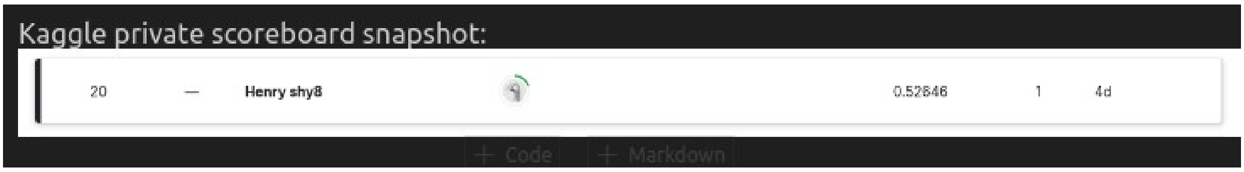
Lab2_Report

Name: 朱冠瑜

Student ID: 113062642



- **Preprocessing & Feature engineering Steps**
  - When creating the DataFrame, I found that some tweet_id entries do not have corresponding emotion labels. Since the number is small, I chose to remove these missing value data.

```
# delete column emotion row that is NaN
train_df = train_df.dropna(subset=['emotion'])
```

  - There are many mention symbols @ in the tweets, but these symbols and the mentioned users are unrelated to the emotion classification task. Therefore, I also removed this noise.

```
    text = re.sub(r'@[A-Za-z0-9_]+', '', text)  # remove @mentions
```

  - Using the emoji library, I converted emojis into text descriptions, allowing the computer to understand their meanings.
  - Social media often contains many hashtag symbols, which I believe do not contribute to the emotion classification task. Therefore, I decided to remove these symbols. However, the

text following the hashtags often includes emotional words and special terms that could be helpful for the task, so I chose to retain this portion of the text.

- Similarly, the data contains many URLs and HTML formats (such as <LH>, https), which I consider to be noise. Therefore, I removed these elements as well.

```
text = re.sub(r'#(\w+)', r'\1', text)       # 保留hashtag内容
text = re.sub(r'<[^>]+>', 'balanced_train_df', text)       # remove HTML
text = re.sub(r'http\S+|www\S+', '', text)  # remove URLs
```

- I also attempted to enhance the dataset by adding features such as the number of emojis, the proportion of uppercase letters, and the tweet length. This was aimed at improving the accuracy of classification predictions. The reason for selecting these features is that text with more emojis or uppercase letters often conveys a stronger, more intense tone or emotion. For tweet length, I hypothesized that tweets expressing anger might be shorter due to the brevity of expression driven by frustration, while trust-related tweets might require more words to explain reasoning, and anticipation-related tweets might include detailed descriptions, resulting in longer tweets.

```
def process_dataframe(df):
    train_df = df[df['split'] == 'train'].copy()
    test_df = df[df['split'] == 'test'].copy()

    # delete column emotion row that is NaN
    train_df = train_df.dropna(subset=['emotion'])

    # add emotion feature
    train_df = add_emotion_features(train_df)
    test_df = add_emotion_features(test_df)
```

```
def add_emotion_features(df):
    df['text_length'] = df['text'].apply(len)
    df['emoji_count'] = df['text'].apply(lambda x: len([c for c in str(x) if c ir
    df['caps_ratio'] = df['text'].apply(lambda x: sum(1 for c in str(x) if c.isur
```

- Additionally, I observed the distribution of emotion labels and found that the number of joy labels is significantly higher than that of other labels (a total of 516,017, accounting for approximately 35.45%)

```
各情緒類別的數量：
emotion
joy             516017
anticipation    248935
trust           205478
sadness         193437
disgust         139101
fear             63999
surprise         48729
anger            39867
Name: count, dtype: int64

各情緒類別的百分比：
emotion
joy             35.45
anticipation    17.10
trust           14.12
sadness         13.29
disgust          9.56
fear             4.40
surprise         3.35
anger            2.74
Name: count, dtype: float64 %

總資料筆數: 1455563
```

- This indicates that the dataset is highly imbalanced. To address this issue, I chose to randomly select 200,000 samples with the joy label as training data. This helps prevent the model from being biased toward the features of joy, which could otherwise result in poor prediction performance for other emotion categories (e.g., misclassifying minority categories as joy).The adjusted proportions are as follows, It can be seen that the adjusted proportions are relatively balanced, which is more favorable for training.

```
各情緒類別的數量：
emotion
anticipation    248935
trust           205478
joy             200000
sadness         193437
disgust         139101
fear             63999
surprise         48729
anger            39867
Name: count, dtype: int64

各情緒類別的百分比：
emotion
anticipation    21.85
trust           18.03
joy             17.55
sadness         16.97
disgust         12.21
fear             5.62
surprise         4.28
anger            3.50
Name: count, dtype: float64 %

總資料筆數: 1139546
```

- **Explanation of my model**
  - Through the lessons and the Lab2 Master assignment, it became clear that using large language models enables a more precise understanding of the context in the text, thereby improving classification tasks. Additionally, since this Kaggle task involves determining the emotion in tweet data, it differs from general classification tasks due to the abundance of abbreviations, misspellings, shorthand, and emojis on Twitter.
  - To address these challenges, I chose a fine-tuned BERT-based model called BERTweet, which was trained on a large dataset

of approximately 850 million tweets (Huggingdace : https://huggingface.co/docs/transformers/model_doc/bertweet). BERTweet specifically emphasizes learning these linguistic features, enabling the model to better understand and handle the unique syntax of social media text. It is particularly suited for non-standard language found on Twitter, such as slang, typos, abbreviations, and emotional expressions (e.g., emojis), which traditional BERT models might struggle to capture effectively.

- Since a portion of the data(joy data) was removed, to avoid overfitting, I further split 20% of the training data as validation data. Additionally, I used a Learning Rate Scheduler to dynamically adjust the learning rate, aiming to achieve better training performance.

```
train, val_df = train_test_split(train_df, test_size=0.2, random_state=42, strat
print(f"Training samples: {len(train)}, Validation samples: {len(val_df)}")
```

```
warmup_steps = len(train_loader) * 0.1
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=warmup_steps,
    num_training_steps=len(train_loader) * num_epochs
)
```

- Below is the performance of each epoch during the training process:

```
Training samples: 911636, Validation samples: 227910

Epoch 1/4
Training batch: 1700/1781
Validation:
Validation batch: 400/446
Epoch 1 Results:
Average train loss: 1.1757
Average val loss: 1.0020
Validation accuracy: 0.6336
Macro F1: 0.5871
Weighted F1: 0.6292

Saved best model


Epoch 2/4
Training batch: 1700/1781
Validation:
Validation batch: 400/446
Epoch 2 Results:
Average train loss: 0.9645
Average val loss: 0.9663
Validation accuracy: 0.6479
Macro F1: 0.6010
Weighted F1: 0.6435

Saved best model


Epoch 3/4
Training batch: 1700/1781
Validation:
Validation batch: 400/446
Epoch 3 Results:
Average train loss: 0.9024
Average val loss: 0.9589
Validation accuracy: 0.6523
Macro F1: 0.6087
Weighted F1: 0.6501

Saved best model


Epoch 4/4
Training batch: 1700/1781
Validation:
Validation batch: 400/446
Epoch 4 Results:
Average train loss: 0.8643
Average val loss: 0.9592
Validation accuracy: 0.6539
Macro F1: 0.6106
Weighted F1: 0.6515
```

- **Experience I gained**
    - During the process of training the model, I clearly observed the challenges of data imbalance, gaining a deeper understanding that in real-world applications, data is rarely perfectly balanced—imbalance is actually the norm.
    - Additionally, many parameters (e.g., batch size, epochs, learning rate) need to be configured, and determining the best combination for a specific dataset and model often relies entirely on trial and error. Early in the training process, I encountered an issue where setting the batch size too large resulted in insufficient GPU memory, highlighting how heavily model training depends on hardware resources. It also requires a significant amount of time. Moving forward, I hope to explore more diverse approaches when training models.