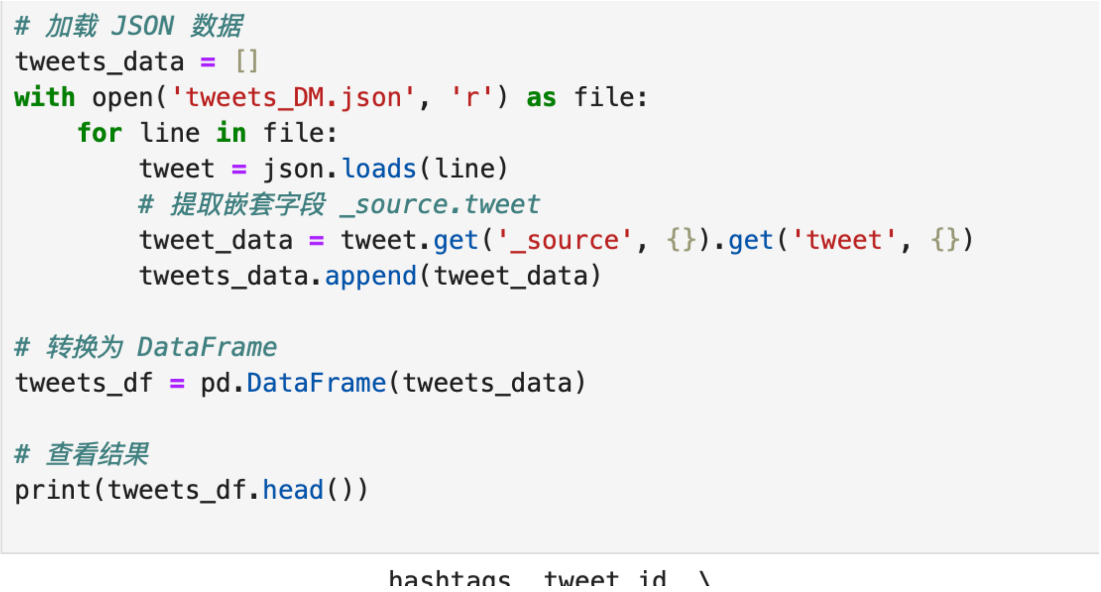
To complete the competition task, I first performed data preprocessing, including extracting tweets, emotion labels, and dataset identifiers from JSON and CSV files, and merging them into a complete dataset. Then, I used regular expressions to clean the tweet texts, removing unnecessary URLs while retaining words and hashtags (# and @). In the feature engineering stage, I applied the Bag-of-Words (BOW) model for text feature extraction and experimented with various feature counts (ranging from 200 to 3000) to find the optimal parameters. Additionally, I tried using TF-IDF for feature selection, testing different feature ranges, and ultimately achieved better results with TF-IDF.

For model selection, I experimented with Decision Trees, Random Forests, and Logistic Regression models and optimized their parameters, but the results were not satisfactory. Subsequently, I attempted a Neural Network (NN) model, which led to significant improvements. I further explored various parameter configurations for the NN model to achieve optimal performance. This competition experience taught me a lot about data processing, feature engineering, and model optimization, and provided me with valuable practical insights.

How to read json



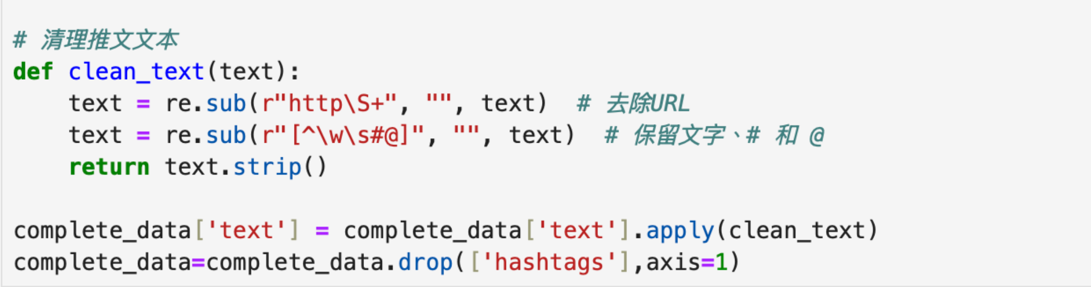
```
# 加载 JSON 数据
tweets_data = []
with open('tweets_DM.json', 'r') as file:
    for line in file:
        tweet = json.loads(line)
        # 提取嵌套字段 _source.tweet
        tweet_data = tweet.get('_source', {}).get('tweet', {})
        tweets_data.append(tweet_data)

# 转换为 DataFrame
tweets_df = pd.DataFrame(tweets_data)

# 查看结果
print(tweets_df.head())
```

```
                    hashtags   tweet_id  \
```

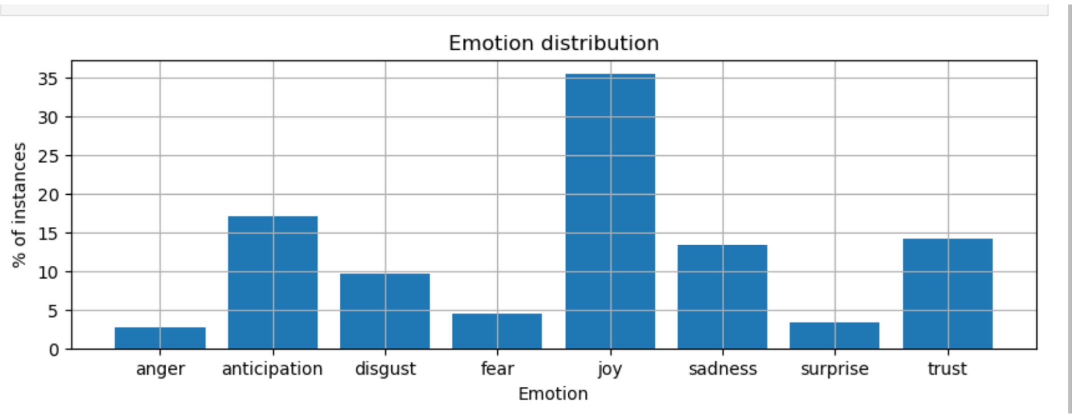
clean the tweet texts



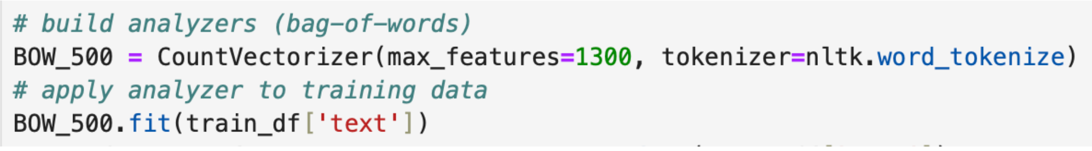
```
# 清理推文文本
def clean_text(text):
    text = re.sub(r"http\S+", "", text)  # 去除URL
    text = re.sub(r"[^\w\s#@]", "", text)  # 保留文字、# 和 @
    return text.strip()

complete_data['text'] = complete_data['text'].apply(clean_text)
complete_data=complete_data.drop(['hashtags'],axis=1)
```

see distribution



try differernt bow



```
# build analyzers (bag-of-words)
BOW_500 = CountVectorizer(max_features=1300, tokenizer=nltk.word_tokenize)
# apply analyzer to training data
BOW_500.fit(train_df['text'])
```

try tfidf

```
import numpy as np

# 建立 TF-IDF Vectorizer
TFIDF = TfidfVectorizer(max_features=2500, tokenizer=nltk.word_tokenize)

# 套用 TF-IDF 分析器到訓練資料
TFIDF.fit(train_df['text'])
X_train = TFIDF.transform(train_df['text'])
y_train = train_df['emotion']

X_test = TFIDF.transform(test_df['text'])
y_test = test_df['emotion']
```

try different modle

```
# 定义模型列表
models = {
    "DecisionTree": DecisionTreeClassifier(random_state=1),
    "RandomForest": RandomForestClassifier(random_state=1, n_estimators=100),
    "LogisticRegression": LogisticRegression(random_state=1)
}
```

try NN

```
# 假设 X_train 是特征矩阵，y_train 是 one-hot 编码的标签
input_dim = X_train.shape[1]  # 输入特征数量
output_dim = y_train.shape[1]  # 输出类别数量

model = Sequential([
    Dense(512, input_dim=input_dim, activation='leaky_relu'),  # 第一隐藏层
    Dropout(0.4),                                              # 防止过拟合
    Dense(256, activation='leaky_relu'),                       # 第二隐藏层
    Dropout(0.4),
    Dense(128, input_dim=input_dim, activation='leaky_relu'),
    Dropout(0.4),
    Dense(64, input_dim=input_dim, activation='leaky_relu'),
    Dropout(0.4),
    Dense(output_dim, activation='softmax')                    # 输出层
])

# 编译模型
model.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
```