Step-by-Step Process to Build the Next Word Prediction Project using NLTK

### 1. Set Up the Environment

- Install the required libraries:

  ```bash
  pip install nltk numpy pandas
  ```

- Import necessary libraries in Python:

  ```python
  import nltk
  import numpy as np
  import pandas as pd
  import random
  from collections import Counter
  ```

### 2. Load and Preprocess the Dataset

- Load a text dataset or define your own sentences.
- Tokenize the text into words:

  ```python
  nltk.download('punkt')
  from nltk.tokenize import word_tokenize

  sentences = ["This is a Data Science course",
               "Machine Learning is fun",
               "Natural Language Processing is a subfield of AI"]

  tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]
  ```

### 3. Generate N-grams (Unigram, Bigram, Trigram)

- Create bigrams and trigrams from the tokenized sentences:
  ```python
  from nltk.util import ngrams

  def generate_ngrams(words, n):
      return list(ngrams(words, n))

  for sentence in tokenized_sentences:
      print("Bigrams:", generate_ngrams(sentence, 2))
      print("Trigrams:", generate_ngrams(sentence, 3))
  ```

### 4. Build a Frequency Distribution Model
- Count occurrences of bigrams and trigrams:
  ```python
  bigram_counts = Counter()
  trigram_counts = Counter()

  for sentence in tokenized_sentences:
      bigram_counts.update(generate_ngrams(sentence, 2))
      trigram_counts.update(generate_ngrams(sentence, 3))
  ```

### 5. Implement Next Word Prediction
- Predict the next word based on previous words using probabilities:
  ```python
  def predict_next_word(previous_words, ngram_counts):
      possible_next_words = [ngram[-1] for ngram in ngram_counts if ngram[:-1] == previous_words]

      if not possible_next_words:
          return None

          next_word = max(possible_next_words, key=lambda word: ngram_counts[previous_words + (word,)])
  ```

```
      return next_word

  # Example Prediction
  prev_bigram = ('data', 'science')
  print("Predicted Next Word:", predict_next_word(prev_bigram, bigram_counts))
```

### 6. Evaluate the Model
- Check the accuracy of predictions by comparing them to actual words in test sentences.
- Use perplexity as a metric to evaluate how well the model predicts words.

### 7. Future Enhancements (Optional)
- Integrate GPT-2 or Transformers for better accuracy.
- Use GPTTokenizer to process larger datasets.
- Train on large corpora to improve predictions.

This step-by-step approach will help you build a basic next-word predictor using NLTK and n-grams.