Configuration Manual

MSc Research Project

MSc Data Analytics

Saif Shuhab Rabbani

Student ID: x21223149

School of Computing

National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Saif Shuhab Rabbani		
Student ID:	21223149		
Programme:	Research Project	**Year:**	2023
Module:	Msc Data Analytics		
Supervisor:	Vladimir Milosavljevic		
Submission Due Date:			
Project Title:	Deep Learning for Enhanced Speech Communication: Integrating Real-time Voice Command Recognition and Emotion Analysis		
Word Count:	1370(Excluding references) **Page Count**: 14		

Signature: Saif Shuhab Rabbani

Date: 14 December 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 the 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

Saif Rabbani
Student ID: 21223149

In an era where seamless and intuitive communication is vital, the convergence of many speech processing domains has arisen as a new path for improving communication technologies. Text-to-speech conversion, speech-to-text transcription, voice command recognition, voice emotion analysis, text classification, and text translation are all covered in this research thesis. When these aspects are combined, they provide a full multimodal speech processing model, laying the framework for a more inclusive and efficient communication ecosystem. The study aims to construct a holistic system surpassing conventional speech processing barriers. It aspires to fuse varied elements like text-to-speech conversion, voice command recognition, emotion analysis in speech, transcription of spoken words to text, categorization of textual data, and language translation. This ambitious concept strives to reshape the human-technology interface, fostering not only seamless communication but also unlocking a myriad of possibilities in realms like education, healthcare, and beyond.

Its robust libraries such as TensorFlow, Keras, and Gradio provided the necessary tools for implementing complex machine learning models seamlessly. The decision to execute the project on Google Colab, a cloud-based platform, was driven by the collaborative nature of the research. Google Colab offered a shared workspace that facilitated real-time collaboration, version control, and the ability to harness the computational power of Google's cloud infrastructure, ensuring scalability and efficiency in model development.

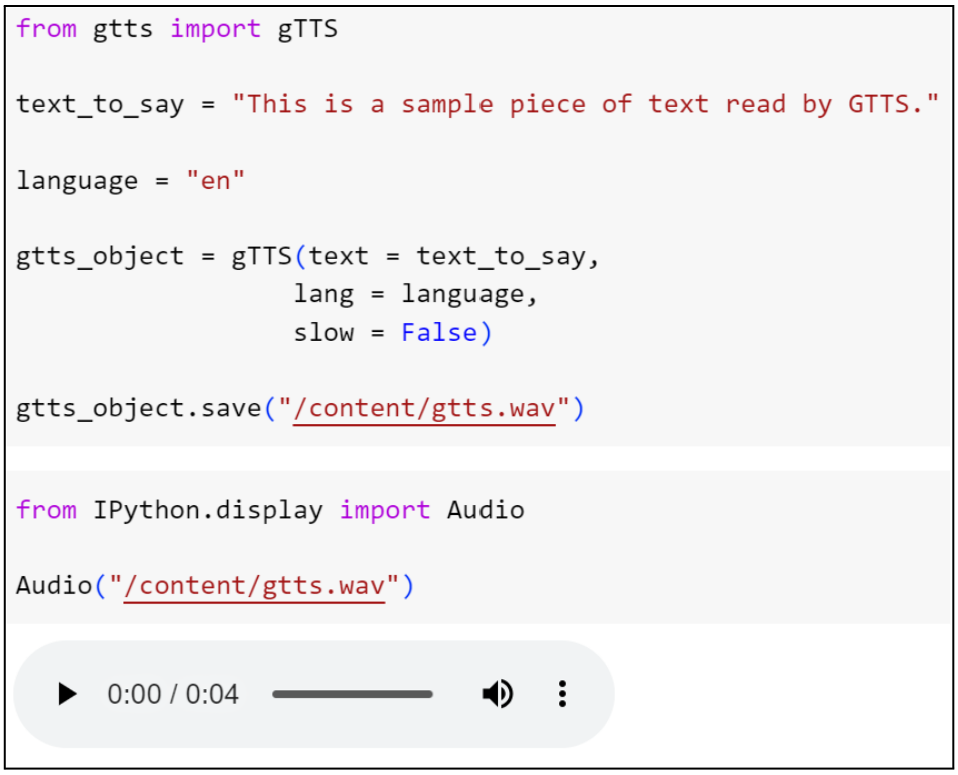
Hardware and Software Requirements

In order to commence this journey into multimodal speech processing, certain hardware and software prerequisites need to be fulfilled:

GPU Acceleration: While not required, GPU acceleration speeds up deep learning model training. It is advised to use a GPU with CUDA support, such as NVIDIA GPUs.
Adequate RAM is required for handling massive datasets and complicated models. It is suggested to have a minimum of 16GB of RAM.

Software Prerequisites:
Python: Install the most recent Python version, ideally Python 3, to take use of the most recent features and libraries.
Google Collaborate Account: Access to Google Colab is required for collaborative development and to make use of Google's cloud services.
Libraries: Install the necessary Python libraries, such as TensorFlow, Keras, Gradio, and others indicated in the manual's relevant parts such as GTTS and Whisper.

1 Text & Speech Processing using Gtts and whisper

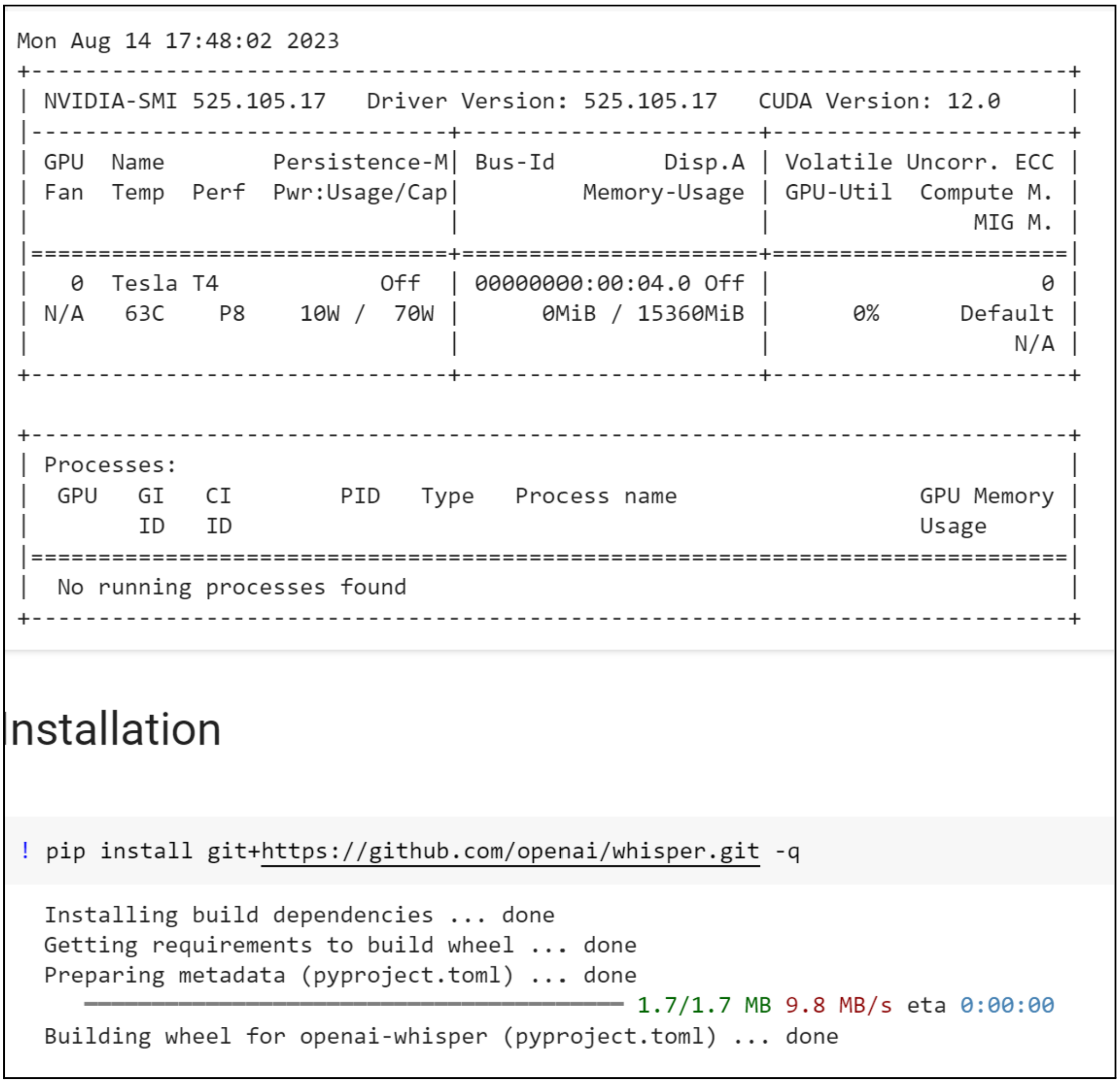


```
from gtts import gTTS

text_to_say = "This is a sample piece of text read by GTTS."

language = "en"

gtts_object = gTTS(text = text_to_say,
                   lang = language,
                   slow = False)

gtts_object.save("/content/gtts.wav")
```

```
from IPython.display import Audio

Audio("/content/gtts.wav")
```

Fig 1. GTTS Requirements

When employing the gTTS library for text-to-speech conversion, it is essential to construct a requirements.txt file, specifying necessary dependencies such as gTTS, requests, and click. The subsequent execution of 'pip install -r requirements.txt' ensures the proper installation of the dependencies. Subsequently, the integration of gTTS into your script facilitates the seamless conversion of both English and French text to speech, enabling the preservation of resulting audio files. Additionally, the utilisation of the IPython.display package further enhances the auditory experience by enabling the playback of the generated audio. By embracing this systematic approach, the setup and execution of your text-to-speech application are uncomplicated, significantly enhancing clarity and repeatability.



```
Mon Aug 14 17:48:02 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   63C    P8    10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Installation

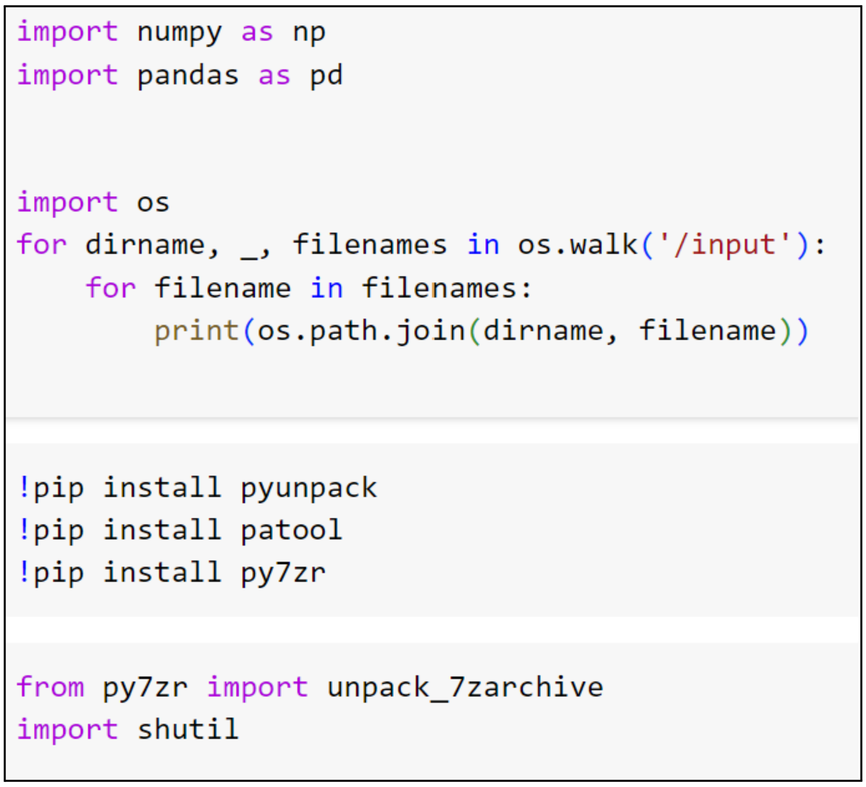
```
! pip install git+https://github.com/openai/whisper.git -q
```

```
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 9.8 MB/s eta 0:00:00
  Building wheel for openai-whisper (pyproject.toml) ... done
```

Fig 2. Whisper Requirements

Following the provided parameters, the improved content is as follows: The specified code necessitates the installation of the openai-whisper library. This can be achieved by executing the command!pip install git+https://github.com/openai/whisper.git -q. Moreover, the approach involves loading a pre-trained model ("medium") sizing at 769M utilising the 'whisper' library. The code employs 'wget' to retrieve an audio file from the URL "http://www.moviesoundclips.net/movies1/batmanbegins/bats.mp3," saving it as 'audio.mp3'. Subsequently, both the Audio and display methods of the IPython.display module are utilised to facilitate the playthrough of the audio file. In conclusion, the openai-whisper library, the 'whisper' module, and the 'wget' function are all indispensable for obtaining the desired audio file.

2 Text to Speech Conversion Model



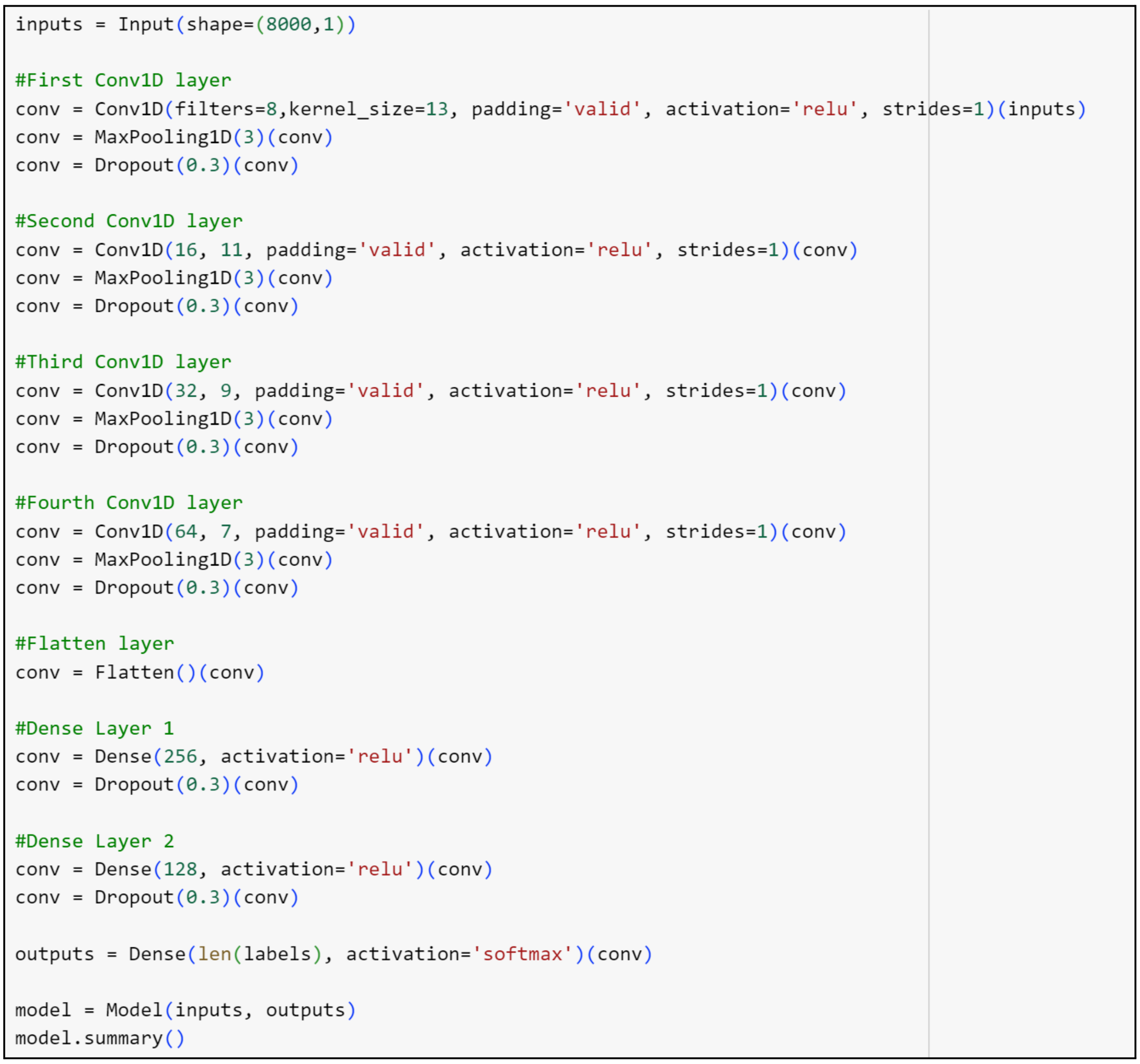
```
import numpy as np
import pandas as pd


import os
for dirname, _, filenames in os.walk('/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
!pip install pyunpack
!pip install patool
!pip install py7zr
```

```
from py7zr import unpack_7zarchive
import shutil
```

Fig 3. Text to Speech Conversion

The given code necessitates the utilisation of various Python libraries and modules for diverse tasks such as data processing, file management, and machine learning. It leverages 'numpy' and 'pandas' for carrying out numerical operations and effectively managing data. Additionally, it makes use of 'os' for seamless communication with the operating system. Furthermore, the code integrates 'pyunpack,' 'patool,' and 'py7zr' to handle compressed files, ensuring efficient file management. Notably, it incorporates 'shutil' to register and unpack 7z archives, 'librosa' for precise audio processing, 'matplotlib.pyplot' for visually representing data, 'IPython.display' for enabling interactive display capabilities, and '

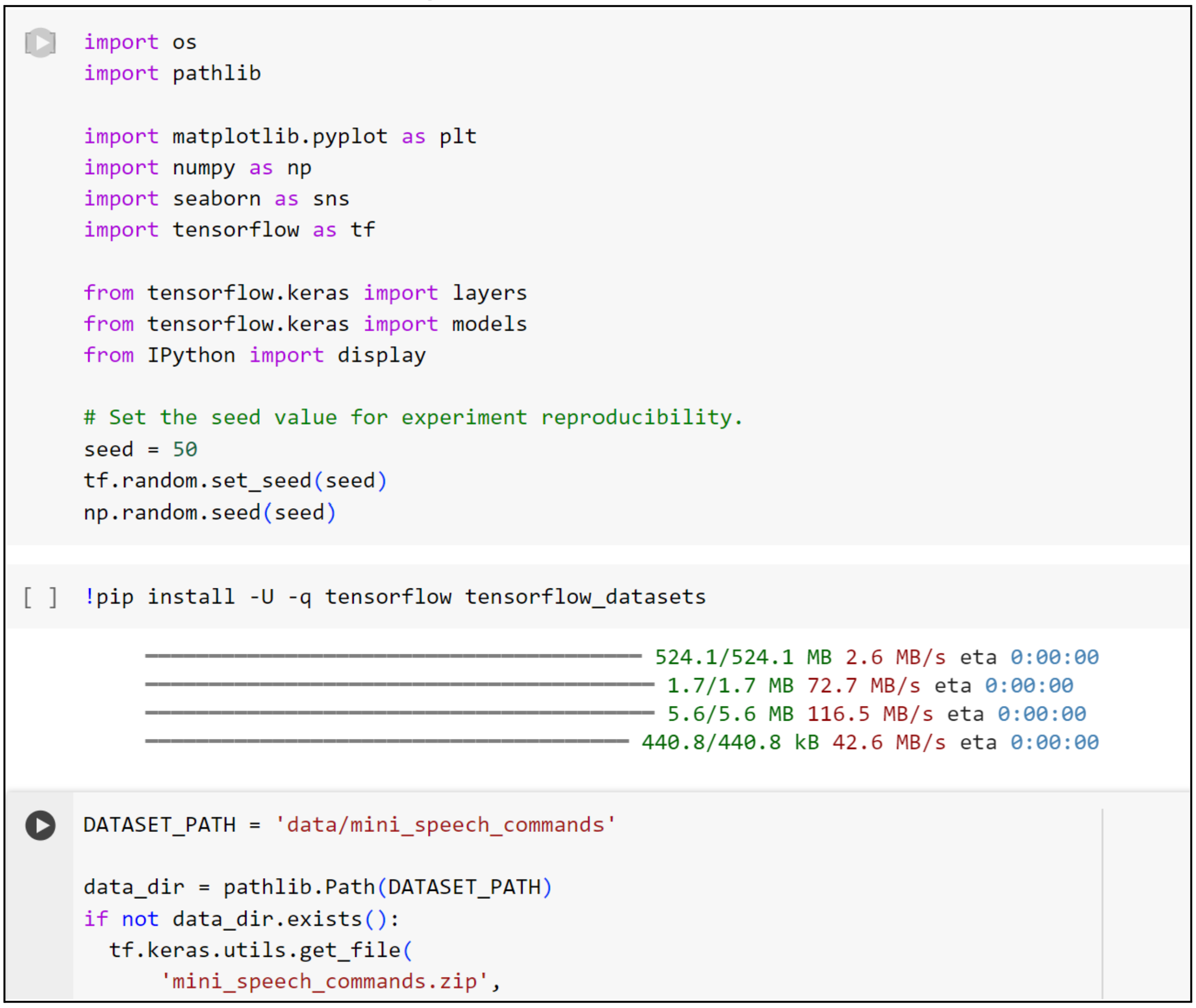


```
inputs = Input(shape=(8000,1))

#First Conv1D layer
conv = Conv1D(filters=8,kernel_size=13, padding='valid', activation='relu', strides=1)(inputs)
conv = MaxPooling1D(3)(conv)
conv = Dropout(0.3)(conv)

#Second Conv1D layer
conv = Conv1D(16, 11, padding='valid', activation='relu', strides=1)(conv)
conv = MaxPooling1D(3)(conv)
conv = Dropout(0.3)(conv)

#Third Conv1D layer
conv = Conv1D(32, 9, padding='valid', activation='relu', strides=1)(conv)
conv = MaxPooling1D(3)(conv)
conv = Dropout(0.3)(conv)

#Fourth Conv1D layer
conv = Conv1D(64, 7, padding='valid', activation='relu', strides=1)(conv)
conv = MaxPooling1D(3)(conv)
conv = Dropout(0.3)(conv)

#Flatten layer
conv = Flatten()(conv)

#Dense Layer 1
conv = Dense(256, activation='relu')(conv)
conv = Dropout(0.3)(conv)

#Dense Layer 2
conv = Dense(128, activation='relu')(conv)
conv = Dropout(0.3)(conv)

outputs = Dense(len(labels), activation='softmax')(conv)

model = Model(inputs, outputs)
model.summary()
```

Fig 4. Model Development requirements for text to speech

In building the machine learning components, 'keras' is utilised to construct a Convolutional Neural Network (CNN) model. This model incorporates specific layers including 'Conv1D', 'MaxPooling1D', 'Flatten', 'Dense', 'Dropout', and 'Input'. The code also encompasses early pausing and model checkpoint callbacks, along with the closure of the Keras session for ensuring repeatability.

3 Voice Command Recognition



```
import os
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import models
from IPython import display

# Set the seed value for experiment reproducibility.
seed = 50
tf.random.set_seed(seed)
np.random.seed(seed)
```

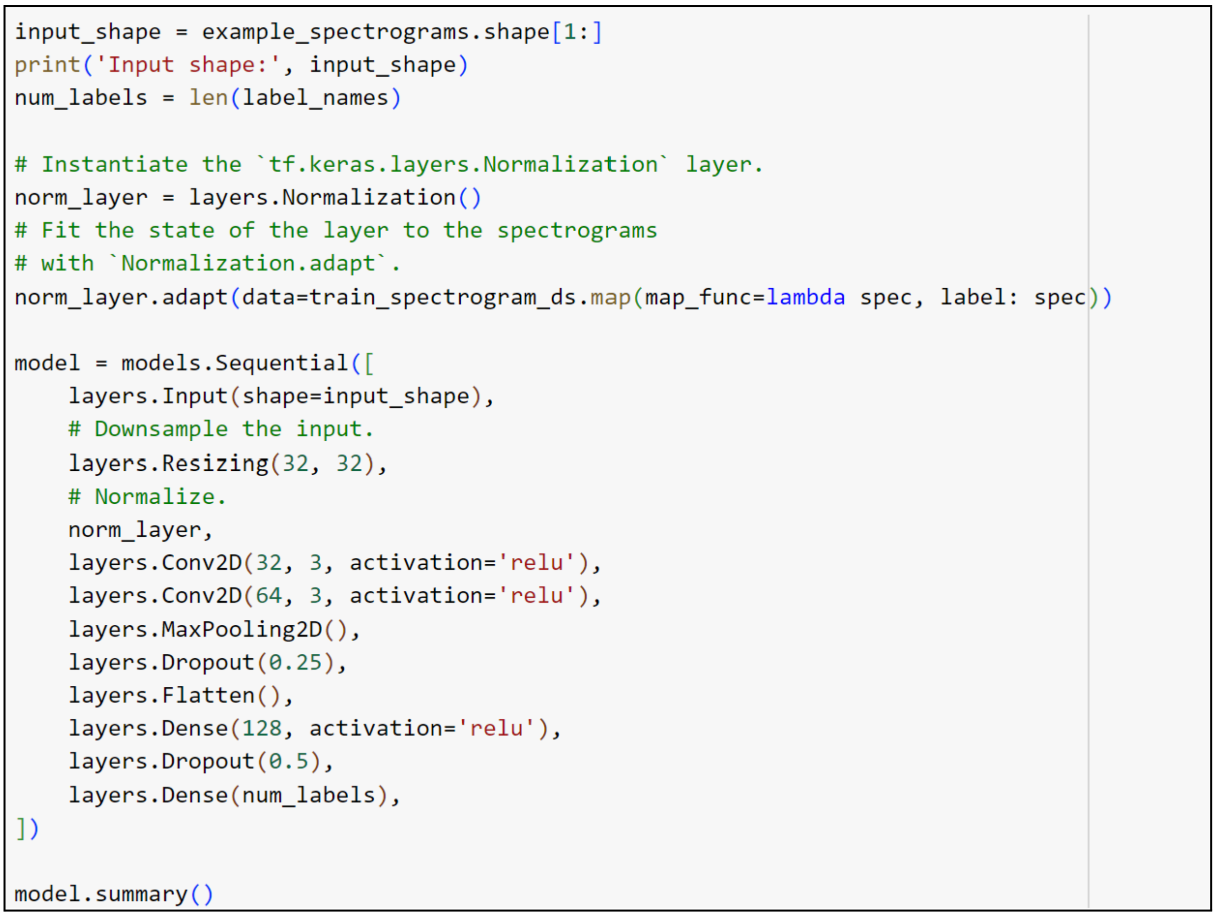
```
[ ] !pip install -U -q tensorflow tensorflow_datasets
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 524.1/524.1 MB 2.6 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 72.7 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.6/5.6 MB 116.5 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 440.8/440.8 kB 42.6 MB/s eta 0:00:00
```

```
DATASET_PATH = 'data/mini_speech_commands'

data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
  tf.keras.utils.get_file(
      'mini_speech_commands.zip',
```

Fig 5. Requirements for Voice Command Recognition

The specified code mandates the utilisation of various Python libraries and frameworks. These include 'os' for interfacing with the operating system, 'pathlib' for managing file paths, 'matplotlib.pyplot' and 'seaborn' for visualising data, 'numpy' for numerical computations, 'tensorflow' for machine learning, 'tensorflow.keras.layers' and 'tensorflow.keras.models' for constructing neural network models, and 'IPython.display' for interactive display functionalities.

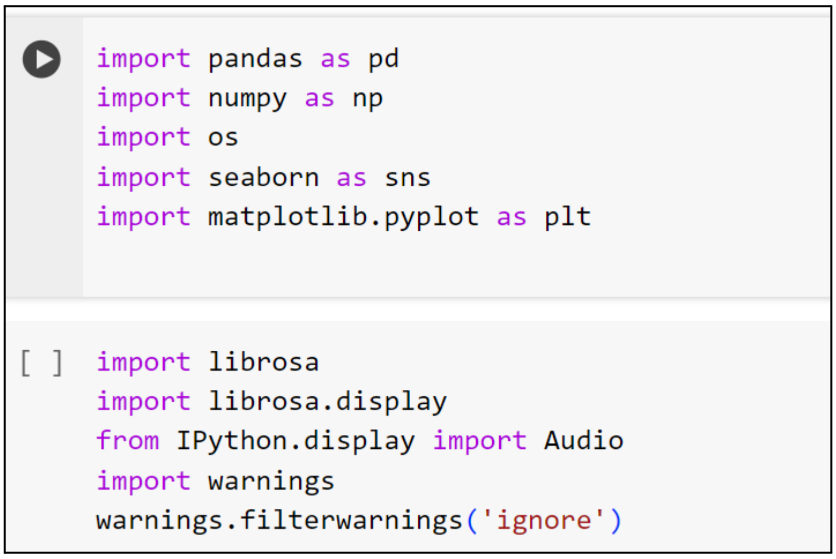


```python
input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)
num_labels = len(label_names)

# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=train_spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input.
    layers.Resizing(32, 32),
    # Normalize.
    norm_layer,
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_labels),
])

model.summary()
```

Fig 6. Model Development Requirements for Command detection

The code also makes use of 'tensorflow_datasets' to handle datasets. The '!pip install -U -q tensorflow tensorflow_datasets' command makes it easier to install the 'tensorflow' and 'tensorflow_datasets' libraries. Additionally, the code integrates particular layers from 'tensorflow.keras.layers' such as 'Normalisation', 'Input', 'Resizing', 'Conv2D', 'MaxPooling2D', 'Dropout', 'Flatten', and 'Dense' to build a convolutional neural network (CNN) model. In short, the prerequisites include os, pathlib, matplotlib, numpy, seaborn, tensorflow, and their accompanying libraries for data visualisation, machine learning, and artificial intelligence.

4 Speech Emotion Recognition

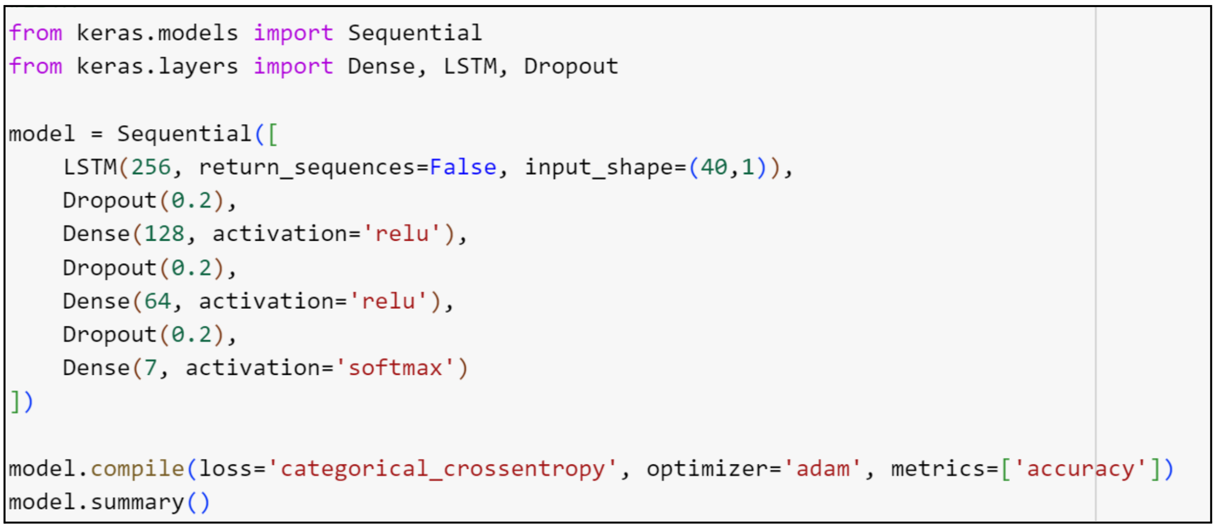


```
import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[ ]  import librosa
     import librosa.display
     from IPython.display import Audio
     import warnings
     warnings.filterwarnings('ignore')
```

Fig 7. Libraries required for Speech Emotion Recognition

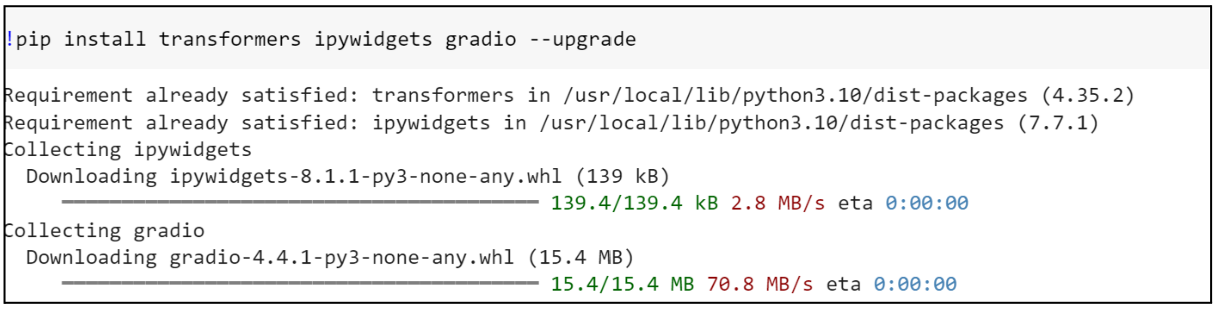
A number of Python libraries and frameworks are needed. Among them are 'pandas' for data manipulation, 'numpy' for numerical operations, 'os' for interacting with the operating system,'seaborn' and 'matplotlib' for data visualisation, 'librosa' for audio processing,'IPython.display' for displaying audio, and 'keras' for building and training neural networks. The neural network architecture uses Keras' 'Sequential' model, which includes layers like 'LSTM' (Long Short-Term Memory), 'Dense' for totally connected layers, and 'Dropout' for regularisation. Furthermore, the code makes use of 'warnings' to filter out unnecessary warnings. In summary, pandas, numpy, os, seaborn, matplotlib, librosa, IPython.display, keras, and their dependencies are required for data analysis, visualisation, audio processing, and neural network implementation.



```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

model = Sequential([
    LSTM(256, return_sequences=False, input_shape=(40,1)),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(7, activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Fig 8. Model Development of Speech Emotion Recognition

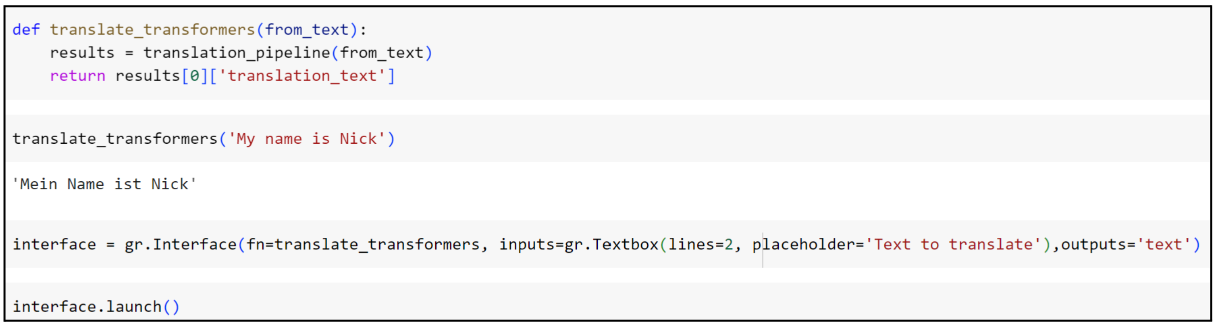
5 Text Classification Web Application



```
!pip install transformers ipywidgets gradio --upgrade

Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.35.2)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (7.7.1)
Collecting ipywidgets
  Downloading ipywidgets-8.1.1-py3-none-any.whl (139 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.4/139.4 kB 2.8 MB/s eta 0:00:00
Collecting gradio
  Downloading gradio-4.4.1-py3-none-any.whl (15.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.4/15.4 MB 70.8 MB/s eta 0:00:00
```

Fig 9. Transformers and gradio update

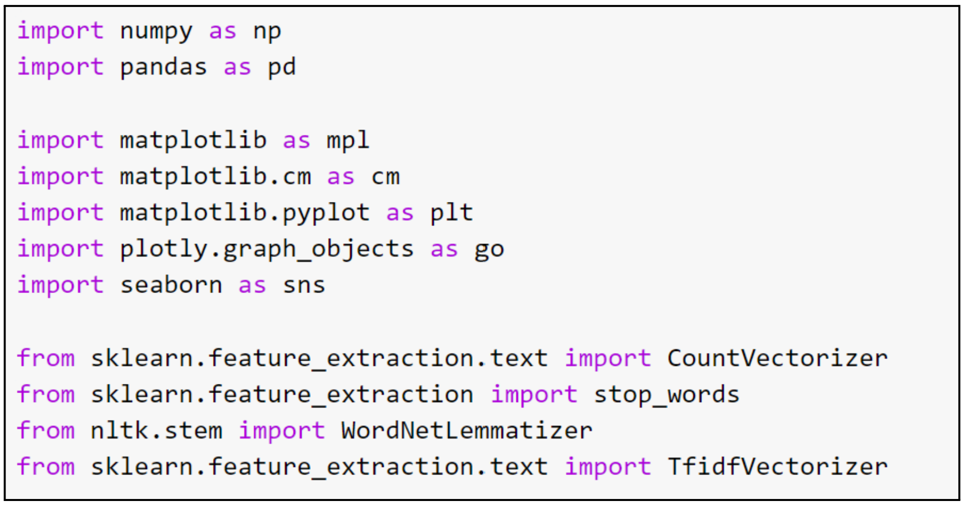
The provided code defines the PyTorch library and its components, ensuring the installation of PyTorch 1.8.1 with support for CUDA 11.1. This command is essential to guarantee the precise installation of PyTorch, torchvision, and torchaudio versions. Within the code, the method 'translate_transformers' is employed for text translation using a translation pipeline. A sample text, 'My name is Nick,' is used as an example, translating to 'Mein Name ist Nick.' Notably, the code utilises Gradio (gr.Interface) to create a user interface facilitating text input for translation and exhibiting the translated output. It is evident that both text translation and user interface creation rely on PyTorch (torch, torchvision, torchaudio) and Gradio. This code intricately weaves together various elements to provide functionalities such as text translation and user interface interaction, underscoring the versatile applications of PyTorch and Gradio.



```python
def translate_transformers(from_text):
    results = translation_pipeline(from_text)
    return results[0]['translation_text']

translate_transformers('My name is Nick')

'Mein Name ist Nick'

interface = gr.Interface(fn=translate_transformers, inputs=gr.Textbox(lines=2, placeholder='Text to translate'),outputs='text')

interface.launch()
```

Fig 10. Launching of the web application

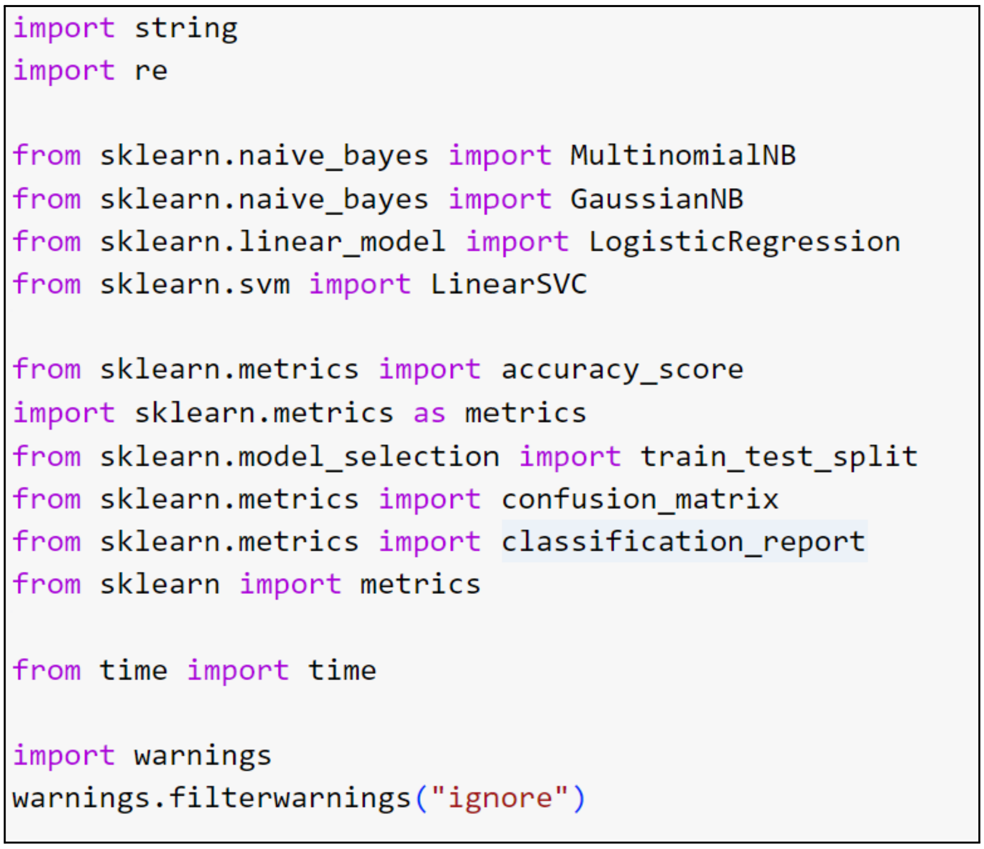
6 Text Classification and Language Translation Model



```python
import numpy as np
import pandas as pd

import matplotlib as mpl
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import stop_words
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

Fig 11. Requirements for Text Classification and language translation

Text processing, and machine learning, the code leverages a myriad of Python libraries and modules. Harnessing the power of 'numpy' and 'pandas' for numerical operations and data manipulation, delving into 'matplotlib' and 'plotly' for data visualisation, and tapping into 'seaborn' to elevate the visual appeal of plots. Moreover, for text processing tasks, the code strategically employs CountVectorizer, stop_words, WordNetLemmatizer, and TfidfVectorizer from scikit-learn and NLTK. It also integrates string and regular expression libraries to prepare the textual data seamlessly.



```python
import string
import re

from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import metrics

from time import time

import warnings
warnings.filterwarnings("ignore")
```

Fig 12. Model development of Text Classification

Scikit-learn machine learning models, a plethora of tools awaits deployment. Among them stand 'MultinomialNB,' 'GaussianNB,' 'LogisticRegression,' and 'LinearSVC.' The evaluation metrics, encompassing the accuracy score, confusion matrix, and classification report, are ushered in by 'sklearn.metrics,' aimed at providing an intricate understanding of model performance. To initialise the groundwork, the 'train_test_split' function steps in, unravelling the dataset with precision, while 'time' takes charge of measuring the execution time, ensuring efficiency at its peak. Meanwhile, 'warnings.filterwarnings("ignore")' steps in to suppress any distracting warnings, allowing a seamless operation