```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from keras.layers import Dense , LSTM , Embedding , Dropout , Activation , Flatten, RNN, G
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.preprocessing import sequence
from tensorflow.keras.utils import to_categorical
from keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import RMSprop, Adam
import tensorflow as tf
%matplotlib inline
```

```
! unzip smsspamcollection.zip
```

```
df = pd.read_csv('SMSSpamCollection',sep ='\t',names=["label", "message"])
```

```
X = df['message']
Y = df['label']
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)
print('Y shape: ', Y.shape)
print('Y unique values: ', np.unique(Y))
```

```
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=0.3, stratify = Y)

# ytrain = to_categorical(ytrain)
# ytest = to_categorical(ytest)

# max
# print('Train data len:'+str(len(xtrain)))
# print('Class distribution'+str(Counter(ytrain)))


# print('Valid data len:'+str(len(xvalid)))
# print('Class distribution'+ str(Counter(yvalid)))

# print('Test data len:'+str(len(xtest)))
# print('Class distribution'+ str(Counter(ytest)))

# # print('Test data len:'+str(len(test_data['text'].tolist())))
# # print('Class distribution'+ str(Counter(test_data['label'].tolist())))


# train_dat =list(zip(ytrain,xtrain))
```

```
# valid_dat =list(zip(yvalid,xvalid))
```

```
max_words = 1000
max_len  =150
tok = Tokenizer(num_words= max_words)
tok.fit_on_texts(xtrain)
sequences = tok.texts_to_sequences(xtrain)
sequences_matrix = sequence.pad_sequences(sequences,maxlen=max_len)
print('sequences[2] length: ', len(sequences[2]))
print('sequences length: ', len(sequences))
```

```
def RNN():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,8,input_length=max_len)(inputs)
    # Embedding (input_dim: size of vocabolary,
    # output_dim: dimension of dense embedding,
    # input_length: length of input sequence)
    layer = SimpleRNN(32)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model

def LSTM1():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,8,input_length=max_len)(inputs)
    # Embedding (input_dim: size of vocabolary,
    # output_dim: dimension of dense embedding,
    # input_length: length of input sequence)
    layer = LSTM(32)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model


def GRU1():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,8,input_length=max_len)(inputs)
    # Embedding (input_dim: size of vocabolary,
    # output_dim: dimension of dense embedding,
    # input_length: length of input sequence)
    layer = GRU(32)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model


model = RNN()
model.summary()
model.compile(loss='binary_crossentropy',metrics=['accuracy'])

model2 = LSTM1()
```

```
model2.summary()
model2.compile(loss='binary_crossentropy',metrics=['accuracy'])


model3 = GRU1()
model3.summary()
model3.compile(loss='binary_crossentropy',metrics=['accuracy'])

history = model.fit(sequences_matrix,ytrain,batch_size=64,epochs=5,validation_split=0.2)
history2 = model2.fit(sequences_matrix,ytrain,batch_size=64,epochs=5,validation_split=0.2)
history3= model3.fit(sequences_matrix,ytrain,batch_size=64,epochs=5,validation_split=0.2)


test_sequences = tok.texts_to_sequences(xtest)
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)


test_sequences = tok.texts_to_sequences(df_large['message'])
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)
ytest = df_large['label']


accr = model.evaluate(test_sequences_matrix,ytest)
accr2 = model2.evaluate(test_sequences_matrix,ytest)
accr3 = model3.evaluate(test_sequences_matrix,ytest)


print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr2[0],accr2[1]))
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr3[0],accr3[1]))


ypred = model2.predict(test_sequences_matrix)
ypred = tf.math.round(ypred)


import matplotlib.pyplot as plt
cf_matrix = confusion_matrix(ytest, ypred)
print(cf_matrix)
from sklearn.metrics import precision_recall_fscore_support, classification_report
print(precision_recall_fscore_support(ytest, ypred, average='binary'))
print(classification_report(ytest, ypred))


import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics

mat = metrics.confusion_matrix(ytest, ypred)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")


xtest
```

```
test = xtest

df = test.to_frame(name="message")


df["message_len"]=df["message"].str.count(' ') + 1
df['label'] = ytest


df.head(5)


import seaborn as sns
import matplotlib.pyplot as plt

fig=plt.figure(figsize=(12,8))
sns.histplot(
    x=df["message_len"]
)
plt.title("Histogram for count of messages with message length")
plt.savefig("Histogram.png")
plt.show()



df_small = df[df['message_len'] <= 8]
df_small.shape


df_medium= df[(df['message_len'] >8) & (df['message_len'] < 18)]
df_medium.shape


df_large = df[df['message_len'] > 18]
df_large.shape
```

Question 2

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip


embeddings_index = {}
f = open('glove.6B.300d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s glove word vectors.' % len(embeddings_index))


max_word = 1000
max_len = 300
```

```
token = Tokenizer(num_words = max_word)
token.fit_on_texts(xtrain)
sequences = token.texts_to_sequences(xtrain)
seq_matrix = sequence.pad_sequences(sequences , maxlen = max_len)

sequences = token.texts_to_sequences(xtest)
seq_matrix_test = sequence.pad_sequences(sequences , maxlen = max_len)
word_index = token.word_index
len(word_index)



import os
path_to_glove_file = "glove.6B.100d.txt"

embeddings_index = {}

with open(path_to_glove_file) as f:
  for line in f:
    word, coefs = line.split(maxsplit = 1)
    coefs = np.fromstring(coefs, "f", sep = " ")
    embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))


embedding_matrix_glove = np.zeros((len(word_index) + 1, max_len))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_glove[i] = embedding_vector


!wget --no-check-certificate \
     https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.vec.zip \
     -O fasttext.1M.zip



! unzip fasttext.1M.zip


embeddings_index = {}
f = open('wiki-news-300d-1M.vec')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s fasttext word vectors.' % len(embeddings_index))


embedding_matrix_fasttext = np.zeros((len(word_index) + 1, max_len))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
```

```
        if embedding_vector is not None:
            embedding_matrix_fasttext[i] = embedding_vector



BATCH = 128
EPOCHS = 10


from keras.models import Sequential
def build_model(model_name = 'RNN', embeddings = 'Glove'):
  model = Sequential()
  if embeddings == 'Glove':
    model.add(Embedding(input_dim=len(word_index) + 1,
                            output_dim=max_len,
                            weights=[embedding_matrix_glove],
                            input_length=max_len,
                            trainable=False))
  else:
    model.add(Embedding(input_dim=len(word_index) + 1,
                            output_dim=max_len,
                            weights=[embedding_matrix_fasttext],
                            input_length=max_len,
                            trainable=False))
  if model_name == 'LSTM':
    model.add(LSTM(64))
  elif model_name == 'GRU':
    model.add(GRU(64))

  model.add(Flatten())

  model.add(Dense(250, activation='relu'))
  model.add(Dropout(0.5))

  model.add(Dense(120, activation='relu'))

  model.add(Dense(1, activation='sigmoid'))

  return model


import keras

from keras import backend as K
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision
```

```
def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

  glove = build_model('GRU', 'Glove')
  glove.compile(loss = 'binary_crossentropy' , optimizer = Adam(0.001) , metrics= ['accura
  es = EarlyStopping(monitor='val_loss', mode = 'min', patience = 2, verbose=1 , restore_b
  glove.fit(seq_matrix, ytrain, epochs = EPOCHS, batch_size = BATCH, validation_split=0.2,



  fast = build_model('GRU', 'Fasttext')
  fast.compile(loss = 'binary_crossentropy' , optimizer = Adam(0.001) , metrics= ['accurac
  es = EarlyStopping(monitor='val_loss', mode = 'min', patience = 2, verbose=1 , restore_b
  fast.fit(seq_matrix, ytrain, epochs = EPOCHS, batch_size = BATCH, validation_split=0.2,



from tensorflow import keras
metricsList= []
models =[]
models.append(glove)
models.append(fast)
for m in models:
  metrics = m.evaluate(seq_matrix_test, ytest, verbose=0)
  metricsList.append([metrics[2], metrics[3], metrics[4]])
  print(f"score of model {m} : f1 - {metrics[2]}, precision - {metrics[3]}, recall - {metr


pd.DataFrame(data = metricsList , columns=['F1', 'Precision', 'Recall'], index = ['Glove',


from sklearn.metrics import confusion_matrix
cms = []
for m in models:
  preds = m.predict(seq_matrix_test, batch_size = BATCH)
  preds = preds.round()
  cm=confusion_matrix(ytest,preds)
  cms.append(cm)


## Glove
ax = plt.axes()
sns.heatmap(cms[0], annot=True,fmt='.4g')
ax.set_title('Confusion Matrix for Test Data using Glove Embeddings')
plt.savefig("GloveE.png")
plt.show()


## Fasttext
ax = plt.axes()
sns.heatmap(cms[1], annot=True,fmt='.4g')
ax.set_title('Confusion Matrix for Test Data using FastText Embeddings')
plt.savefig("fastText.png")
plt.show()
```