Configuration Manual

MSc Research Project
MSc. Data Analytics

Neha Eldho
Student ID: x21217793

School of Computing
National College of Ireland

Supervisor: Qurrat Ul Ain

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Neha Eldho
Student ID:	x21217793
Programme:	MSc Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Qurrat Ul Ain
Submission Due Date:	14/08/2023
Project Title:	Emotion Detection Using Deep Learning Models on Speech and Text Data
Word Count:	4798
Page Count:	38

Signature:	
Date:	13th August 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.	☐

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

Configuration Manual

Neha Eldho
x21217793

1 Introduction

This project is a comprehensive exploration of emotion detection from both text and audio data using advanced machine learning and deep learning techniques. This manual provides all the necessary information to set up, configure, and run the project in your own environment. The Emotion Detection Project, encapsulated in a Jupyter notebook, utilizes various Python libraries such as ‘pandas‘, ‘numpy‘, ‘matplotlib‘, ‘seaborn‘, ‘sklearn‘, ‘nltk‘, ‘re‘, and ‘keras‘ to preprocess data, build models, and evaluate their performance. This manual will guide you through the process of installing these libraries and setting up your environment to successfully run the project. The project involves several stages, including data preprocessing, model building with LSTM networks, and evaluation of the models. We will cover how to install and configure the required software, how to use the notebook, and how to interpret the results. Additionally, we will provide some troubleshooting advice for common issues. This manual assumes a basic understanding of Python programming and familiarity with Jupyter notebooks. While knowledge of machine learning concepts and natural language processing is beneficial, the Emotion Detection Project notebook provides explanations and comments that make it accessible to individuals new to these concepts.

2 Required Specifications

2.1 Expected Hardware Requirements

The exact hardware specifications can vary depending on the size of the data you are processing, the following are general recommendations:

- Processor: Intel or AMD processor, 2 GHz or faster
- Memory: Minimum 8 GB RAM. For larger datasets or complex computations, 16 GB or more is recommended.
- Storage: At least 2 GB of free disk space for the installation of Python, required libraries, and the project itself. Additional space will be needed for data storage.

2.2 Expected Software Requirements

- Operating System: Windows, macOS, or Linux. The project is platform-independent as long as you can run Python and Jupyter notebooks.

- Python: Python 3.7 or newer. Python can be downloaded from the official website: `https://www.python.org/downloads/`
- Jupyter Notebook: The project is provided as a Jupyter notebook. You can install Jupyter via pip with pip install jupyter or if you prefer using Anaconda, you can install it from there.

2.3 Actual System on Which Notebook was Executed

2.3.1 Hardware

- Processor: Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz
- RAM: 32.0 GB (31.9 GB usable)
- System Type: 64-bit architecture

2.3.2 Software

- Operating System: Windows 10 Pro
- Edition: Windows 10 Pro
- Version: 22H2
- OS build: 19045.3208
- Experience: Windows Feature Experience Pack 1000.19041.1000.0

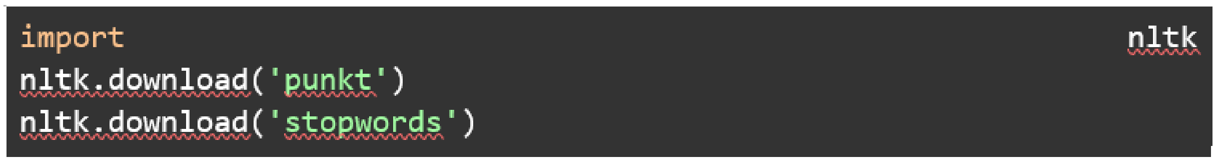
2.4 Python Packages Installation

Once Python is installed, the following packages are required to run the project. These packages can be installed via pip, Python's package installer. Open your terminal or command prompt and run the following commands:

- Pandas: ‘pip install pandas‘
- Numpy: ‘pip install numpy‘
- Matplotlib: ‘pip install matplotlib‘
- Seaborn: ‘pip install seaborn‘
- Scikit-learn (sklearn): ‘pip install -U scikit-learn‘
- NLTK: ‘pip install nltk‘
- Regular expressions (re): This is a built-in module and does not require installation.
- Keras: ‘pip install keras‘
- Tensorflow: Keras is a high-level API and needs a backend engine for computation. Tensorflow is one such engine. Install it using pip install tensorflow
- tqdm: ‘pip install tqdm‘

- Wordcloud: ‘pip install wordcloud‘

After installing NLTK, you need to download the ’punkt’ and ’stopwords’ datasets. Open a Python shell (or a Jupyter notebook) and run the following commands: Figure 1 shows the code for downloading ’punkt’ and ’stopwords’.



```
import                                                              nltk
nltk.download('punkt')
nltk.download('stopwords')
```

Figure 1: downloading ’punkt’ and ’stopwords’

Please ensure that your Python and pip installations are set up correctly and are accessible via the terminal or command prompt. If you encounter any issues, refer to the official installation instructions provided by each package or consult the troubleshooting section of this manual.

3 Data Collection

The Emotion Detection Project leverages two primary datasets, one for textual data and the other for audio data. These datasets are essential for training and evaluating the models built during the project.

3.1 Emotion Detection from Text

Source: `https://www.kaggle.com/datasets/pashupatigupta/emotion-detection-from-text` Kaggle - Emotion Detection from Text

3.1.1 Description:

This dataset provides a collection of text entries labelled with their corresponding emotions. It is ideal for training machine learning models that aim to detect or predict emotions from textual data.

3.1.2 Download and Usage Instructions:

- Visit the dataset page on Kaggle: Emotion Detection from Text
- Click on the ”Download” button to download the dataset as a ZIP file.
- Extract the ZIP file to get the dataset in CSV format.
- Load the CSV file into the project using pandas or any preferred data manipulation library.

3.2 RAVDESS Emotional Speech Audio

Source: `https://www.kaggle.com/datasets/uwrfkaggler/ravdess-emotional-speech-audio` Kaggle - RAVDESS Emotional Speech Audio

3.2.1 Description:

The RAVDESS dataset consists of audio files of actors vocalising emotional expressions. Each audio file is labelled with the emotion it represents. This dataset provides a rich collection of emotional speech audio, which is valuable for training models for emotion detection from audio data.

3.2.2 Download and Usage Instructions:

- Visit the dataset page on Kaggle: RAVDESS Emotional Speech Audio
- Click on the "Download" button to download the dataset as a ZIP file.
- Extract the ZIP file to access the audio files.
- Integrate the audio files into the project as needed.

Please ensure you have the necessary permissions to access and use these datasets for your project. Always acknowledge the data source and respect any licensing or usage constraints associated with the datasets.

4 Exploratory Data Analysis (Text)

4.1 Importing Libraries

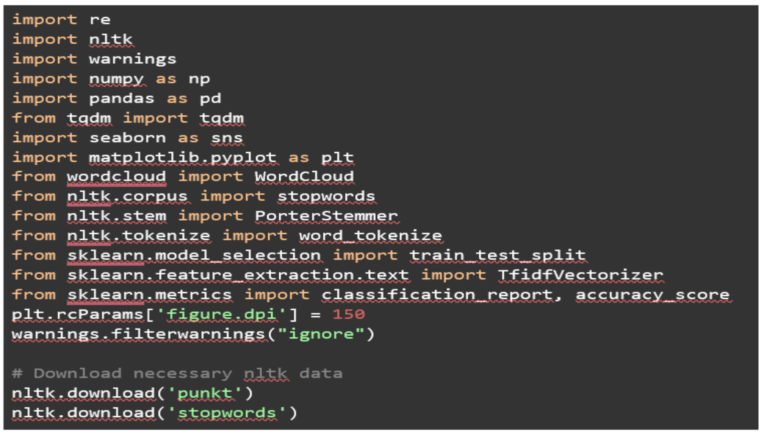
This code is setting up the tools and materials needed for a project. Think of it as gathering ingredients for a recipe.

- Firstly, it's collecting a set of tools (or libraries) from a workshop. These tools help in tasks like data organisation, drawing graphs, and processing language.
- Next, it adjusts some settings to make sure the pictures it draws are clear, and it turns off any distracting or unnecessary alerts.
- Lastly, it fetches two special language tools to help it understand and break down sentences into words.

In essence, this code is just preparing everything needed before the main work starts. Figure 2 shows the code for preparing everything needed before the main work starts.

4.2 Loading and Preprocessing the Text Data

- Loads a dataset named 'text_emotion' which has information about text content and the sentiment (or emotion) associated with it.
- From the entire dataset, it chooses to work with two columns: the actual content of the text and its sentiment.
- It then defines a set of cleaning steps for the text content. These steps include:
  - Converting everything to lowercase.
  - Removing mentions, links, and special codes.



```python
import re
import nltk
import warnings
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, accuracy_score
plt.rcParams['figure.dpi'] = 150
warnings.filterwarnings("ignore")

# Download necessary nltk data
nltk.download('punkt')
nltk.download('stopwords')
```

Figure 2:

  - Breaking down the content into individual words.
  - Removing common, unimportant words.
  - Simplifying words to their basic form.

- With these cleaning steps defined, it goes through each text entry, applies the cleaning, and then adds this cleaned version to the dataset.

- Finally, it displays the cleaned dataset. In essence, it's about loading a dataset, selecting important parts, cleaning the text content, and displaying the cleaned dataset.

Figure 3 represents the code for Loading and Preprocessing the Text Data

4.3 Analysing Data

This code checks and counts any missing or incomplete information in the 'emotion_data'. It then displays how many pieces of missing information are present for each feature (or column) in the data. Figure 4 shows the code for analysing the Text data with the distribution of sentiments. Figure 5 shows the code for analysing the Text data with the distribution of word counts in a content and the distribution of average word length in content.

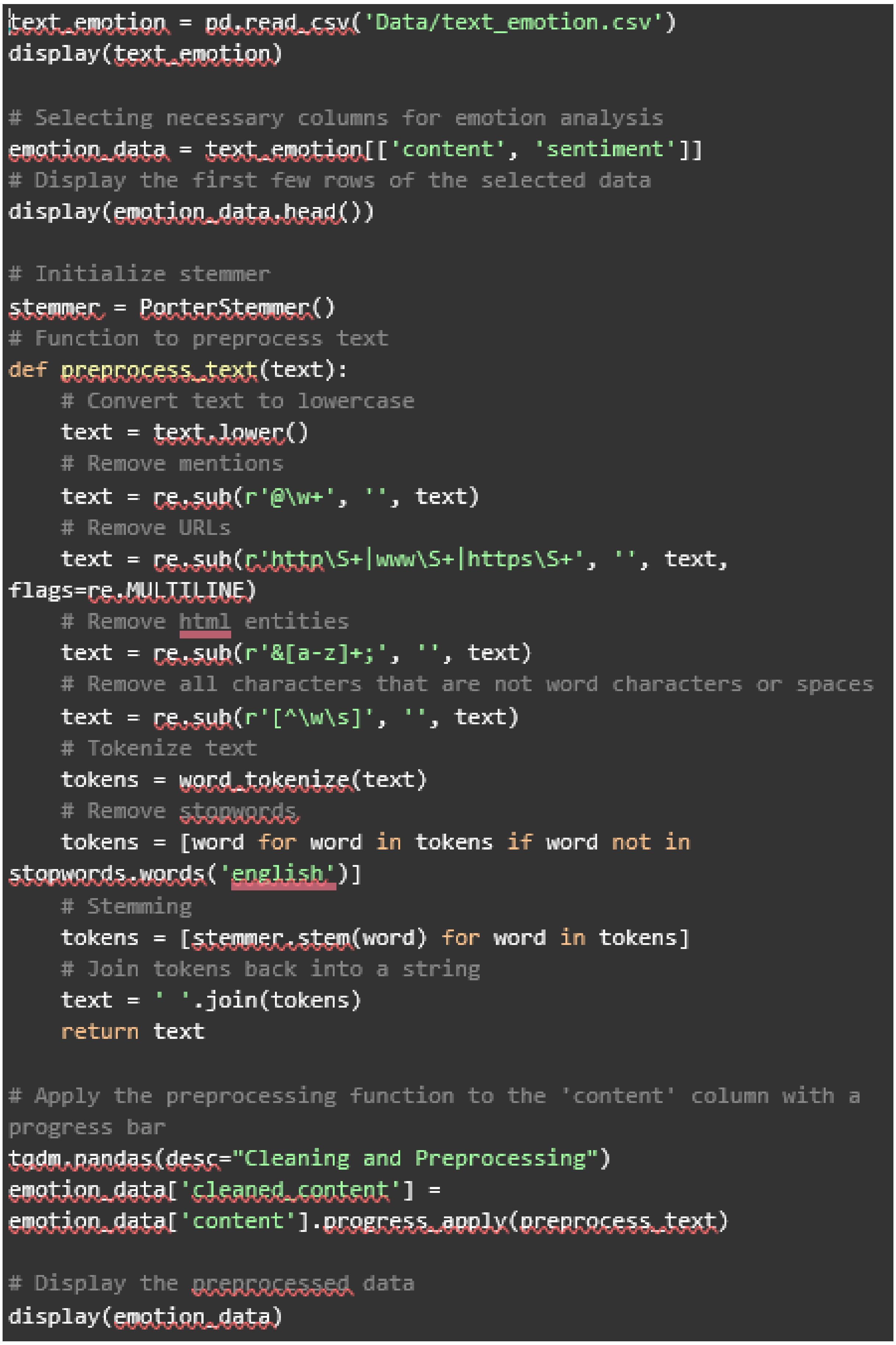
This Figure 6 below provides a quick summary of two specific features in the 'emotion_data:' the number of words ('word_count') and the average length of words ('avg_word_length'). It shows things like the average, minimum, and maximum values for these features.

This code in the Figure 7 does the following:

- It identifies the different types of sentiments (like happy, sad, or angry) present in the emotion_data.

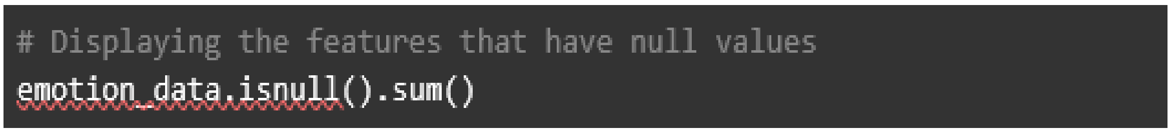
- For each type of sentiment, it gathers all the related text content.

- Using this content, it creates a visual called a "word cloud" which shows words that appear more often in a bigger size.

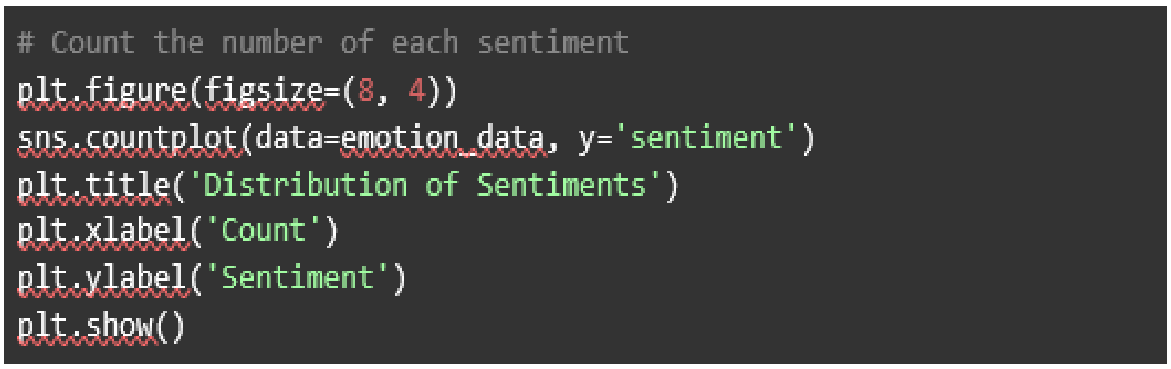


```python
text_emotion = pd.read_csv('Data/text_emotion.csv')
display(text_emotion)

# Selecting necessary columns for emotion analysis
emotion_data = text_emotion[['content', 'sentiment']]
# Display the first few rows of the selected data
display(emotion_data.head())

# Initialize stemmer
stemmer = PorterStemmer()
# Function to preprocess text
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove mentions
    text = re.sub(r'@\w+', '', text)
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text,
flags=re.MULTILINE)
    # Remove html entities
    text = re.sub(r'&[a-z]+;', '', text)
    # Remove all characters that are not word characters or spaces
    text = re.sub(r'[^\w\s]', '', text)
    # Tokenize text
    tokens = word_tokenize(text)
    # Remove stopwords
    tokens = [word for word in tokens if word not in
stopwords.words('english')]
    # Stemming
    tokens = [stemmer.stem(word) for word in tokens]
    # Join tokens back into a string
    text = ' '.join(tokens)
    return text

# Apply the preprocessing function to the 'content' column with a
progress bar
tqdm.pandas(desc="Cleaning and Preprocessing")
emotion_data['cleaned_content'] =
emotion_data['content'].progress_apply(preprocess_text)

# Display the preprocessed data
display(emotion_data)
```

Figure 3: Loading and Preprocessing the Text Data



```python
# Displaying the features that have null values
emotion_data.isnull().sum()
```



```python
# Count the number of each sentiment
plt.figure(figsize=(8, 4))
sns.countplot(data=emotion_data, y='sentiment')
plt.title('Distribution of Sentiments')
plt.xlabel('Count')
plt.ylabel('Sentiment')
plt.show()
```

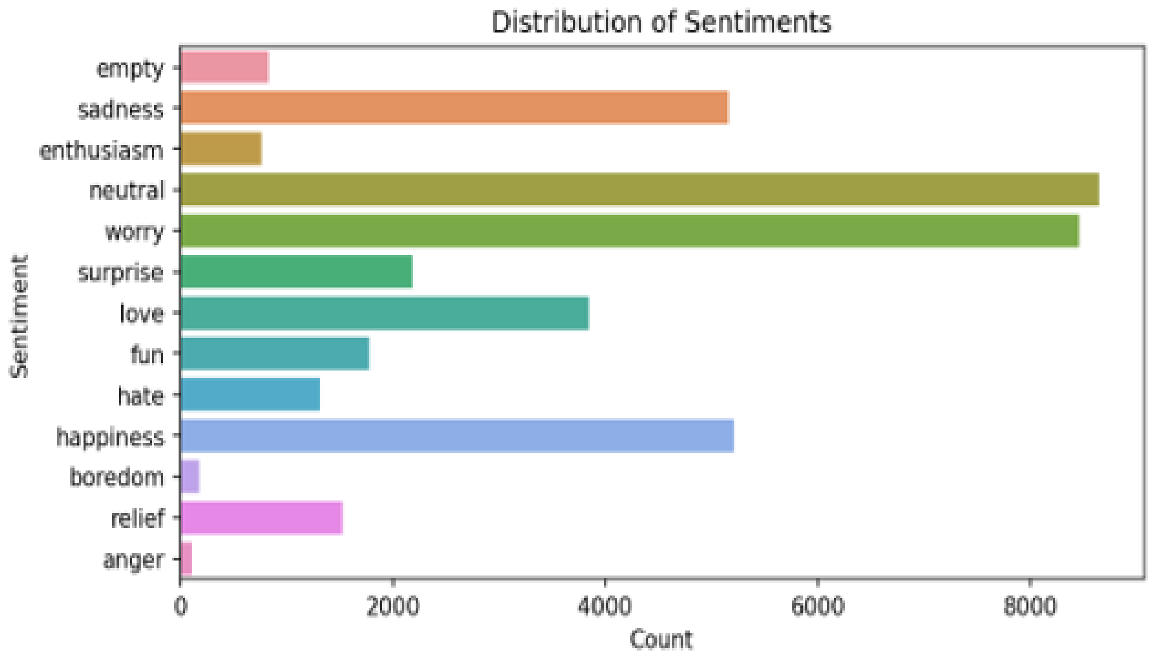
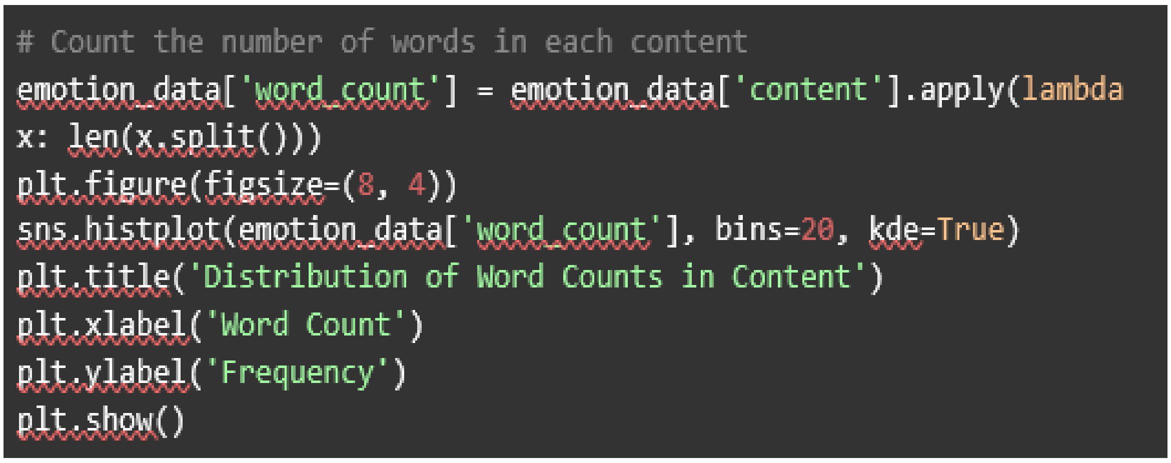


Figure 1: Distribution of Sentiments



```python
# Count the number of words in each content
emotion_data['word_count'] = emotion_data['content'].apply(lambda
x: len(x.split()))
plt.figure(figsize=(8, 4))
sns.histplot(emotion_data['word_count'], bins=20, kde=True)
plt.title('Distribution of Word Counts in Content')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()
```

Figure 4: Analysing Text Data

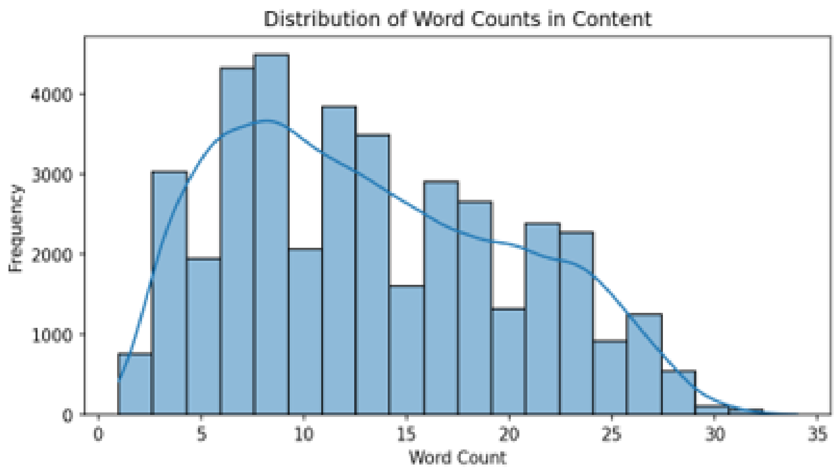
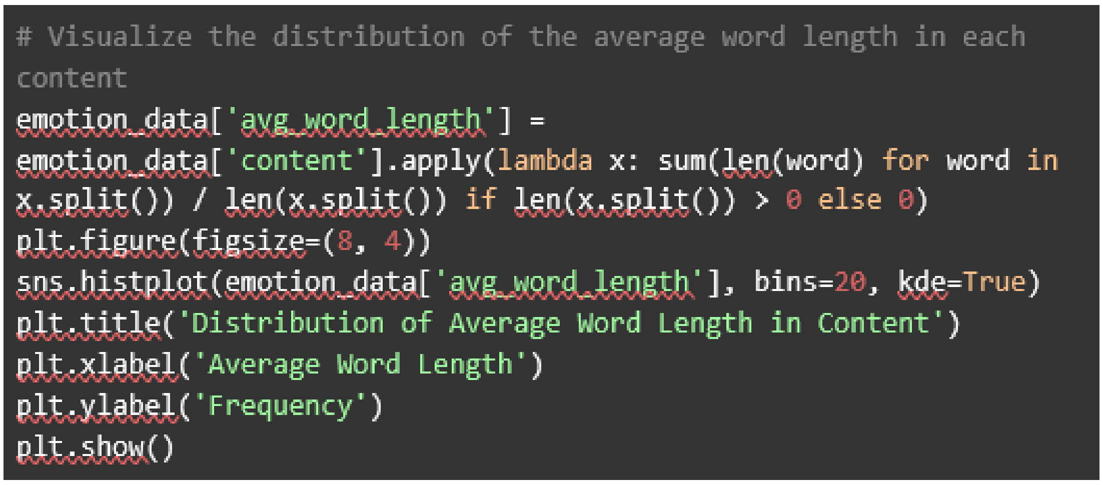


Figure 2: Distribution of Word Counts in Content



```
# Visualize the distribution of the average word length in each
content
emotion_data['avg_word_length'] =
emotion_data['content'].apply(lambda x: sum(len(word) for word in
x.split()) / len(x.split()) if len(x.split()) > 0 else 0)
plt.figure(figsize=(8, 4))
sns.histplot(emotion_data['avg_word_length'], bins=20, kde=True)
plt.title('Distribution of Average Word Length in Content')
plt.xlabel('Average Word Length')
plt.ylabel('Frequency')
plt.show()
```

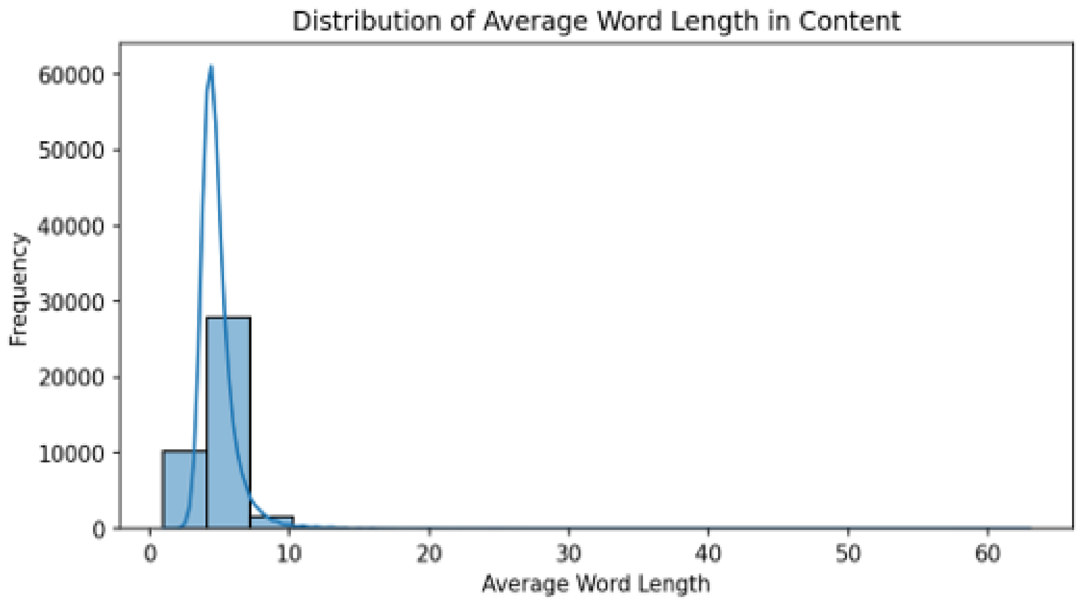
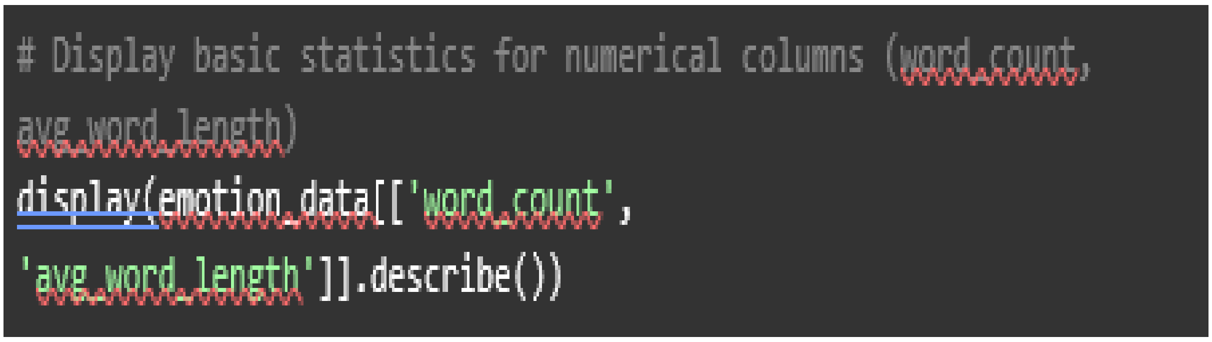


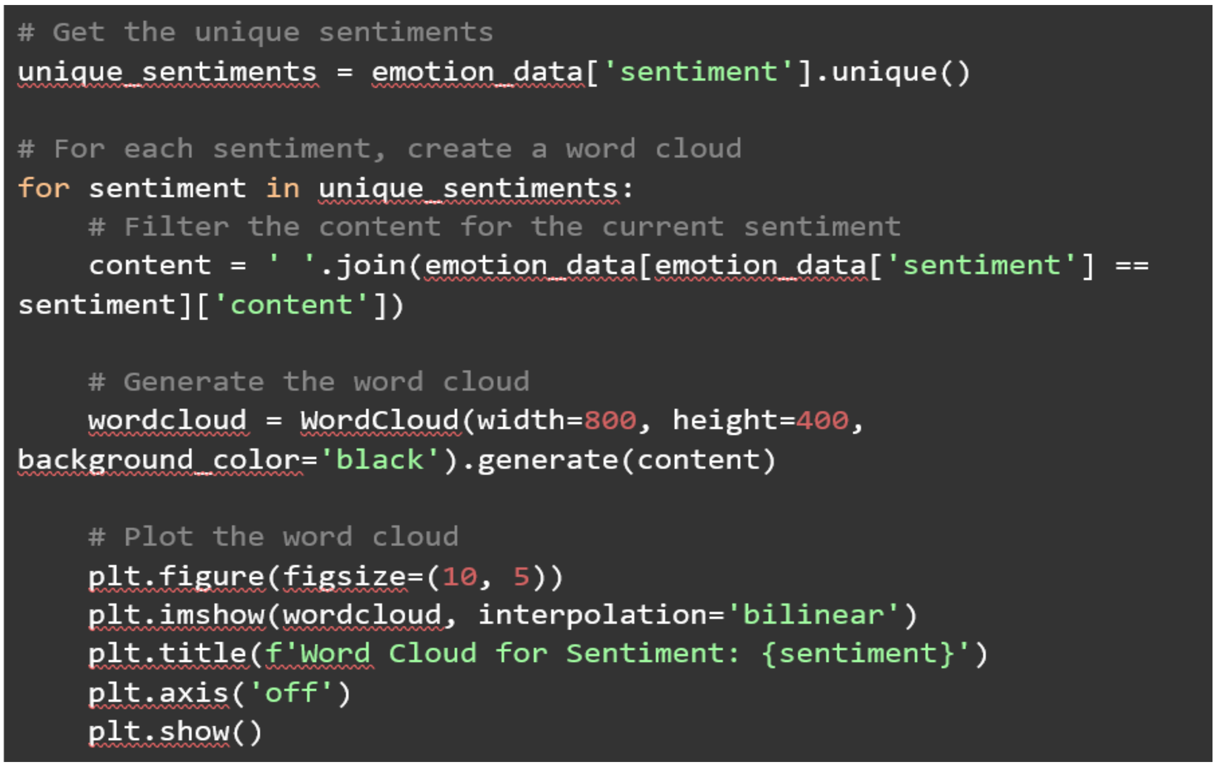
Figure 3: Distribution of Average Word Length in Content

Figure 5: Analysing Text Data



```python
# Display basic statistics for numerical columns (word_count, avg_word_length)
display(emotion_data[['word_count', 'avg_word_length']].describe())
```

Figure 6: summary of two specific features in the 'emotion_data



```python
# Get the unique sentiments
unique_sentiments = emotion_data['sentiment'].unique()

# For each sentiment, create a word cloud
for sentiment in unique_sentiments:
    # Filter the content for the current sentiment
    content = ' '.join(emotion_data[emotion_data['sentiment'] ==
sentiment]['content'])

    # Generate the word cloud
    wordcloud = WordCloud(width=800, height=400,
background_color='black').generate(content)

    # Plot the word cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud for Sentiment: {sentiment}')
    plt.axis('off')
    plt.show()
```

Figure 7: code for type of sentiments

- It then displays these word clouds for each sentiment, allowing you to see which words are most associated with each feeling.

Figure 8represents the word cloud of all the sentiments.

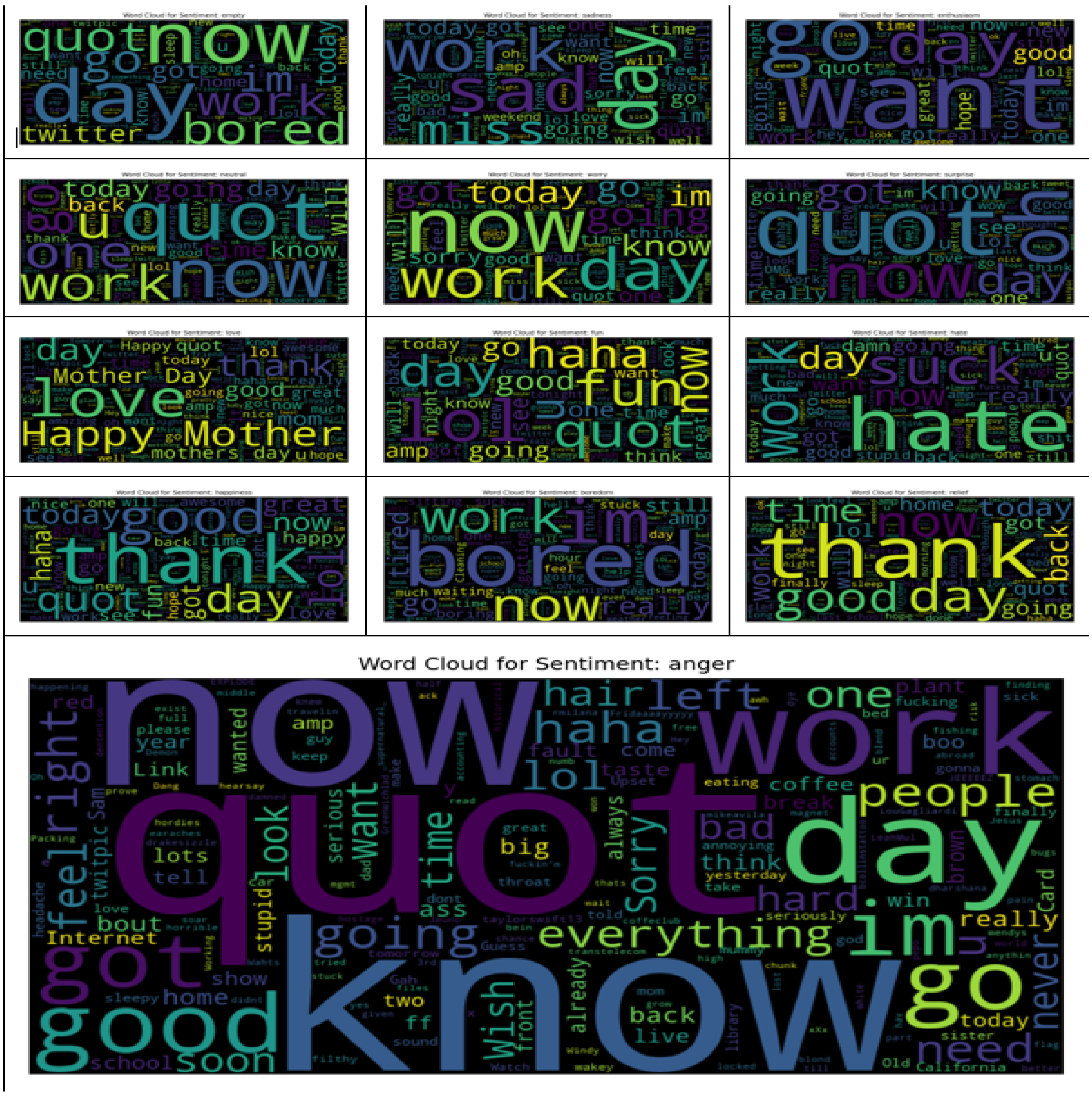
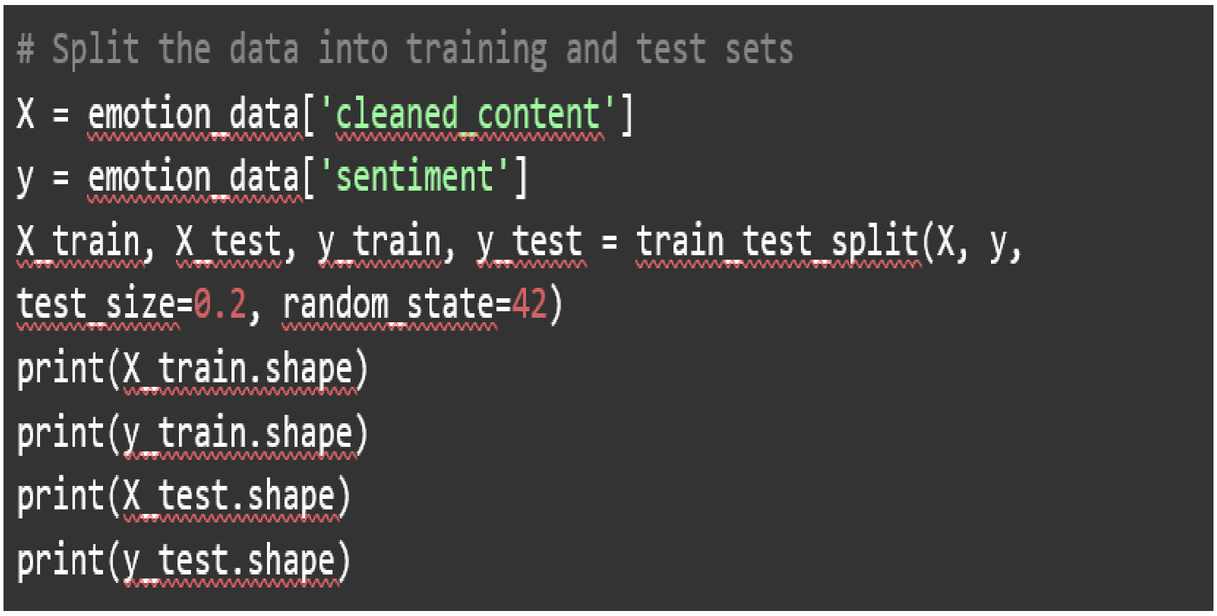


Figure 8: word cloud

4.3.1 Splitting the Data

This code in Figure 9 represents the splitting of data

- Take the cleaned-up text and their emotions.
- Divides them into a training set (to learn from) and a test set (to check the learning).
- Displays the number of items in each set.



```python
# Split the data into training and test sets
X = emotion_data['cleaned_content']
y = emotion_data['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

Figure 9: Splitting the Data

5 5 Modelling for Text Data

5.1 Machine Learning Model

This code in Figure 10:

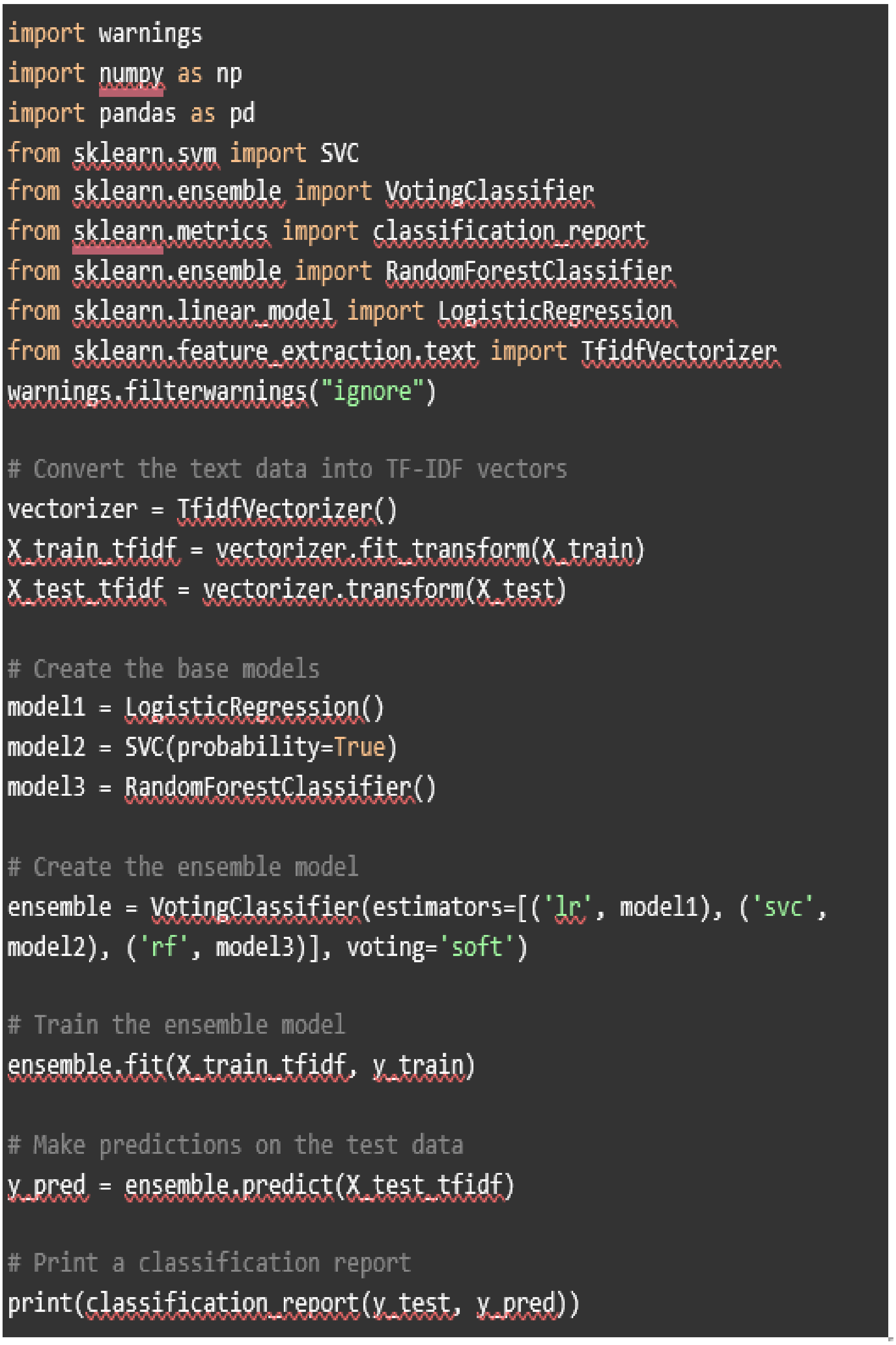
- Prepares tools to work with text data and models.
- Converts the text data into a format that's easier for models to understand.
- Sets up three different models (methods) to predict emotions from text.
- Combines these models into one unified model for better predictions.
- Trains the unified model using the training data.
- Tests the model's predictions on new data.
- Displays how well the model did in the test.

Figure 11 represents the Classification report of the Ensemble model.

5.2 Base LSTM

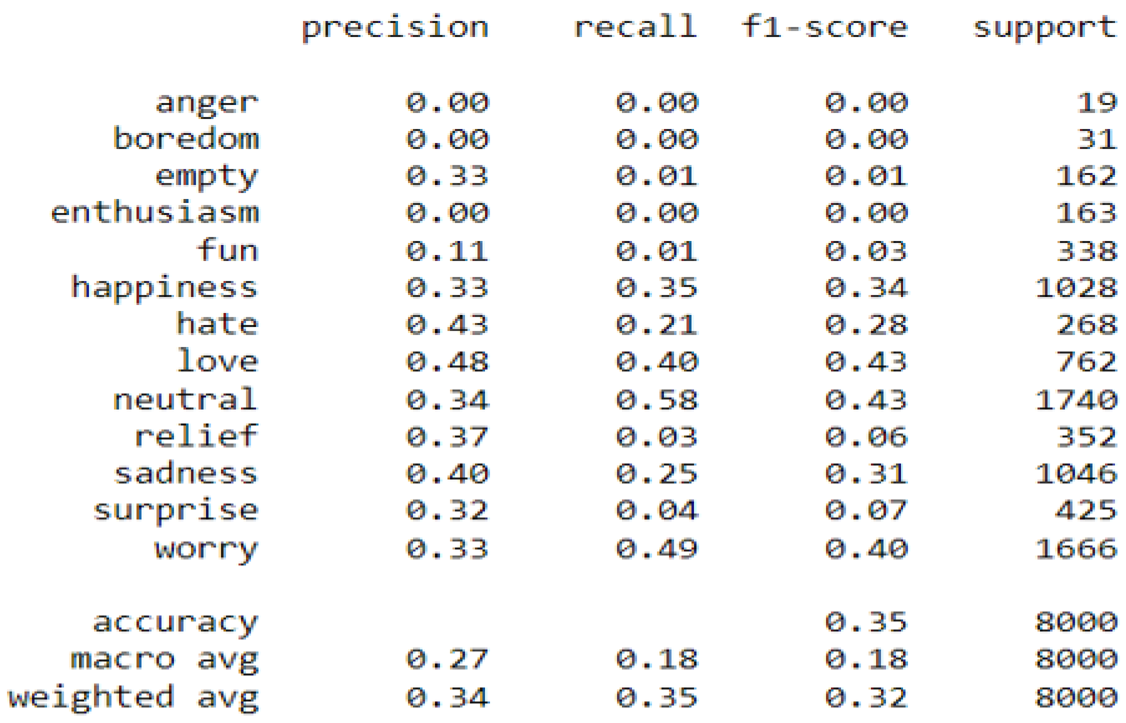
This code in Figure 12 andFigure 13 :

- Sets up tools to work with and process text data.
- Converts the text into numbers and ensures they are of equal length.
- Converts emotions into a format the model can understand.
- Creates a model (a kind of neural network) to predict emotions based on the text.



```python
import warnings
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")

# Convert the text data into TF-IDF vectors
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Create the base models
model1 = LogisticRegression()
model2 = SVC(probability=True)
model3 = RandomForestClassifier()

# Create the ensemble model
ensemble = VotingClassifier(estimators=[('lr', model1), ('svc',
model2), ('rf', model3)], voting='soft')

# Train the ensemble model
ensemble.fit(X_train_tfidf, y_train)

# Make predictions on the test data
y_pred = ensemble.predict(X_test_tfidf)

# Print a classification report
print(classification_report(y_test, y_pred))
```

Figure 10: Ensemble model



```
              precision    recall  f1-score   support

       anger       0.00      0.00      0.00        19
     boredom       0.00      0.00      0.00        31
       empty       0.33      0.01      0.01       162
  enthusiasm       0.00      0.00      0.00       163
         fun       0.11      0.01      0.03       338
   happiness       0.33      0.35      0.34      1028
        hate       0.43      0.21      0.28       268
        love       0.48      0.40      0.43       762
     neutral       0.34      0.58      0.43      1740
      relief       0.37      0.03      0.06       352
     sadness       0.40      0.25      0.31      1046
    surprise       0.32      0.04      0.07       425
       worry       0.33      0.49      0.40      1666

    accuracy                           0.35      8000
   macro avg       0.27      0.18      0.18      8000
weighted avg       0.34      0.35      0.32      8000
```

Figure 11: Classification report of ensemble model

- Trains this model with the training data.
- Plots how well the model learned over time.
- Saves the trained model for future use.

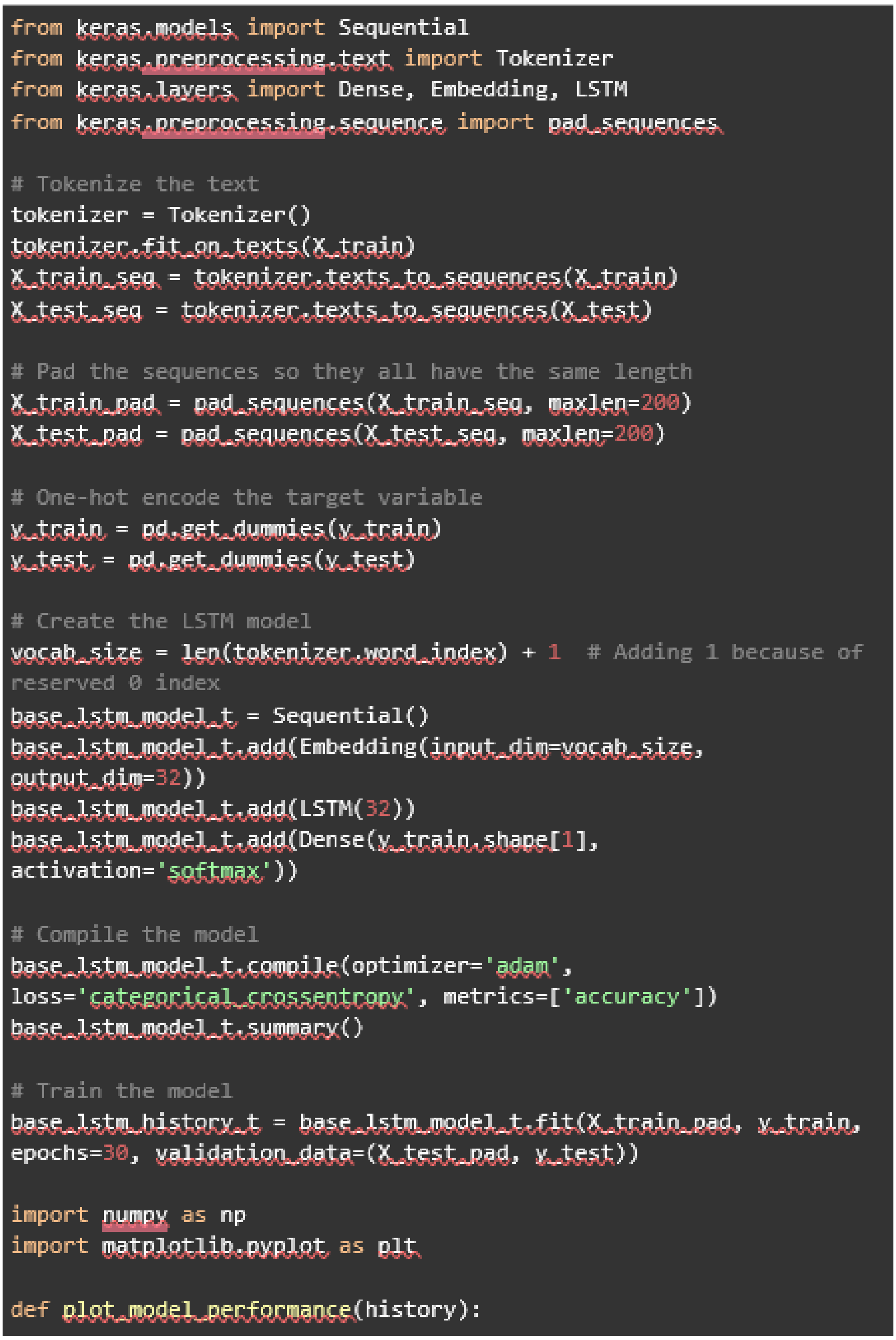
The Accuarcy and Average loss of the model is shown in Figure 14

5.3 Tuned-LSTM

This code in Figure 15 and Figure 16

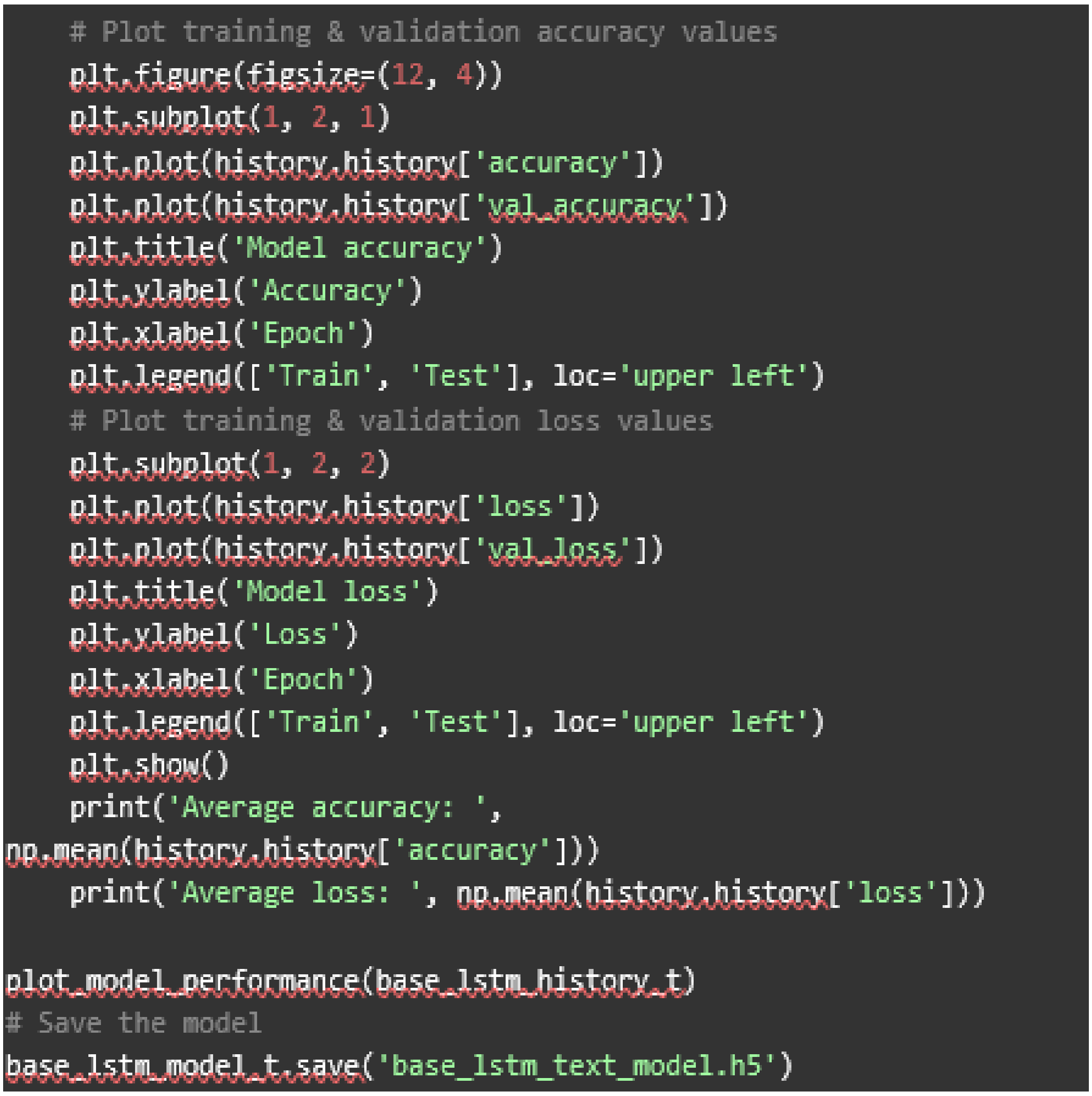
- Prepares tools for processing text data and building a neural network.
- Converts text data into a numeric format, making sure they're all the same length.
- Turns emotions into a format the model understands.
- Divides the data into a training set and a test set.
- Designs a slightly more advanced model to predict emotions from text.
- Trains this model with the training data.
- Displays how well the model learned over time.
- Saves the trained model for future use.

The Accuarcy and Average loss of the model is shown in Figure 17



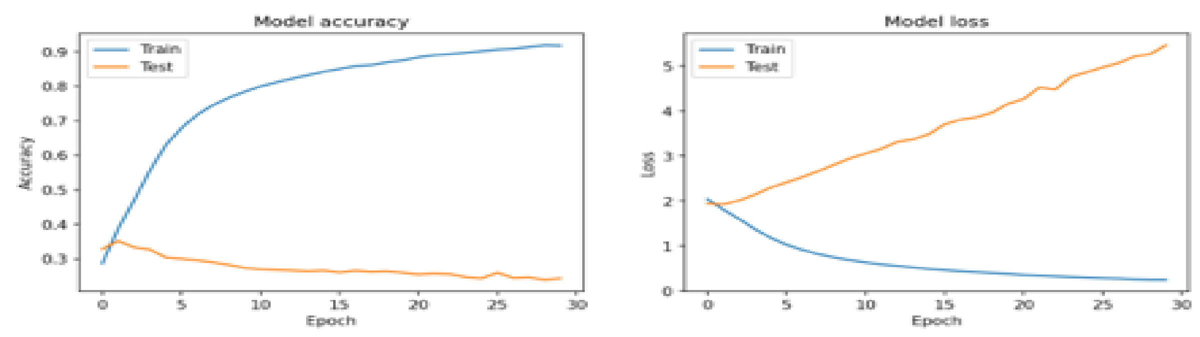
```python
from keras.models import Sequential
from keras.preprocessing.text import Tokenizer
from keras.layers import Dense, Embedding, LSTM
from keras.preprocessing.sequence import pad_sequences

# Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Pad the sequences so they all have the same length
X_train_pad = pad_sequences(X_train_seq, maxlen=200)
X_test_pad = pad_sequences(X_test_seq, maxlen=200)

# One-hot encode the target variable
y_train = pd.get_dummies(y_train)
y_test = pd.get_dummies(y_test)

# Create the LSTM model
vocab_size = len(tokenizer.word_index) + 1  # Adding 1 because of
reserved 0 index
base_lstm_model_t = Sequential()
base_lstm_model_t.add(Embedding(input_dim=vocab_size,
output_dim=32))
base_lstm_model_t.add(LSTM(32))
base_lstm_model_t.add(Dense(y_train.shape[1],
activation='softmax'))

# Compile the model
base_lstm_model_t.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
base_lstm_model_t.summary()

# Train the model
base_lstm_history_t = base_lstm_model_t.fit(X_train_pad, y_train,
epochs=30, validation_data=(X_test_pad, y_test))

import numpy as np
import matplotlib.pyplot as plt

def plot_model_performance(history):
```

Figure 12: Code for Base LSTM



```
    # Plot training & validation accuracy values
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
    print('Average accuracy: ',
np.mean(history.history['accuracy']))
    print('Average loss: ', np.mean(history.history['loss']))

plot_model_performance(base_lstm_history_t)
# Save the model
base_lstm_model_t.save('base_lstm_text_model.h5')
```

Figure 13: Code for Base LSTM



```
Average accuracy:  0.7823895821968715
Average loss:  0.6700021566271782
```

Figure 14: Accuarcy and Average loss of the Base LSTM

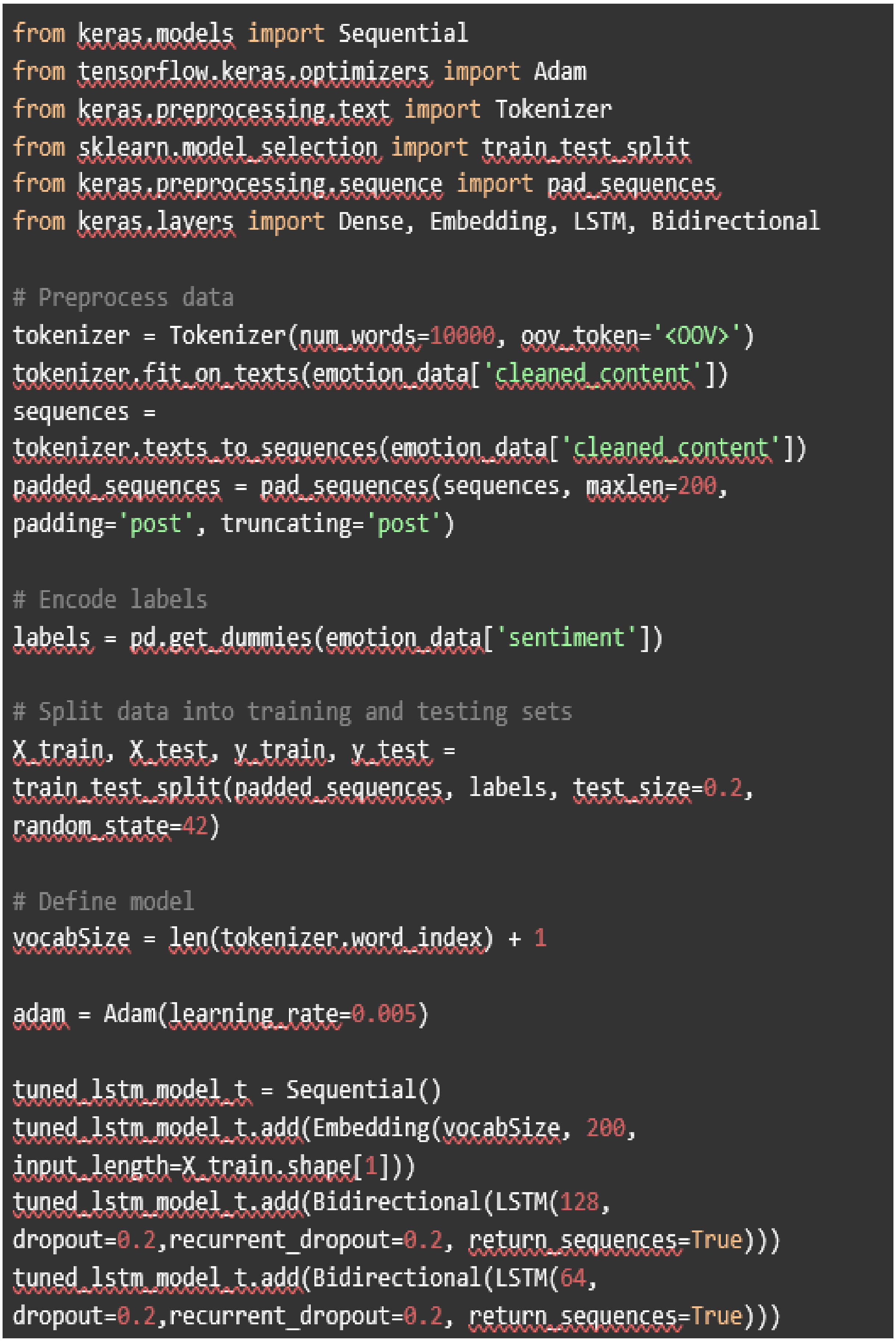
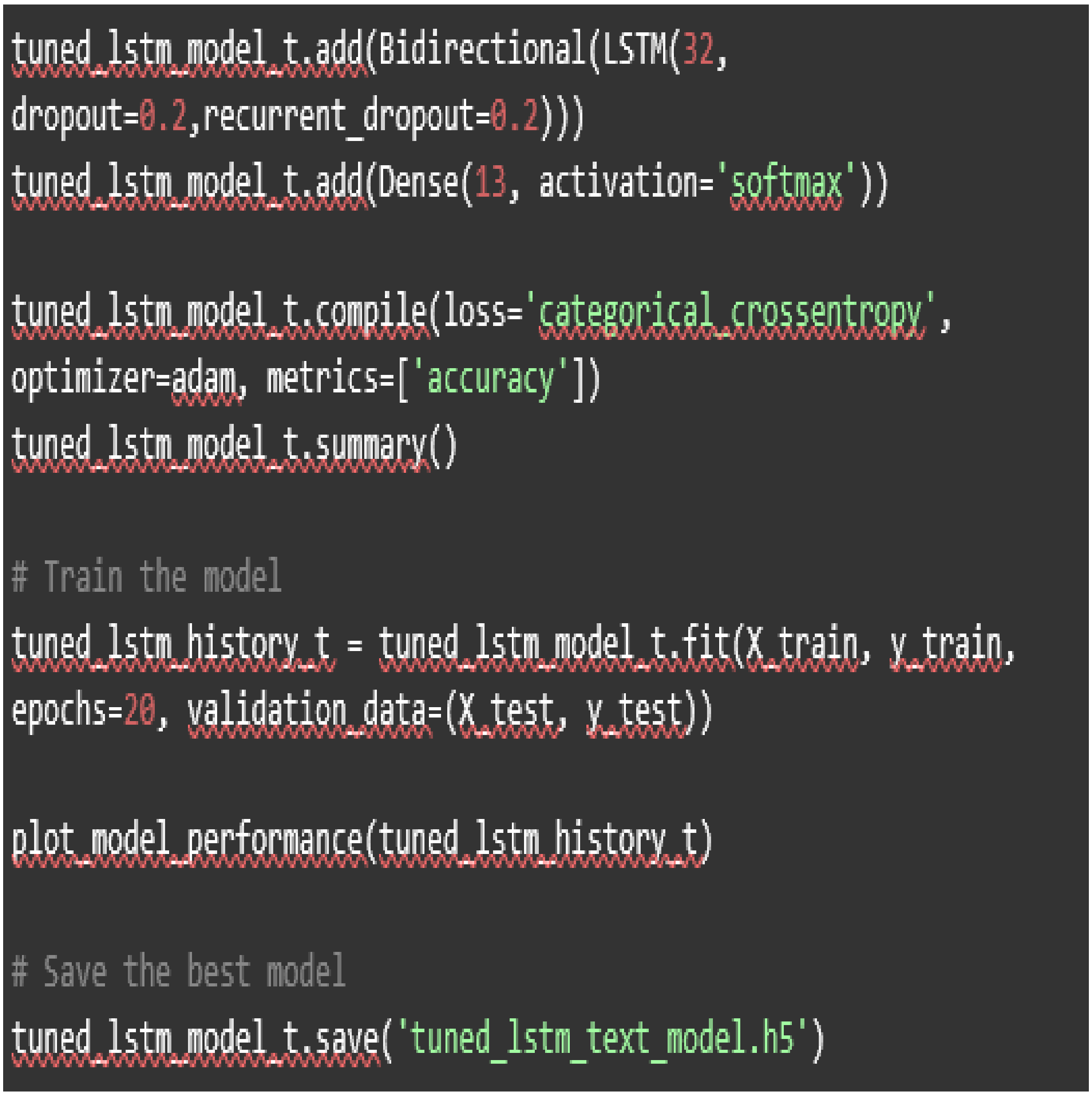
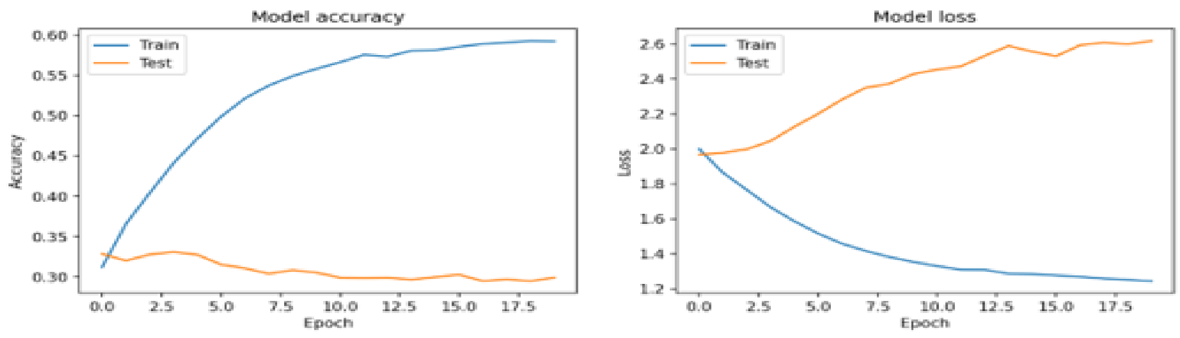

```
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Embedding, LSTM, Bidirectional

# Preprocess data
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
tokenizer.fit_on_texts(emotion_data['cleaned_content'])
sequences =
tokenizer.texts_to_sequences(emotion_data['cleaned_content'])
padded_sequences = pad_sequences(sequences, maxlen=200,
padding='post', truncating='post')

# Encode labels
labels = pd.get_dummies(emotion_data['sentiment'])

# Split data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(padded_sequences, labels, test_size=0.2,
random_state=42)

# Define model
vocabSize = len(tokenizer.word_index) + 1

adam = Adam(learning_rate=0.005)

tuned_lstm_model_t = Sequential()
tuned_lstm_model_t.add(Embedding(vocabSize, 200,
input_length=X_train.shape[1]))
tuned_lstm_model_t.add(Bidirectional(LSTM(128,
dropout=0.2,recurrent_dropout=0.2, return_sequences=True)))
tuned_lstm_model_t.add(Bidirectional(LSTM(64,
dropout=0.2,recurrent_dropout=0.2, return_sequences=True)))
```

Figure 15: Code for Tuned LSTM



```
tuned_lstm_model_t.add(Bidirectional(LSTM(32,
dropout=0.2,recurrent_dropout=0.2)))
tuned_lstm_model_t.add(Dense(13, activation='softmax'))

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

# Train the model
tuned_lstm_history_t = tuned_lstm_model_t.fit(X_train, y_train,
epochs=20, validation_data=(X_test, y_test))

plot_model_performance(tuned_lstm_history_t)

# Save the best model
tuned_lstm_model_t.save('tuned_lstm_text_model.h5')
```

Figure 16: Code for Tuned LSTM



```
Average accuracy:  0.5244093775749207
Average loss:  1.4420320749282838
```

Figure 17: Accuarcy and Average loss of the Tuned LSTM

5.4 Hybrid Model

This code in Figure 18 and Figure 19 :

- Prepares tools for processing text and building a neural network.
- Converts text into numbers, ensuring they're uniform in length.
- Transforms emotion labels into a format the model can understand.
- Splits the data into a part for learning and another for testing.
- Designs a combined (hybrid) model using both filtering techniques and memory units to predict emotions from text.
- Trains this combined model using the learning data.
- Displays how well the model learned over time.
- Saves the trained model for later use.

The Accuarcy and Average loss of the model is shown in Figure 20.

5.5 Classifying Custom Sample

This code in Figure 21 and Figure 22

- Sets up tools for working with text and models.
- Provides a list of sentences to identify emotions.
- Loads three trained models from files.
- Converts the sentences into a format the models understand.
- Uses the models to predict the emotion for each sentence.
- Displays the predicted emotions for each sentence, using each of the three models.

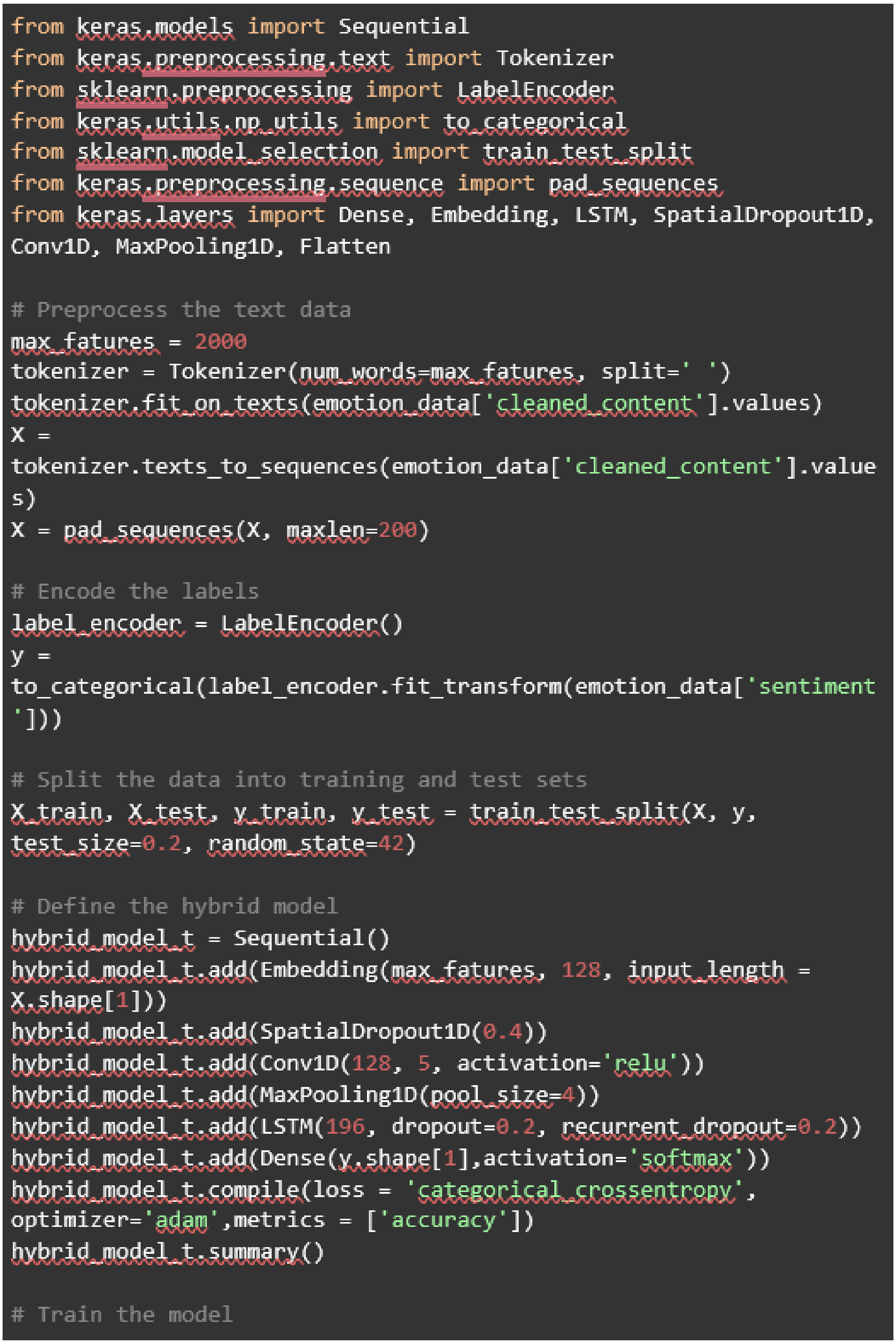
The classified report of the three models are shown in Figure 23.

6 Exploratory Data Analysis (Speech)

6.1 Loading and Preprocessing the Speech Data

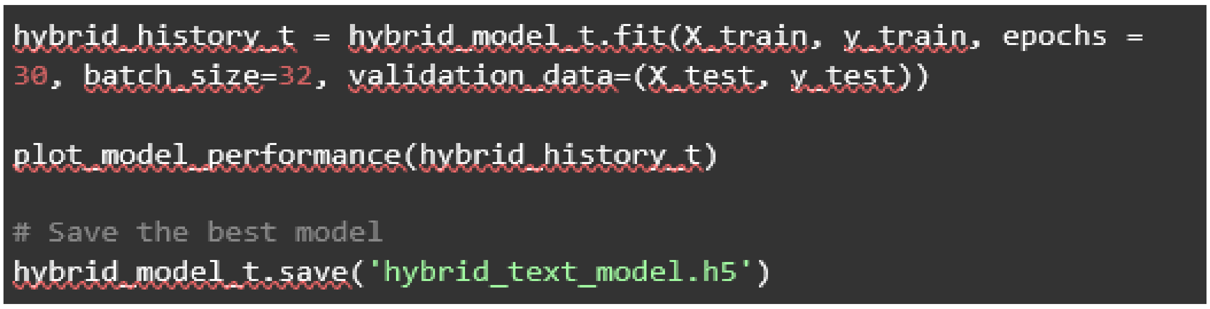
This code in Figure 24:

- Sets up tools to work with audio files.
- Defines a method to pull out meaningful characteristics from audio.
- Defines a method to determine the emotion based on a file's name.
- Goes through each audio file in a folder, extracts its characteristics and determines its emotion.
- Gather these characteristics and emotions into a table for viewing.



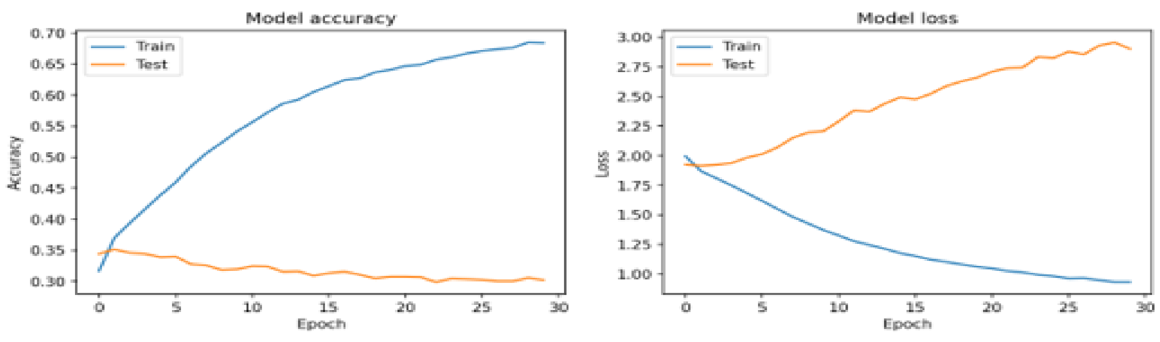
```
from keras.models import Sequential
from keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D,
Conv1D, MaxPooling1D, Flatten

# Preprocess the text data
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(emotion_data['cleaned_content'].values)
X =
tokenizer.texts_to_sequences(emotion_data['cleaned_content'].value
s)
X = pad_sequences(X, maxlen=200)

# Encode the labels
label_encoder = LabelEncoder()
y =
to_categorical(label_encoder.fit_transform(emotion_data['sentiment
']))

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the hybrid model
hybrid_model_t = Sequential()
hybrid_model_t.add(Embedding(max_fatures, 128, input_length =
X.shape[1]))
hybrid_model_t.add(SpatialDropout1D(0.4))
hybrid_model_t.add(Conv1D(128, 5, activation='relu'))
hybrid_model_t.add(MaxPooling1D(pool_size=4))
hybrid_model_t.add(LSTM(196, dropout=0.2, recurrent_dropout=0.2))
hybrid_model_t.add(Dense(y.shape[1],activation='softmax'))
hybrid_model_t.compile(loss = 'categorical_crossentropy',
optimizer='adam',metrics = ['accuracy'])
hybrid_model_t.summary()

# Train the model
```

Figure 18: Code for Hybrid model



```
hybrid_history_t = hybrid_model_t.fit(X_train, y_train, epochs =
30, batch_size=32, validation_data=(X_test, y_test))

plot_model_performance(hybrid_history_t)

# Save the best model
hybrid_model_t.save('hybrid_text_model.h5')
```

Figure 19: Code for Hybrid model



```
Average accuracy:  0.5726239601771037
Average loss:  1.270220955212911
```

Figure 20: Accuarcy and Average loss of the Hybrid Model

6.2 Analysing Data

This code Figure 25:

- Displays a summary of the data's characteristics.
- Shows how often each emotion appears in the data.
- Visualises how the data's characteristics relate to each other.
- Displays the distribution of each characteristic in the data.

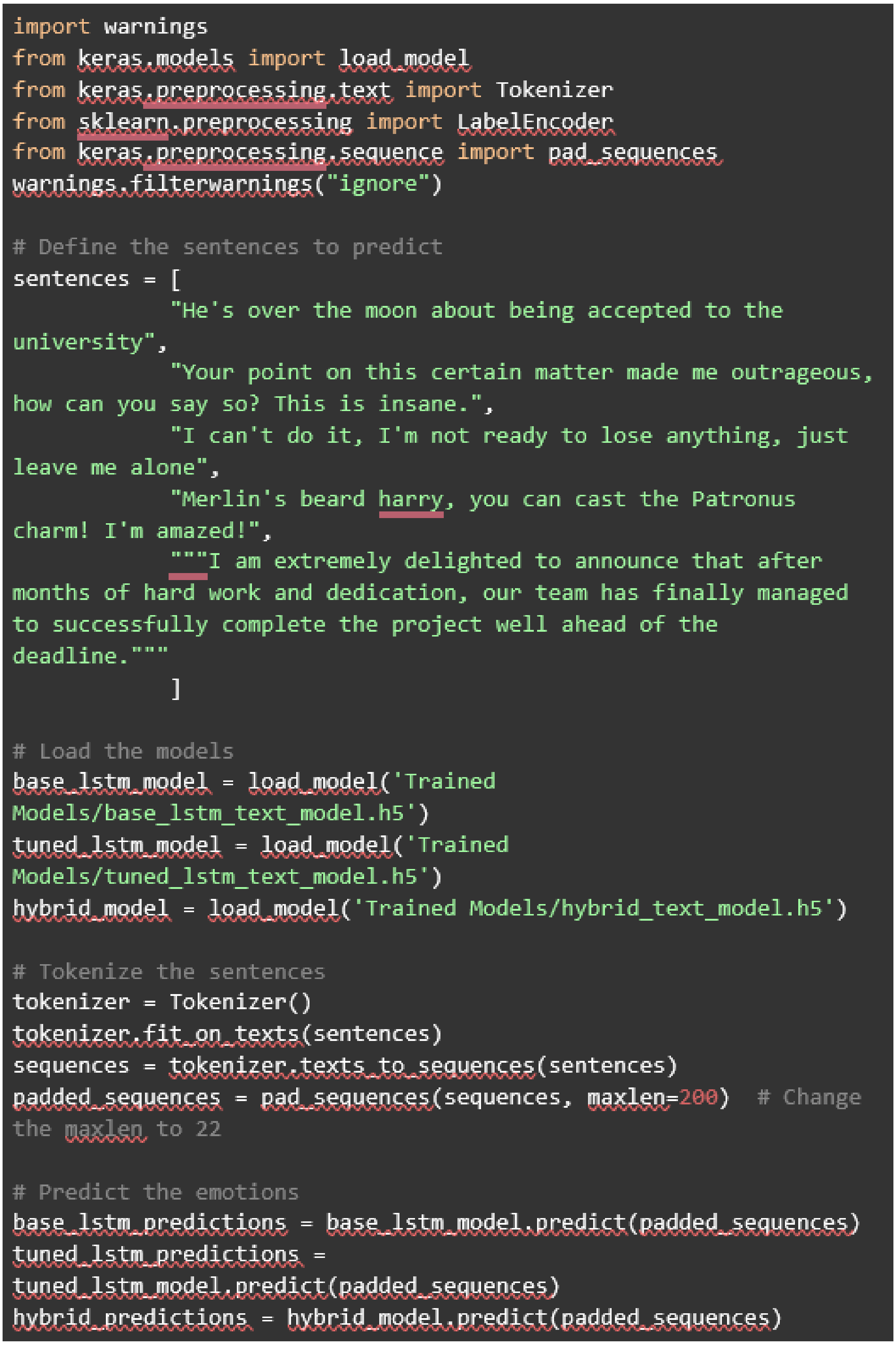
Figure 26 shows the distribution of emotions in the speech data. while Figure 27the Correlation Matrix of MFCC Features in speech. And Figure 28 Numerical Feature Distribution (MFCC).

7 7 Modelling for Speech Data

7.1 Machine Learning Model for Speech

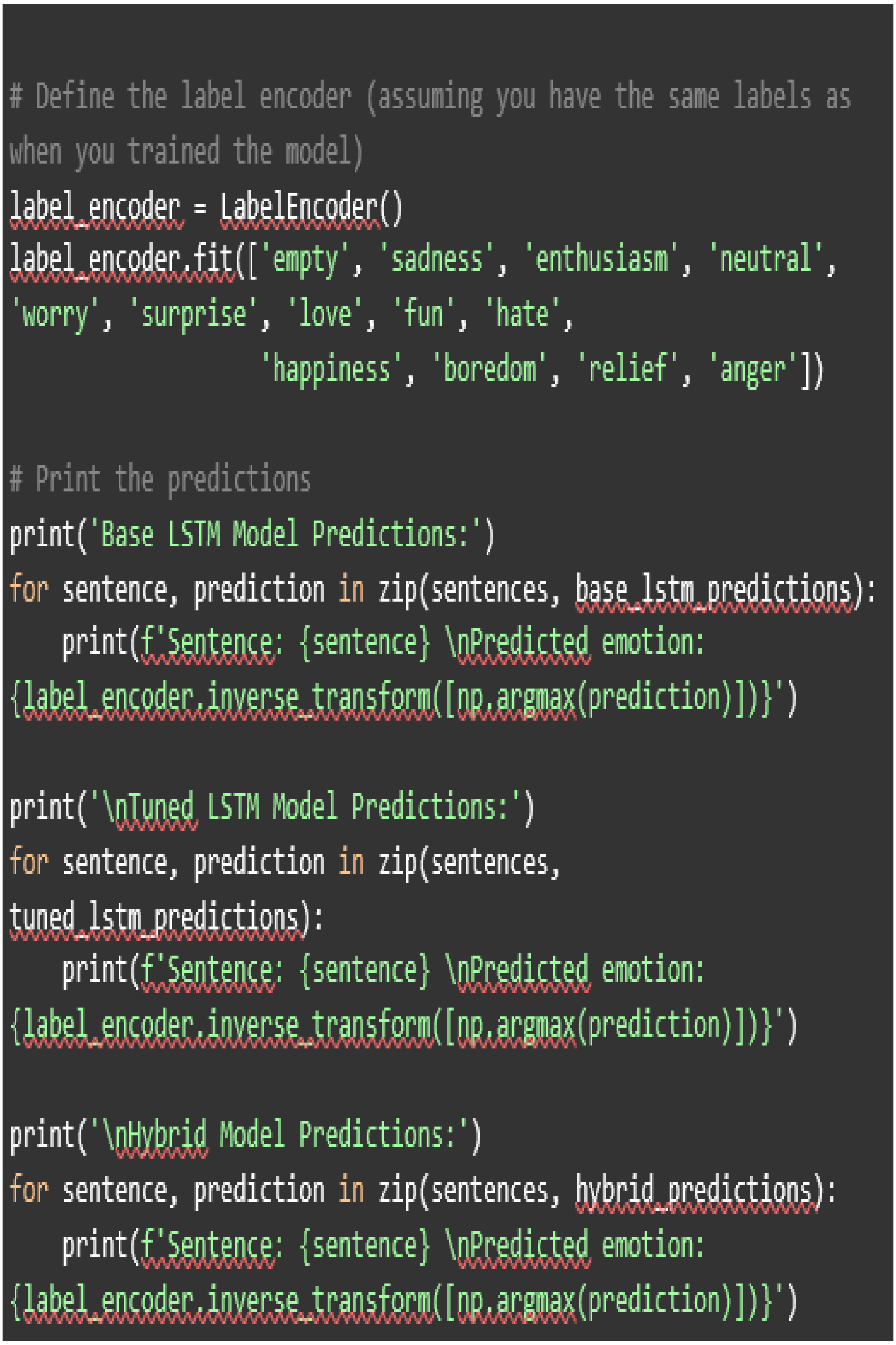
This code in Figure 29:

- Prepares tools for working with data and models.
- Splits the data into what we want to predict (emotions) and the characteristics used for predictions.



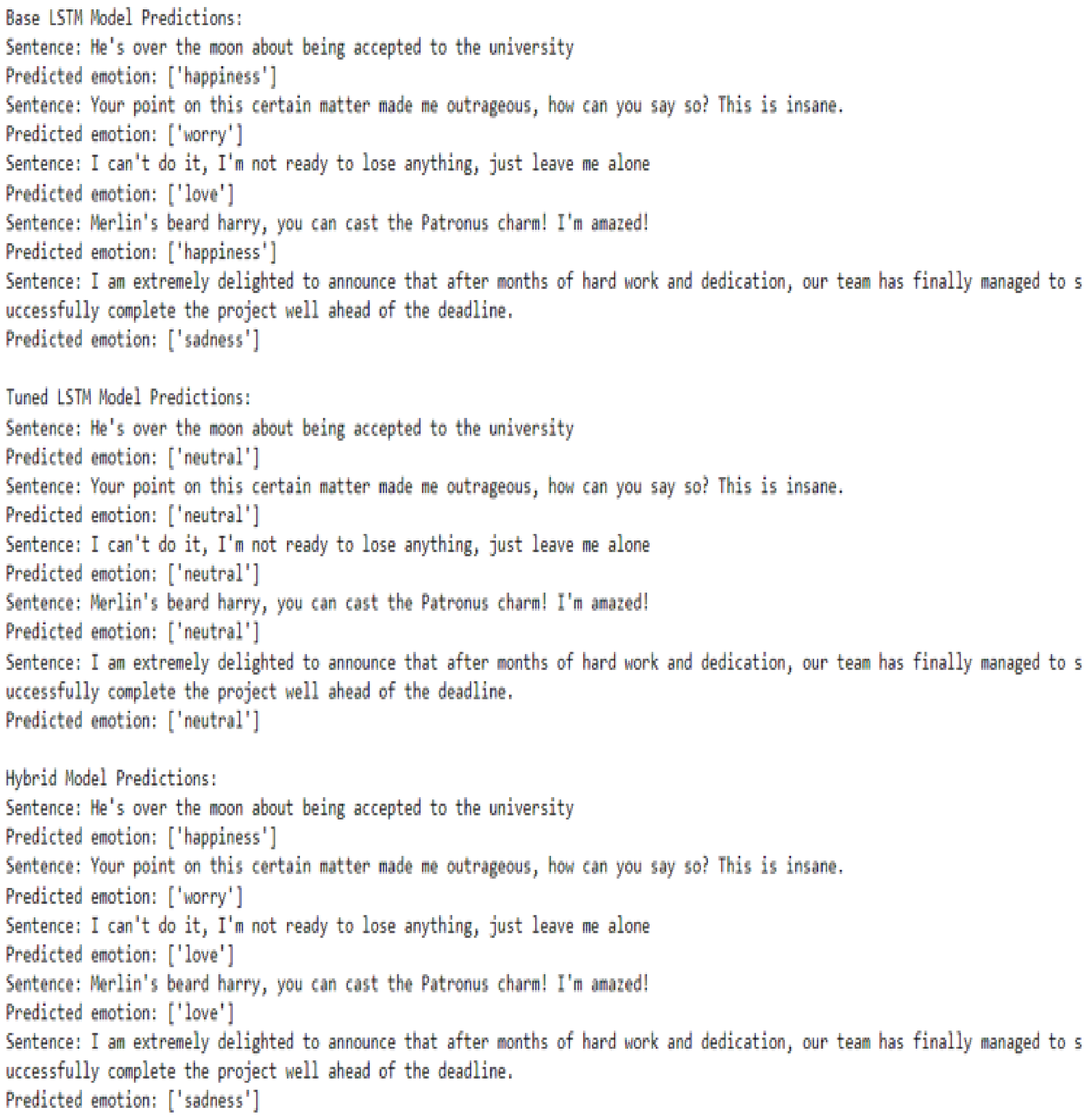
```
import warnings
from keras.models import load_model
from keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.sequence import pad_sequences
warnings.filterwarnings("ignore")

# Define the sentences to predict
sentences = [
            "He's over the moon about being accepted to the
university",
            "Your point on this certain matter made me outrageous,
how can you say so? This is insane.",
            "I can't do it, I'm not ready to lose anything, just
leave me alone",
            "Merlin's beard harry, you can cast the Patronus
charm! I'm amazed!",
            """I am extremely delighted to announce that after
months of hard work and dedication, our team has finally managed
to successfully complete the project well ahead of the
deadline."""
            ]

# Load the models
base_lstm_model = load_model('Trained
Models/base_lstm_text_model.h5')
tuned_lstm_model = load_model('Trained
Models/tuned_lstm_text_model.h5')
hybrid_model = load_model('Trained Models/hybrid_text_model.h5')

# Tokenize the sentences
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
sequences = tokenizer.texts_to_sequences(sentences)
padded_sequences = pad_sequences(sequences, maxlen=200)  # Change
the maxlen to 22

# Predict the emotions
base_lstm_predictions = base_lstm_model.predict(padded_sequences)
tuned_lstm_predictions =
tuned_lstm_model.predict(padded_sequences)
hybrid_predictions = hybrid_model.predict(padded_sequences)
```

Figure 21: Code for Classifying Custom Sample



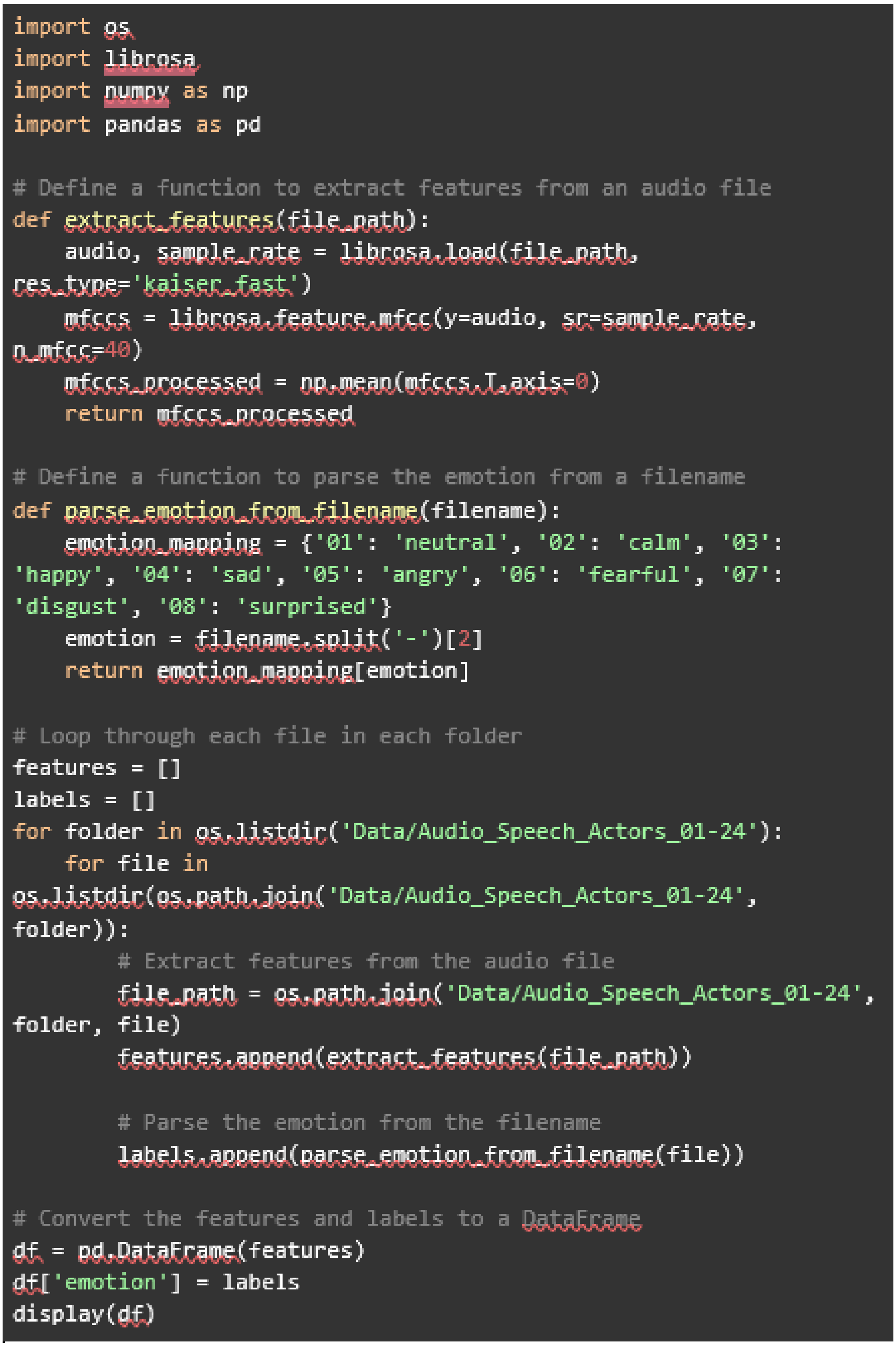
```
# Define the label encoder (assuming you have the same labels as
when you trained the model)
label_encoder = LabelEncoder()
label_encoder.fit(['empty', 'sadness', 'enthusiasm', 'neutral',
'worry', 'surprise', 'love', 'fun', 'hate',
                  'happiness', 'boredom', 'relief', 'anger'])

# Print the predictions
print('Base LSTM Model Predictions:')
for sentence, prediction in zip(sentences, base_lstm_predictions):
    print(f'Sentence: {sentence} \nPredicted emotion:
{label_encoder.inverse_transform([np.argmax(prediction)])}')

print('\nTuned LSTM Model Predictions:')
for sentence, prediction in zip(sentences,
tuned_lstm_predictions):
    print(f'Sentence: {sentence} \nPredicted emotion:
{label_encoder.inverse_transform([np.argmax(prediction)])}')

print('\nHybrid Model Predictions:')
for sentence, prediction in zip(sentences, hybrid_predictions):
    print(f'Sentence: {sentence} \nPredicted emotion:
{label_encoder.inverse_transform([np.argmax(prediction)])}')
```

Figure 22: Code for Classifying Custom Sample



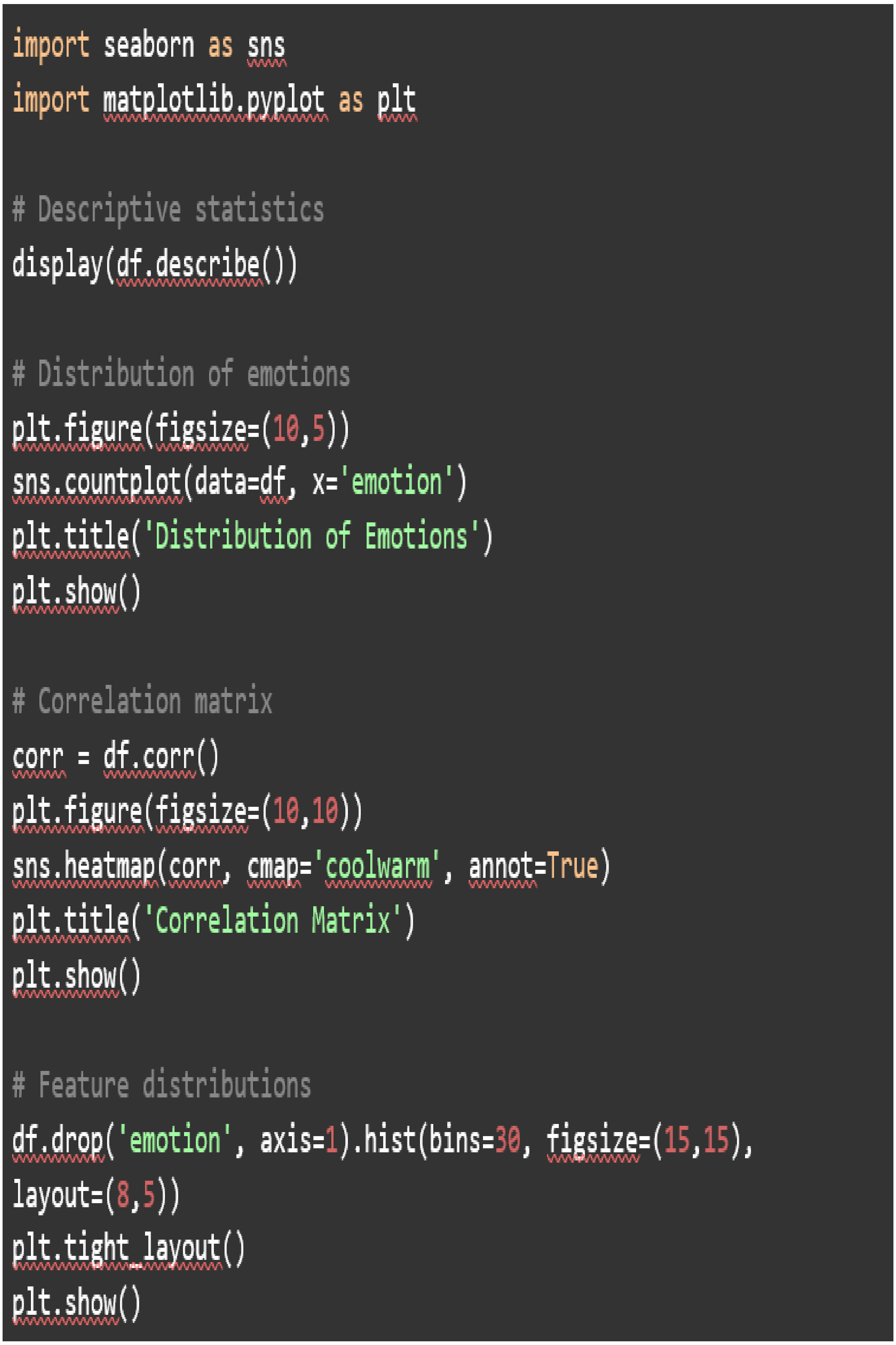
```
Base LSTM Model Predictions:
Sentence: He's over the moon about being accepted to the university
Predicted emotion: ['happiness']
Sentence: Your point on this certain matter made me outrageous, how can you say so? This is insane.
Predicted emotion: ['worry']
Sentence: I can't do it, I'm not ready to lose anything, just leave me alone
Predicted emotion: ['love']
Sentence: Merlin's beard harry, you can cast the Patronus charm! I'm amazed!
Predicted emotion: ['happiness']
Sentence: I am extremely delighted to announce that after months of hard work and dedication, our team has finally managed to s
uccessfully complete the project well ahead of the deadline.
Predicted emotion: ['sadness']

Tuned LSTM Model Predictions:
Sentence: He's over the moon about being accepted to the university
Predicted emotion: ['neutral']
Sentence: Your point on this certain matter made me outrageous, how can you say so? This is insane.
Predicted emotion: ['neutral']
Sentence: I can't do it, I'm not ready to lose anything, just leave me alone
Predicted emotion: ['neutral']
Sentence: Merlin's beard harry, you can cast the Patronus charm! I'm amazed!
Predicted emotion: ['neutral']
Sentence: I am extremely delighted to announce that after months of hard work and dedication, our team has finally managed to s
uccessfully complete the project well ahead of the deadline.
Predicted emotion: ['neutral']

Hybrid Model Predictions:
Sentence: He's over the moon about being accepted to the university
Predicted emotion: ['happiness']
Sentence: Your point on this certain matter made me outrageous, how can you say so? This is insane.
Predicted emotion: ['worry']
Sentence: I can't do it, I'm not ready to lose anything, just leave me alone
Predicted emotion: ['love']
Sentence: Merlin's beard harry, you can cast the Patronus charm! I'm amazed!
Predicted emotion: ['love']
Sentence: I am extremely delighted to announce that after months of hard work and dedication, our team has finally managed to s
uccessfully complete the project well ahead of the deadline.
Predicted emotion: ['sadness']
```

Figure 23: Classifying the Custom Samples using 3 Models



```
import os
import librosa
import numpy as np
import pandas as pd

# Define a function to extract features from an audio file
def extract_features(file_path):
    audio, sample_rate = librosa.load(file_path,
res_type='kaiser_fast')
    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate,
n_mfcc=40)
    mfccs_processed = np.mean(mfccs.T,axis=0)
    return mfccs_processed

# Define a function to parse the emotion from a filename
def parse_emotion_from_filename(filename):
    emotion_mapping = {'01': 'neutral', '02': 'calm', '03':
'happy', '04': 'sad', '05': 'angry', '06': 'fearful', '07':
'disgust', '08': 'surprised'}
    emotion = filename.split('-')[2]
    return emotion_mapping[emotion]

# Loop through each file in each folder
features = []
labels = []
for folder in os.listdir('Data/Audio_Speech_Actors_01-24'):
    for file in
os.listdir(os.path.join('Data/Audio_Speech_Actors_01-24',
folder)):
        # Extract features from the audio file
        file_path = os.path.join('Data/Audio_Speech_Actors_01-24',
folder, file)
        features.append(extract_features(file_path))

        # Parse the emotion from the filename
        labels.append(parse_emotion_from_filename(file))

# Convert the features and labels to a DataFrame
df = pd.DataFrame(features)
df['emotion'] = labels
display(df)
```

Figure 24: Loading and Preprocessing the Speech Data



```python
import seaborn as sns
import matplotlib.pyplot as plt

# Descriptive statistics
display(df.describe())

# Distribution of emotions
plt.figure(figsize=(10,5))
sns.countplot(data=df, x='emotion')
plt.title('Distribution of Emotions')
plt.show()

# Correlation matrix
corr = df.corr()
plt.figure(figsize=(10,10))
sns.heatmap(corr, cmap='coolwarm', annot=True)
plt.title('Correlation Matrix')
plt.show()

# Feature distributions
df.drop('emotion', axis=1).hist(bins=30, figsize=(15,15),
layout=(8,5))
plt.tight_layout()
plt.show()
```

Figure 25: Analysing Speech Data

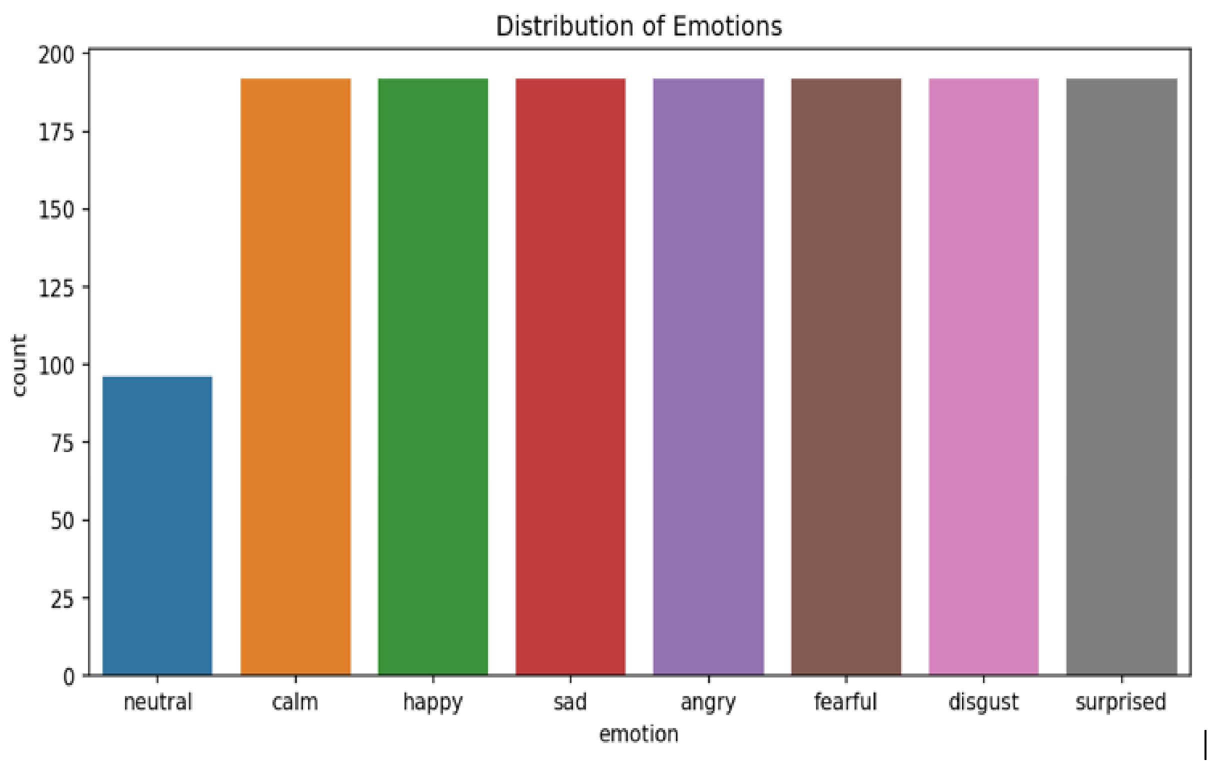


Figure 26: Distribution of Emotions

- Converts emotion names into numbers for the model to understand.
- Divides the data into a learning part and a testing
- Sets up three different methods to predict emotions based on the characteristics.
- Combines these methods into one unified method for v Teaches this unified method using the learning data.
- Checks the method's predictions on new data.
- Displays how well the method did in the test.

The classification report of the ensemble model for speech data is in Figure 30

7.2 Base LSTM for Speech

This code in Figure 31 and Figure 32 :

- Prepares tools for working with data and building a neural network.
- Splits the data into characteristics and the emotions to predict.
- Converts emotion names into a format the model can understand.
- Divides the data into a learning part and a testing part.
- Adjusts the data's shape for a special kind of model called LSTM.

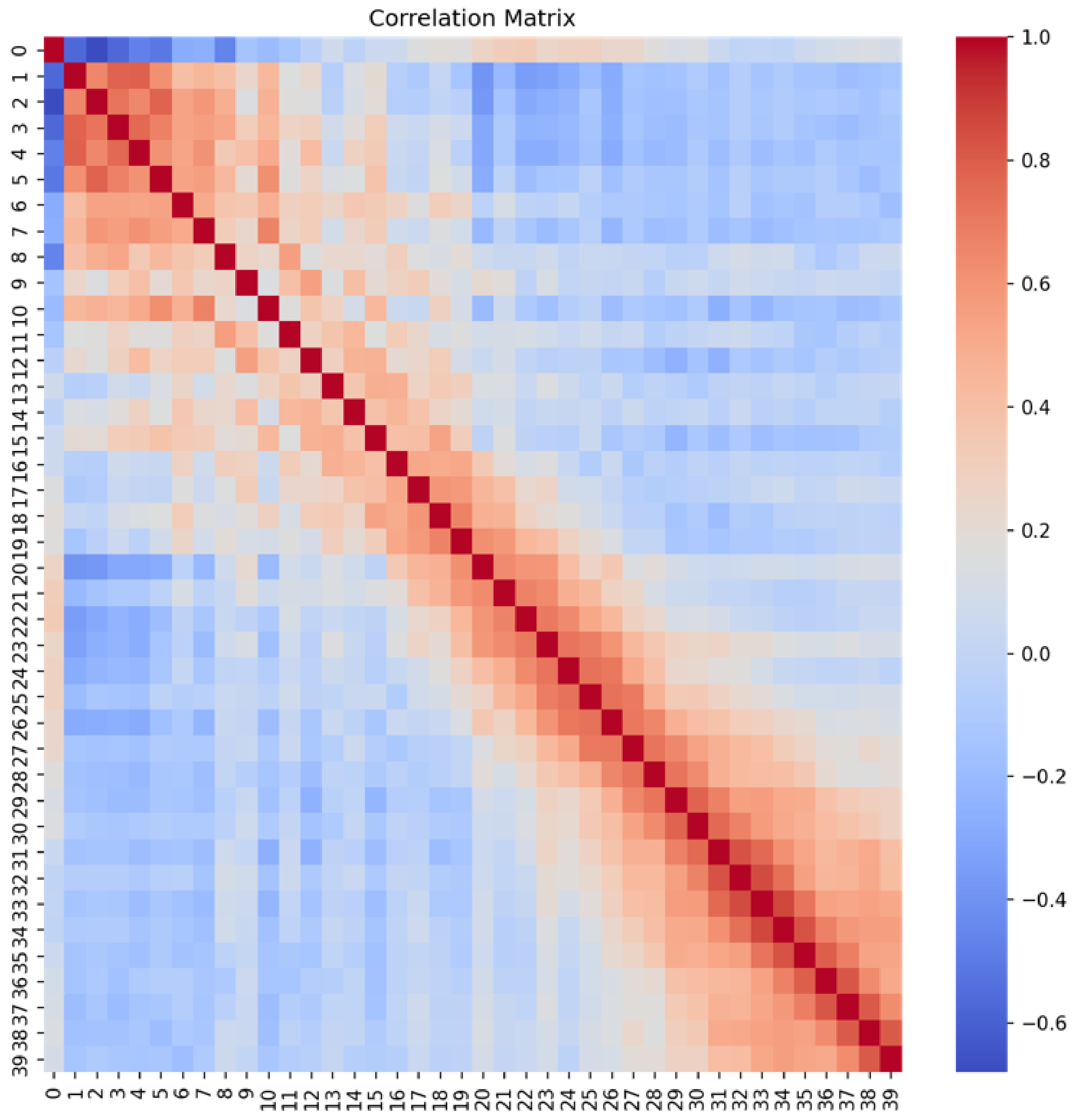


Figure 27: Correlation Matrix of MFCC Features in speech

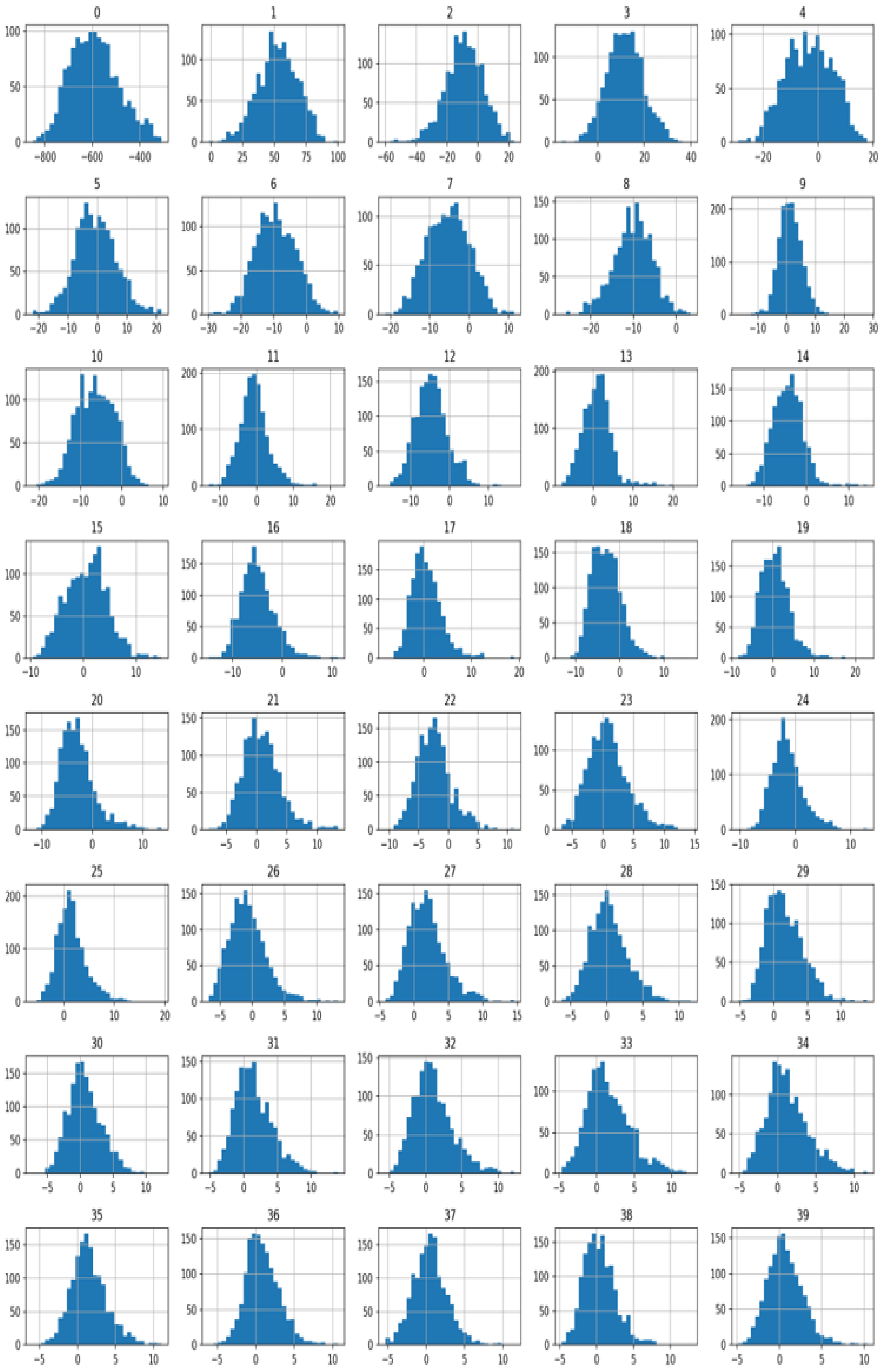
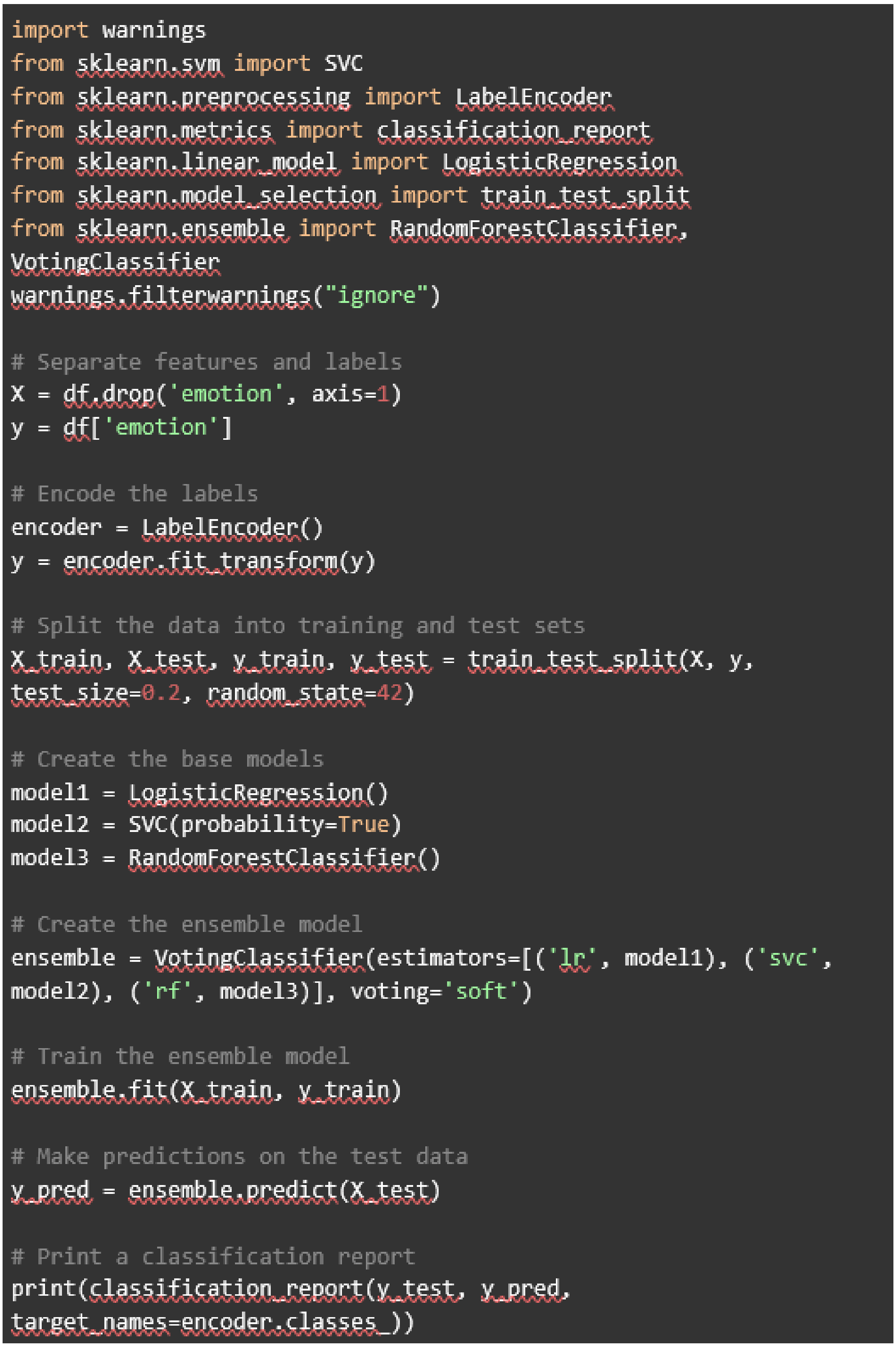
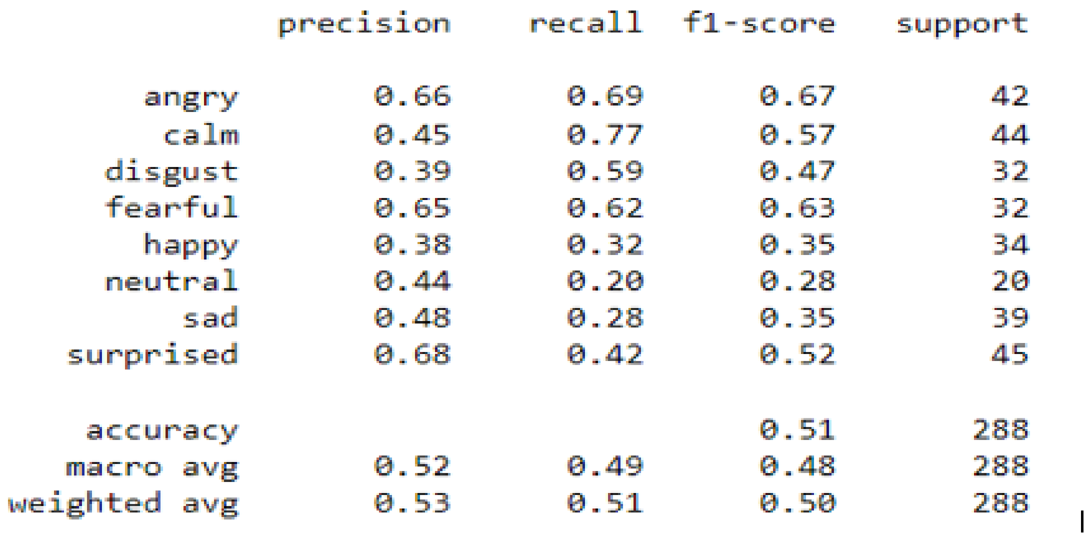


Figure 28: Numerical Feature Distribution (MFCC)



```python
import warnings
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier,
VotingClassifier
warnings.filterwarnings("ignore")

# Separate features and labels
X = df.drop('emotion', axis=1)
y = df['emotion']

# Encode the labels
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create the base models
model1 = LogisticRegression()
model2 = SVC(probability=True)
model3 = RandomForestClassifier()

# Create the ensemble model
ensemble = VotingClassifier(estimators=[('lr', model1), ('svc',
model2), ('rf', model3)], voting='soft')

# Train the ensemble model
ensemble.fit(X_train, y_train)

# Make predictions on the test data
y_pred = ensemble.predict(X_test)

# Print a classification report
print(classification_report(y_test, y_pred,
target_names=encoder.classes_))
```

Figure 29: Ensemble model for speech data



```
              precision    recall  f1-score   support

       angry       0.66      0.69      0.67        42
        calm       0.45      0.77      0.57        44
     disgust       0.39      0.59      0.47        32
     fearful       0.65      0.62      0.63        32
       happy       0.38      0.32      0.35        34
     neutral       0.44      0.20      0.28        20
         sad       0.48      0.28      0.35        39
   surprised       0.68      0.42      0.52        45

    accuracy                           0.51       288
   macro avg       0.52      0.49      0.48       288
weighted avg       0.53      0.51      0.50       288
```

Figure 30: classification report for speech data

- Designs an LSTM model to predict emotions based on the characteristics.
- Teaches this model using the learning data.
- Shows how well the model learned over time.
- Saves the trained model for future use.

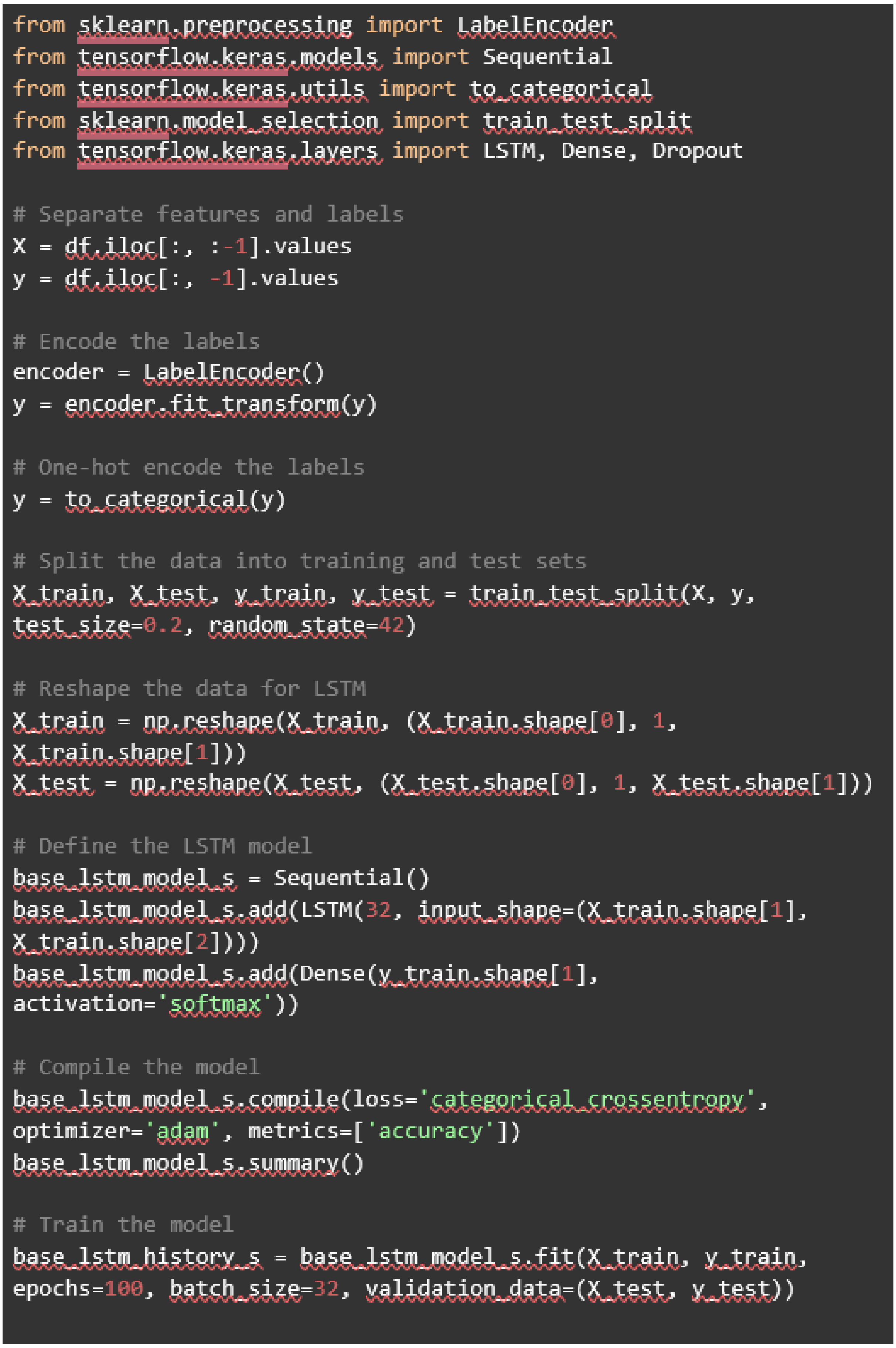
The Accuracy and Avg loss of the base LSTM model for speech is shown in Figure 33.

7.3 Tuned-LSTM for speech

This code in Figure 34 and Figure 35 :

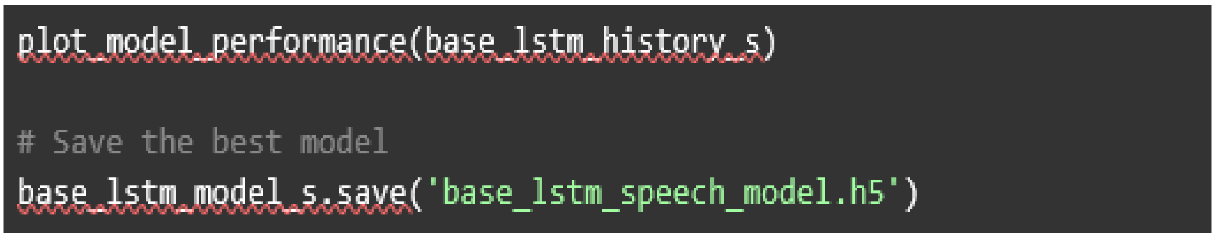
- Prepares tools for working with data and designing a neural network.
- Organises the data into characteristics and emotions.
- Converts emotion names into a model-friendly format.
- Divides the data into learning and testing portions.
- Adjusts the shape of the data for a neural network.
- Designs a more advanced model (with layers that remember and use both directions of data) to predict emotions.
- Trains this model with the learning data.
- Visualises the model's learning progress.
- Stores the model for later use.

The Accuracy and Avg loss of the Tuned LSTM model for speech is shown in Figure 36.



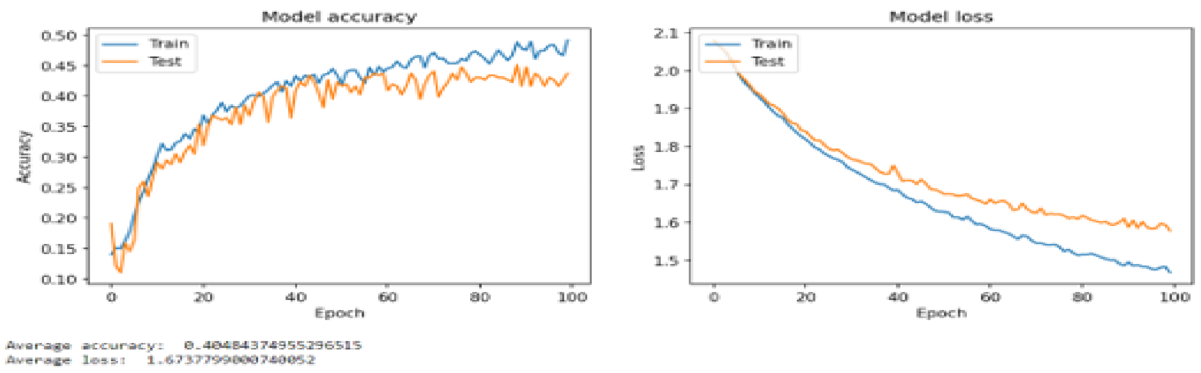
```python
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Separate features and labels
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Encode the labels
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# One-hot encode the labels
y = to_categorical(y)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Reshape the data for LSTM
X_train = np.reshape(X_train, (X_train.shape[0], 1,
X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Define the LSTM model
base_lstm_model_s = Sequential()
base_lstm_model_s.add(LSTM(32, input_shape=(X_train.shape[1],
X_train.shape[2])))
base_lstm_model_s.add(Dense(y_train.shape[1],
activation='softmax'))

# Compile the model
base_lstm_model_s.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
base_lstm_model_s.summary()

# Train the model
base_lstm_history_s = base_lstm_model_s.fit(X_train, y_train,
epochs=100, batch_size=32, validation_data=(X_test, y_test))
```

Figure 31: Base LSTM for speech data



```
plot_model_performance(base_lstm_history_s)

# Save the best model
base_lstm_model_s.save('base_lstm_speech_model.h5')
```

Figure 32: Base LSTM for speech data



```
Average accuracy:  0.40484374955296515
Average loss:  1.6737799000740052
```

Figure 33: Accuracy and Loss Plot of Speech Base-LSTM Model

7.4 Hybrid Model for speech

This code in Figure 37 and Figure 38:

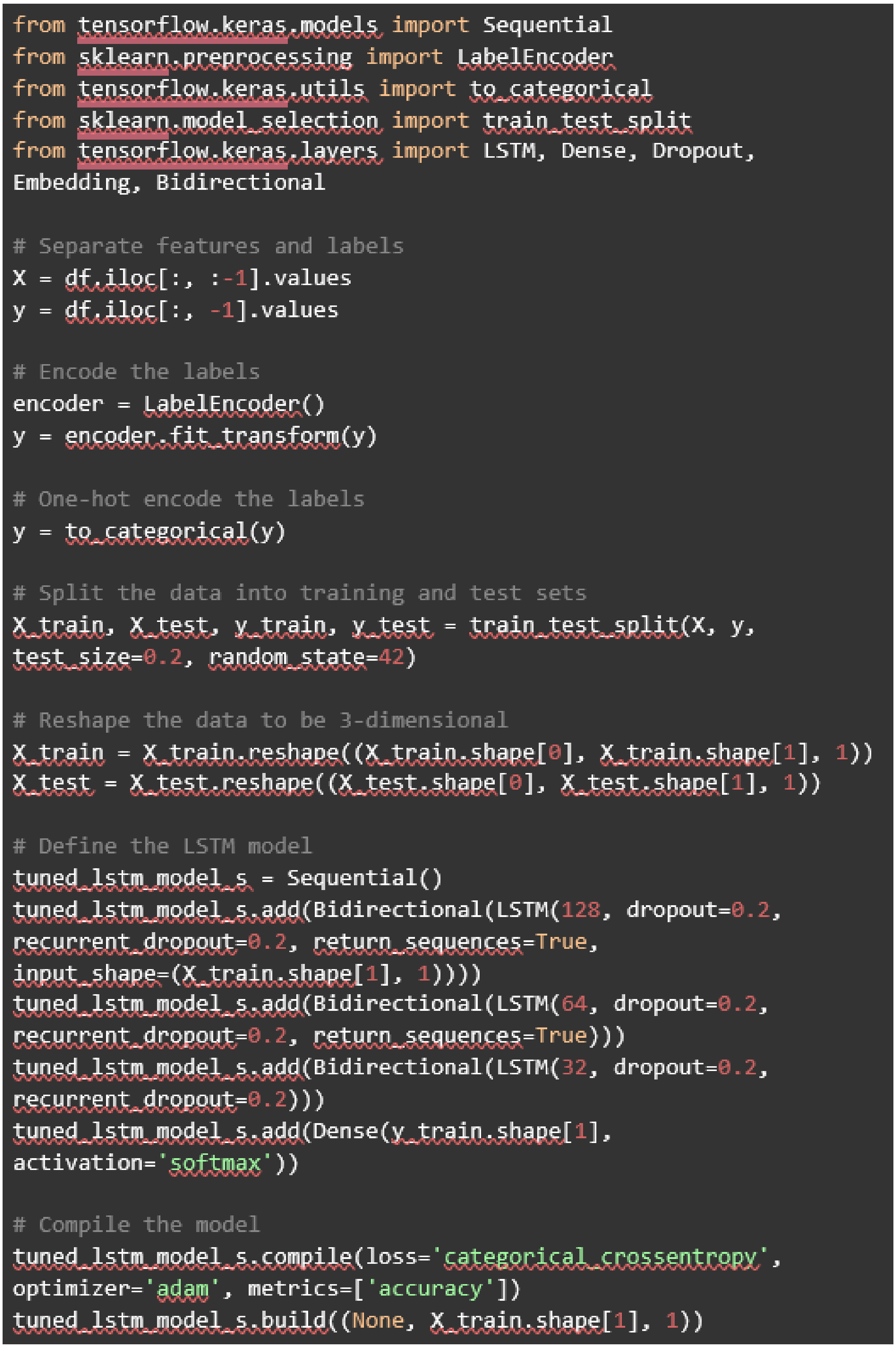
- Organises the data into characteristics and emotions.
- Converts emotion names into a model-friendly format.
- Splits the data into training and testing sets.
- Adjusts the data's shape for the neural network.
- Designs a combined model using both filtering techniques and memory units to predict emotions.
- Trains this model using the training data.
- Shows the model's learning progress.
- Saves the model for later use.

The Accuracy and Avg loss of the Hybrid Model for speech is shown in Figure 39.

7.5 Classifying Custom Sample for speech data

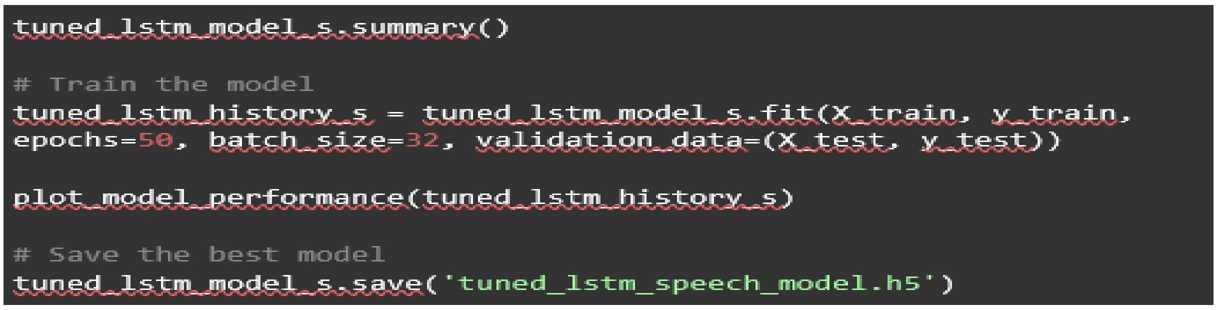
This code in Figure 40 and Figure 41:

- Prepares tools and loads pre-trained models for emotion prediction.



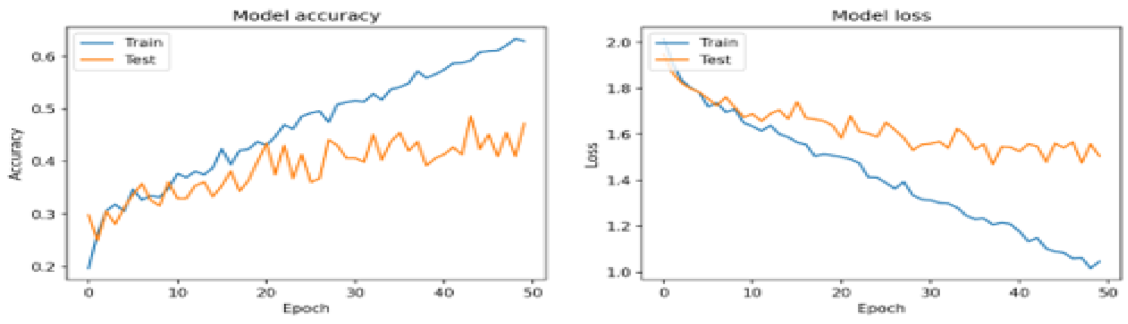
```
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import LSTM, Dense, Dropout,
Embedding, Bidirectional

# Separate features and labels
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Encode the labels
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# One-hot encode the labels
y = to_categorical(y)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Reshape the data to be 3-dimensional
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Define the LSTM model
tuned_lstm_model_s = Sequential()
tuned_lstm_model_s.add(Bidirectional(LSTM(128, dropout=0.2,
recurrent_dropout=0.2, return_sequences=True,
input_shape=(X_train.shape[1], 1))))
tuned_lstm_model_s.add(Bidirectional(LSTM(64, dropout=0.2,
recurrent_dropout=0.2, return_sequences=True)))
tuned_lstm_model_s.add(Bidirectional(LSTM(32, dropout=0.2,
recurrent_dropout=0.2)))
tuned_lstm_model_s.add(Dense(y_train.shape[1],
activation='softmax'))

# Compile the model
tuned_lstm_model_s.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
tuned_lstm_model_s.build((None, X_train.shape[1], 1))
```

Figure 34: Tuned LSTM for speech data



```
tuned_lstm_model_s.summary()

# Train the model
tuned_lstm_history_s = tuned_lstm_model_s.fit(X_train, y_train,
epochs=50, batch_size=32, validation_data=(X_test, y_test))

plot_model_performance(tuned_lstm_history_s)

# Save the best model
tuned_lstm_model_s.save('tuned_lstm_speech_model.h5')
```

Figure 35: Tuned LSTM for speech data

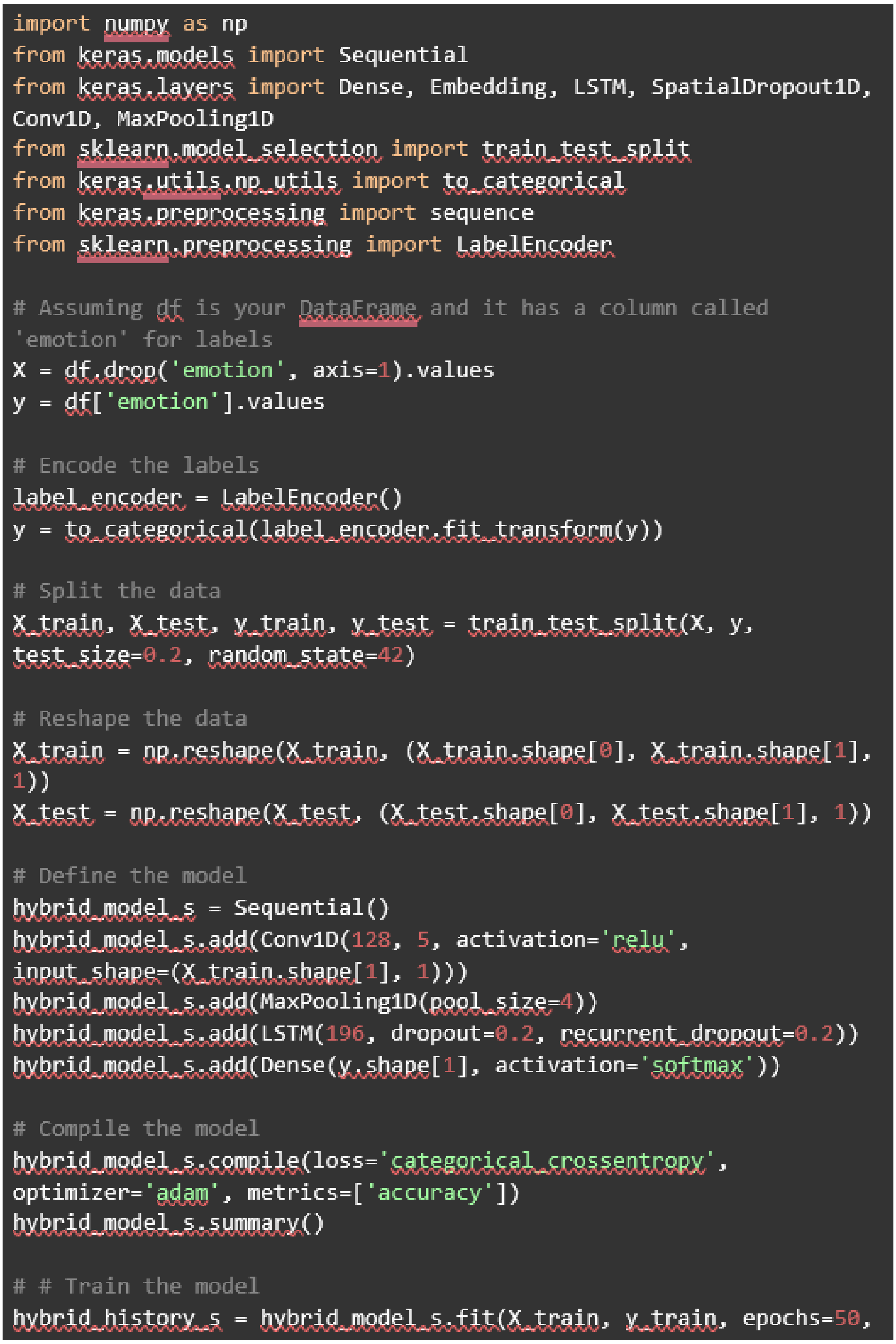


```
Average accuracy:  0.46611111164093016
Average loss:  1.4224433612823486
```

Figure 36: Accuracy and Loss Plot of Speech Tuned-LSTM Model

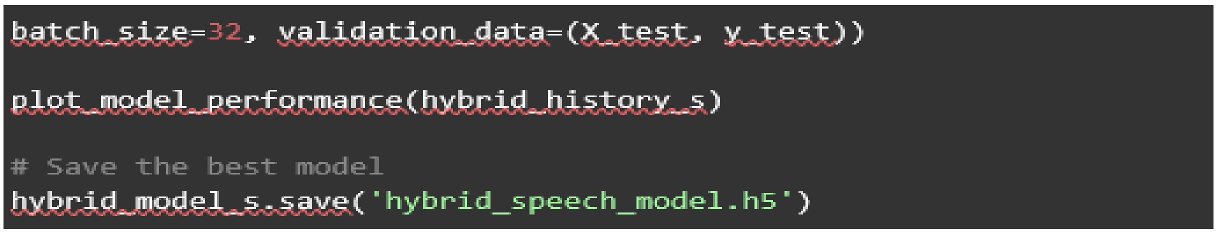
- Lists several audio files.
- For each audio file:
  - Identifies the actual emotion from its name.
  - Loads the audio and processes it to extract features.
  - Adjusts the shape of the features for different models.
  - Uses each model to predict the emotion from the audio's features.
  - Displays the real emotion and the predicted emotions.

The classification of the samples using three models are shown in Figure 42



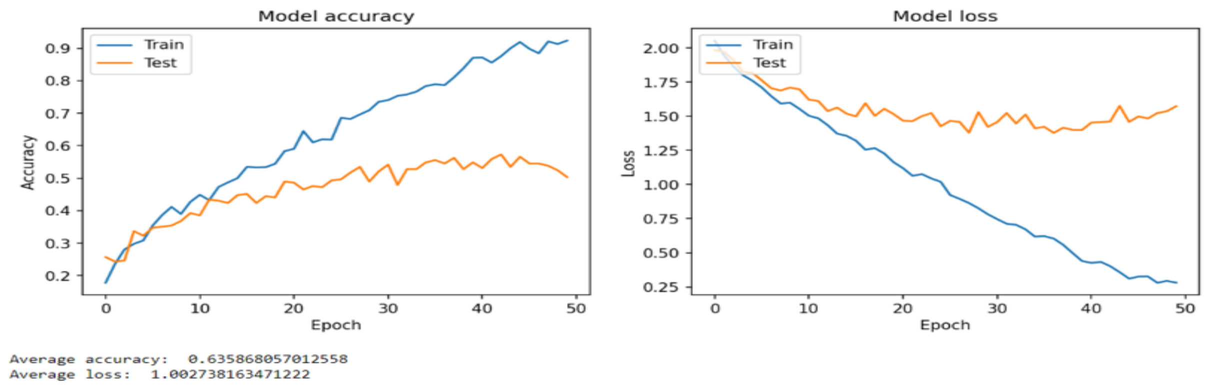
```python
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D,
Conv1D, MaxPooling1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.preprocessing import sequence
from sklearn.preprocessing import LabelEncoder

# Assuming df is your DataFrame and it has a column called
'emotion' for labels
X = df.drop('emotion', axis=1).values
y = df['emotion'].values

# Encode the labels
label_encoder = LabelEncoder()
y = to_categorical(label_encoder.fit_transform(y))

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Reshape the data
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],
1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Define the model
hybrid_model_s = Sequential()
hybrid_model_s.add(Conv1D(128, 5, activation='relu',
input_shape=(X_train.shape[1], 1)))
hybrid_model_s.add(MaxPooling1D(pool_size=4))
hybrid_model_s.add(LSTM(196, dropout=0.2, recurrent_dropout=0.2))
hybrid_model_s.add(Dense(y.shape[1], activation='softmax'))

# Compile the model
hybrid_model_s.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
hybrid_model_s.summary()

# # Train the model
hybrid_history_s = hybrid_model_s.fit(X_train, y_train, epochs=50,
```

Figure 37: Hybrid Model for speech data



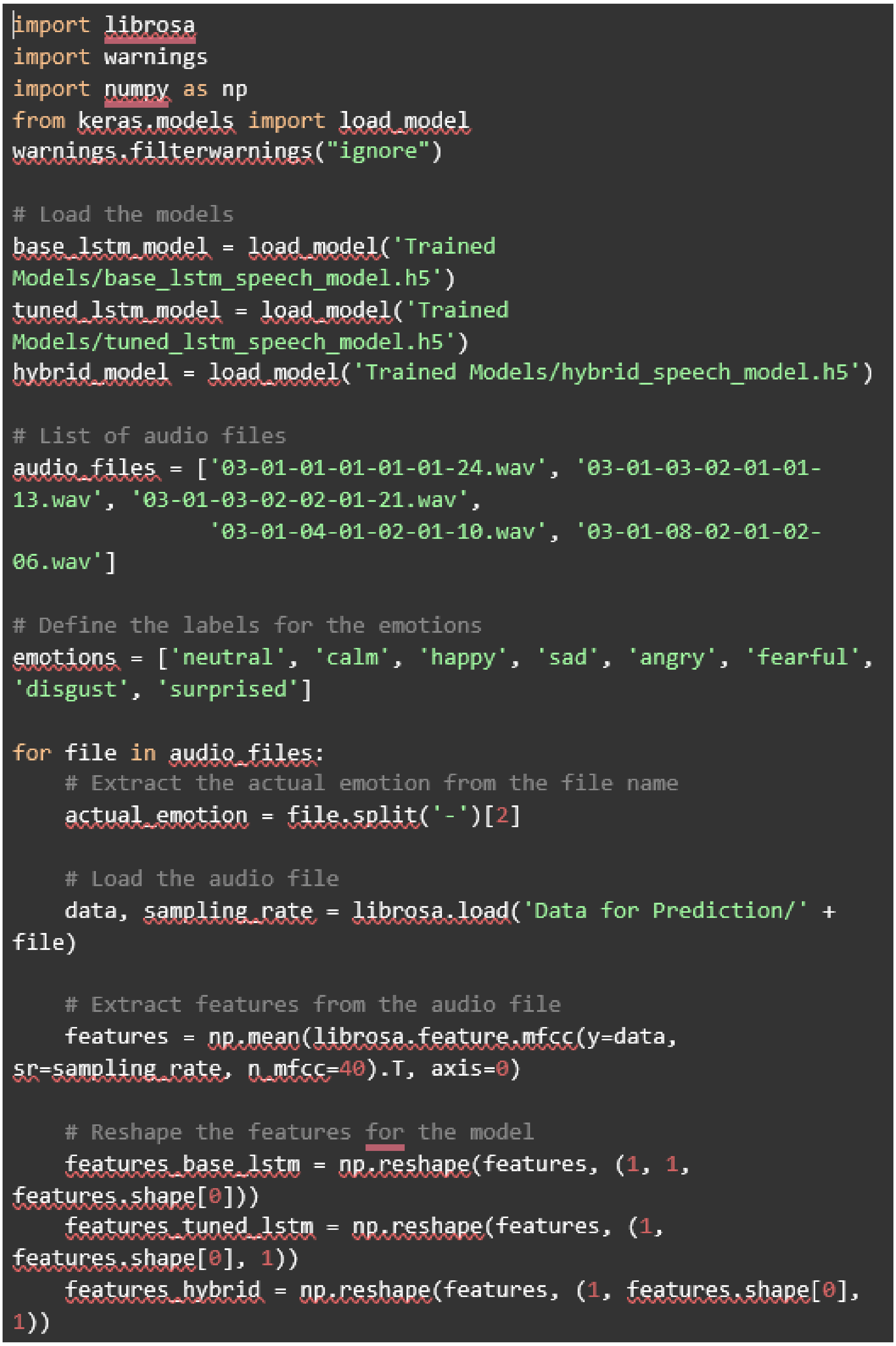
```
batch_size=32, validation_data=(X_test, y_test))

plot_model_performance(hybrid_history_s)

# Save the best model
hybrid_model_s.save('hybrid_speech_model.h5')
```

Figure 38: Hybrid Model for speech data



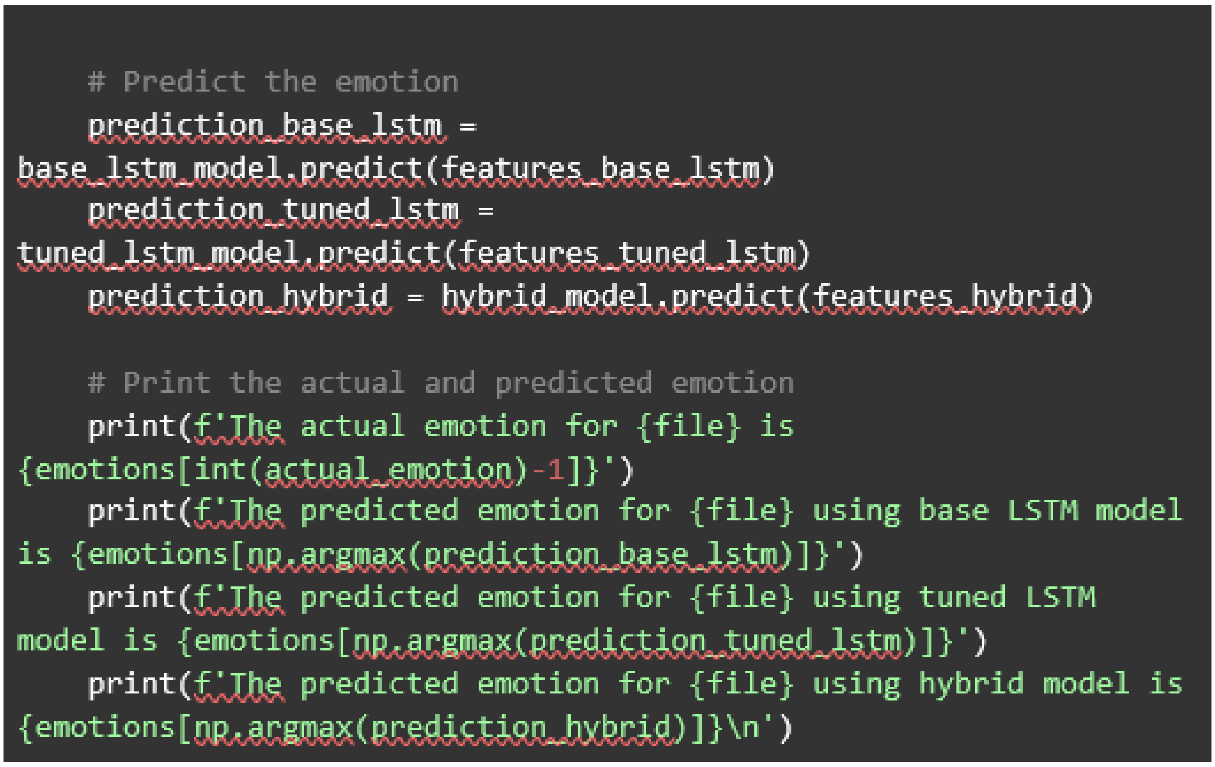
```
Average accuracy:  0.635868057012558
Average loss:  1.002738163471222
```

Figure 39: Accuracy and Loss Plot of Speech Hybrid Model Model



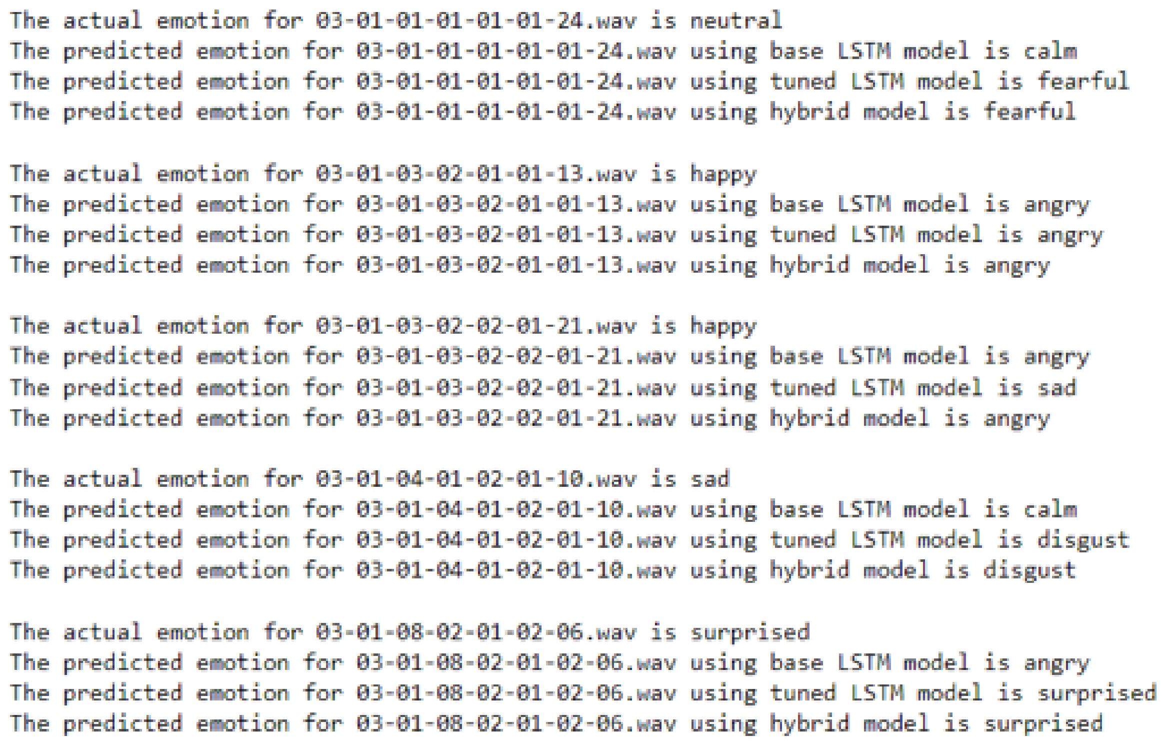
```python
import librosa
import warnings
import numpy as np
from keras.models import load_model
warnings.filterwarnings("ignore")

# Load the models
base_lstm_model = load_model('Trained
Models/base_lstm_speech_model.h5')
tuned_lstm_model = load_model('Trained
Models/tuned_lstm_speech_model.h5')
hybrid_model = load_model('Trained Models/hybrid_speech_model.h5')

# List of audio files
audio_files = ['03-01-01-01-01-01-24.wav', '03-01-03-02-01-01-
13.wav', '03-01-03-02-02-01-21.wav',
               '03-01-04-01-02-01-10.wav', '03-01-08-02-01-02-
06.wav']

# Define the labels for the emotions
emotions = ['neutral', 'calm', 'happy', 'sad', 'angry', 'fearful',
'disgust', 'surprised']

for file in audio_files:
    # Extract the actual emotion from the file name
    actual_emotion = file.split('-')[2]

    # Load the audio file
    data, sampling_rate = librosa.load('Data for Prediction/' +
file)

    # Extract features from the audio file
    features = np.mean(librosa.feature.mfcc(y=data,
sr=sampling_rate, n_mfcc=40).T, axis=0)

    # Reshape the features for the model
    features_base_lstm = np.reshape(features, (1, 1,
features.shape[0]))
    features_tuned_lstm = np.reshape(features, (1,
features.shape[0], 1))
    features_hybrid = np.reshape(features, (1, features.shape[0],
1))
```

Figure 40: Classifying Custom Sample for speech data



```
    # Predict the emotion
    prediction_base_lstm =
base_lstm_model.predict(features_base_lstm)
    prediction_tuned_lstm =
tuned_lstm_model.predict(features_tuned_lstm)
    prediction_hybrid = hybrid_model.predict(features_hybrid)

    # Print the actual and predicted emotion
    print(f'The actual emotion for {file} is
{emotions[int(actual_emotion)-1]}')
    print(f'The predicted emotion for {file} using base LSTM model
is {emotions[np.argmax(prediction_base_lstm)]}')
    print(f'The predicted emotion for {file} using tuned LSTM
model is {emotions[np.argmax(prediction_tuned_lstm)]}')
    print(f'The predicted emotion for {file} using hybrid model is
{emotions[np.argmax(prediction_hybrid)]}\n')
```

Figure 41: Classifying Custom Sample for speech data



```
The actual emotion for 03-01-01-01-01-01-24.wav is neutral
The predicted emotion for 03-01-01-01-01-01-24.wav using base LSTM model is calm
The predicted emotion for 03-01-01-01-01-01-24.wav using tuned LSTM model is fearful
The predicted emotion for 03-01-01-01-01-01-24.wav using hybrid model is fearful

The actual emotion for 03-01-03-02-01-01-13.wav is happy
The predicted emotion for 03-01-03-02-01-01-13.wav using base LSTM model is angry
The predicted emotion for 03-01-03-02-01-01-13.wav using tuned LSTM model is angry
The predicted emotion for 03-01-03-02-01-01-13.wav using hybrid model is angry

The actual emotion for 03-01-03-02-02-01-21.wav is happy
The predicted emotion for 03-01-03-02-02-01-21.wav using base LSTM model is angry
The predicted emotion for 03-01-03-02-02-01-21.wav using tuned LSTM model is sad
The predicted emotion for 03-01-03-02-02-01-21.wav using hybrid model is angry

The actual emotion for 03-01-04-01-02-01-10.wav is sad
The predicted emotion for 03-01-04-01-02-01-10.wav using base LSTM model is calm
The predicted emotion for 03-01-04-01-02-01-10.wav using tuned LSTM model is disgust
The predicted emotion for 03-01-04-01-02-01-10.wav using hybrid model is disgust

The actual emotion for 03-01-08-02-01-02-06.wav is surprised
The predicted emotion for 03-01-08-02-01-02-06.wav using base LSTM model is angry
The predicted emotion for 03-01-08-02-01-02-06.wav using tuned LSTM model is surprised
The predicted emotion for 03-01-08-02-01-02-06.wav using hybrid model is surprised
```

Figure 42: Classifying the Custom Speech Samples using 3 Models