```
%%capture
!pip install transformers
```

```
import tensorflow as tf
from tensorflow.python.client import device_lib
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

from transformers import BertTokenizer
from transformers import TFBertForSequenceClassification

import pandas as pd
import numpy as np
import plotly.figure_factory as ff
import plotly.express as px
import plotly.graph_objects as go

import re
import nltk
from nltk.corpus import stopwords
from google.colab import drive
```

```
print(device_lib.list_local_devices())

if tf.config.list_physical_devices('GPU'):
    tf.config.experimental.set_memory_growth(tf.config.list_physical_devices('GPU')[0], True)
    tf.config.set_visible_devices(tf.config.list_physical_devices('GPU')[0], 'GPU')
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 6917018627078311131
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14328594432
locality {
  bus_id: 1
  links {
  }
}
incarnation: 11862668901081598525
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5"
xla_global_id: 416903419
]
```

```
drive.mount('/content/drive')
nltk.download('stopwords')
```

```
Mounted at /content/drive
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
df = pd.read_csv('/content/drive/MyDrive/Datasets/Playstore_Reviews/reviews.csv')
```

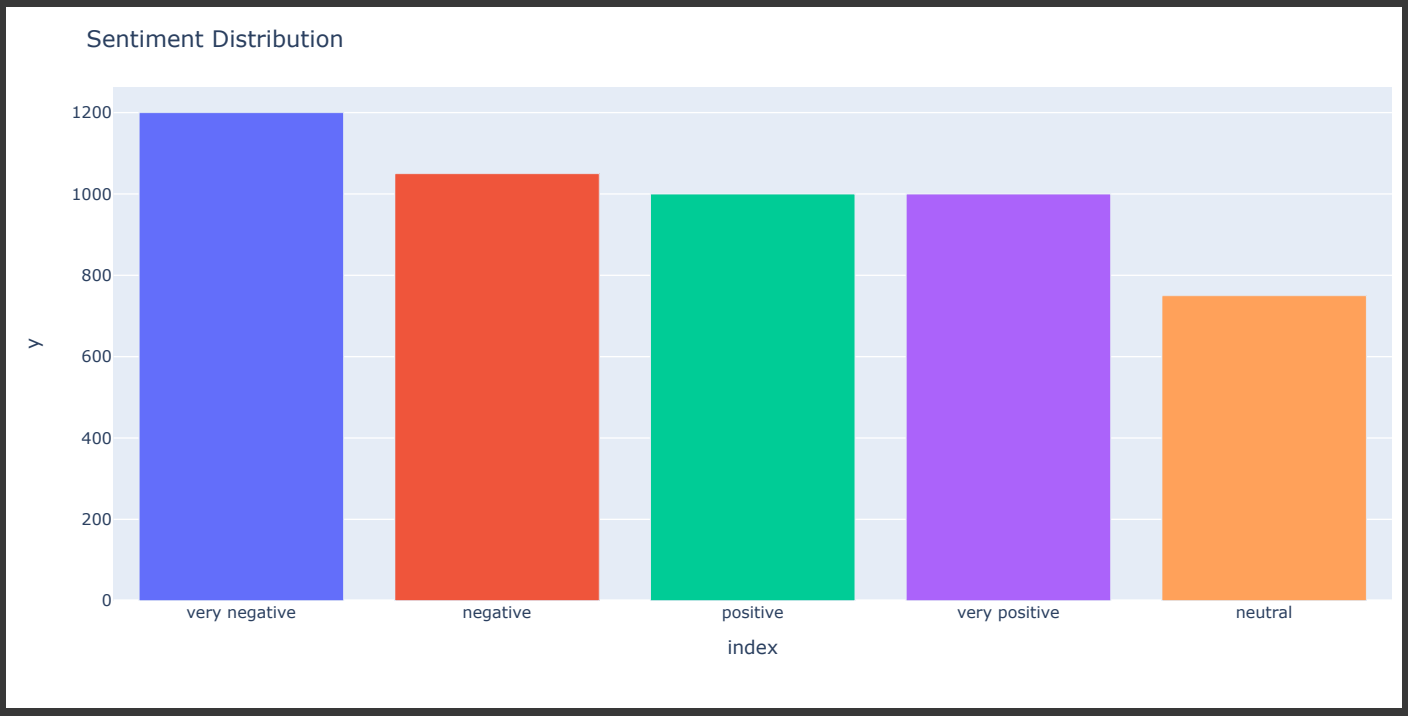
```
df = df.head(5000)
```

```
def preprocess_text(text):
    # text = text.lower()
    # text = re.sub(r"http\S+|www\S+|https\S+", "", text, flags=re.MULTILINE)
    # text = re.sub(r"[^\w\s]", "", text)
    # text = re.sub(r"\d+", "", text)
    # stop_words = set(stopwords.words('english'))
    # text = " ".join([word for word in text.split() if word not in stop_words])

    return text
```

```
def assign_5_types(score):
    if score == 1:
        return 'very negative'
    elif score == 2:
        return 'negative'
    elif score == 3:
        return 'neutral'
    elif score == 4:
        return 'positive'
    else:
        return 'very positive'
```

```
X = df['content'].apply(preprocess_text)
y = df['score'].apply(assign_5_types)
```

```
sentiment_counts = y.value_counts()
fig = px.bar(sentiment_counts, x=sentiment_counts.index, y=sentiment_counts.values, color=sentiment_counts.index)
fig.update_layout(title="Sentiment Distribution")
fig.show()
```




```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

X_validation, X_train = X_train[:300], X_train[300:]
y_validation, y_train = y_train[:300], y_train[300:]

X_train.shape, X_validation.shape, X_test.shape, y_train.shape, y_validation.shape, y_test.shape
```

```
((3700,), (300,), (1000,), (3700,), (300,), (1000,))
```

```
with tf.device('GPU'):
    bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    bert_model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=5)
```

```
All PyTorch model weights were used when initializing TFBertForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not initialized from the PyTorch model and a
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
def create_input_tensors(input_X, tokenizer):
    input_ids = []
    attention_masks = []

    for text in input_X:
        tokens = tokenize_text(text, tokenizer)
```

```
        input_ids.append(tokens['input_ids'][0])
        attention_masks.append(tokens['attention_mask'][0])

    input_ids = np.array(input_ids)
    attention_masks = np.array(attention_masks)

    return {'input_ids': input_ids, 'attention_mask': attention_masks}


def tokenize_text(text, tokenizer):
    tokens = tokenizer.encode_plus(text,
                                   max_length=128,
                                   truncation=True,
                                   padding='max_length',
                                   add_special_tokens=True,
                                   return_attention_mask=True,
                                   return_tensors='tf')
    return tokens


def convert_labels_to_one_hot(labels, num_classes):
    label_mapping = {'very positive': 4,
                     'positive': 3,
                     'neutral': 2,
                     'negative': 1,
                     'very negative': 0}
    labels = [label_mapping[label] for label in labels]

    return tf.keras.utils.to_categorical(labels, num_classes=num_classes)
```

```
new_train_X = create_input_tensors(X_train, bert_tokenizer)
new_train_y = convert_labels_to_one_hot(y_train, 5)

new_validation_X = create_input_tensors(X_validation, bert_tokenizer)
new_validation_y = convert_labels_to_one_hot(y_validation, 5)

new_test_X = create_input_tensors(X_test, bert_tokenizer)
new_test_y = convert_labels_to_one_hot(y_test, 5)
```

```
with tf.device('GPU'):

  epochs = 5

  bert_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-5),
                loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
                metrics=[tf.keras.metrics.CategoricalAccuracy('accuracy')])

  bert_model_history = bert_model.fit(new_train_X,
                                      new_train_y,
                                      batch_size=32,
                                      epochs=epochs,
                                      validation_data=(new_validation_X, new_validation_y))
```

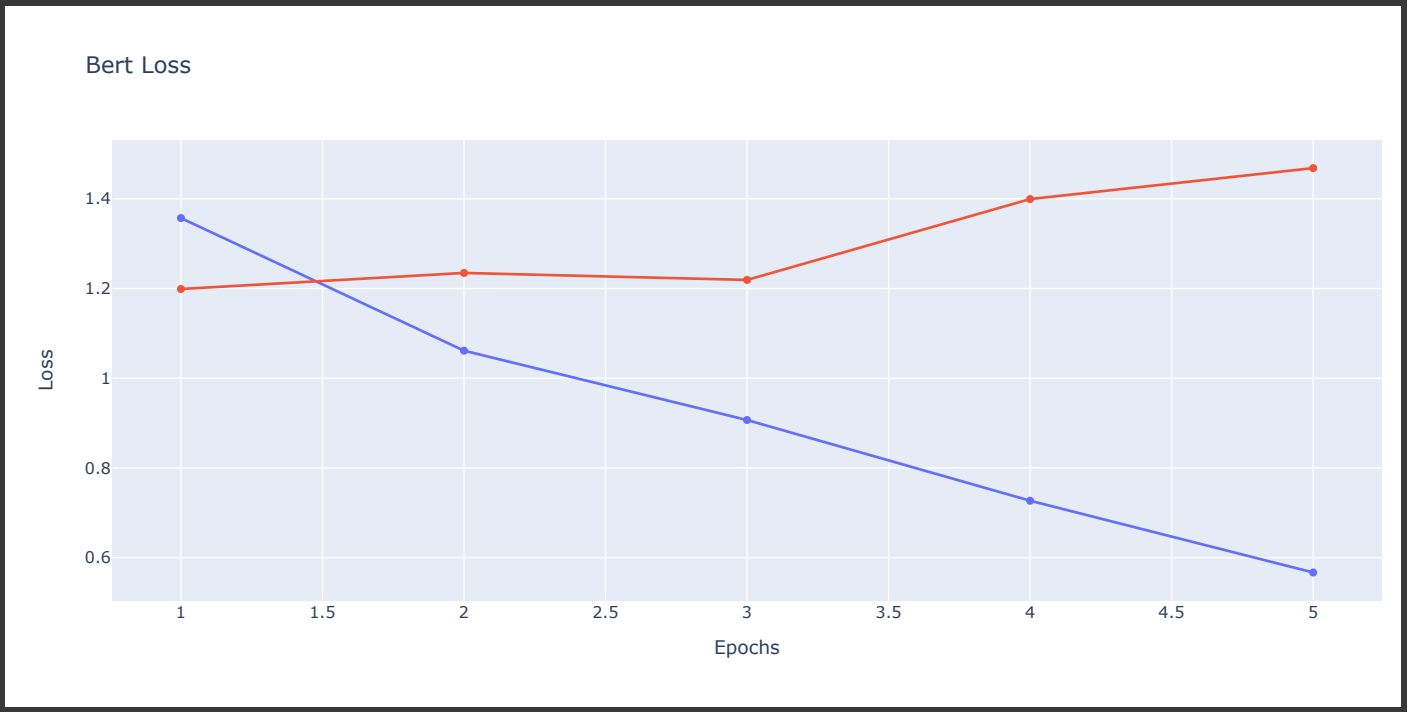
```
Epoch 1/5
116/116 [==============================] - 150s 811ms/step - loss: 1.3570 - accuracy: 0.4057 - val_loss: 1.1988 - val_accurac
Epoch 2/5
116/116 [==============================] - 91s 782ms/step - loss: 1.0613 - accuracy: 0.5484 - val_loss: 1.2346 - val_accuracy
Epoch 3/5
116/116 [==============================] - 90s 775ms/step - loss: 0.9067 - accuracy: 0.6335 - val_loss: 1.2190 - val_accuracy
Epoch 4/5
116/116 [==============================] - 90s 774ms/step - loss: 0.7268 - accuracy: 0.7254 - val_loss: 1.3993 - val_accuracy
Epoch 5/5
116/116 [==============================] - 90s 774ms/step - loss: 0.5669 - accuracy: 0.7951 - val_loss: 1.4683 - val_accuracy
```

```
train_loss_history = bert_model_history.history['loss']
validation_loss_history = bert_model_history.history['val_loss']

train_acc_history = bert_model_history.history['accuracy']
validation_acc_history = bert_model_history.history['val_accuracy']
```

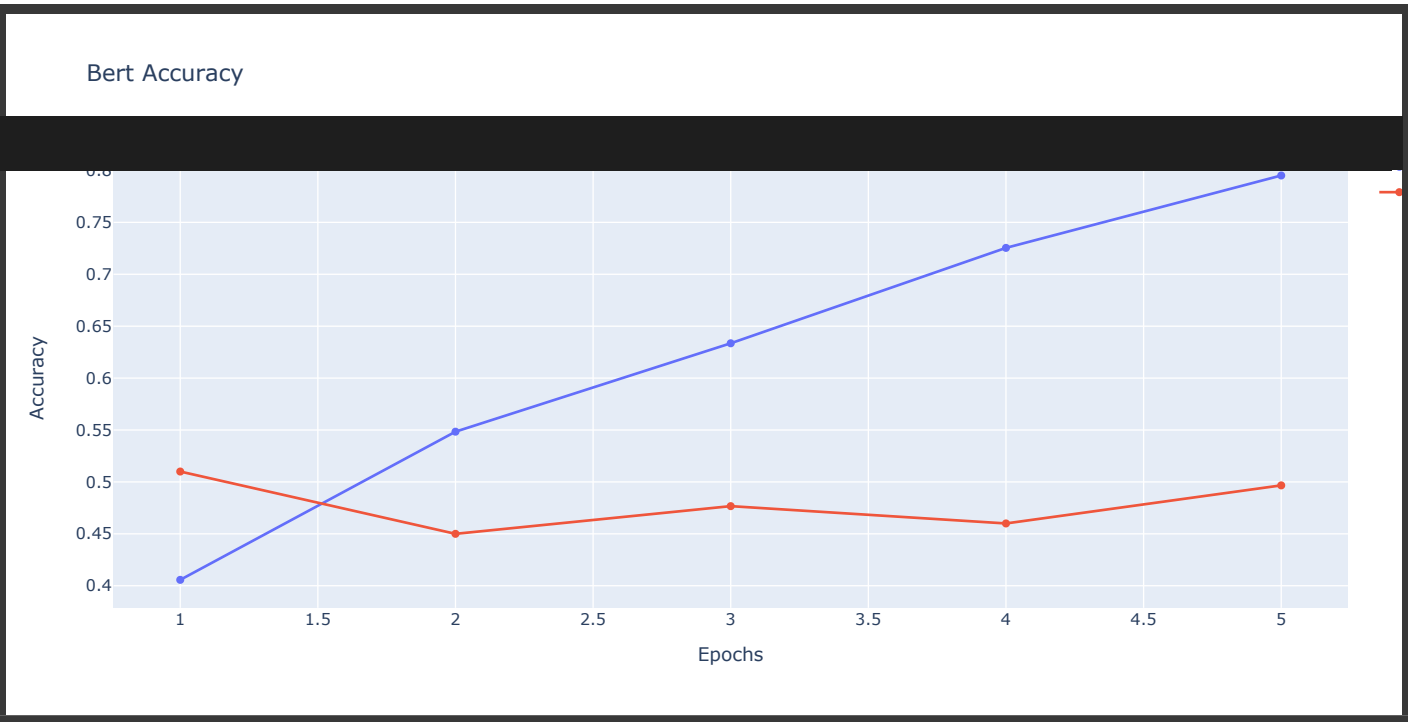
```
bert_model.save('/content/drive/MyDrive/Datasets/Playstore_Reviews/bert_model_5')
```

```
WARNING:absl:Found untraced functions such as embeddings_layer_call_fn, embeddings_layer_call_and_return_conditional_losses,
```

```
fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=train_loss_history, mode='lines+markers', name='Train Loss'))
fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=validation_loss_history, mode='lines+markers', name='Validation Lo

fig.update_layout(title="Bert Loss",
                  xaxis_title="Epochs",
                  yaxis_title="Loss")
fig.show()
```




```
fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=train_acc_history, mode='lines+markers', name='Train Accuracy'))
fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=validation_acc_history, mode='lines+markers', name='Validation Acc

fig.update_layout(title="Bert Accuracy",
                  xaxis_title="Epochs",
                  yaxis_title="Accuracy")
fig.show()
```


Bert Accuracy
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
Accuracy
1
1.5
2
2.5
3
3.5
4
4.5
5
Epochs