Configuration Manual

MSc Research Project
Data Analytics

Vrushali Bhanudas Darade
Student ID: x21123764

School of Computing
National College of Ireland

Supervisor: Athanasios Staikopoulos

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vrushali Bhanudas Darade
Student ID:	x21123764
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Athanasios Staikopoulos
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual
Word Count:	463
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Vrushali Bhanudas Darade
Date:	31st January 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vrushali Bhanudas Darade
x21123764

1 Overview

This is the research project manual for "Deep Learning and Natural Language Processing for Suicidal Ideation Using Instagram Posts." This will be a step-by-step guide for setting up the environment, pre-requests, and running the code.

2 Hardware/Software Requirements

2.1 Hardware Requirements

The hardware configuration of the system on which this research project is build and executed are as follow:

- Operating System: Windows 10 Home Single Language, version - 21H2.
- Processor: 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 2.61 GHz
- Storage: 458 GB
- RAM: 16.0 GB

2.2 Software Requirements

Software's required for build and execution:

- Integrated Development Environment: Google Colab
- Scripting Language: Python 3.7
- Cloud Storage: Google Drive
- Other Tool: Notepad ++, overleaf, Excel

3 Setting up environment

3.1 Google Colab

The setup begins with Google Colabs. First go to the official website of Google Colab, followed by enabling the GPU for processing.

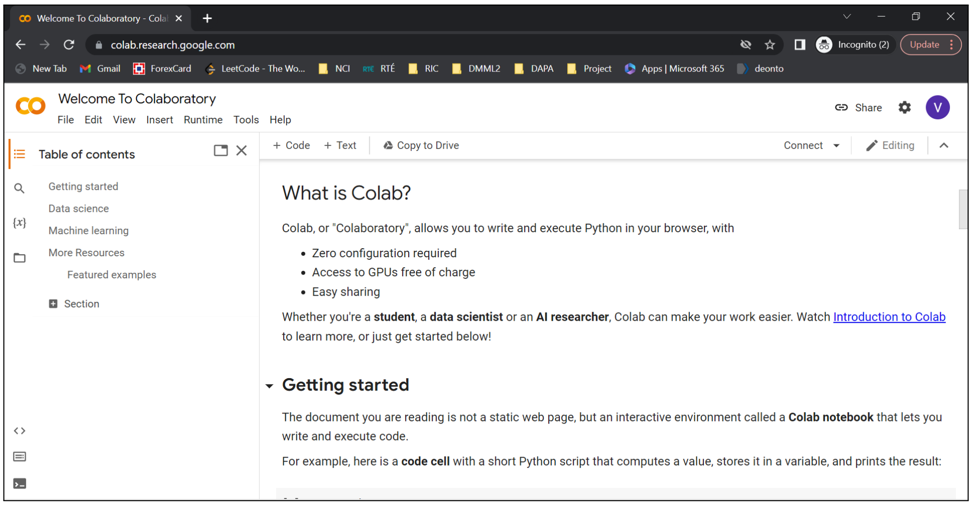

Figure 1: Google Colab

4 Data Selection

The data used in this study was obtained from Kaggle's dataset repository. This dataset contains over 20k Instagram images along with a.csv file containing the caption data.

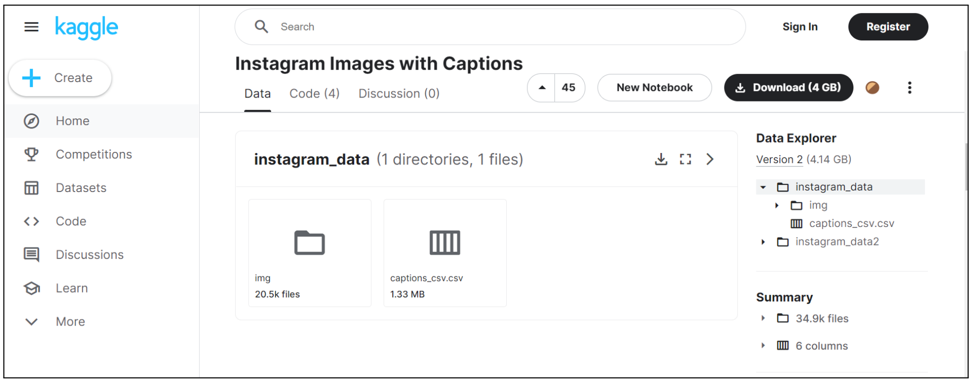

Figure 2: Google Colab

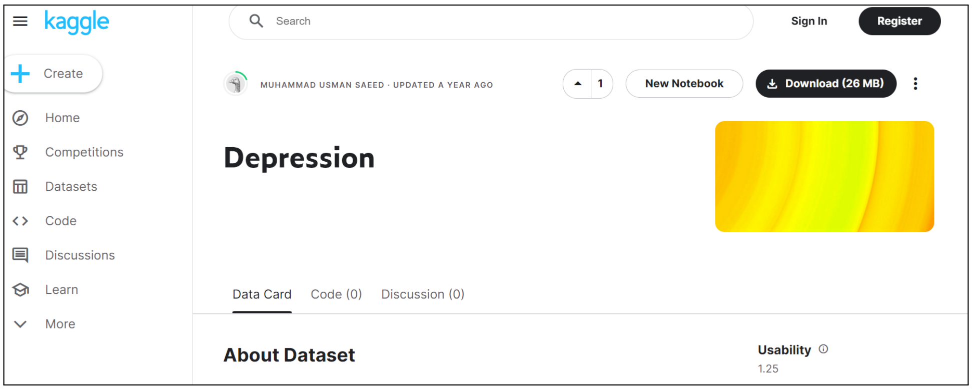

Figure 3: Google Colab

5 Data transformation and Model Building

5.1 Upload data on Google Drive:

Data is directly uploaded to Google Drive using the python package provided by Kaggle, as shown in the Figure 4 below.

Figure 4: Data upload on Google drive

5.2 Package installations and library importing

This research applied use of the following libraries for data pre-processing, model building, and evaluation:

- Ftfy
- NTTK
- genism.models
- DeepFace
- CV2

- TensorFlow
- Keras
- Sklearn.metrics

```python
from google.colab import drive
import tensorflow as tf
#from deepface import DeepFace
#import face_recognition
import cv2
import matplotlib.pyplot as plt
import os
from os import listdir
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import splitfolders
```

Figure 5: Imported Libraries and packages

```python
import numpy as np # Array and mathematical operations
import pandas as pd # working with datasets
import re
from math import exp
from numpy import sign
import random
import matplotlib.pyplot as plt # Visualisation

import ftfy
import nltk # text processing
nltk.download('punkt')
nltk.download('stopwords')
from nltk import word_tokenize,sent_tokenize
from nltk.corpus import stopwords # removing unwanted stop words
from nltk import PorterStemmer # used for convering word to its root format by removing prefix or suffix

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

from sklearn.metrics import  classification_report, confusion_matrix, accuracy_score #model evaluation
from gensim.models import KeyedVectors # used for text processing, word2vec

from keras_preprocessing.sequence import pad_sequences # Ensures all the captions have same length
from keras.models import Model, Sequential #Im porting models
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import Conv1D, Dense, Input, LSTM, Embedding, Dropout, Activation, MaxPooling1D, SpatialDropout1D
from keras.preprocessing.text import Tokenizer # Converting string into substring or words
from keras.utils import plot_model
from keras.metrics import categorical_accuracy
```

Figure 6: Imported Libraries and packages

```
# ftfy package is used to handle back unicode
!pip install ftfy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ftfy
  Downloading ftfy-6.1.1-py3-none-any.whl (53 kB)
     |████████████████████████████████| 53 kB 1.4 MB/s
Requirement already satisfied: wcwidth>=0.2.5 in /usr/local/lib/python3.8/dist-packages (from ftfy) (0.2.5)
Installing collected packages: ftfy
Successfully installed ftfy-6.1.1
```

Figure 7: : ftfy package downloading and installation

5.3 Google drive and Google Colab connection:

Google Colab is linked to Google drive for data access, shown in the below Figure 8:

```
# Google drive mount for accessing data file
from google.colab import drive

#Connecting to Google Drive
drive.mount('/content/drive')
```

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

Figure 8: Google drive mount

5.4 Data read, transform and splitting:

5.4.1 Text Data

The data is read into the Dataframe and verified in the first step, as shown in Figure 9.

```
CAPTION_CSV = '/content/drive/MyDrive/colab_data/captions_csv.csv'

insta_caption_df = pd.read_csv(CAPTION_CSV, encoding = "ISO-8859-1", usecols = range(0,3))
insta_caption_df
```

	Sr No	Image File	Caption
0	1	img/insta1	NaN
1	2	img/insta2	bye
2	3	img/insta3	Ok, a few more... sorry I just had so much fun...
3	4	img/insta4	This was one of my favorite shoots lâ□□ve ever...
4	5	img/insta5	Wrapped round my finger like a ring
...	...	...	...
20510	20511	img/insta20511	Cowgirl :)
20511	20512	img/insta20512	<3
20512	20513	img/insta20513	I love me and Kylie's nail polish colors!
20513	20514	img/insta20514	Fammm
20514	20515	img/insta20515	Disneyland!

20515 rows × 3 columns

Figure 9: Caption Data file

Following that, writen custom functions for cleaning up the text data using various Python libraries and methods, as shown in the Figure 10 below.

```
# Replace contractions with normal words
def replaceContractions(text, c_re=c_re):
    def replace(match):
        return Contractions_words[match.group(0)]
    return c_re.sub(replace, text)
```

```
[20] # Remove various extra items from the caption such as emojis, hashtag, special characters and so on
def caption_cleanup(captions):
    caption_cleanup = []
    for caption in captions:
        caption = str(caption)
        # if url links then dont append to avoid news articles
        # To focus on depression word, check for length > 10
        if re.match("(\w+:\/\/\S+)", caption) == None and len(caption) > 10:
            # removing emojis, hashtags and special characters
            caption = ' '.join(
                re.sub("(@[A-Za-z0-9]+)|(\#[A-Za-z0-9]+)|(<Emoji:.*>)", " ", caption).split())

            # Unicode changes
            caption = ftfy.fix_text(caption)

            # calling replace method for contraction replace
            caption = replaceContractions(caption)

            # handling punctuation
            caption = ' '.join(re.sub("([^0-9A-Za-z \t])", " ", caption).split())

            # Removal of stop words
            stop_words = set(stopwords.words('english'))
            word_tokens = nltk.word_tokenize(caption)
            filtered_sentence = [w for w in word_tokens if not w in stop_words]
            caption = ' '.join(filtered_sentence)

            # Removing stemming words
            caption = PorterStemmer().stem(caption)

            caption_cleanup.append(caption)

    return caption_cleanup
```

Figure 10: Clean up methods

After cleaning up the data, I tokenised the word and converted the list into a 2D array for model building.

```
tokenizer = Tokenizer(num_words=max_words)

# Provides number of caption data available
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(suicide_d + non_suicide_d)

# Assign number to words
sequences_d = tokenizer.texts_to_sequences(suicide_d)
sequences_r = tokenizer.texts_to_sequences(non_suicide_d)

# Assign index to words
word_index = tokenizer.word_index
print('Found %s unique tokens' % len(word_index))

Found 11545 unique tokens

#Converting data into 2D array
data_d = pad_sequences(sequences_d, maxlen=140)
data_r = pad_sequences(sequences_r, maxlen=140)
print('Shape of data_d tensor:', data_d.shape)
print('Shape of data_r tensor:', data_r.shape)

Shape of data_d tensor: (268, 140)
Shape of data_r tensor: (13623, 140)
```

Figure 11: Word Tokenisation

Data is then split for text, train and validation into 60:20:20

```
# Data split into test (60%), validation (20%), and train data (20%)
random.seed(1)

perm_r = np.random.permutation(len(data_r))
perm_d = np.random.permutation(len(data_d))

train_d = perm_d[:int(len(data_d)*(Train_split))]
test_d = perm_d[int(len(data_d)*(Train_split)):int(len(data_d)*(Train_split+Test_split))]
val_d = perm_d[int(len(data_d)*(Train_split+Test_split)):]

train_r = perm_r[:int(len(data_r)*(Train_split))]
test_r = perm_r[int(len(data_r)*(Train_split)):int(len(data_r)*(Train_split+Test_split))]
val_r = perm_r[int(len(data_r)*(Train_split+Test_split)):]

data_train = np.concatenate((data_d[train_d], data_r[train_r]))
labels_train = np.concatenate((labels_d[train_d], labels_r[train_r]))
data_test = np.concatenate((data_d[test_d], data_r[test_r]))
labels_test = np.concatenate((labels_d[test_d], labels_r[test_r]))
data_val = np.concatenate((data_d[val_d], data_r[val_r]))
labels_val = np.concatenate((labels_d[val_d], labels_r[val_r]))
```

Figure 12: Data split into 60:20:20

5.4.2 Image Data

Loading train and test data for model implementation

```
train_path = '/content/drive/MyDrive/colab_data/dataset/train'
valid_path = '/content/drive/MyDrive/colab_data/dataset/train/sad'
```

Figure 13: data load from Google drive

Data Augmentation for data transformation and pre-processing

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

Figure 14: Image data transformation

6 Model Implementation:

6.1 Long-Short term memory:

The Figure 15 below depicts an LSTM model with 128 LSTM layers, an input gate value of 300, a forget gate value of 300, and an output gate value of 140.

```
# Define parameter
n_lstm = 128
emb = len(embedding_matrix)+1

# Define LSTM Model
model1 = Sequential()
model1.add(Embedding(emb, EMBEDDING_DIM, input_length=Max_caption_len))
model1.add(LSTM(n_lstm,return_sequences=False))
model1.add(Dense(1, activation='sigmoid'))
```

Figure 15: LSTM model for Caption Analysis

Finally compiling the model with "binary_crossentropy" as loss and "adam" optimizer.

```
model1.compile(loss = 'binary_crossentropy',optimizer = 'adam',metrics = ('accuracy'))
```

Figure 16: LSTM model compile and model.fit

The LSTM model is enhanced by the addition of CNN for improved performance. Before the LSTM layers, CNN layers with filter 32, activation 'relu', and pool size 2 are added. The Figure 17 below depicts this.

```
model_m2 = Sequential()
# Embedded layer
model_m2.add(Embedding(len(embedding_matrix), EMBEDDING_DIM, weights=[embedding_matrix],input_length=Max_caption_len, trainable=False))
# Convolutional Layer
model_m2.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model_m2.add(MaxPooling1D(pool_size=2))
model_m2.add(Dropout(0.2))

# LSTM Layer
model_m2.add(LSTM(300))
model_m2.add(Dropout(0.2))
model_m2.add(Dense(1, activation='sigmoid'))
```

Figure 17: LSTM with CNN Model

6.2 VGG16:

The below Figure 18 illustrates the implementation of VGG16 model. While implementing the model, last layer of the model is removed.

```
# add preprocessing layer to the front of VGG
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

Figure 18: Pre-trained VGG1 model

```
# our layers - you can add more if you want
x = Flatten()(vgg.output)
prediction = Dense(len(folders_one_hot_enc), activation='sigmoid')(x)

# create a model object
model = Model(inputs=vgg.input, outputs=prediction)
```

Figure 19: VGG1 model

```
# fit the model
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=10,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

Figure 20: VGG16 Model fit

7 Model Evaluation:

The models are then evaluated by comparing train and test accuracy and loss using various graphs. Then, for each evaluation, a separate classifier report is generated to provide a better understanding of accuracy, precision, and recall.

```
# Plot for Model Accuracy
plt.plot(hist_m1.history['accuracy'])
plt.plot(hist_m1.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Figure 21: Model Accuracy Plot

```
# Plot for model Loss
plt.plot(hist_m1.history['loss'])
plt.plot(hist_m1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Figure 22: Model Loss plot

```
labels_pred = model1.predict(data_test)
labels_pred = np.round(labels_pred.flatten())
accuracy = accuracy_score(labels_test, labels_pred)
print("Accuracy: %.2f%%" % (accuracy*100))
```

Figure 23: Model Prediction

```
print(classification_report(labels_test, labels_pred))
```

Figure 24: Model classification

8 Testing:

The implemented model is then tested individually against a text and image dataset to determine the predictability of the developed models. Models that have been saved are reloaded and checked by passing an image to determine whether it is or is not suicidal.

```
model = load_model('/content/drive/MyDrive/colab_data/InstagramSuicidalIdeationImages.h5')
img = cv2.imread("/content/drive/MyDrive/colab_data/face_detect/face/insta46.jpg")
plt.imshow(img)
image_x = 224
image_y = 224
img = cv2.resize(img, (image_x, image_y))
img = np.reshape(img, (1,224,224,3))
pred_probab = model.predict(img)[0]
pred_class = list(pred_probab).index(max(pred_probab))
if (pred_class==0):
  print("Suicidal")
else:
    print("Non Suicidal")
```

Figure 25: Model reload and testing

The classification of suicidal and non-suicidal identification is shown in the images below.

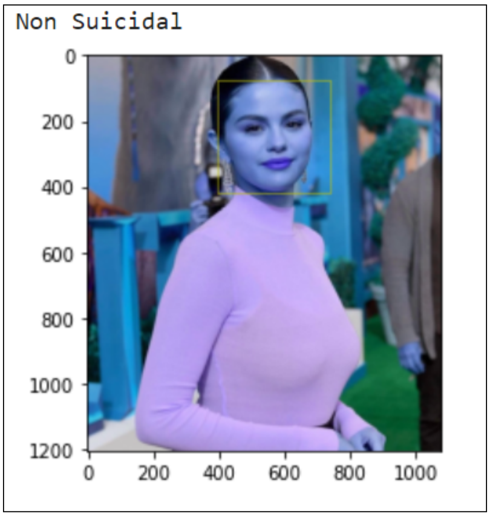

Figure 26: Non-suicidal post

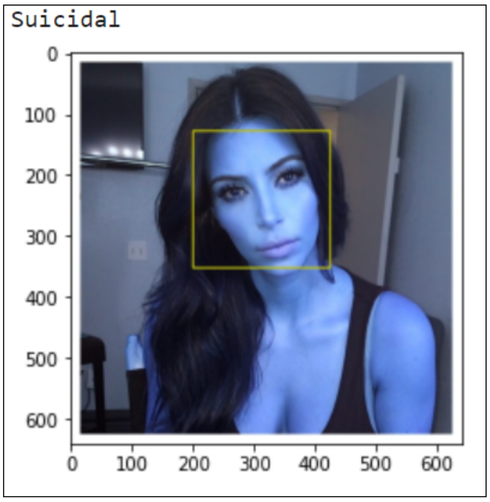

Figure 27: Suicidal post