Problem Statement

This weeks problem was to parse through a dataset of tweets and determine which of the tweets in the data were about real disasters and which are not. Kaggle submissions will be evaluated using F1 between the predicted and expected answers.

The provided starter notebook was used as a skeleton to connect and get started with this project (but not for the actual model building and training).

```
# Connect to Kaggle
import kagglehub
kagglehub.login()
```

```
401 Client Error.

access resource at URL: https://www.kaggle.com/api/v1/hello. The server reported the follo

sure you are authenticated if you are trying to access a private resource or a resource requ
```

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

# nlp_getting_started_path = kagglehub.competition_download('nlp-getting-started')

# print('Data source import complete.')
```

Exploratory Data Analysis

First install/import the relevant packages

```
import numpy as np
import pandas as pd
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
```

Import the test and training datasets provided by Kaggle

```
from google.colab import files
uploaded = files.upload()
file_names = list(uploaded.keys())
print("Uploaded files:", file_names)
```

```
Choose Files 2 files
- test.csv(text/csv) - 420783 bytes, last modified: 4/14/2025 - 100% done
- train.csv(text/csv) - 987712 bytes, last modified: 4/14/2025 - 100% done
Saving test.csv to test.csv
Saving train.csv to train.csv
```

Exploratory Data Analysis

The training data consists of 7,613 observations and the test data consists of 3,263 observations to make predictions on. The data includesd the following fields:

- id: a unique key for each tweet (separate from the pandas-designated id)
- keyword: a keyword from the tweet text, but may be blank
- location: location the tweet was sent from, but may also be blank

- text: the actual text of the tweet
- target: 1 if the tweet referenced an actual disaster, and 0 otherwise. This is obviously absent from the test data set.

There is a note on Kaggle's website that all the tweets are in English so no translation will be necessary.

```
df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")

print('Training Set Shape = {}'.format(df_train.shape))
print('Test Set Shape = {}'.format(df_test.shape))
```

```
Training Set Shape = (7613, 5)
Test Set Shape = (3263, 4)
```

```
df_train.head()
```

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

Next steps: Generate code with df_train | View recommended plots | New interactive sheet

```
df_test.head()
```

	id	keyword	location	text
0	0	NaN	NaN	Just happened a terrible car crash
1	2	NaN	NaN	Heard about #earthquake is different cities, s...
2	3	NaN	NaN	there is a forest fire at spot pond, geese are...
3	9	NaN	NaN	Apocalypse lighting. #Spokane #wildfires
4	11	NaN	NaN	Typhoon Soudelor kills 28 in China and Taiwan

Next steps: Generate code with df_test | View recommended plots | New interactive sheet

Continuing the EDA, it would be good to see how many of the tweets in the test and training data have missing values for the keyword and location fields as well as how long, on average, each tweet was.

Of the 7,613 tweets in the training data set, only 61 were missing keywords, (approx 0.8%) and 2,533 were missing location data (approx 33%). Of the 3,263 tweets in the test data, only 26 were missing keywords (approx 0.8%) and 1,105 were missing location data (approx 34%). Proportionally the training and test data sets are equivalent in terms of the missing values.

```
# check the average length of the tweet
train_word_count  = df_train['text'].apply(lambda x: len(x.split()))
test_word_count = df_test['text'].apply(lambda x: len(x.split()))

print(f"Average # of words per tweet in training data: {train_word_count.mean()}")
print(f"Average # of words per tweet in test data: {test_word_count.mean()}")
```

```
Average # of words per tweet in training data: 14.903585971364771
Average # of words per tweet in test data: 14.965369292062519
```

```
# check to see how many tweets are missing keywords and location values
df_train.info()
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        7613 non-null   int64
 1   keyword   7552 non-null   object
```

```
 2   location  5080 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64
dtypes: int64(2), object(3)
memory usage: 297.5+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        3263 non-null   int64
 1   keyword   3237 non-null   object
 2   location  2158 non-null   object
 3   text      3263 non-null   object
dtypes: int64(1), object(3)
memory usage: 102.1+ KB
```

Another statistic that may prove to be relevant is the number of tweets that include hashtags in the tweets themselves, which may be indicative of a reference to an actual disaster and significant for the model. A summary look at the head of the data frames indicated this may be the case.

```
# how many tweets contain hashtags - could they possibly be referencing actual events as seen from the initial inspection above?
hashtag_train = df_train['text'].str.contains('#').sum()
hashtag_test = df_test['text'].str.contains('#').sum()
print(f"Hashtags in Training Set: {hashtag_train}")
print(f"Hashtags in Test Set: {hashtag_test}")
```

```
Hashtags in Training Set: 1761
Hashtags in Test Set: 808
```

Model Building and Training

For this project I elected to use a RoBERTa (Robustly Optimized BERT Pretraining Approach) Transformer model rather than a traditional BERT model as it promises improved performance while still being efficient and relatively easy to use. It has been traied on a larger and more diverse dataset, trained for a longer duration with larger batch sizes, etc. when compared to a traditional BERT model. Documentation shows RoBERTa can outperform BERT in tasks like text classification.

```
# load the pretrained RoBERTa model and tokenizer and configure for binary classification
model_name = "roberta-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

# tokenize and prepare the data for the model
def preprocess_function(examples):
    return tokenizer(examples['text'], truncation=True, padding='max_length', max_length=128)

# split the training dataframe to train and test sets
train_texts, val_texts, train_target, val_target = train_test_split(
    df_train['text'].tolist(), df_train['target'].tolist(), test_size=0.2, random_state=13)

train_dataset = preprocess_function({'text': train_texts})
val_dataset = preprocess_function({'text': val_texts})

# convert to pytorch datasets
class DisasterDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        if self.labels:
            item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])

train_dataset = DisasterDataset(train_dataset, train_target)
val_dataset = DisasterDataset(val_dataset, val_target)

test_encodings = preprocess_function({'text': df_test['text'].tolist()})
test_dataset = DisasterDataset(test_encodings)
```

```
# train the model
def compute_metrics(p):
    predictions, labels = p
    predictions = np.argmax(predictions, axis=1)
    accuracy = accuracy_score(labels, predictions)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, predictions, average='binary')
    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1
    }

training_args = TrainingArguments(
    output_dir='./results',
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model='f1',
    logging_dir='./logs',
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)

trainer.train()
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  25.0/25.0 [00:00<00:00, 2.97kB/s]
config.json: 100%  481/481 [00:00<00:00, 58.4kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 4.35MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 3.29MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 5.09MB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfo
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
model.safetensors: 100%  499M/499M [00:06<00:00, 119MB/s]
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newly initialize
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
[1143/1143 4:53:08, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	No log	0.406218	0.822718	0.819048	0.767857	0.792627
2	0.435100	0.440589	0.829941	0.848229	0.748512	0.795257
3	0.303500	0.480566	0.831911	0.829114	0.779762	0.803681

```
TrainOutput(global_step=1143, training_loss=0.3564928176626237, metrics={'train_runtime': 17603.9304, 'train_samples_per_second':
1.038, 'train_steps_per_second': 0.065, 'total_flos': 1201759745356800.0, 'train_loss': 0.3564928176626237, 'epoch': 3.0})
```

Finally, create the predictions and save to a file for Kaggle submission.

```
predictions = trainer.predict(test_dataset)
probabilities = torch.nn.functional.softmax(torch.tensor(predictions.predictions), dim=-1).numpy()
predicted_classes = np.argmax(probabilities, axis=1)
```

```
# add the predictions to the provided test df
df_test['target'] = predicted_classes

# export the test df for submission
df_test.to_csv('submission_file.csv', index=False)
```