we observed the following distribution:
        - ingredient: 74.82%
        - quantity: 13.78%
        - unit: 11.40%
          This highlights the imbalance, with ingredient being the most prevalent label.
    - **Top Frequent Items:**
        - Categorizing tokens into labels (unit, ingredient, quantity)
            - List top 10 frequent items in ingredient and unit lists for **Training Data**





- List top 10 frequent items in ingredient and unit lists for **Validation Data**





5. **Feature Extraction for CRF Model:**
    - Preform feature extraction to extract each token from recipe (word2features)
    - Applying weights to feature sets {'quantity': 2.4197 , 'unit': 2.9240 , 'ingredient':0.4455}
    - Penalising ingredient label with 0.5 reducing weights_dict to {'quantity': 2.4197 , 'unit': 2.9240 , 'ingredient': 0.2227}
6. **Model Building & Training:**
    - Initializing CRF model with specified hyperparameters

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

```
# initialise CRF model with the specified hyperparameters and use weight_dict
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',           # L-BFGS optimization algorithm
    c1=0.5,                      # L1 regularization coefficient
    c2=1.0,                      # L2 regularization coefficient
    max_iterations=100,          # Maximum number of iterations
    all_possible_transitions=True  # Consider all possible state transitions
)
# train the CRF model with the weighted training data
print("Training CRF model...")
crf.fit(X_train_weighted_features, y_train_labels)
print("CRF model training completed!")
✓ [127] 214ms

Training CRF model...
CRF model training completed!
```

7. **Prediction & Evaluation:**
    - Evaluate training data and print classification report

```
# specify the flat classification report by using training data for evaluation
print("\nTraining Set - Classification Report:")
print("=" * 60)
labels = sorted(list(set(y_train_flat)))
train_report = flat_classification_report(y_train_labels, y_pred_train, labels=labels)
print(train_report)
✓ [129] 31ms

Training Set - Classification Report:
============================================================
              precision    recall  f1-score   support

  ingredient       1.00      1.00      1.00      5323
    quantity       1.00      1.00      1.00       980
        unit       1.00      1.00      1.00       811

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

- Display Confusion Matrix on Training Data

```
Training Set - Confusion Matrix:
            ingredient  quantity  unit
ingredient        5323         0     0
quantity             0       980     0
unit                 0         0   811
```

- Evaluate validation data and print classification report, from which we can draw that our model has performed well

```
# specify flat classification report
print("\nValidation Set - Classification Report:")
print("=" * 60)
val_report = flat_classification_report(y_val_labels, y_pred_val, labels=labels)
print(val_report)
```

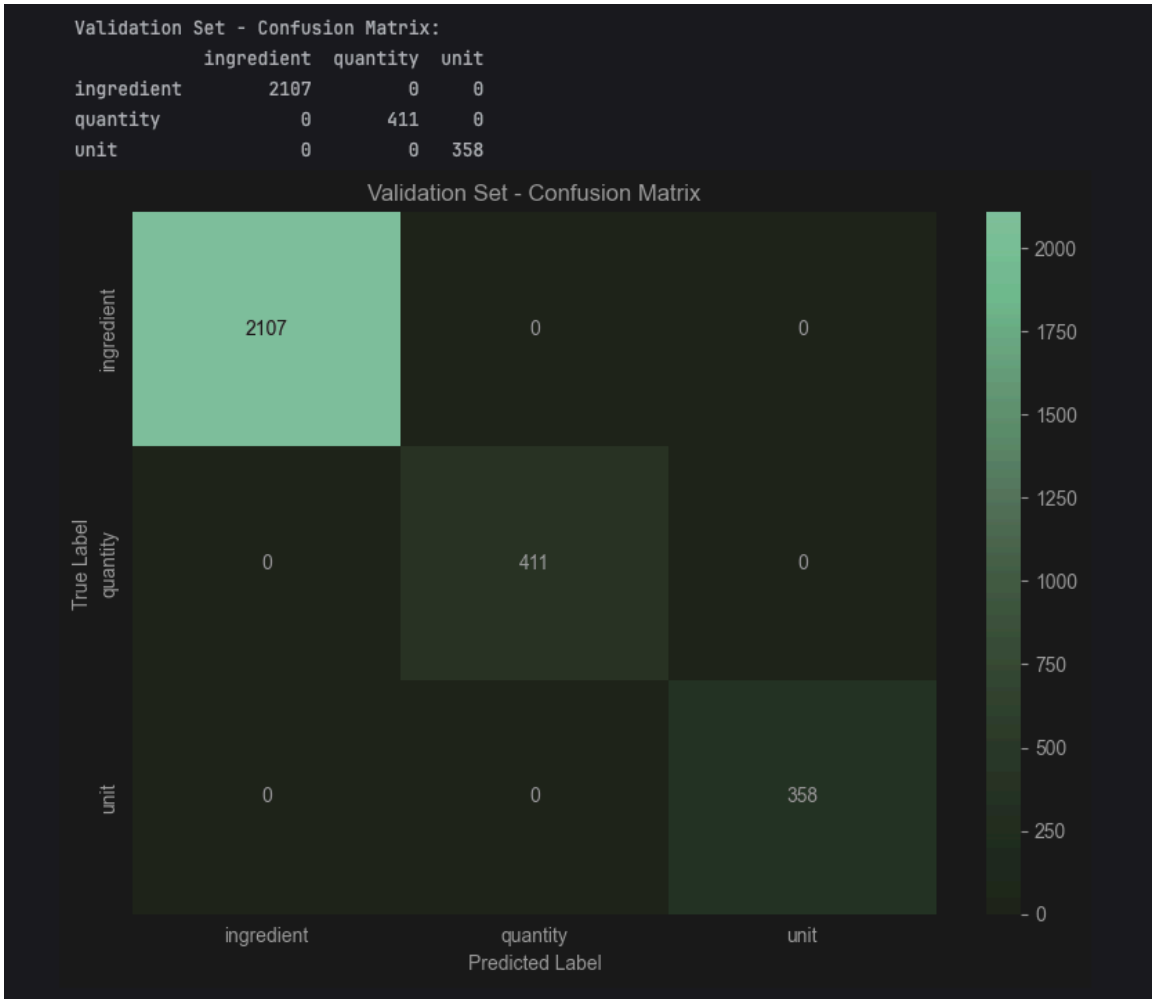
✓ [133] 15ms

```
Validation Set - Classification Report:
============================================================
              precision    recall  f1-score   support

  ingredient       1.00      1.00      1.00      2107
    quantity       1.00      1.00      1.00       411
        unit       1.00      1.00      1.00       358

    accuracy                           1.00      2876
   macro avg       1.00      1.00      1.00      2876
weighted avg       1.00      1.00      1.00      2876
```

- Display Confusion Matrix on Validation Data



8. **Error Analysis on Validation Data:** Given the **100% accuracy** on the validation dataset, no misclassified samples were found. This means the model perfectly predicted the labels for all tokens in the validation set. While the framework for detailed error analysis (identifying token context, true vs. predicted labels, and class weights) was implemented, it did not identify any errors to report.

```python
# Create DataFrame and Print Overall Accuracy
if error_data:
    error_df = pd.DataFrame(error_data)
    print(f"\nError DataFrame created with {len(error_df)} error cases")
else:
    error_df = pd.DataFrame()
    print("\nNo errors found - perfect predictions!")

# Calculate overall accuracy
correct_predictions = len(y_true_flat) - len(error_data)
overall_accuracy = correct_predictions / len(y_true_flat)

print(f"\nOVERALL ACCURACY ANALYSIS:")
print(f"  Total tokens: {len(y_true_flat):,}")
print(f"  Correct predictions: {correct_predictions:,}")
print(f"  Incorrect predictions: {len(error_data):,}")
print(f"  Overall accuracy: {overall_accuracy:.4f} ({overall_accuracy*100:.2f}%)")
```

✓ [137] < 10 ms

```
No errors found - perfect predictions!

OVERALL ACCURACY ANALYSIS:
  Total tokens: 2,876
  Correct predictions: 2,876
  Incorrect predictions: 0
  Overall accuracy: 1.0000 (100.00%)
```

5. Conclusion & Future Enhancements

The model's **100% accuracy on validation data**, the model demonstrates excellent performance and generalization on validation set.

For further analysis consider:

- Cross-validation for model’s robustness.
- Monitoring performance when scaling to new recipes
- Collect more training data for underperforming categories