National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Steffi Veientlena
Student ID: x21202109

School of Computing
National College of Ireland

Supervisor: Prof.Prashanth Nayak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Steffi Veientlena
Student ID:	x21202109
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof.Prashanth Nayak
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Steffi Veientlena
Date:	13th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Steffi Veientlena
x21202109

1 Introduction

The provided guide will give information about hardware and software set up required to reproduce this work.

2 Specifications

The table below gives information on the different software and hardware specifications used for the purpose of this study.

Table 1: Software and Hardware Specifications

Configuration	**Description**
Language	Python
Framework	Pytorch
Integrated Development Environment	Google Collab and Jupyter Notebook
Computation	GPU
libraries	transformers, pytorch lightning,pandas,spacy,numpy

The models were trained, tested, and evaluated using the aforementioned tools and technologies on a MacBook. An Intel Core i7 processor, 16 GB of RAM, and integrated Intel Iris Xe graphics where the configuration of the Macbook.

3 Data Set

Data is sourced from Kaggle website, a platform for machine learning datasets, were used in the study. 2225 papers from the BBC were used to create news items in the fields of technology, business, sports, entertainment, and politics. The data, which covered the years 2004 and 2005, was saved in CSV format, uploaded to Google Drive, and examined using a Colab notebook for exploratory data analysis. The data contained category, Title and Content of the nes articles.[1]

[1]Data: `https://www.kaggle.com/datasets/hgultekin/bbcnewsarchive`

4 Data Pre-processing

Several data-preprocessing steps was involves before the data was passed onto various experiments using the transformer model.The dataset was in csv format which was loaded into jupyter notebook pre-processing steps are performed. The Figure 1 displays all the necessary libraries that are required before the pre-processing step is involved.

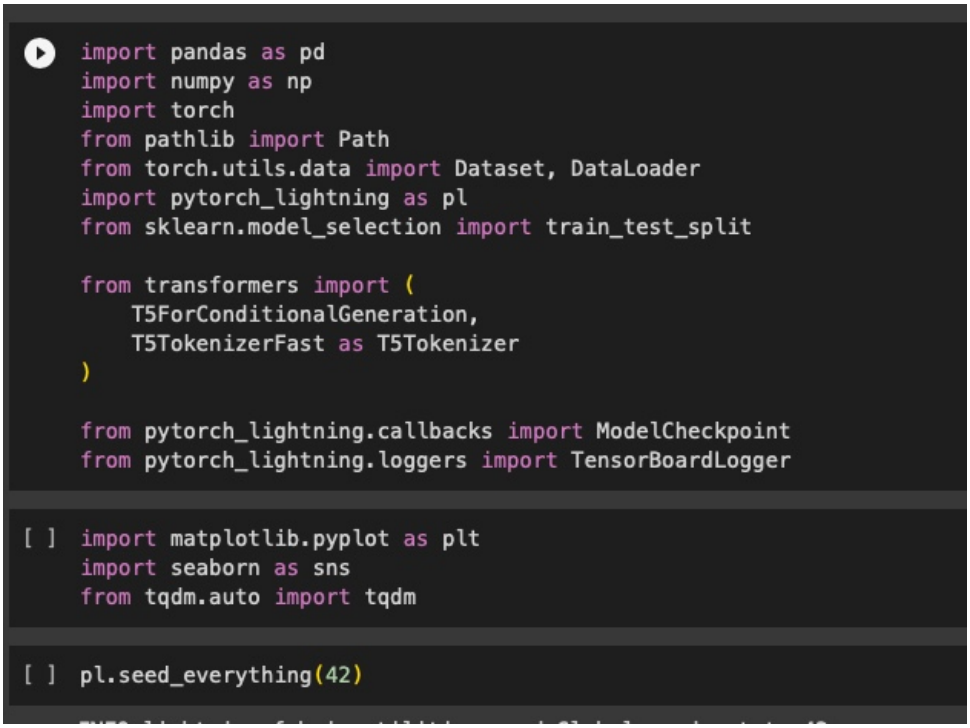
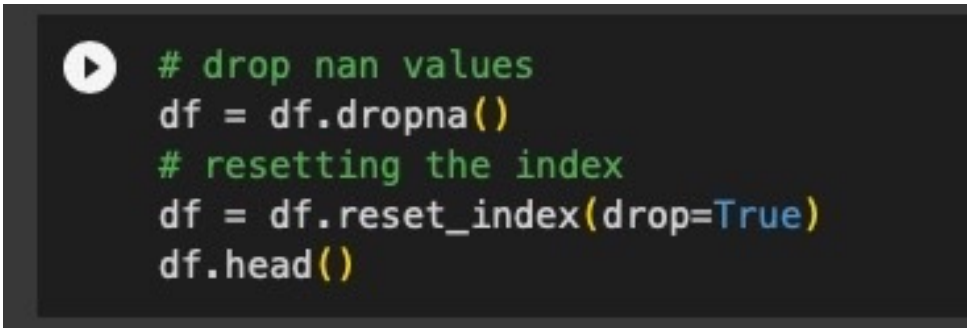


Figure 1: Import Necessary Library

The figures Figure 2 and Figure 3 shows the different pre-processing steps undertaken for this study. The Porter technique was used to stem the words in this dataset, improving generalisation and decreasing data dimensionality by reducing words to their simplest form. Stop words that had little to no meaning were eliminated to improve the clarity of the data. To improve dataset relevance, low-frequency phrases (occurring 3 times) were deleted.

Figure 2: Data Pre-processing

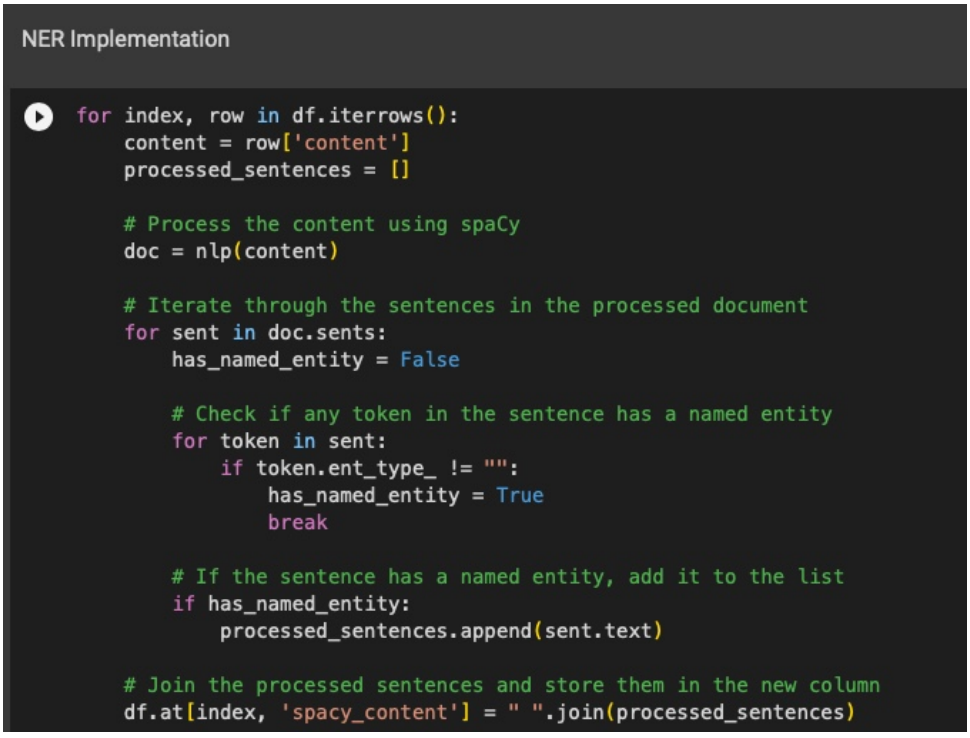


```python
# drop nan values
df = df.dropna()
# resetting the index
df = df.reset_index(drop=True)
df.head()
```

Figure 3: Data Pre-processing 2

5 NER Implementation

The figure Figure 4 describes the implementation of NER (Named Entity Recognition) which is implemented before the T-5 model was used. Spacy library from python package was used to conduct this implementation. A function takes all the articles utilizes the library and identifies the sentences with NER.



NER Implementation

```python
for index, row in df.iterrows():
    content = row['content']
    processed_sentences = []

    # Process the content using spaCy
    doc = nlp(content)

    # Iterate through the sentences in the processed document
    for sent in doc.sents:
        has_named_entity = False

        # Check if any token in the sentence has a named entity
        for token in sent:
            if token.ent_type_ != "":
                has_named_entity = True
                break

        # If the sentence has a named entity, add it to the list
        if has_named_entity:
            processed_sentences.append(sent.text)

    # Join the processed sentences and store them in the new column
    df.at[index, 'spacy_content'] = " ".join(processed_sentences)
```

Figure 4: NER Implementation

The identified NER sentences are stored in a new column called Spacy content. This sentences are further used by models to generate the headlines.

6 Model Implementation

The research thesis was proposed and several experiments where conducted to understand the effectiveness of the related study. There are three major experiments conducted as a part of this study and are implemented as discussed in the below section.The T-5 model was first loaded to the colab environment.

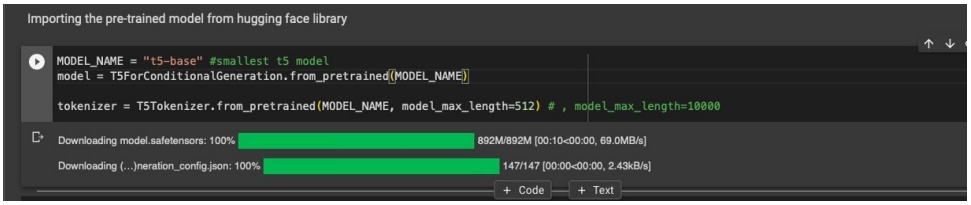


Figure 5: Model Extraction

6.1 Experiment 1

The first experiment was conducted with the pre-trained model itself with the testing set. The Figure 6 below explains the code on how the T-5 model was implemented. This was implemented on the original articles where NER sentences where not obtained and the model is not trained on the existing dataset.

```
generated_headlines = []
for index, row in test_df.iterrows():
    article = row["article"]
    headline = row["headline"]

    input_text = f"summarize: {article} </s> {headline} </s>"
    input_ids = tokenizer.encode(input_text, return_tensors="pt", max_length=512, truncation=True)

    # Generate the combined headline
    output = model.generate(input_ids, max_length=10, num_beams=4, early_stopping=True)
    generated_headline = tokenizer.decode(output[0], skip_special_tokens=True)

    print("Article:", article)
    print("Existing Headline:", headline)
    print("Generated Headline:", generated_headline)
    print("="*50)
    generated_headlines.append(generated_headline)


# Add the generated headlines to the DataFrame
test_df["generated_headline"] = generated_headlines
```

Figure 6: Experiment 1 with pre-trained model

6.2 Experiment 2

The second experiment was conducted by fine-tuning the T-5 model without implementing the NER implementation as described in section 5. The figures below displays the different sections on how the model was trained and then used to generate headline.

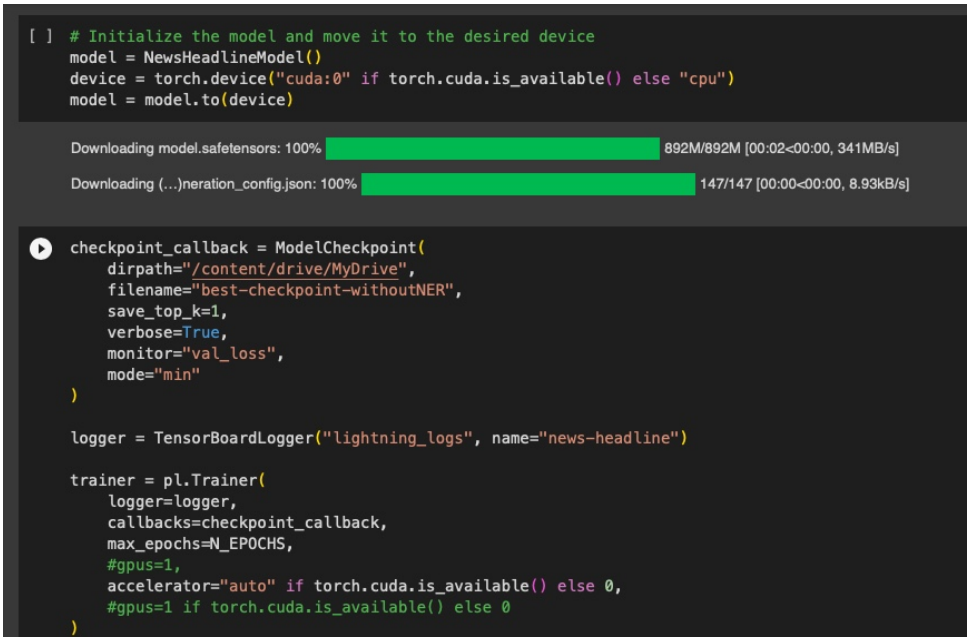


Figure 7: Model Training

In the Figure 7 the model configuration is set to use check if GPU compution is present. If its present the model is moved to GPU. This ensures that the model cmpetency is performed on specified device for optimal performance.

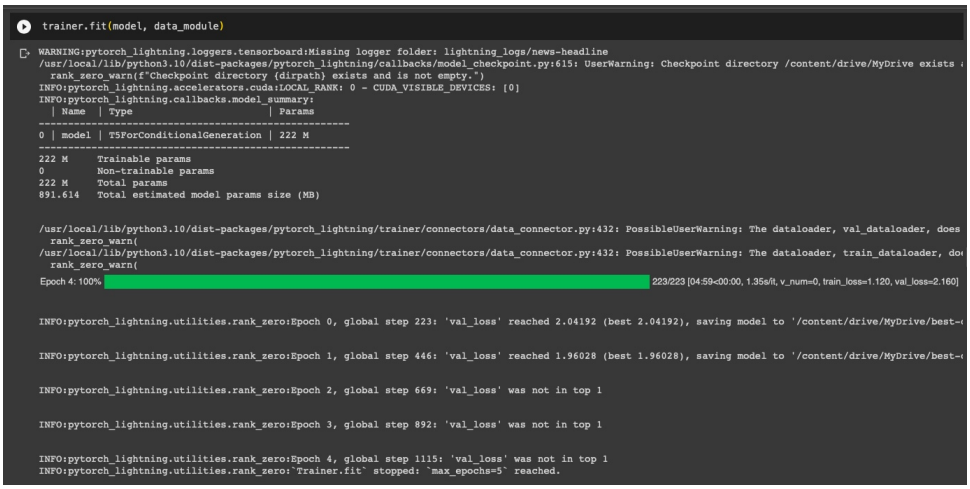


Figure 8: Training the model

In Figure 8 the model was then trained on testing data for 5 epochs in order to get he best training model. It seems the epochs 2 is the one with less validation loss and hence the model with less epochs loss is saved and later used for testing.

The final function written in Figure 9 is generate headline function which now uses trained model for headline generation rather than the pre-trained model.

```python
def generate_headline(text):
  text_encoding = tokenizer(
      text,
      max_length=512,
      padding="max_length",
      truncation=True,
      return_attention_mask=True,
      add_special_tokens=True,
      return_tensors="pt"
  )

  generated_ids = trained_model.model.generate(
      input_ids=text_encoding["input_ids"].to(trained_model.device),
      attention_mask=text_encoding["attention_mask"].to(trained_model.device),
      max_length=150,
      num_beams=2, # beam search
      repetition_penalty=2.5,
      length_penalty=1.0,
      early_stopping=True # To speed up the process
  )

  preds = [
      tokenizer.decode(gen_id, skip_special_tokens=True, clean_up_tokenization_spaces=True)
      for gen_id in generated_ids
  ]

  return "".join(preds)
```

Figure 9: Headline generation using fine-tuned model

6.3 Experiment 3

The last experiment was conducted similar to experiment 2 but it also included NER implementation explain in section 5. The code files will be same as section 5 and section 6 but instead of using the actual content of the article. the model is first trained on NER sentences and then tested with generated headline function to generate headlines.

7 Evaluation

7.1 ROUGE

The code to evalaute ROUGE metrics is given in Figure 10. This ROUGE metrics gives 3 evaluation metric namely ROUGE-1, ROUGE-2 and ROUGE-L. The average of all three gives the total ROUGE score for the model.ROUGE-1 calculates the no og uni grams present in generated text compared to original test.ROUGE-2 calculates the no of bi grams present in generated text when compared with original text. and ROUGE - L calcualtes the longest lenth sequence in both the texts.

7.2 BERT Score

The Figure 11below shows code for Bert score implementation.The quality of machine-generated text, such as summaries or translations, is evaluated using a metric called BERT (Bidirectional Encoder Representations from Transformers) score. It analyses context from both ends of a sentence to determine how similar the generated text and reference text are to one another. In comparison to more conventional metrics like BLEU or ROUGE, the BERT score is useful for evaluating how fluent and informative generated content is.

```
res = {}
res['rouge-1'] = {}
res['rouge-2'] = {}
res['rouge-l'] = {}
for key in res:
  res[key]['r'] = 0
  res[key]['p'] = 0
  res[key]['f'] = 0
for index in range(len(test_df.index)):
  sample_row = test_df.iloc[index]
  text = sample_row["article"]
  reference = sample_row["headline"]
  candidate = generate_headline(text)
  score = ROUGE.get_scores(candidate, reference)
  for key in res:
    res[key]['r'] += score[0][key]['r']
    res[key]['p'] += score[0][key]['p']
    res[key]['f'] += score[0][key]['f']
for key in res:
  res[key]['r'] /= len(test_df.index)
  res[key]['p'] /= len(test_df.index)
  res[key]['f'] /= len(test_df.index)
res
```

Figure 10: ROUGE Evaluation Metric

```
[ ] # Loop over the test dataset
    for index in range(len(test_df.index)):
        sample_row = test_df.iloc[index]
        text = sample_row["article"]
        reference = sample_row["headline"]
        candidate = generate_headline(text)  # Generate the headline using your model

        # Append candidate and reference to the lists
        candidates.append(candidate)
        references.append(reference)

        # Calculate BERTScore
        P, R, F1 = score(candidates, references, lang='en',verbose=False)

        # Aggregate the scores
        #res['bert-score']['p'] += P.mean().item()
        #res['bert-score']['r'] += R.mean().item()
        #res['bert-score']['f'] += F1.mean().item()

    # Calculate average BERTScore
    #num_samples = len(test_df.index)
    #res['bert-score']['p'] /= num_samples
    #res['bert-score']['r'] /= num_samples
    #res['bert-score']['f'] /= num_samples

    avg_P = P.mean().item()
    avg_R = R.mean().item()
    avg_F1 = F1.mean().item()
```

Figure 11: BERT Score Evaluation Metric