BÜYÜK DİL MODELLERİNE GİRİŞ
DÖNEM PROJESİ RAPORU

Büyük Dil Modelleriyle Duygu Analizinin

Kişisel Günlüklerde Kullanımı

Seçil Şener - 200701077
Şevval Kapçak - 200701073

GİRİŞ

Duygusal zeka, günlük davranışlarımızı ve etkileşimlerimizi önemli ölçüde etkiler. Duygusal ipuçlarını anlamak ve bunlara yanıt vermek, insanlara problem çözmede belirgin bir avantaj sağlar.

Büyük Dil Modelleri (LLM'ler), insanların yazılı ve sözlü ifadelerini anlama kapasitesine sahiptir. Duygu analizi, bu modellerin insanların metinlerindeki duygusal tonları anlamasına ve etkileşimleri daha iyi değerlendirmesine olanak tanır. Kişinin günlük yaşamında hissettiği duyguları anlamasına yardımcı olur. Bu, bireyin kendi iç dünyasını keşfetmesini, duygusal süreçleri anlamasını ve kişisel gelişimine katkıda bulunmasını sağlar.

LLM'ler, kişisel günlük programımızdaki yazılardan çıkarılan duygu analizi sonuçlarıyla kullanıcının duygusal durumunu izleyebilir. Bu, kullanıcıya zaman içindeki duygusal değişiklikleri anlama ve takip etme fırsatı sunar. Duygu analizi, kişisel günlükler üzerinde yapıldığında, kullanıcının günlük yazılarındaki duygu tonlarına göre kendini nasıl hissettiğini anlamak için kullanılabilir. Bu, kişisel gelişim hedeflerini belirleme ve destekleme sürecine katkıda bulunabilir.

Bu projenin amacı, kullanıcının günlük tutmasına olanak sağlayan ve bu günlük yazıları analiz edip kişinin duygu durumuna göre destek mesajı ve beraberinde müzik önerisi sunan bir program tasarlamaktır.

VERİ SETİ

İnternet üzerinden bize sunulan veri setleri incelendikten sonra hugging face'de bulunan "dair-ai/emotion" adlı dataset seçilmiştir.
Bu veri seti metin tabanlı duygusal etiketlemeye odaklanan bir duygu analizi veri setidir. Metin örnekleri, farklı duygusal kategorilere (öfke, korku, sevinç, aşk, üzüntü ve şaşkınlık) ait etiketlerle etiketlenmiştir.

Veri Seti Bölümleri

Veri seti 2 konfigürasyona sahiptir:

split: Toplamda 20,000 örneği içeren ve eğitim, doğrulama ve bölme (test) olarak bölünen bir konfigürasyon.
unsplit: Tek bir eğitim bölümünde toplamda 416,809 örneği içeren bir konfigürasyon.

name	train	validation	test
split	16000	2000	2000
unsplit	416809	n/a	n/a

Veri Seti Örneği

Veri setinden bir örnek aşağıdaki gibi görünmektedir:


```
{
 "text": "im feeling quite sad and sorry for myself but ill snap out of it soon",
 "label": 0

}
```


Veri Alanları

Veri setinde bulunan veri alanları şunlardır:
text: Bir dize (string) özellik.

label: Sınıflandırma etiketi, olası değerleri arasında sadness (üzüntü) (0), joy (sevinç) (1), love (aşk) (2), anger (öfke) (3), fear (korku) (4), surprise (şaşkınlık) (5) bulunan bir özellik.

TEXT	LABEL	OBJECT
im feeling quite sad and sorry for myself but ill snap out of it soon	sadness	String
i feel romantic too	love	String
i do not feel reassured anxiety is on each side	joy	String
i now feel compromised and skeptical of the value of every unit of work i put in	fear	String
i feel kinda appalled that she feels like she needs to explain in wide and lenghth her body measures etc pp	anger	String
i have seen heard and read over the past couple of days i am left feeling impressed by more than a few companies	suprise	String

Kullanılan Kütüphaneler:

datasets: Veri setlerini yüklemek ve işlemek için kullanılan kütüphanedir.

transformers: Hugging Face'in modellerini ve tokenizasyon araçlarını kullanmak için kullanılan kütüphanedir. Bu projede, duygusal analiz için bir modeli ve tokenizer'ı yüklemek için kullanılmaktadır.

tensorflow: Derin öğrenme modelinin eğitimi ve tahminleri için kullanılan kütüphanedir.

evaluate: Metrik değerlendirmeler için kullanılan bir kütüphanedir.

git-lfs: Büyük dosyaların (Large File Storage) yönetimini sağlayan bir Git eklentisidir.

huggingface_hub: Hugging Face model hub'ına bağlanmak için kullanılan bir kütüphanedir.

KerasMetricCallback ve PushToHubCallback: Model eğitimi sırasında Keras için metrik geri çağırıcılarıdır.

Google Colab: Google Colab ortamında çalıştığımız için, çeşitli Google Colab özelliklerini kullanmak adına bazı komutlar mevcuttur. Bu kütüphane ile onlar kullanılmıştır.

PyQt5.QtWidgets: PyQt5, Qt kütüphanesini Python ile kullanmamıza olanak tanıyan bir Python bağlayıcısıdır. Bu kütüphane, kullanıcı arayüzü oluşturmak için kullanılır. QApplication, QMainWindow, QLabel, QVBoxLayout, QPushButton, QTextEdit, QMessageBox, QWidget ve QTabWidget gibi sınıfları içerir.

numpy: NumPy, çok boyutlu diziler ve matrisler üzerinde çalışmak için yüksek performanslı bir kütüphanedir. Bu örnekte, sayısal hesaplamalar için kullanılmaktadır.

spotipy: Spotify API'sine erişim sağlamak için kullanılan bir kütüphanedir. Spotify ile etkileşim kurmak ve önerilen müzikleri almak için kullanılmaktadır.

random: Rastgele sayılar ve öğeler oluşturmak için kullanılan Python'un standart kütüphanesidir. Bu projede, rastgele bir şarkı seçimi için kullanılmaktadır.

os: İşletim sistemine bağlı işlemleri gerçekleştirmek için kullanılan kütüphanedir. Bu projede, dosya yollarını birleştirmek ve dosya varlığını kontrol etmek için kullanılmaktadır.

sys: Python dilinin sistemle ilgili işlemleri yöneten bir modüldür.

QPixmap: PyQt5 ile resim işleme işlevselliği sağlayan bir sınıftır.

AutoTokenizer, TFAutoModelForSequenceClassification: Hugging Face tarafından sağlanan `transformers` kütüphanesinde bulunan, önceden eğitilmiş dil modellerini kullanmak için gerekli olan sınıflardır.
Kullanılan Model ve Tercih Sebebi:

BERT, 2018 yılında Google tarafından geliştirilen ve Transformer mimarisine dayanan bir dil modelidir. BERT, özellikle doğal dil işleme görevlerinde başarılı bir şekilde kullanılan bir önceden eğitimli modeldir. BERT'in belirgin bir özelliği, bir metni bağlamını anlamak için bir kelimenin etrafındaki tüm bağlamı kullanma yeteneğidir. Geleneksel dil modellerinden farklı olarak, BERT hem soldan sağa hem de sağdan sola yönlü bağlamı dikkate alır, bu da daha iyi bir anlam sağlar.

BERT, büyük miktarda metin verisiyle önceden eğitilir ve bu sayede genel dil anlama yeteneği geliştirilir. Ardından, belirli görevler için ince ayar yaparak (örneğin, duygusal analiz, metin sınıflandırma gibi) görev spesifik verilerle eğitilebilir.

DistilBERT, Hugging Face tarafından geliştirilen ve BERT modelinin hafifletilmiş bir versiyonudur. BERT'in tam boyutlu modeli büyük parametre sayısı ve hesaplama gücü gerektirdiğinden, bazı durumlarda bu modeli kullanmak maliyetli olabilir. DistilBERT, BERT'in özünü koruyarak modeli daha hafif hale getirir.

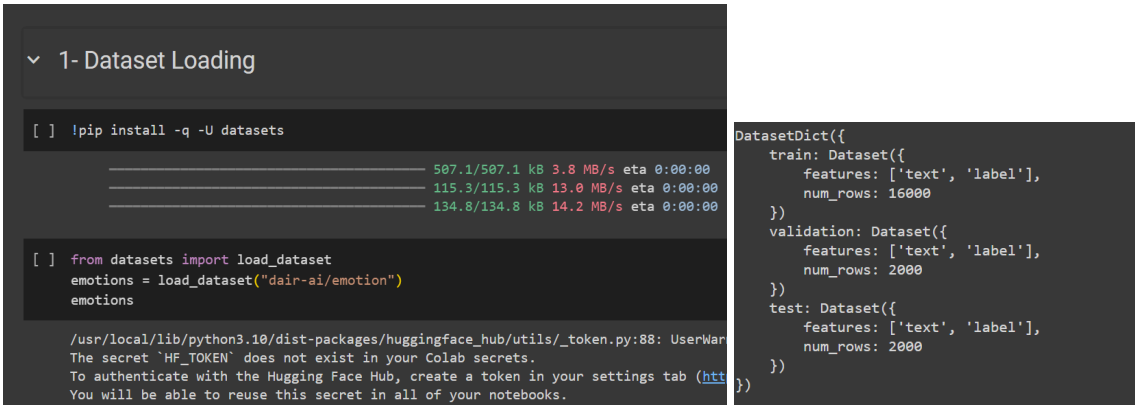
DistilBERT, öğrenme sürecinde bazı modifikasyonlar ve hafifletmelerle, orijinal BERT modelinin performansını korurken daha az hesaplama gücü ve bellek kullanır. Bu sayede, daha az kaynakla çalışan uygulamalarda veya cihazlarda kullanılabilecek bir çözüm sunar. DistilBERT, özellikle sınırlı kaynaklara sahip ortamlarda veya hızlı çözümler gerektiren durumlarda tercih edilebilir.

Her iki model de genel dil anlama yetenekleriyle öne çıkan ve çeşitli doğal dil işleme görevlerinde başarılı sonuçlar elde eden modellerdir. Ancak DistilBERT, daha hafif ve kaynak dostu bir alternatif sunar. Bu sebeple projemizde **DistilBERT** tercih edilmiştir.

KOD AÇIKLAMALARI

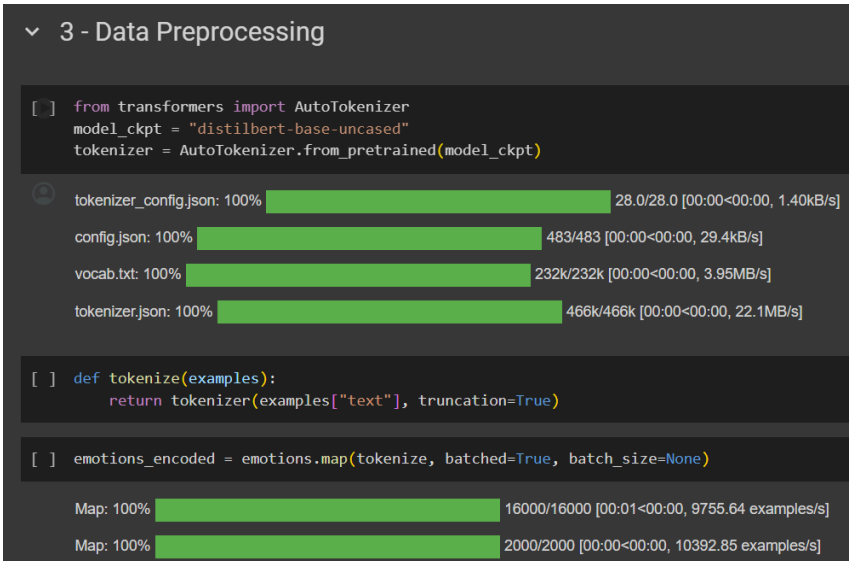
MODEL EĞİTİMİ

Birinci bölümde, önce Hugging Face'in Datasets kütüphanesinden "dair-ai/emotion" veri setini yüklüyoruz ve daha sonra bu veri setini incelemek için emotions değişkenine atıyoruz.

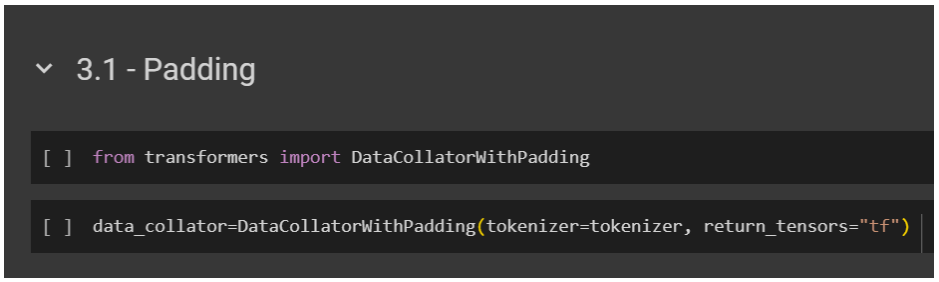


İkinci bölümde, ilk kod bloğunda yüklenen veri setinin train(eğitim) bölümündeki ilk örneği gösteriyoruz. Sonraki kod bloklarında ise sütun isimlerini ve özelliklerini gösteriyoruz.

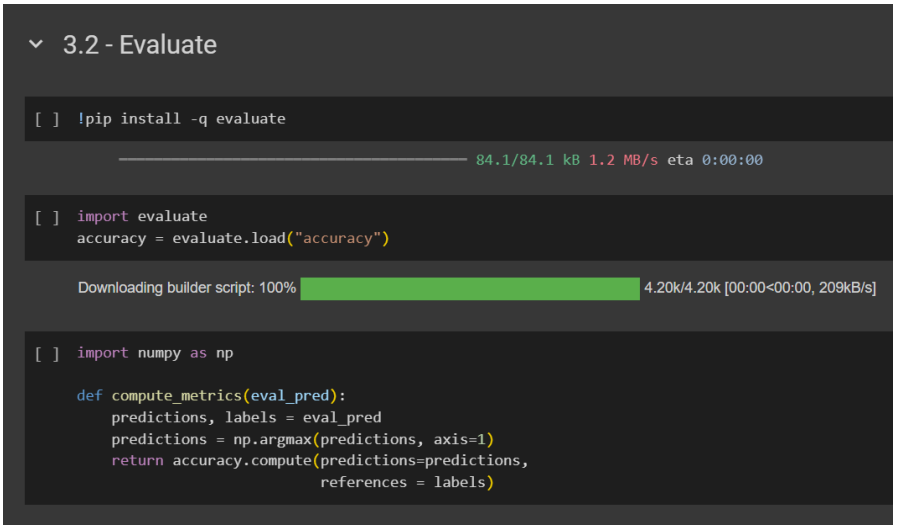
Üçüncü bölümde Hugging Face'in Transformers kütüphanesini kullanarak **DistilBERT** modelini yüklüyoruz ve bu model için bir tokenizer oluşturuyoruz.Sonrasında veri setindeki metin verilerini tokenize etmek için önceden tanımlanan **tokenizer** fonksiyonunu kullanarak bir haritalama işlemi gerçekleştiriyoruz. Bu fonksiyon, her bir metin örneğini belirtilen modelin giriş formatına dönüştürmek için kullanılır. Ardından, **emotions_encoded** adlı yeni bir veri seti oluşturulur.



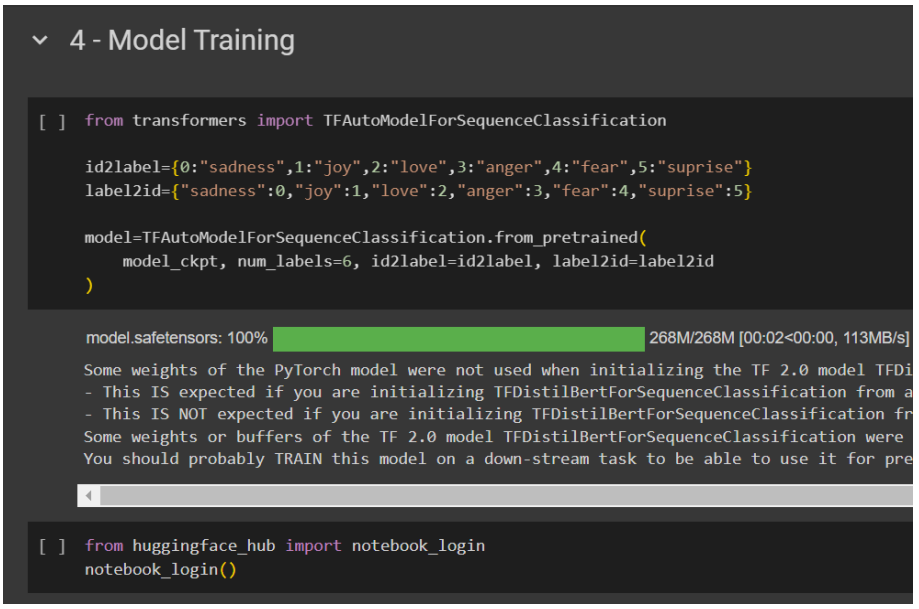
Üçüncü bölümün ilk kısmında, metin verilerini padding işlemi yapabilmek için **DataCollatorWithPadding** sınıfını kullanıyoruz.Veri setini işlerken, modele veri girişlerini aynı uzunlukta tutmak için tercih edilmiştir.



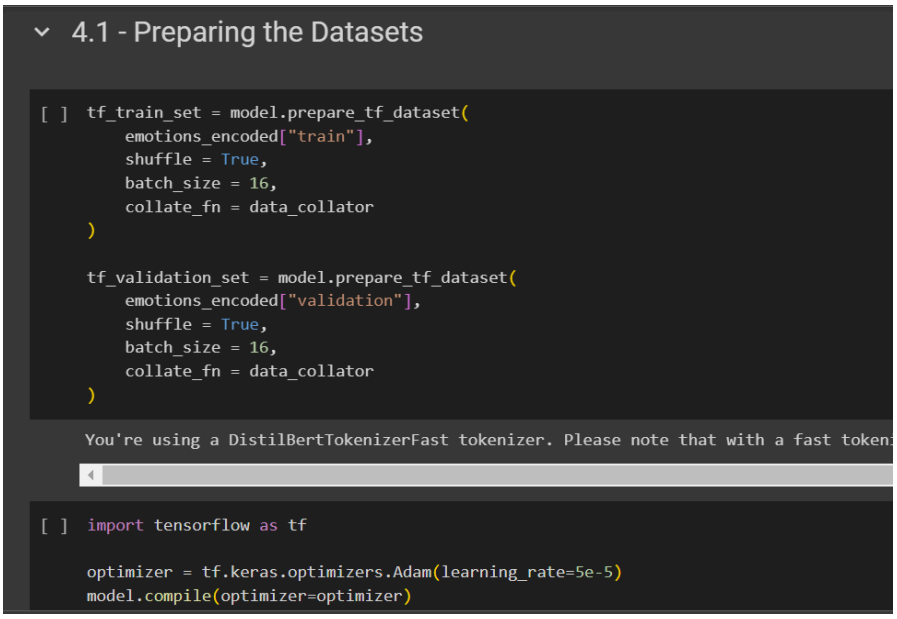
Üçüncü bölümün ikinci kısmında, modelin eğitimi sırasında kullanılacak metrikleri değerlendirmek için evaluate kütüphanesini yükleyip, accuracy metrik fonksiyonunu kullanıma hazırlıyoruz. Bu modelin performansını değerlendirmek için kullanılır.



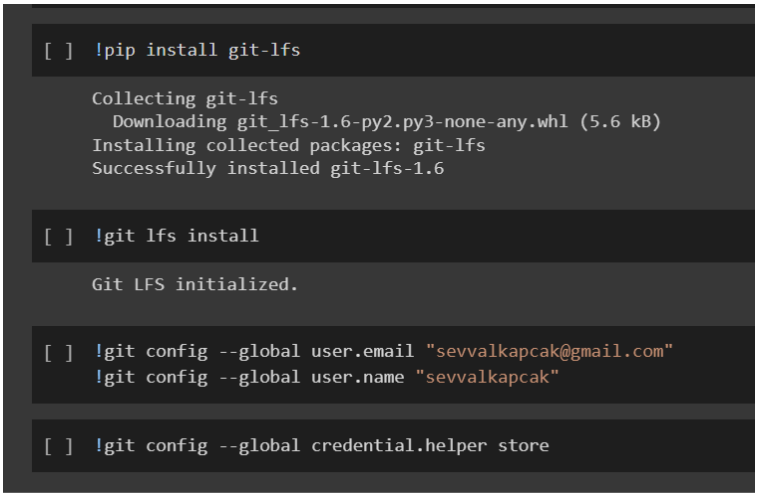
Dördüncü bölümde ilk adımda önceden yüklediğimiz DistilBERT modelini kullanarak bir duygu analizi modeli oluşturuyoruz. Modeli tanımlamak ve önceden eğitilmiş ağırlıkları yüklemek için **TFAutoModelForSequenceClassification** kullanılır. İlk kod bloğunda model oluşturulurken, duygu sınıflandırma görevi için 6 farklı etiket ve bu etiketlere karşılık gelen özel etiket-id dönüşümlerini tanımlıyoruz. Sonraki kod bloğunda, Hugging Face Hub'a modelimizi kaydetmek için giriş yapma işlemi gerçekleştiriyoruz.



Dördüncü bölümün ilk kısmında TensorFlow veri setlerini hazırlayarak modelin eğitimi için kullanılacak veriyi oluşturuyoruz. İlk kod bloğunda bu işlem gerçekleştirilirken ikinci kod bloğunda modelin eğitimi için TensorFlow'da kullanılacak optimizer'ı tanımlayarak modeli derliyoruz.

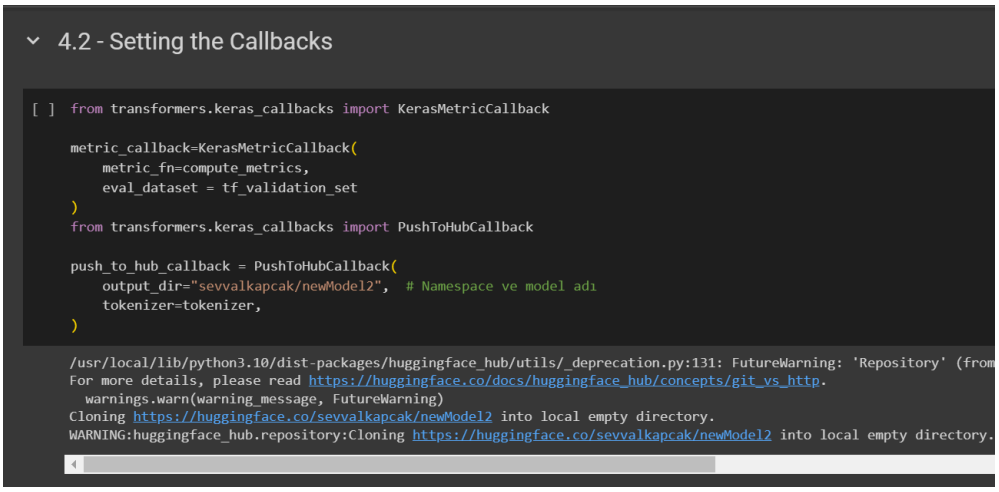


Devamında aşağıdaki kod bloklarında, Git LFS (Large File Storage) kütüphanesini yükleyip ayarlıyoruz. Bu, büyük model dosyalarını daha verimli bir şekilde yönetmemize olanak tanır. Git konfigürasyonu yaparak, kullanıcı adı, e-posta ve kimlik doğrulama ayarlarını tanımlıyoruz.



```
[ ]  !pip install git-lfs

     Collecting git-lfs
       Downloading git_lfs-1.6-py2.py3-none-any.whl (5.6 kB)
     Installing collected packages: git-lfs
     Successfully installed git-lfs-1.6

[ ]  !git lfs install

     Git LFS initialized.

[ ]  !git config --global user.email "sevvalkapcak@gmail.com"
     !git config --global user.name "sevvalkapcak"

[ ]  !git config --global credential.helper store
```

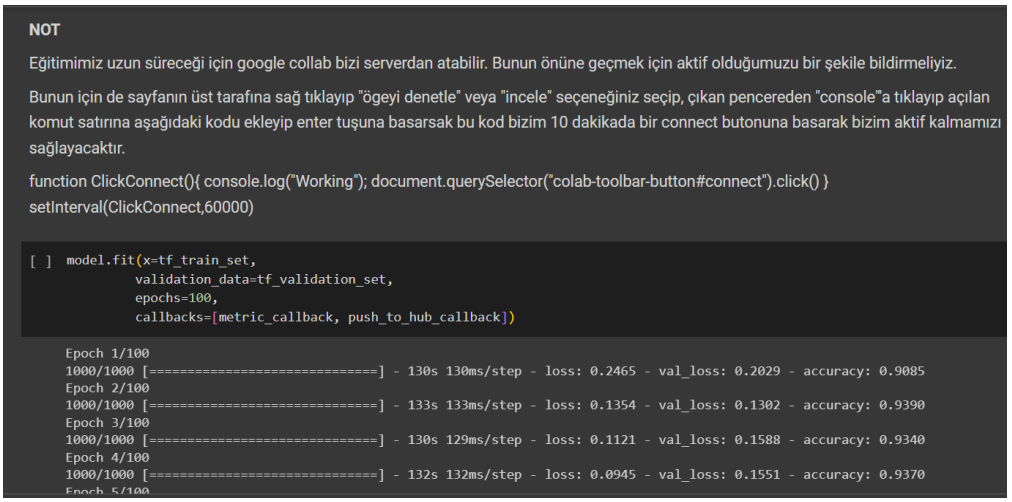
Dördüncü bölümün ikinci kısmında modelin eğitimi sırasında kullanılacak metrik geri çağırımını ve modelin eğitimi sırasında Hugging Face Hub'a yüklenmesini sağlayacak geri çağırımı tanımlıyoruz.



4.2 - Setting the Callbacks

```
[ ]  from transformers.keras_callbacks import KerasMetricCallback

     metric_callback=KerasMetricCallback(
         metric_fn=compute_metrics,
         eval_dataset = tf_validation_set
     )
     from transformers.keras_callbacks import PushToHubCallback

     push_to_hub_callback = PushToHubCallback(
         output_dir="sevvalkapcak/newModel2",  # Namespace ve model adı
         tokenizer=tokenizer,
     )

     /usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_deprecation.py:131: FutureWarning: 'Repository' (from
     For more details, please read https://huggingface.co/docs/huggingface_hub/concepts/git_vs_http.
       warnings.warn(warning_message, FutureWarning)
     Cloning https://huggingface.co/sevvalkapcak/newModel2 into local empty directory.
     WARNING:huggingface_hub.repository:Cloning https://huggingface.co/sevvalkapcak/newModel2 into local empty directory.
```

Bu kısmın son kod bloğunda ise, önceden tanımladığımız veri setleri üzerinde modelin 100 epochs ile eğitimini gerçekleştiriyoruz.



NOT

Eğitimimiz uzun süreceği için google collab bizi serverdan atabilir. Bunun önüne geçmek için aktif olduğumuzu bir şekile bildirmeliyiz.

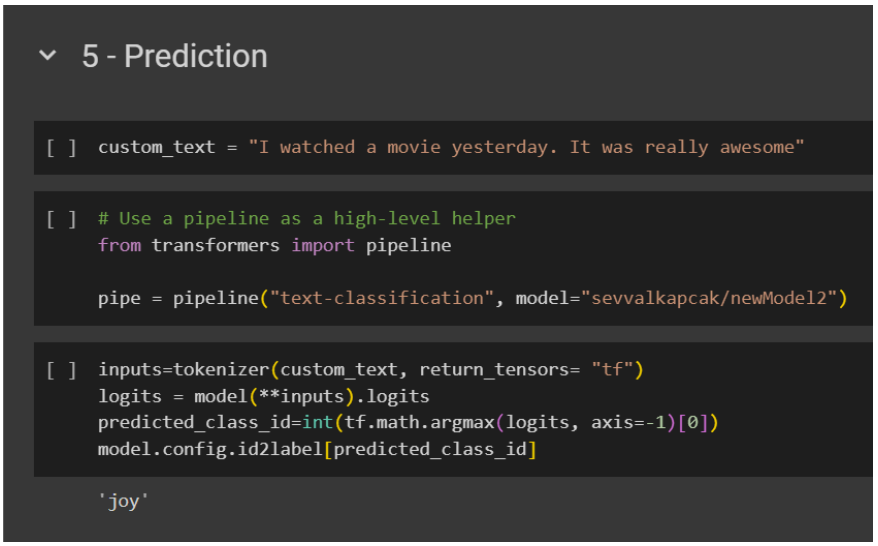
Bunun için de sayfanın üst tarafına sağ tıklayıp "ögeyi denetle" veya "incele" seçeneğiniz seçip, çıkan pencereden "console"a tıklayıp açılan komut satırına aşağıdaki kodu ekleyip enter tuşuna basarsak bu kod bizim 10 dakikada bir connect butonuna basarak bizim aktif kalmamızı sağlayacaktır.

```
function ClickConnect(){ console.log("Working"); document.querySelector("colab-toolbar-button#connect").click() }
setInterval(ClickConnect,60000)
```

```
[ ]  model.fit(x=tf_train_set,
               validation_data=tf_validation_set,
               epochs=100,
               callbacks=[metric_callback, push_to_hub_callback])
```

```
Epoch 1/100
1000/1000 [==============================] - 130s 130ms/step - loss: 0.2465 - val_loss: 0.2029 - accuracy: 0.9085
Epoch 2/100
1000/1000 [==============================] - 133s 133ms/step - loss: 0.1354 - val_loss: 0.1302 - accuracy: 0.9390
Epoch 3/100
1000/1000 [==============================] - 130s 129ms/step - loss: 0.1121 - val_loss: 0.1588 - accuracy: 0.9340
Epoch 4/100
1000/1000 [==============================] - 132s 132ms/step - loss: 0.0945 - val_loss: 0.1551 - accuracy: 0.9370
Epoch 5/100
```

Beşinci ve aynı zamanda son bölümde, eğittiğimiz model kullanarak bir metni sınıflandırıyoruz. İlk olarak, örnek bir metin olan custom_text'i belirliyoruz. Ardından, Transformers kütüphanesindeki pipeline fonksiyonunu kullanarak bir yüksek seviyeli yardımcı nesne oluşturuyoruz. Bu pipeline, belirli bir görev için önceden eğitilmiş bir modeli kullanmayı kolaylaştırır. Sonrasında, giriş metni tokenizer kullanılarak tensorlere dönüştürülüyor. Ardından, model üzerinden geçirilerek logit değerleri elde edilir. Logit değerleri, her bir sınıfın olasılığını temsil eden sayısal değerlerdir. argmax fonksiyonu kullanılarak en yüksek olasılığa sahip sınıfın endeksi bulunur. Son olarak, bu endeks kullanılarak model.config.id2label sözlüğünden tahmin edilen etiket çekilir. Bu şekilde, verilen bir metnin model tarafından tahmin edilen duygu etiketi elde edilmiş olur. Bu modelin eğitimi tamamlandıktan sonra gerçekleştirilen bir örnek tahmindir ve gerçek uygulama bağlamına göre özelleştirilecektir.



5 - Prediction

```
[ ]  custom_text = "I watched a movie yesterday. It was really awesome"
```

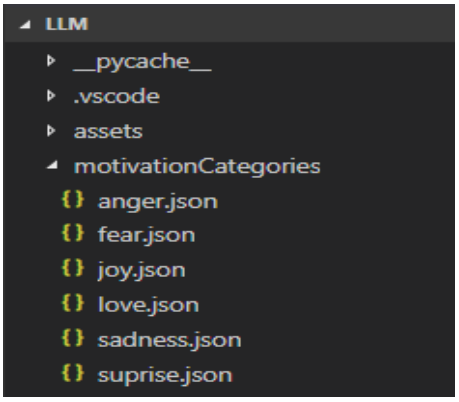
```
[ ]  # Use a pipeline as a high-level helper
     from transformers import pipeline

     pipe = pipeline("text-classification", model="sevvalkapcak/newModel2")
```

```
[ ]  inputs=tokenizer(custom_text, return_tensors= "tf")
     logits = model(**inputs).logits
     predicted_class_id=int(tf.math.argmax(logits, axis=-1)[0])
     model.config.id2label[predicted_class_id]
```

```
'joy'
```

MOTİVASYON CÜMLESİ ÖNERİSİ

Projemizde her duygu durumu için random bir şekilde data çekilerek oluşturulmuş json dosyaları bulunmaktadır. Motivasyon mesajları, duygu analizi sonucunda belirlenen duygu durumuna uygun olarak seçilen rastgele bir mesajı içeren JSON dosyalarından çekilir ve kullanıcıya sunulur. Aşağıda kodun detaylarına ve json dosya dizinine yer verilmiştir.



arayuz.py | motivationMessage.py

```python
import json
import random
import os
def get_random_motivation(emotion):

    model_directory = os.path.join("C:\\Users\\secii\\OneDrive\\Masaüstü\\LLM\\motivationCategories")
    motivation_file_path = os.path.join(model_directory, f'{emotion}.json')
    try:
        with open(motivation_file_path, 'r', encoding='utf-8') as file:
            motivations = json.load(file).get('motivations', [])
        if motivations:
            random_motivation = random.choice(motivations)
            return random_motivation
        else:
            return f"No text was found in the motivation file."
    except FileNotFoundError:
        return f"{emotion.capitalize()} Motivation file not found."

```

motivationMessage.py

Bu fonksiyon, belirli bir duygu durumu için motivasyon metinlerini içeren JSON dosyalarının bulunduğu bir dizin yapısını kullanarak, her duygu durumu için farklı motivasyon metinleri sağlamak üzere tasarlanmıştır.

Bu fonksiyonun ana işlevleri şu şekildedir;

1. Parametreler:
- `emotion`: Kullanıcının günlük metni analiz edildikten sonra belirlenen duygu durumu.

2. Dosya Yolu Belirleme:
- `model_directory` değişkeni, motivasyon metinlerinin bulunduğu dizini temsil eder. Bu dizin, duygu durumlarına özel JSON dosyalarını içerir.
- `motivation_file_path` değişkeni, belirli bir duygu durumu için kullanılacak olan JSON dosyasının tam dosya yolunu oluşturur.

3. Dosyayı Okuma ve Kontrol:
- `try` bloğu içinde, belirtilen duygu durumu için motivasyon metinlerini içeren JSON dosyası okunmaya çalışılır.
- `with open(...)` ifadesi, dosyanın güvenli bir şekilde açılmasını ve işlemlerin tamamlandıktan sonra otomatik olarak kapatılmasını sağlar.
- `json.load(file)` ifadesi ile dosyadaki JSON verisi yüklenir.

4. Motivasyon Metni Seçme:
- Eğer dosya başarıyla okunursa, `motivations` listesi içinden rastgele bir motivasyon metni seçilir.
- Eğer `motivations` boşsa, dosyanın içinde motivasyon metni bulunamamış demektir ve buna göre bir bilgilendirme metni döndürülür.

5. Dosya Bulunamazsa:
- Eğer belirtilen duygu durumu için bir motivasyon dosyası bulunamazsa (`FileNotFoundError`), ilgili bilgilendirme metni döndürülür.

6. Sonuç Döndürme:
- Fonksiyon, seçilen rastgele motivasyon metnini döndürür veya dosya bulunamazsa veya motivasyon metni yoksa ilgili bilgilendirme mesajını döndürür.

MÜZİK ÖNERİSİ

Bu proje, kullanıcının günlük metnini analiz eden bir doğal dil işleme modeli kullanarak duygu durumunu belirledikten sonra her bir duygu durumu için kullanıcıya random müzik önerisi yapar. Böylece kişiler o günkü duygularını, onunla eşleşen müzikleri dinleyerek besleyebileceklerdir.

Bu öneri sistemi şu şekilde çalışır:

Kullanıcının günlük metnini analiz eden bir doğal dil işleme modeli kullanarak duygu durumunu belirlenir. Duygu durumu, altı sınıftan birine atanır: 'sadness', 'joy', 'love', 'anger', 'fear', 'surprise'.

Her bir duygu durumu için önceden belirlenmiş Spotify çalma listeleri bulunur, bu çalma listeleri belirli bir duygusal atmosferi yansıtan şarkıları içerir.

Proje, Spotify API aracılığıyla kullanıcı adına çalma listelerine erişir ve belirlenen duygu durumuna uygun çalma listesinden rastgele bir şarkı seçer. Eğer çalma listesinde şarkılar bulunmuyorsa, genel bir mutluluk çalma listesi kullanılır.

Seçilen şarkının Spotify URI'si alınarak, bu URI ile bir Spotify şarkısı için web bağlantısı oluşturulur. Kullanıcıya, önerilen şarkıyı dinlemesi için bir bağlantı sağlanır. Bu süreç, kullanıcının günlük metni üzerinden duygu durumu analizi yaparak, uygun bir müzik önerisi sunma mantığına dayanır. Aşağıda müzik öneri kısmının kodları yer almaktadır.

arayuz.py getmusic.py

```python
import spotipy
from spotipy.oauth2 import SpotifyOAuth
import random

class MusicRecommendation:
    def __init__(self):
        self.sp = spotipy.Spotify(auth_manager=SpotifyOAuth(client_id='72bddd956a82499fa51a51bd2d6a1ceb',
                                                            client_secret='e80dacd16db84b4d830025398a685424',
                                                            redirect_uri='http://localhost:8888/callback',
                                                            scope='playlist-modify-public'))

    def get_random_track_from_playlist(self, emotion):
        mood_playlists = {
            'sadness': 'https://open.spotify.com/playlist/37i9dQZF1DXbrUpGvoi3TS?si=17fdae277c0d4018',
            'joy': 'https://open.spotify.com/playlist/37i9dQZF1DX9XIFQuFvzM4?si=1cc04be28e904f1a',
            'love': 'https://open.spotify.com/playlist/37i9dQZF1DWVf1Phr4ZVgO?si=4aec072e2e7541ca',
            'anger': 'https://open.spotify.com/playlist/37i9dQZF1DWZVAVMhIe3pV?si=f712325f58f0412b',
            'fear': 'https://open.spotify.com/playlist/37i9dQZF1DX4WYpdgoIcn6?si=8bbd7a195c8a48a9',
            'surprise': 'https://open.spotify.com/playlist/37i9dQZF1DX3rxVfibe1L0?si=88c2597081b1421f'
        }

        playlist_name = mood_playlists.get(emotion, 'Happy Hits')
        playlist_uri = self.sp.user_playlist(self.sp.me()['id'], playlist_name)['uri']
        tracks = self.sp.playlist_tracks(playlist_uri)['items']

        if not tracks:
            return None  # Eğer listede şarkı yoksa None döndür

        random_track = random.choice(tracks)
        track_uri = random_track['track']['uri']

        return f"https://open.spotify.com/track/{track_uri.split(':')[2]}"
```

getmusic.py

Bu Python kodu, Spotify API'sini kullanarak duygu durumuna uygun müzik önerisi yapabilen bir sınıf içerir. İşlevselliği aşağıdaki gibidir:

1. Spotipy Kütüphanesi ve Spotify API Yetkilendirmesi:

- "**spotipy**" kütüphanesi, Spotify API'sini kullanmak için bir Python istemcisidir.
- "**SpotifyOAuth**" sınıfı, Spotify kullanıcısının yetkilendirilmesi için OAuth2 kimlik doğrulama akışını yönetir. Bu, uygulamanın Spotify API'sine erişim sağlamasını sağlar.
- Bu sınıfın "**__init__**" metodu, Spotify API'ye yetkilendirme sağlamak için gerekli bilgileri içerir.

2. Playlist ve Duygu Durumu Eşleştirmesi:

- "**get_random_track_from_playlist**" metodu, duygu durumuna uygun bir Spotify çalma listesi belirler. Önceden belirlenmiş duygu durumlarına göre çalma listeleri bir sözlük içinde tanımlanmıştır.
- Belirtilen duygu durumu için çalma listesinin adını alır ve Spotify API'sini kullanarak çalma listesinin URI'sini elde eder.

3. Rastgele Şarkı Seçimi:

- Çalma listesindeki şarkıları alır ("**tracks**").
- Eğer çalma listesi boşsa "**None**" döndürür.
- Eğer çalma listesinde şarkılar varsa, rastgele bir şarkı seçer ve şarkının URI'sini elde eder.

4. Sonuç:

- Seçilen şarkının URI'si kullanılarak Spotify'da dinlemeye yönlendirme linki oluşturulur ve bu link "**get_random_track_from_playlist**" tarafından döndürülür.

KULLANICI ARAYÜZÜ

Projenin kullanıcı arayüzü, PyQt5 kütüphanesi kullanılarak oluşturulmuş bir masaüstü uygulamasını içerir. Uygulama, bir ana pencere ve bu pencerede iki sekme içerir: "Home" ve "Diary". "Home" sekmesi, uygulamanın ana sayfasını temsil eder ve basit bir arka plan resmi ve yazı içeren bir etiket içerir.

"Diary" sekmesi ise günlük yazılarını analiz etme ve kullanıcıya duygu durumuna uygun müzik ve motivasyon önerileri sunma işlevselliğini barındırır. Bu sekmede bir etiket, metin düzenleme alanı, analiz butonu ve bir bağlantı içeren bir etiket bulunur. Kullanıcı, günlük yazısını girdikten sonra "Analyze It" butonuna tıklayarak yazısının duygu durumunu analiz edebilir. Analiz sonucunda, kullanıcıya önerilen müzik ve motivasyon mesajları bağlantılar aracılığıyla sunulur.

Arayüz, birkaç görsel öğe ve düğme ile kullanıcı dostu bir deneyim sunar, böylece kullanıcılar günlük yazılarını analiz edebilir ve duygusal önerilere ulaşabilir.

Aşağıda arayüz sayfasının kod açıklamalarına yer verilmiştir.

Kod Açıklamaları

Kütüphane ve Modüller:

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel, QVBoxLayout,
QPushButton, QTextEdit, QMessageBox, QWidget, QTabWidget
from PyQt5.QtGui import QPixmap
import numpy as np
from motivationMessage import get_random_motivation
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from PyQt5.QtWidgets import QMessageBox
from getmusic import MusicRecommendation
```

Kullanılan kütüphane ve modüller raporun başında detaylı bir şekilde açıklanmıştır. Onlara ek olarak aşağıda belirtilen modüller bu dosyada import edilmiştir.

get_random_motivation: Bu, `motivationMessage` adlı dosyadan rastgele motivasyon mesajları almak için kullanılan bir fonksiyondur.

MusicRecommendation: Bu, `getmusic` adlı dosyadan müzik önerileri almak için kullanılan bir sınıftır.

class DailyJournalApp(QMainWinow):

```
class DailyJournalApp(QMainWindow):
  def __init__(self):
      super().__init__()
      self.setWindowTitle("Diary Analysis Application")
      self.setGeometry(100, 100, 300, 500)
      background_image = QLabel(self)
      pixmap = QPixmap("arkaplann.jpeg")
      background_image.setPixmap(pixmap)
      background_image.setGeometry(0, 0, self.width() + 800, self.height())
      background_image.setScaledContents(True)
      self.tab_widget = QTabWidget()
      self.login_tab = QWidget()
      self.setup_login_tab()
      self.journal_tab = QWidget()
      self.setup_journal_tab()
      self.tab_widget.addTab(self.login_tab, "Home")
      self.tab_widget.addTab(self.journal_tab, "Diary")
      self.setCentralWidget(self.tab_widget)
      self.model, self.tokenizer = self.load_model()
      self.music_recommendation = MusicRecommendation()
```

Bu blok, `DailyJournalApp` sınıfının Init metodunu (constructor) tanımlar ve uygulamanın ana penceresinin başlangıç durumunu ayarlar. Önemli detaylar şu şekildedir;

1. `super().__init__():`
  - `QMainWindow` sınıfının kurucu metodunu çağırarak, `DailyJournalApp` sınıfını başlatır. Bu, PyQt5 uygulamalarında pencere oluşturmanın genel bir yöntemidir.

2. `self.setup_login_tab()`:
  - "Home" sekmesinin içeriğini ayarlamak için `setup_login_tab` fonksiyonunu çağırır.

3. `self.setup_journal_tab()`:
  - "Diary" sekmesinin içeriğini ayarlamak için `setup_journal_tab` fonksiyonunu çağırır.

4. `self.model, self.tokenizer = self.load_model()`:
  - Modeli ve tokenizer'ı yüklemek için `load_model` fonksiyonunu çağırır.

5. `self.music_recommendation = MusicRecommendation()`:
  - `MusicRecommendation` sınıfından bir nesne oluşturur. Bu, müzik önerileri için kullanılacak olan sınıfı temsil eder.

setup_login_tab fonksiyonu:

```
def setup_login_tab(self):

    layout = QVBoxLayout()
    image_label = QLabel(self.login_tab)
    pixmap = QPixmap("home_page.png")
    image_label.setPixmap(pixmap)
    text_label = QLabel("HOME PAGE", self.login_tab)
    text_label.setStyleSheet("font-size: 20px; color: white;")
    layout.addWidget(image_label)
    layout.addWidget(text_label)
    self.login_tab.setLayout(layout)
```

Bu kod parçası, GUI uygulamasının "Home" sekmesinin içeriğini düzenlemek için kullanılır. Detaylar şu şekildedir:

1. VBox (QVBoxLayout) Oluşturma: İlk olarak, dikey bir düzen oluşturmak için `QVBoxLayout` sınıfından bir örnek olan `layout` oluşturulur. Bu düzen, içerisine eklenen widget'ları dikey bir düzende sıralar.

2. Resim Ekleme: Ardından, `QLabel` sınıfından bir örnek olan `image_label` oluşturulur. Bu etiket, arka plan resmini göstermek için kullanılır.
  - `QPixmap` sınıfı, resmin yüksek kaliteli bir haritasını oluşturmak için kullanılır.
  - `setPixmap` yöntemi, etiketin içeriğini pixmap ile ayarlar, yani resmi etikete ekler.

3. Metin Etiketi Ekleme: Bir `QLabel` örneği olan `text_label` oluşturulur. Bu etiket, "HOME PAGE" metnini gösterir ve beyaz renkte 20 piksel boyutunda yazılmıştır.
  - `setStyleSheet` yöntemi, metin etiketinin stilini belirlemek için kullanılır.

4. Widget'ları Düzen Ekleme: `addWidget` yöntemiyle, oluşturulan `image_label` ve `text_label` widget'ları `layout`'a eklenir. Bu, widget'ların dikey düzende sıralanmasını sağlar.

5. Düzeni Belirli Sekmeye Atamak: Son olarak, oluşturulan dikey düzen (`layout`), "Home" sekmesine atanır.
  - `self.login_tab.setLayout(layout)` satırı, bu dikey düzenin `login_tab` adlı sekmenin içeriği olarak belirlenmesini sağlar.

setup_journal_tab fonksiyonu:

```
def setup_journal_tab(self):
    layout = QVBoxLayout()

    self.label = QLabel("Dear Diary;")
    self.label.setStyleSheet("background-color: white; color: #d686ed; font-size: 20px;
font-style: italic; font-weight: bold;")

    self.text_edit = QTextEdit()
    self.text_edit.setStyleSheet("font-size: 17px;font-style: italic;")
    self.analyze_button = QPushButton("Analyze It")
    self.analyze_button.clicked.connect(self.analyze_journal)
    self.analyze_button.setStyleSheet("background-color: #d686ed; color: white; font-size:
20px")

    self.link_label = QLabel("<a href='https://www.youtube.com/' style='color:
#d686ed;font-weight: bold; '>This is exactly your mood today :)</a>")
    self.link_label.setStyleSheet("color: white; font-size: 17px")
    self.link_label.setOpenExternalLinks(True)

    layout.addWidget(self.label)
    layout.addWidget(self.text_edit)
    layout.addWidget(self.analyze_button)
    layout.addWidget(self.link_label)

    journal_widget = QWidget(self.journal_tab)
    journal_widget.setGeometry(self.width()/2.3, 0, self.width()+400, self.height()+35)
    journal_widget.setLayout(layout)
```

Bu kod bloğu, GUI uygulamasının "Diary" sekmesinin içeriğini düzenlemek için kullanılır.

Aşağıda yapılan işlemlerin detaylı açıklamaları:

1. Dikey Düzen Oluşturma (`layout`):
- `QVBoxLayout` sınıfından bir örnek olan `layout` oluşturulur. Bu, içerisine eklenen widget'ları dikey bir düzende sıralar.

2. Başlık Etiketi (`self.label`):
- `QLabel` sınıfından bir örnek olan `self.label` oluşturulur. Bu etiket, "Dear Diary;" metnini içerir.
- `setStyleSheet` yöntemiyle etiketin stil özellikleri belirlenir. Arka plan rengi beyaz, metin rengi pembe-tonu (`#d686ed`), yazı stil'i italik ve kalın olarak ayarlanır.

3. Metin Giriş Kutusu (`self.text_edit`):
- `QTextEdit` sınıfından bir örnek olan `self.text_edit` oluşturulur. Bu, kullanıcının günlük metnini girebileceği bir metin giriş kutusudur.
- `setStyleSheet` yöntemiyle metin giriş kutusunun stil özellikleri belirlenir. Metin boyutu 17 piksel ve italik olarak ayarlanır.

4. Analiz Butonu (`self.analyze_button`):
- `QPushButton` sınıfından bir örnek olan `self.analyze_button` oluşturulur. Bu buton, günlük metni analiz etmek için kullanılır.
- `clicked.connect` yöntemiyle butona tıklandığında çağrılacak olan `self.analyze_journal` fonksiyonu belirlenir.
- `setStyleSheet` yöntemiyle butonun stil özellikleri belirlenir. Arka plan rengi pembe-tonu (`#d686ed`), metin rengi beyaz, yazı boyutu 20 piksel olarak ayarlanır.

5. Müzik Bağlantı Etiketi (`self.link_label`):
- `QLabel` sınıfından bir örnek olan `self.link_label` oluşturulur. Bu etiket, kullanıcının analiz sonucuna göre önerilen müziğe yönlendiren bir bağlantı içerir.
- `setStyleSheet` yöntemiyle etiketin stil özellikleri belirlenir. Metin rengi beyaz, yazı boyutu 17 piksel olarak ayarlanır.
- `setOpenExternalLinks(True)` yöntemi ile etiket içindeki bağlantının tarayıcıda açılmasına izin verilir.

6. Düzeni Belirli Sekmeye Atama:
- Oluşturulan düzen elemanları, `layout.addWidget` yöntemi ile sırasıyla eklenir.
- `journal_widget` adlı bir widget oluşturulur ve bu widget, `self.journal_tab` adlı sekmenin içeriği olarak atanır.
- `setGeometry` yöntemi ile widget'ın boyutu ve konumu ayarlanır.

load_model fonksiyonu:

```
def load_model(self):
    # Hugging Face'de kaydettiğiniz modelin repository ID'si
    repo_id = 'sevvalkapcak/newModel2'

    # Load model directly
    tokenizer = AutoTokenizer.from_pretrained(repo_id)
    model = TFAutoModelForSequenceClassification.from_pretrained(repo_id)

    return model, tokenizer
```

Bu fonksiyon, Hugging Face model hub'ında kaydedilmiş bir duygusal analiz modelini yüklemek ve kullanıma hazır hale getirmek için kullanılır. Aşağıda fonksiyonun adım adım açıklamaları bulunmaktadır.

1. Model ve Tokenizer ID'si Belirleme:
- `repo_id` değişkenine, Hugging Face modelinin repository ID'si olan 'sevvalkapcak/newModel2' atanır. Bu ID, modelin ve tokenizer'ın bulunduğu depoyu belirtir.

2. Tokenizer ve Modeli Yükleme:
- `AutoTokenizer.from_pretrained(repo_id)` kullanılarak, belirtilen repository ID'sine sahip tokenizer yüklenir. `from_pretrained` yöntemi, Hugging Face model hub'ından önceden eğitilmiş bir model veya tokenizer yüklemek için kullanılır.
- `TFAutoModelForSequenceClassification.from_pretrained(repo_id)` kullanılarak, aynı repository ID'sine sahip duygusal analiz modeli yüklenir. Bu model, metin sıralı sınıflandırma (sequence classification) görevi için eğitilmiştir.

3. Tokenizer ve Modeli İade Etme:
- Oluşturulan tokenizer ve model, fonksiyonun dönüş değeri olarak bir demet (tuple) içinde iade edilir. Bu sayede fonksiyon çağrıldığında, yüklenen model ve tokenizer kolayca elde edilebilir.

preprocess_text fonksiyonu:

```
def preprocess_text(self, text, max_length=512):
    inputs = self.tokenizer(text, truncation=True, padding=True, return_tensors="tf",
max_length=max_length)
    return tuple((inputs['input_ids'], inputs['attention_mask']))
```

Bu fonksiyon, kullanıcının giriş metnini modelin anlayabileceği bir forma dönüştürmek için kullanılır. Aşağıda fonksiyonun adım adım açıklamaları bulunmaktadır.

1. Giriş Parametreleri:
- `text`: Analiz edilecek metin.
- `max_length` (varsayılan değeri 512): Modelin işleyebileceği maksimum metin uzunluğu. Bu, giriş metinin uzunluğunu kontrol etmek ve gerektiğinde kısaltmak için kullanılır.

2. Tokenization (Simgeleme):
- `self.tokenizer(text, truncation=True, padding=True, return_tensors="tf", max_length=max_length)` kullanılarak, Hugging Face modelinin tokenizer'ı kullanılarak metin tokenize edilir. Bu işlem, metni modelin anlayabileceği simgeler (token'lar) haline getirir.
- `truncation=True` ile metin, belirtilen maksimum uzunluğu aşarsa kırpılır.
- `padding=True` ile metin, belirtilen maksimum uzunluğa ulaşmazsa, eksik kısımlar belirtilen bir simge ile doldurulur.
- `return_tensors="tf"` ile TensorFlow tensor formatında bir çıkış elde edilir.

3. Tuple İade Etme:
- Elde edilen token'lar, input_ids ve attention_mask olarak adlandırılan iki anahtarla bir sözlük içindedir.
- Bu sözlük, fonksiyonun dönüş değeri olan bir tuple içinde iade edilir. Tuple içindeki iki eleman, sırasıyla input_ids ve attention_mask'tir.
- Bu sayede model eğitimi veya tahminleme için kullanılacak giriş verisi hazır hale getirilmiş olur.

analyze_journal fonksiyonu:

```
def analyze_journal(self):
    user_text = self.text_edit.toPlainText()
    inputs_tuple = self.preprocess_text(user_text)
    logits = self.model.predict(inputs_tuple)[0]
    probabilities = tf.nn.softmax(logits)
    predicted_class = np.argmax(probabilities)
    probability_predicted_emotion = probabilities[0][predicted_class].numpy()
    emotion_labels = ['sadness', 'joy', 'love', 'anger', 'fear', 'surprise']
    predicted_emotion = emotion_labels[predicted_class]
    result_text = f"Analysis result: {predicted_emotion} (Probability:
{probability_predicted_emotion})"
    print(result_text)
    song_link =
self.music_recommendation.get_random_track_from_playlist(predicted_emotion)
    if song_link:
        self.link_label.setText(f"<a href='{song_link}' style='color:  #d686ed;font-weight:
bold;font-style: italic; '>This is exactly your mood today :)</a>")
    else:
        self.link_label.setText("<a href='https://www.youtube.com/' style='color:
#d686ed;font-weight: bold; '>This is exactly your mood today :)</a>")
```

```
    random_motivation = get_random_motivation(predicted_emotion)
    print(f"{predicted_emotion.capitalize()} {random_motivation}")
    QMessageBox.information(None, "MOTIVATION", f"{random_motivation}")
```

Bu fonksiyon, kullanıcının girdiği günlük metni analiz etmek ve sonuçları göstermek için kullanılır. Aşağıda fonksiyonun adım adım açıklamaları bulunmaktadır:

1. Giriş Parametreleri:
- `self.text_edit.toPlainText()`: PyQt5 kütüphanesindeki bir metin düzenleyiciden alınan metni elde eder. Bu metin, kullanıcının günlük yazısıdır.
- `inputs_tuple = self.preprocess_text(user_text)`: Önce, kullanıcının girdiği metni modelin anlayabileceği forma dönüştürmek üzere `preprocess_text` fonksiyonu kullanılır. Bu, metni tokenize eder ve modelin beklentilerine uygun bir formatta hazırlar.

2. Duygu Tahmini:
- `logits = self.model.predict(inputs_tuple)[0]`: Hazırlanan giriş verisi, duygu analizi modeline verilir ve tahmin sonuçları elde edilir.
- `probabilities = tf.nn.softmax(logits)`: Modelin çıkışındaki logit değerleri softmax fonksiyonu ile olasılıklara dönüştürülür.
- `predicted_class = np.argmax(probabilities)`: En yüksek olasılığa sahip sınıf (duygu) belirlenir.
- `probability_predicted_emotion = probabilities[0][predicted_class].numpy()`: Tahmin edilen duygunun olasılık değeri elde edilir.

3. Sonuçların Gösterilmesi:
- `result_text = f"Analysis result: {predicted_emotion} (Probability: {probability_predicted_emotion})"`: Elde edilen duygu ve olasılık değeriyle bir sonuç metni oluşturulur.
- `print(result_text)`: Sonuç konsola yazdırılır.

4. Müzik Önerisi:
- `song_link = self.music_recommendation.get_random_track_from_playlist(predicted_emotion)`: Tahmin edilen duygu temel alınarak, `getmusic` modülündeki `get_random_track_from_playlist` fonksiyonu kullanılarak Spotify üzerinden bir müzik önerisi alınır.
- Alınan müzik önerisi, bir Spotify şarkısının paylaşım bağlantısıdır.

5. Arayüz Güncellemeleri:
- `self.link_label.setText(...)`: Arayüzdeki bir etiket (label) üzerindeki metni günceller. Bu, kullanıcıya tahmin edilen duygu için Spotify müzik önerisi sunar.

6. Motivasyon Mesajı:
- `random_motivation = get_random_motivation(predicted_emotion)`: `motivationMessage` modülündeki `get_random_motivation` fonksiyonu kullanılarak, tahmin edilen duygu temel alınarak bir motivasyon mesajı alınır.
- `QMessageBox.information(None, "MOTIVATION", f"{random_motivation}")`: PyQt5 kütüphanesindeki bir bilgi penceresi ile kullanıcıya motivasyon mesajı gösterilir.

main fonksiyonu:

```
if __name__ == "__main__":
    app = QApplication([])
    window = DailyJournalApp()
    window.show()
    sys.exit(app.exec_())
```

Bu bölüm, uygulamanın nasıl başlatılacağını ve çalıştırılacağını tanımlar. Bu bölümdeki satırların ne yaptığını açıklayan adımlara aşağıda yer verilmiştir:

1. if __name__ == "__main__":
   - Python'da bir script çalıştırıldığında, `__name__` değişkeni o anki dosyanın adını içerir. Eğer bir dosya doğrudan çalıştırılıyorsa, `__name__` değişkeni `"__main__"` olarak ayarlanır. Bu durumu kontrol etmek için `if __name__ == "__main__":` bloğu kullanılır.

2. app = QApplication([]):
   - PyQt5 kütüphanesinin `QApplication` sınıfından bir nesne oluşturulur. Bu, bir PyQt uygulamasını başlatmak için gereklidir. `[]` içinde boş bir liste, komut satırı argümanlarına yer vermediğimizi belirtir.

3. window = DailyJournalApp():
   - `DailyJournalApp` sınıfından bir nesne oluşturulur. Bu, günlük uygulamasının ana penceresini temsil eder.

4. window.show():
   - Oluşturulan pencereyi görünür hale getirir. Bu, kullanıcıya uygulamayı gösterir.

5. sys.exit(app.exec_()):
   - Uygulamanın çalışmasını başlatır ve PyQt5 uygulamasının ana döngüsünü çalıştırır. Bu satır, uygulama penceresi kapatıldığında veya uygulama sonlandığında programın düzgün bir şekilde kapanmasını sağlar. `sys.exit()` ile Python yorumlayıcısına programın kapatılmasını bildirir.

ÇIKTILAR

Örnek Metin

Örnek bir günlük metni yazılıp analiz edildi. Metin aşağıdaki gibidir;

“ Today was filled with an overwhelming sense of joy. The sun's warmth embraced everything, casting a golden glow on the day. Every moment seemed to dance with positivity, and a genuine smile found its way to my face effortlessly.

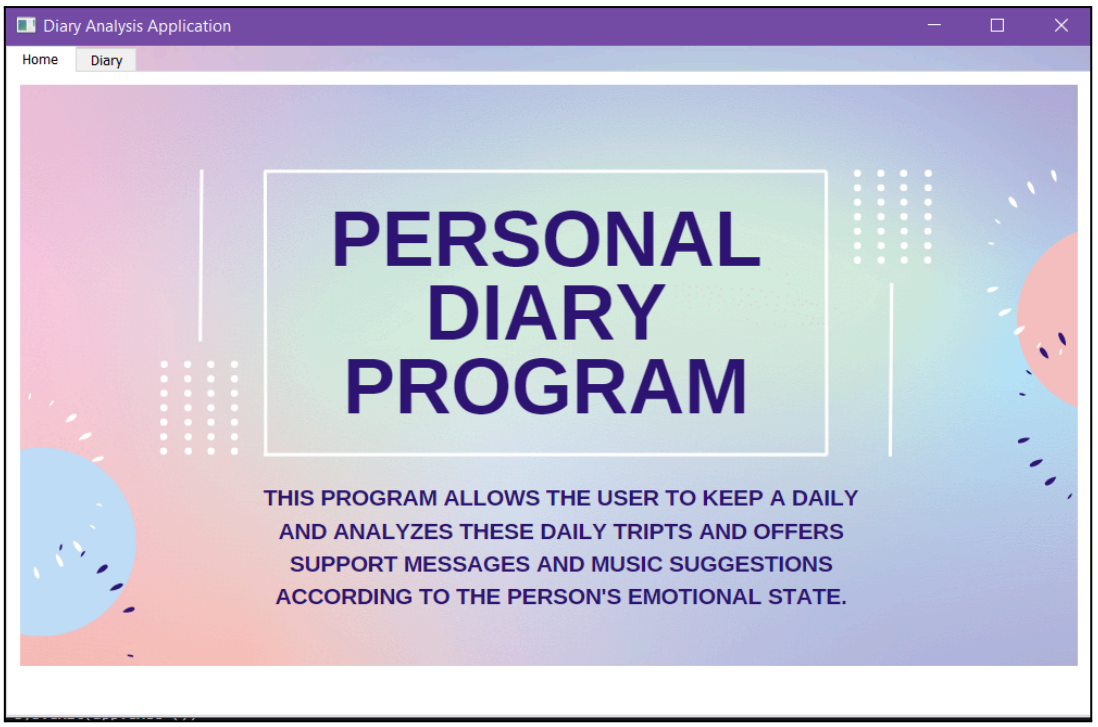
Work, usually a routine, became a joyful journey of accomplishments. The simple pleasures, like a warm cup of coffee and the laughter of colleagues, magnified the happiness within. It's amazing how one emotion can color the entire canvas of the day.

As the day concludes, I carry this joy in my heart, grateful for the simple and beautiful moments that made today special.”

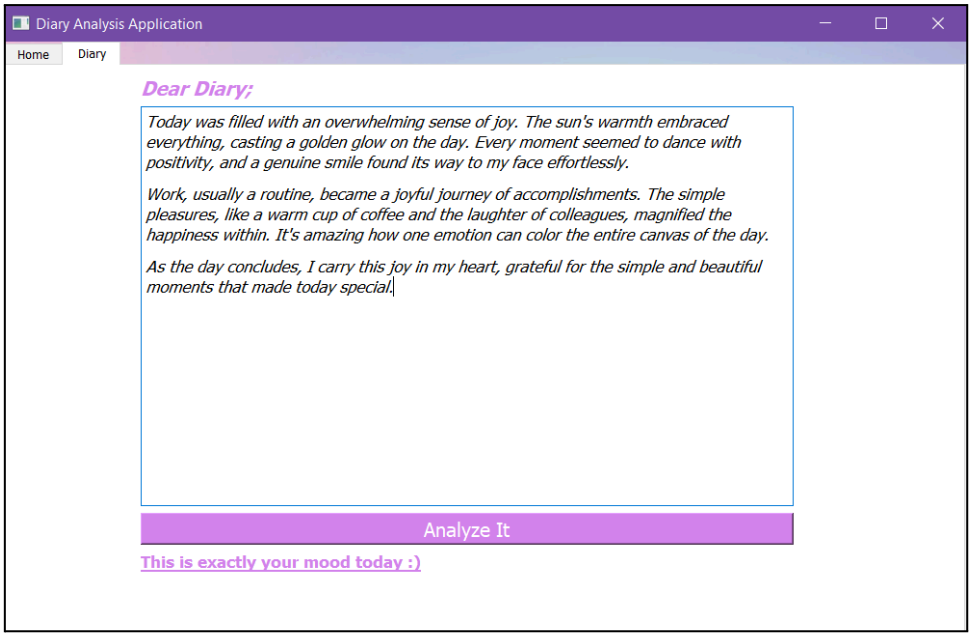
Terminal Çıktısı

```
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFDistilBertForSequenceClassification for predictions without further training.

1/1 [==============================] - 3s 3s/step

Analysis result: joy (Probability: 0.9996223449707031)

Joy Joy is the ability to celebrate the beauties of life. Evaluating every joyful moment can create a sense of internal gratitude.
```

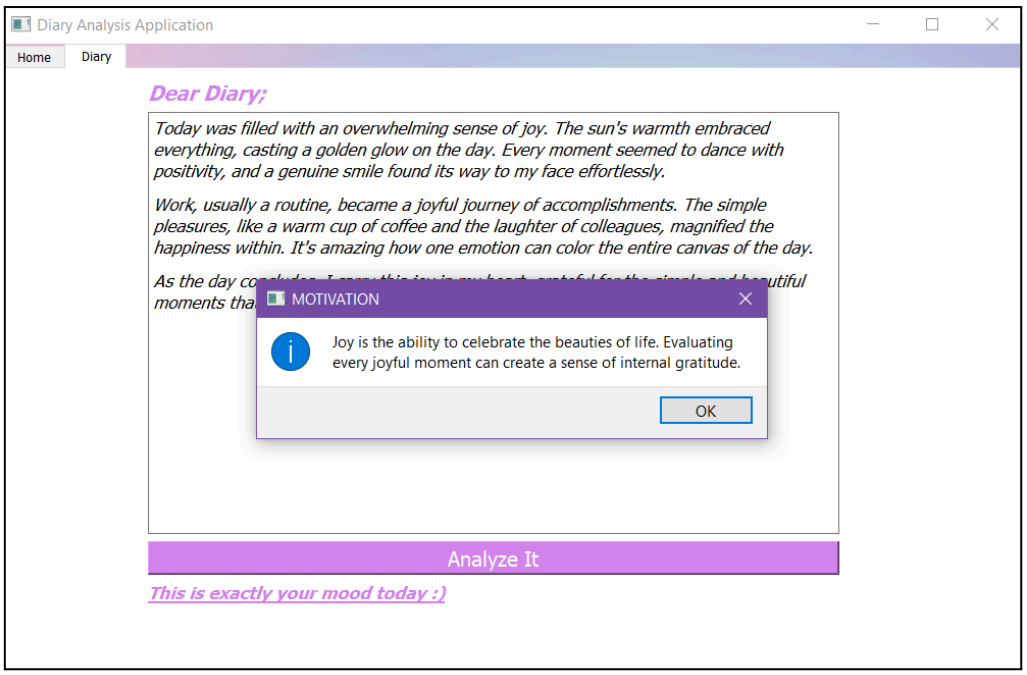
Program Çıktısı



Aşağıda kullanıcının günlüğünü yazdığı bir örnek yer almaktadır. Kullanıcı günlük yazmayı tamamladıktan sonra "**Analyze It**" butonuna basacak ve tespit edilen duygu durumuna göre kullanıcıya motivasyon mesajı verilecek ve müzik önerisi yapılacaktır.



Aşağıda ise kullanıcı butona bastıktan sonra motivasyon mesajının ekranda belirdiği ve "**This is exactly your mood today :)**" yazısının link haline geldiği görülmektedir.



Aşağıdaki görselde ise, oluşan linke tıklandığında duygu durumuna atanmış olan çalma listesinden random seçilen bir müzik Spotify'da açılmaktadır.