# Methodology

Let's go through the complete project step by step.

**NOTE**: The analysing, preprocessing and modelling is done in the **train.ipynb** notebook which is located in the notebooks directory in the root folder. The notebook is well documented so that it could be understood easily.

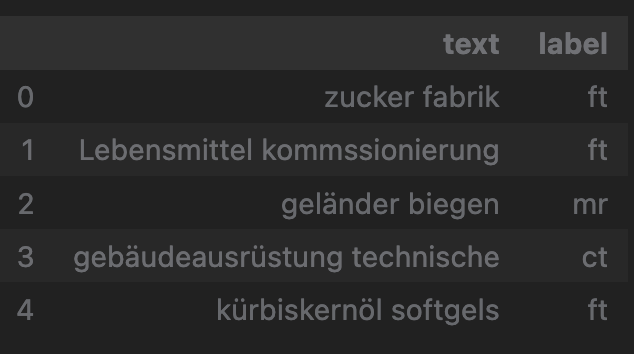
### Data Preprocessing

A “Configuration File” was used (config.json) to generalise the script rather than hard coding it. In this task, config file was used to store the paths for:

* Dataset
* Model file

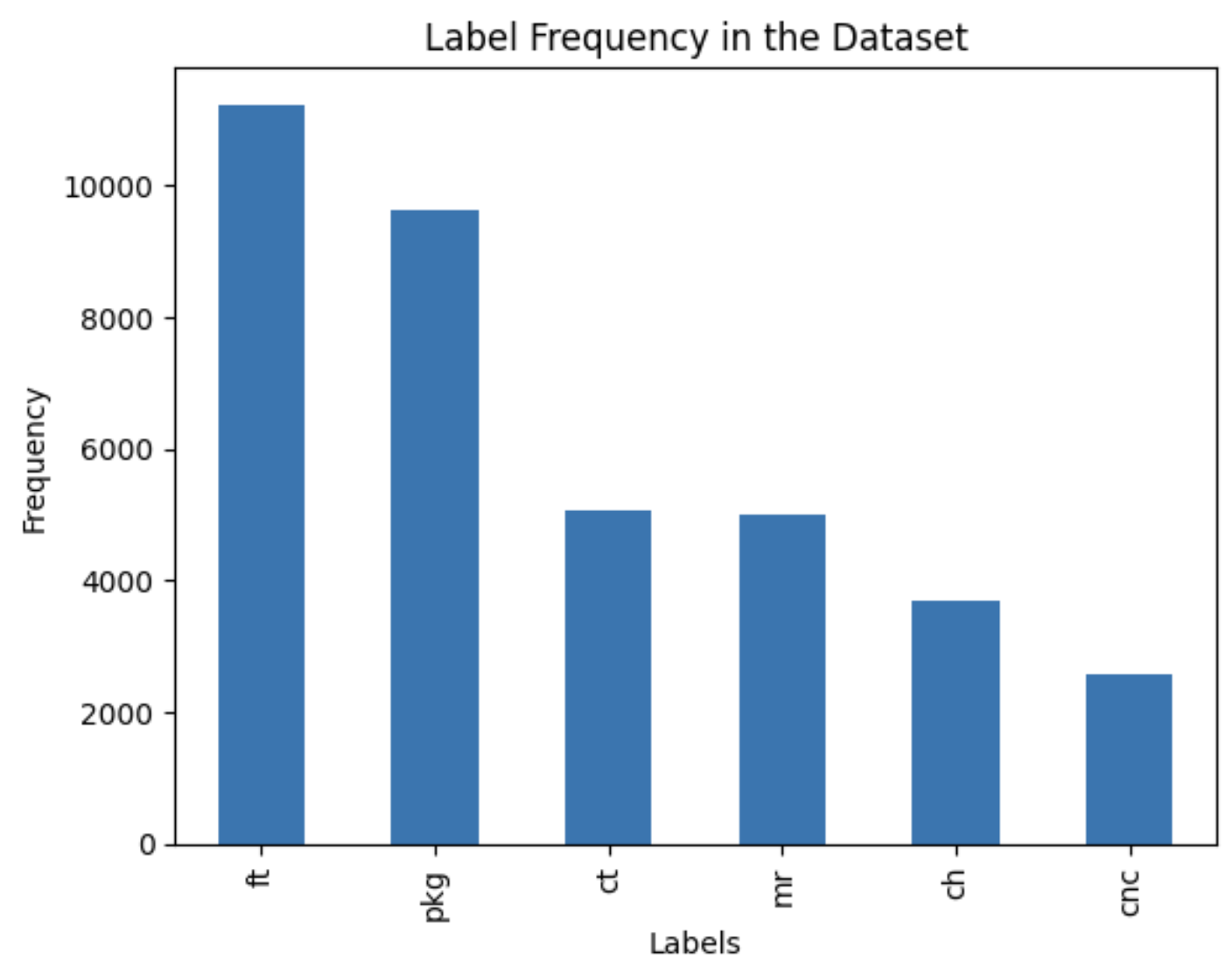
The config file was used in the notebook as well as in the app.py. This helped in reducing redundancy from the project. The config file generally contains all the values which can be changed. The hyperparameter tuning parameters and the related information can also be stored in this file.

The data was loaded to a pandas dataframe. This is what the data looked like:



There were 6 unique labels in the dataset. The data had a total of **37,295** rows in which **100** contained **null** values. Since this was a small number of rows, these 100 rows were dropped.

The data was further analysed. A bar graph was plotted to get an idea of the frequencies of the labels.



As it is very evident from the graph, the data was imbalanced. For this assignment, we will be calculating “F1 score” as well along with the accuracy, since it considers the imbalanced nature of the dataset.

### Modelling

The data was divided into training and testing data with a **70:30** split between the two. Since the data was textual, it had to be vectorized. TF-IDF is used to vectorize data.

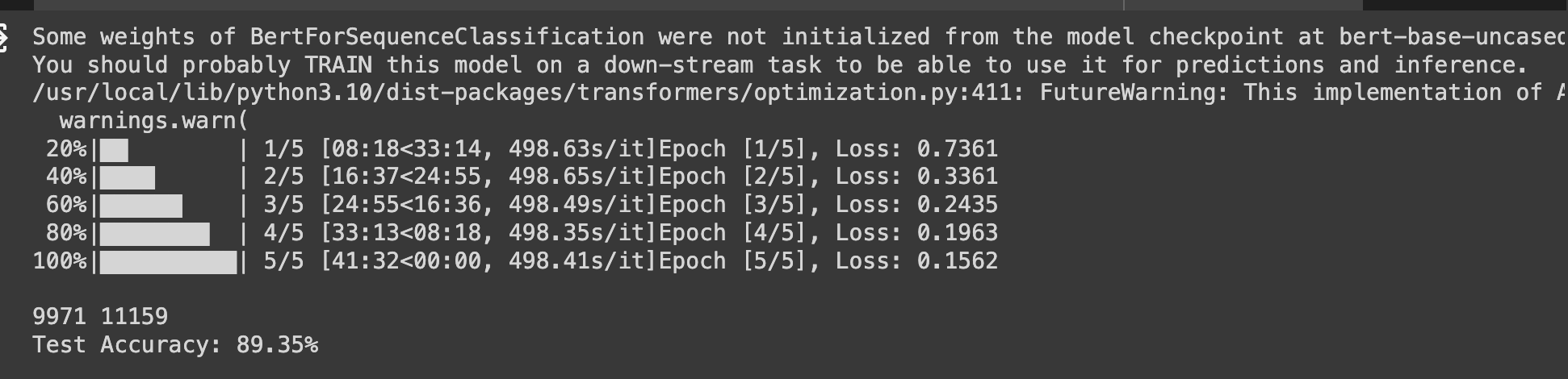
The vectorized data is used to fit a Simple Support Vector Classifier. The model was saved to be used in the app.py for prediction. The classifier performed really well on the given data, giving the following metrics:

* **Accuracy**: 0.887
* **F1 score**: 0.887

The metrics show that the model worked really well on the given dataset.

**BERT Approach**

Even though the model worked really well on the Support Vector Classifier, Bert model was also tried for modelling. My machine could not **fine tune** the already trained Bert model. However, I ran it on google colab, and it gave me an accuracy of **89%,** which was very close to the support vector classifier.



Since there was no major difference between the results of both models, it seemed that using **Bert** was an overkill for this task. The model being used by the API in the project will be the **Support Vector Classifier.** The code for Bert is added to the notebook as well **(The code for Bert was not entirely mine, I updated the code used already on the internet for my dataset)**. I ran it with a very limited dataset (100 rows) on the notebook, which is why it is giving such bad results.

### Deployment

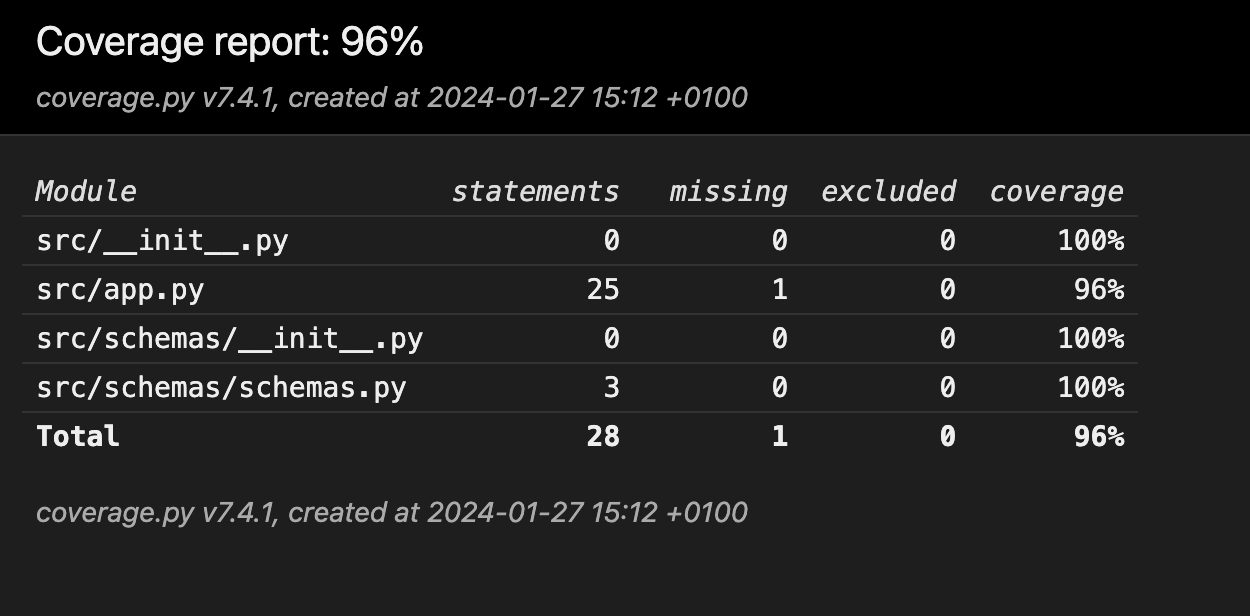
The web framework used for building the API is FastAPI (as advised). A hierarchical directory structure has also been followed.

The API uses a POST endpoint for prediction, the path of which is /predict.

An example request is as follows:  
{ "phrase": "Example Phrase"}

Upon receiving this request, the API uses the loaded model and calls the predict method which outputs an array containing the predicted label.  
An example response is as follows:  
{ "prediction": ["mr"] }

Test cases for the API have also been written, giving **96%** coverage as shown below:



The test coverage of the API code can be seen in the HTML page located in the following path htmlcov/index.html as well.

The application is dockerized as well for quick deployment.

**Note:** The instructions to write a run the application or the Docker container are given in the README.md file present in the root directory of the project.

### Future Work / Improvements

As an improvement, one could work on the following aspects of the pipeline:

* Since the dataset is **imbalanced**, try different techniques to overcome this problem.
* If you have computational resources, **RoBERTa** (Robustly optimized BERT approach) could be explored.