```
%%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 sklearn.metrics import classification_report

from transformers import BertTokenizer
from transformers import BertTokenizerFast
from transformers import TFBertForSequenceClassification
from transformers import DistilBertTokenizerFast, TFDistilBertForSequenceClassification

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: 8439228498734052733
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14328594432
locality {
  bus_id: 1
  links {
  }
}
incarnation: 18142777843843554931
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
```

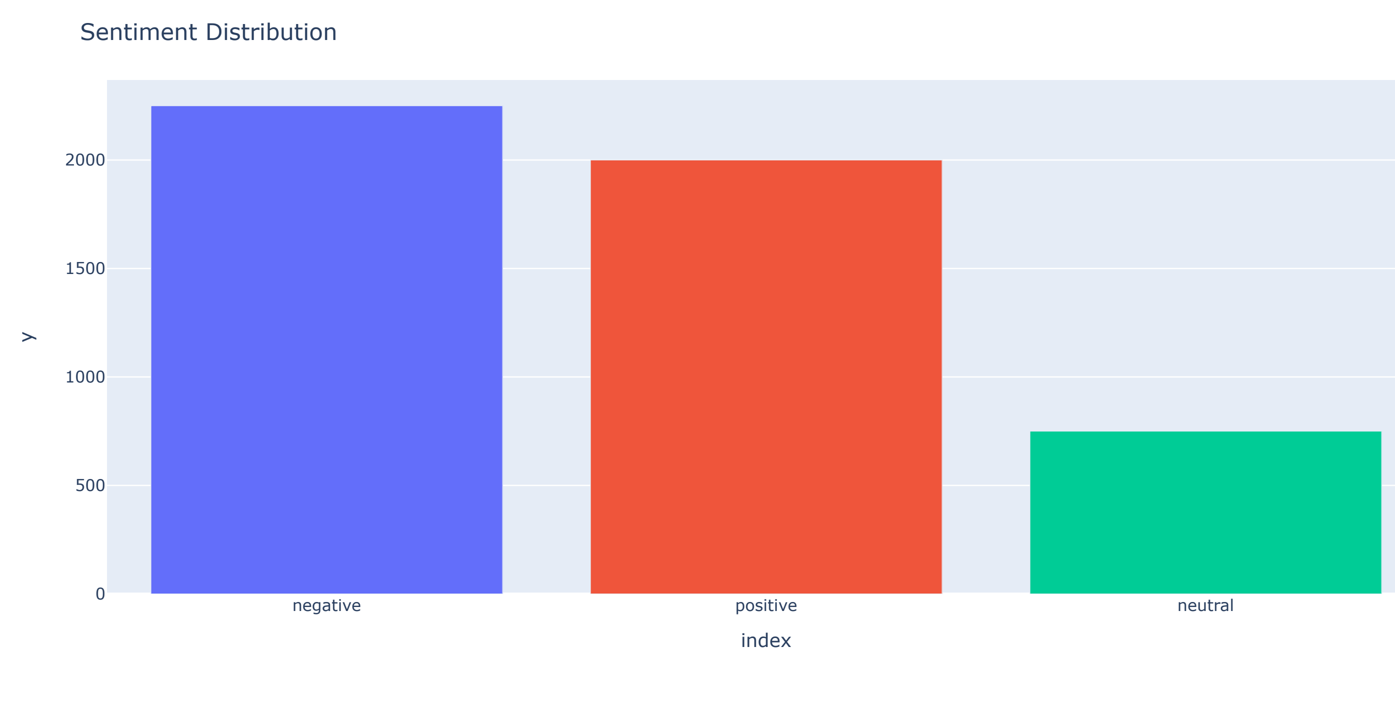
```
df = pd.read_csv('/content/drive/MyDrive/Datasets/Playstore_Reviews/reviews.csv')
```

```
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_labels_3(score):
    if score <= 2:
        return 'negative'
    elif score >= 4:
        return 'positive'
    else:
        return 'neutral'
```

```
num_labels = 3

X = df['content'].apply(preprocess_text)
y = df['score'].apply(assign_labels_3)
```

```
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)
```

```
validation_num = 500

X_validation, X_train = X_train[:validation_num], X_train[validation_num:]
y_validation, y_train = y_train[:validation_num], y_train[validation_num:]
```

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

```
((3500,), (500,), (1000,), (3500,), (500,), (1000,))
```

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

# with tf.device('GPU'):
#     bert_tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
#     bert_model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=num_labels
```

```
Downloading (…)solve/main/vocab.txt: 100%  232k/232k [00:00<00:00, 1.09MB/s]

Downloading (…)okenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 1.20kB/s]

Downloading (…)lve/main/config.json: 100%  570/570 [00:00<00:00, 21.7kB/s]

Downloading model.safetensors: 100%  440M/440M [00:02<00:00, 152MB/s]
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 = {'positive': 2,
                     'negative': 0,
                     'neutral': 1}
    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, num_labels)

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

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

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

  epochs = 3
  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))
```

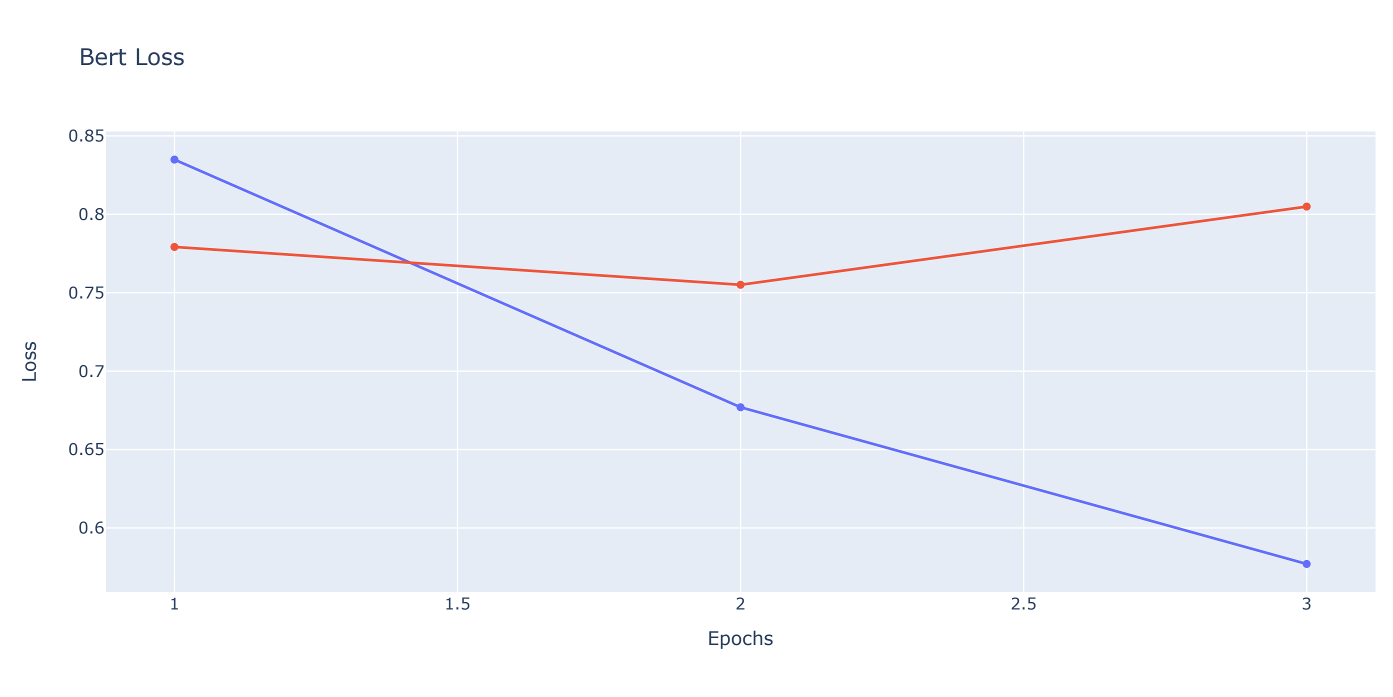
```
    Epoch 1/3
    110/110 [==============================] - 159s 922ms/step - loss: 0.8349 - accuracy: 0.6671 - val_loss: 0.7792 - val_accurac
    Epoch 2/3
    110/110 [==============================] - 90s 815ms/step - loss: 0.6770 - accuracy: 0.7454 - val_loss: 0.7551 - val_accuracy
    Epoch 3/3
    110/110 [==============================] - 88s 804ms/step - loss: 0.5770 - accuracy: 0.7869 - val_loss: 0.8049 - 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']
```

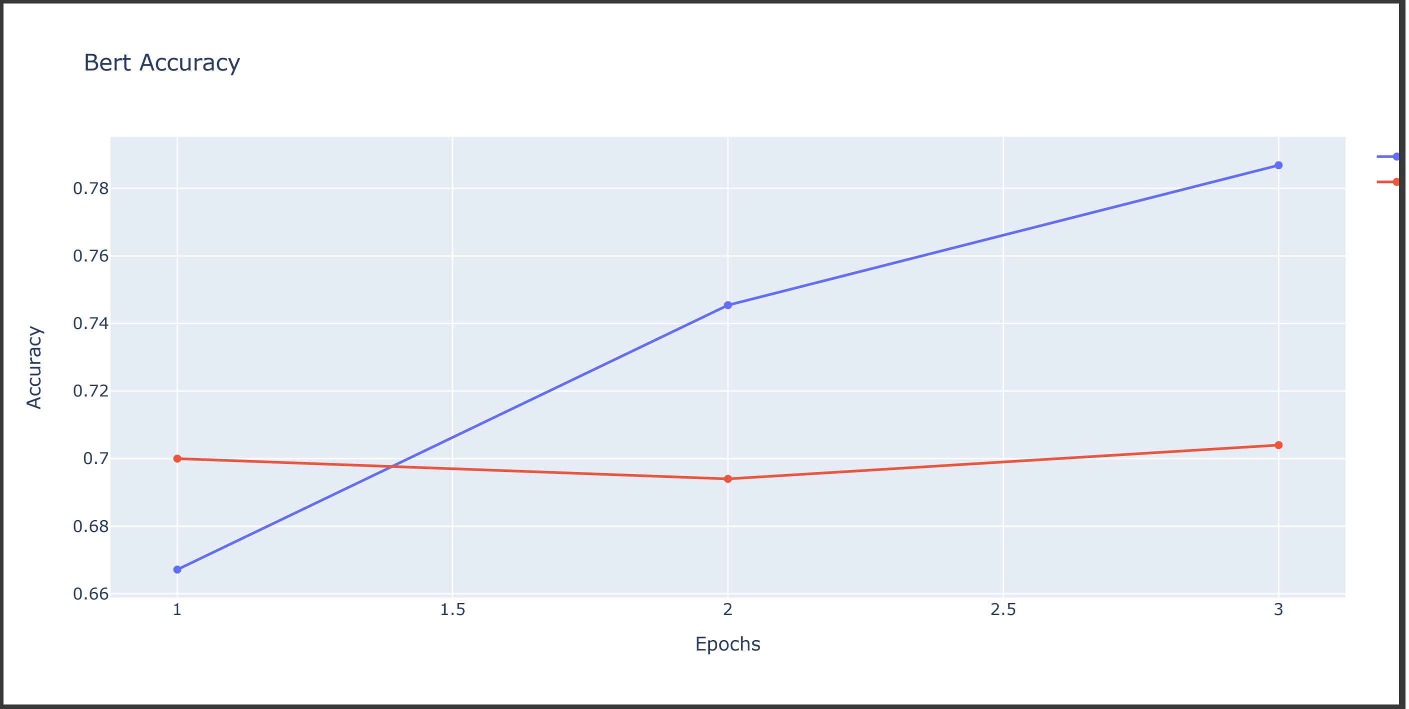
```
bert_model.save('/content/drive/MyDrive/Datasets/Playstore_Reviews/bert_model_3')
```

```
    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()
```



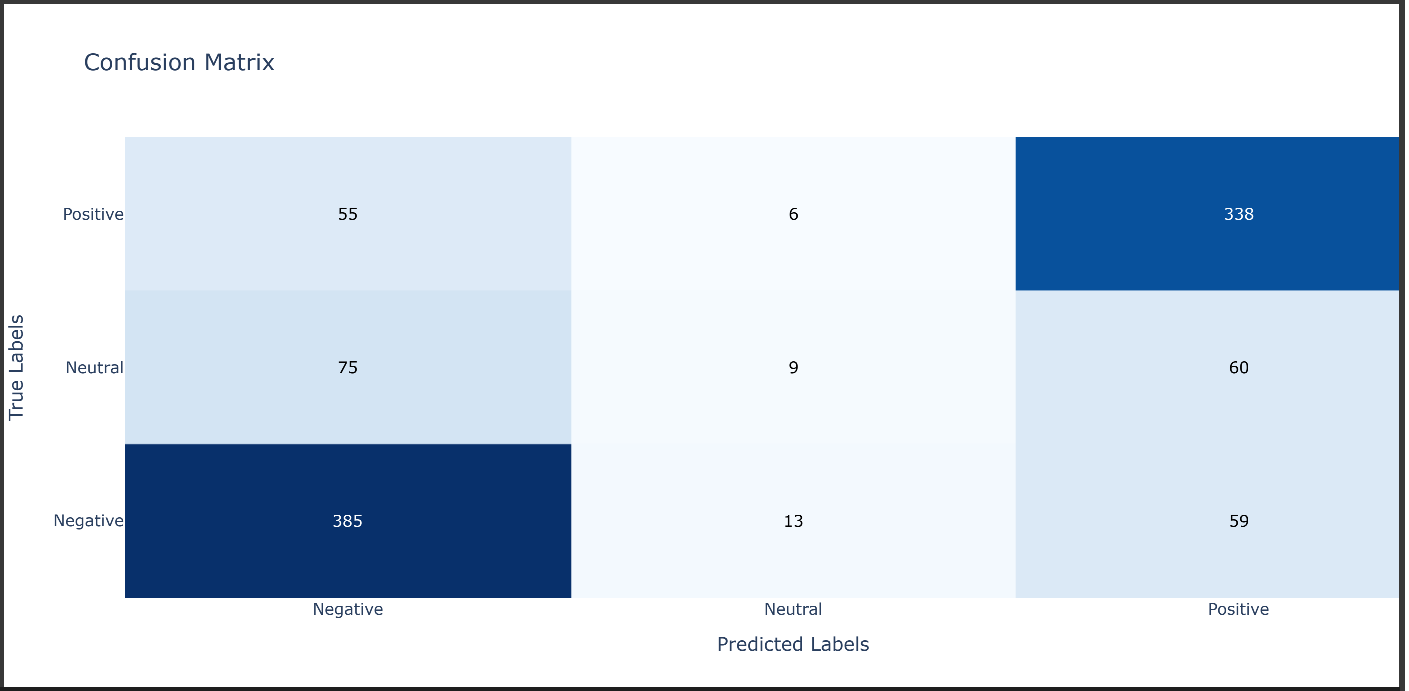
```
test_predictions = bert_model.predict(new_test_X)

predicted_labels = np.argmax(test_predictions.logits, axis=1)

test_labels_1d = np.argmax(new_test_y, axis=1)
```

```
    32/32 [==============================] - 12s 269ms/step
```

```
print(classification_report(test_labels_1d, predicted_labels))
```

```
              precision    recall  f1-score   support

           0       0.75      0.84      0.79       457
           1       0.32      0.06      0.10       144
           2       0.74      0.85      0.79       399

    accuracy                           0.73      1000
   macro avg       0.60      0.58      0.56      1000
weighted avg       0.68      0.73      0.69      1000
```

```
confusion_matrix = confusion_matrix(test_labels_1d, predicted_labels)

fig = go.Figure(data=go.Heatmap(z=confusion_matrix,
                               x=['Negative', 'Neutral', 'Positive'],
                               y=['Negative', 'Neutral', 'Positive'],
                               colorscale="Blues",
                               text=confusion_matrix))

for i in range(len(confusion_matrix)):
    for j in range(len(confusion_matrix[0])):
        fig.add_annotation(x=j, y=i, text=str(confusion_matrix[i][j]),
                           showarrow=False,
                           font=dict(color="white" if confusion_matrix[i][j] > np.max(confusion_matrix) / 2 else "black")

fig.update_layout(title="Confusion Matrix",
                  xaxis_title="Predicted Labels",
                  yaxis_title="True Labels")

fig.show()
```



	Negative	Neutral	Positive
Positive	55	6	338
Neutral	75	9	60
Negative	385	13	59