LOADING FILES

In [1]:

```
!pip3 install plotly

import pandas as pd
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
from keras.preprocessing.sequence import pad_sequences
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import tensorflow as tf
# tf.compat.v1.enable_eager_execution()
from tensorflow import keras
from tensorflow.keras.layers import *
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import UpSampling2D
from tensorflow.keras.layers import MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.layers import Multiply
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNo
rmalization, Flatten, Conv2D, AveragePooling2D, MaxPooling2D, GlobalMaxPooling2D
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.initializers import glorot_uniform
K.set_image_data_format('channels_last')
K.set_learning_phase(1)
import pickle

from tqdm import tqdm
import os
import tensorflow as tf
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (4.4.1)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from plotly
) (1.15.0)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-packages
(from plotly) (1.3.3)
```

In [2]:

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

LOADING PICKLE FILES FOR PREDICTION

In [13]:

```
#opening vocab files and embeddings
import pickle

open_file = open('/content/drive/MyDrive/CASE STUDY 2/modi_word_files.pkl', "rb")
loaded_list = pickle.load(open_file)
open_file.close()

token_corr=loaded_list[2]
token_eng=loaded_list[3]
vocab_size_eng=loaded_list[4]
vocab_size_corr=loaded_list[5]
```

LOADING STORED BEST MODEL

In [14]:

```
# Defining Custom Loss Function
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True, reductio
n = 'none')

@tf.function
def loss_function(real, pred):
    # Custom loss function that will not consider the loss for padded zeros.
    # Refer https://www.tensorflow.org/tutorials/text/nmt_with_attention
    # optimizer = tf.keras.optimizers.Adam()
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)
    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask
    return tf.reduce_mean(loss_)

# Loading trained model
best_model = tf.keras.models.load_model('/content/drive/MyDrive/CASE STUDY 2/best_model_g
eneral', custom_objects={"loss_function": loss_function})
```

DEFINING PREDICT FUNCTION

In [61]:

```
def function_1(input_sentence):
  ''' This fucntion wil take a raw text input and using the prediction, it will return pr
edicted model'''

  model=best_model
  UNITS=200

  # Tokenizing and Padding the sentence
  inputs = [token_corr.word_index.get(i, 0) for i in input_sentence]
  inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs], maxlen = 38, padding
= 'post')
  inputs = tf.convert_to_tensor(inputs)


  # Initializing result string and hidden states
  result = ''
  hidden = tf.zeros([1, UNITS]), tf.zeros([1, UNITS])

  # Getting Encoder outputs
  enc_out, state_h, state_c = model.encoder([inputs, hidden])
```

```
  dec_hidden = [state_h, state_c]
  dec_input = tf.expand_dims([token_eng.word_index['<start>']], 0)


  # Running loop until max length or the prediction is '>' token
  for t in range(38):
      # Getting Decoder outputs fot timestep t
      output, state_h, state_c = model.decoder.timestepdecoder([dec_input, enc_out, stat
e_h, state_c])
      # Getting token index having highest probability
      predicted_id = tf.argmax(output[0]).numpy()
      # Getting output token
      if token_eng.index_word.get(predicted_id, '') == '<end>':
          break
      else:
          result +=token_eng.index_word.get(predicted_id, '')+' '
          dec_input = tf.expand_dims([predicted_id], 0)
  # Postprocessing the result string to remove spaces between punctuations

  return result
```

DEFINING SCORING FUNCTION

In [62]:

```
def function_2(input_text,corrected):
  ''' This funciton will return the bleu score of actual and predictext text, actual text
will be one of the input'''

  score = nltk.translate.bleu_score.sentence_bleu([corrected.split(' ')],function_1(input
_text).split(' '))
  return score
```

VALIDATING THE RESULTS

In [63]:

```
input='hw r u'
predicted=function_1(input)
print('The Predicted output of:',input,'is----->',predicted)
```

```
The Predicted output of: hw r u is-----> how are you
```

In [64]:

```
print('The Bleu Score of the:',input,'is---------->',function_2(input,'How are you ?'))
```

```
The Bleu Score of the: hw r u is----------> 0.6389431042462724
```