Part 1: Defining classes

We started by analyzing the corpus to identify the most frequently appearing speaker names. Then, we examined these top speakers to detect alternate spellings or aliases for the same individual (e.g., "ר' ריבלין" and "ראובן ריבלין"). Using a mapping of aliases to their primary names, we grouped sentences belonging to the same speaker without modifying the original names. Finally, we classified the data:

1. **Binary Classification**: Selected sentences of the top two speakers.
2. **Multi-class Classification**: Created three categories: the two top speakers and "Others" for all remaining speakers.

```
Primary speakers: ['וילבירז ובואר', "א' גרוב"]
```

The names with the highest frequency before mapping:

1. **ראובן ריבלין**: 2800 appearances
2. **א' בורג**: 1761 appearances

After Mapping:

1. **ראובן ריבלין**: 3113 appearances
2. **א' בורג**: 1898 appearances

Note: The names in the corpus were not changed; they were simply categorized into the appropriate classes.

```
map_speakers_to_aliases = {
    "ר' ריבלין": "ראובן ריבלין",
    "ראובן ריבלין": "ראובן ריבלין",
    "הכנסת ראובן ריבלין": "ראובן ריבלין",
    "רובי ריבלין": "ראובן ריבלין",
    "אברהם בורג": "א' בורג",
    "א' בורג": "א' בורג"
}
```

Part 2: Balancing

How Down-Sampling Was Performed:

For each classification task:

- We identify the class with the smallest number of sentences (minority class).
- Then, we randomly sampled a subset of sentences from larger classes to match the size of the minority class.

This approach resulted in balanced datasets for both binary and multi-class classification tasks.

Here is the results:

Binary Classification:

- **Before Balancing:**
  - Class 1: 3137 sentences.
  - Class 2: 1898 sentences.
- **After Balancing:**
  - Class 1: 1898 sentences.
  - Class 2: 1898 sentences.

Multi-Class Classification:

- **Before Balancing:**
  - Class 1: 3137 sentences.
  - Class 2: 1898 sentences.
  - Class 3 ("Others"): 96769 sentences.
- **After Balancing:**
  - Class 1: 1898 sentences.
  - Class 2: 1898 sentences.
  - Class 3: 1898 sentences.

Part 3: feature vector

1. Bag of Words(BoW or TF-IDF):

In this step, we created a feature vector for each sentence using the **TF-IDF** method. This was chosen over CountVectorizer because TF-IDF provides a weighted representation of words, giving more importance to unique and informative words while reducing the influence of common words. Here's what we did:

- **Data Source:** We used the `sentence_text` field from the corpus.
- **Feature Extraction:** We extracted TF-IDF features for each sentence in the balanced binary and multi-class datasets.
- **Feature Shape:** The resulting matrices represent each sentence as a vector in a high-dimensional space where each dimension corresponds to a specific word in the corpus vocabulary.

2. Custom Features (Style and Content):

1. Feature Creation:

- Sentence and Text Length:
    - The function calculates the number of words in a sentence (sentence_length) and the total number of characters (text_length).
- Punctuation Counts:
    - Counts specific punctuation marks like commas, periods, questions, exclamations, and dashes, which are key style-based features.
- Protocol Metadata:
    - Retrieves numerical metadata fields like protocol_number and knesset_number.
- Protocol Type:
    - Extracts the protocol type and prepares it for encoding.

2. Encoding Categorical Features:

- Uses LabelEncoder to convert categorical protocol types into numeric values, adding these as additional features.

At the end, all the extracted features, including text-based features (like sentence and text length), style-based features (such as punctuation counts), and metadata

(like protocol numbers and encoded protocol types), were combined into a single feature matrix. This matrix represents each sentence with a comprehensive set of features, capturing both its linguistic characteristics and contextual information.

Reasoning Behind Choices:

- **TF-IDF:** Chosen for its ability to capture the importance of words in a sentence relative to the entire corpus, helping differentiate between sentences with unique content.
- **Custom Features:** Designed to complement the text-based representation by capturing stylistic and contextual nuances, such as sentence structure, punctuation usage, and protocol metadata, providing a richer and more holistic understanding of each sentence.

Part 4: Training

To evaluate the performance of our classification models, we implemented the cross_validate_and_report function that provides a detailed analysis of model performance:

We evaluated two classifiers for each dataset (binary and multi-class):

- **K-Nearest Neighbors (KNN)**
- **Logistic Regression**

Then , we applied **5-fold Cross Validation** to each classifier.

For every classification type (binary and multi-class) and feature type (TF-IDF and custom features), we generated a **classification report**. This report detailed:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-Score**

To optimize the performance of the classifiers, we chose the hyperparameters using **Grid Search**:

- For **KNN**, we tuned parameters such as the number of neighbors (n_neighbors) , the distance metric (metric) and weighting scheme (weights)
- For **Logistic Regression**, we adjusted parameters like the regularization strength (C), maximum iterations (max_iter) and the solver (solver).

Example of Parameter Tuning for KNN:
For the binary classification task using TF-IDF features, we applied **Grid Search** to identify the best parameters for the K-Nearest Neighbors (KNN) classifier. Below is the output showing the optimal combination of hyperparameters found during this process:

```
Grid search for KNN : binary class, tfidf
Best Parameters for KNN: {'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'distance'}
```

Classification Report:

Here is the detailed classification report generated for each classifier and feature type:

```
Processing Binary Classification with TF-IDF Features...

K-Nearest Neighbors with TF-IDF Features Classification Report:
              precision    recall  f1-score   support

     Class 1       0.84      0.87      0.86      1898
     Class 2       0.86      0.84      0.85      1898

    accuracy                           0.85      3796
   macro avg       0.85      0.85      0.85      3796
weighted avg       0.85      0.85      0.85      3796


Logistic Regression with TF-IDF Features Classification Report:
              precision    recall  f1-score   support

     Class 1       0.84      0.94      0.89      1898
     Class 2       0.93      0.83      0.87      1898

    accuracy                           0.88      3796
   macro avg       0.89      0.88      0.88      3796
weighted avg       0.89      0.88      0.88      3796
```

```
Processing Binary Classification with Custom Features...

K-Nearest Neighbors with Custom Features Classification Report:
              precision    recall  f1-score   support

     Class 1       0.90      0.91      0.90      1898
     Class 2       0.91      0.90      0.90      1898

    accuracy                           0.90      3796
   macro avg       0.90      0.90      0.90      3796
weighted avg       0.90      0.90      0.90      3796


Logistic Regression with Custom Features Classification Report:
              precision    recall  f1-score   support

     Class 1       0.92      0.86      0.89      1898
     Class 2       0.87      0.92      0.89      1898

    accuracy                           0.89      3796
   macro avg       0.89      0.89      0.89      3796
weighted avg       0.89      0.89      0.89      3796
```

```
Processing Multi-Class Classification with Custom Features...

K-Nearest Neighbors with Custom Features Classification Report:
              precision    recall  f1-score   support

     Class 1       0.73      0.79      0.76      1898
     Class 2       0.82      0.87      0.84      1898
     Class 3       0.79      0.68      0.73      1898

    accuracy                           0.78      5694
   macro avg       0.78      0.78      0.78      5694
weighted avg       0.78      0.78      0.78      5694


Logistic Regression with Custom Features Classification Report:
              precision    recall  f1-score   support

     Class 1       0.54      0.57      0.55      1898
     Class 2       0.62      0.79      0.69      1898
     Class 3       0.72      0.49      0.58      1898

    accuracy                           0.61      5694
   macro avg       0.63      0.61      0.61      5694
weighted avg       0.63      0.61      0.61      5694
```

```
Processing Multi-Class Classification with TF-IDF Features...

K-Nearest Neighbors with TF-IDF Features Classification Report:
              precision    recall  f1-score   support

     Class 1       0.65      0.63      0.64      1898
     Class 2       0.80      0.82      0.81      1898
     Class 3       0.63      0.64      0.63      1898

    accuracy                           0.69      5694
   macro avg       0.69      0.69      0.69      5694
weighted avg       0.69      0.69      0.69      5694


Logistic Regression with TF-IDF Features Classification Report:
              precision    recall  f1-score   support

     Class 1       0.69      0.67      0.68      1898
     Class 2       0.88      0.81      0.84      1898
     Class 3       0.65      0.71      0.68      1898

    accuracy                           0.73      5694
   macro avg       0.74      0.73      0.73      5694
weighted avg       0.74      0.73      0.73      5694
```

Part 5: Sentence Classification

We implemented the classify_sentences function, which classifies sentences from the knesset_sentences.txt file into predefined categories. First, the sentences were loaded and stripped of unnecessary whitespace. Next, the pre-trained TF-IDF vectorizer transformed the text into numerical features that captured the importance of words within the sentences. Using the trained K-Nearest Neighbors (KNN) model, each sentence was classified into one of three categories: "first" , "second" or "other" . The numeric predictions were mapped to readable class labels using a label mapping dictionary, and the results were saved to the output file classification_results.txt, with each sentence's predicted category written on a new line.