این دیتاست شامل اخباری از بازارهای مالی میباشد که احاسات هر خبر توسط یک سرمایه گذار مشخص شده است. این دیتاست دارای دو ستون «جمله» و «احساسات» میباشد. احساسات در یکی از دسته‌های خنثی، منفی و مثبت هستند. این کد به استخراج داده و تحلیل احساسات با استفاده از این دیتاست میپردازد.

(کدهای موجود در این سند فاقد کامنت هستند. کامنت‌های کد درون فایل کد موجود هستند.)

پیش‌پردازش و مصورسازی داده‌ها

```
import re, nltk
import numpy as np
import pandas as pd
from textblob import TextBlob
from nltk import word_tokenize
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk.corpus import stopwords
from xgboost import XGBClassifier
from gensim.models import KeyedVectors
from nltk.stem import WordNetLemmatizer
from keras.preprocessing import sequence
from keras.metrics import Precision, Recall
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.utils import to_categorical
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
from tensorflow.keras.layers import Embedding, LSTM, Dense,
SpatialDropout1D, Dropout, Activation, TextVectorization
```

در این سلول کتابخانه‌های مورد نیاز برای پیش‌پردازش داده‌ها، تحلیل، مدل‌سازی و ارزیابی نتایج وارد شده‌اند. به عنوان مثال، کتابخانه nltk برای مباحث مربوط به زبان طبیعی، کتابخانه matplotlib برای مصورسازی داده‌ها و نتایج، کتابخانه XGBClassifier برای پیاده‌سازی مدل XGBoost و کتابخانه RandomForestClassifier برای پیاده‌سازی مدل جنگل تصادفی مورد استفاده قرار میگیرند.

```
nltk.download('wordnet')
nltk.download('stopwords')
```

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

در این سلول دیتابیس‌های مورد نیاز برای کتابخانه nltk توسط خود کتابخانه دانلود میشود. دیتابیس wordnet شامل کلمات بسیار زیاد، دیتابیس stopwords شامل stopword های زبان انگلیسی مانند is, the, are, ...، و دیتابیس punkt شامل انواع گوناگونی از علائم نگارشی میباشد.

```
df = pd.read_csv('./all-data.csv', encoding='ISO-8859-1')
```

در این سلول وظیفه اصلی برنامه آغاز شده و فایل دیتاست توسط کتابخانه pandas باز میشود. برخی نظرات در این دیتاست از انکدینگ ISO-8859-1 تبعیت میکنند و به همین جهت این انکدینگ مشخص شده است.

```
df['Label'] = LabelEncoder().fit_transform(df['Sentiment'])
```

در این خط ستون جدیدی تحت عنوان Label ایجاد میشود و سه احساس مختلف منفی، خنثی و مثبت به سه عدد مختلف انکد میشوند و در این ستون قرار میگیرند. انکد به این صورت خواهد بود که مقدار منفی عدد صفر، مقدار خنثی عدد یک و مقدار مثبت عدد دو را اختیار خواهد کرد.

```
stopwords.words("english").extend(['rt', 'mkr', 'didn', 'bc', 'n',
'm','im', 'll', 'y', 've', 'u', 'ur', 'don','p', 't', 's', 'aren', 'kp',
'o', 'kat', 'de', 're', 'amp', 'will'])

Stopwords = set(stopwords.words('english')) - set(['not'])
```

در این قسمت ابتدا کلماتی که ممکن است پس از پیش‌پردازش داده در دیتاست ظاهر شوند را به مجموعه stopwords اضافه میکنیم و در نهایت کلمه not را از آن حذف میکنیم به دلیل اینکه این کلمه ممکن است که بار جمله را تغییر دهد و برای تشخیص قطبیت جمله نیاز است.

```
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    text = text.replace(r"http\S+"," ")
    text = text.replace(r"http"," ")
    text = text.replace(r"@","at")
    text = text.replace("#[A-Za-z0-9_]+", ' ')
    text = text.replace(r"[^A-Za-z(),!?@\'\"_\n]"," ")
    text = text.lower()
    text = re.sub(r"won\'t", "will not", text)
```

```
    text = re.sub(r"can\'t", "can not", text)
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would",text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    text = re.sub('[^a-zA-Z]',' ',text)
    text = re.sub(r'[^\x00-\x7f]','',text)
    text = [lemmatizer.lemmatize(word) for word in text.split() if not
word in Stopwords]
    text = ' '.join(text)
    return text
```

در این قسمت ابتدا از WordNetLemmatizer یک نمونه ساخته میشود. وظیفه این آبجکت تبدیل کلمات انگلیسی به ریشه یا مصدر آن‌هاست که در تابع پیش‌پردازش از این آبجکت استفاده میشود. پس از آن تابع preprocess_text تعریف شده است که روی ورودی خود اعمالی را انجام میدهد و آن را بازمیگرداند. این اعکال از قبیل حذف علائم نگارشی، تبدیل کلمات متن به ریشه‌ها یا مصدرها، حذف لینک‌ها، جدا سازی مخفف های انگلیسی مثل won’t به will not و ...، کوچک کردن الفبای کلمات و ... است.

```
df.drop_duplicates(subset=['Sentence'], keep='first', inplace=True)
```

در این قسمت نظرات تکراری موجود در دیتاست حذف میشوند.

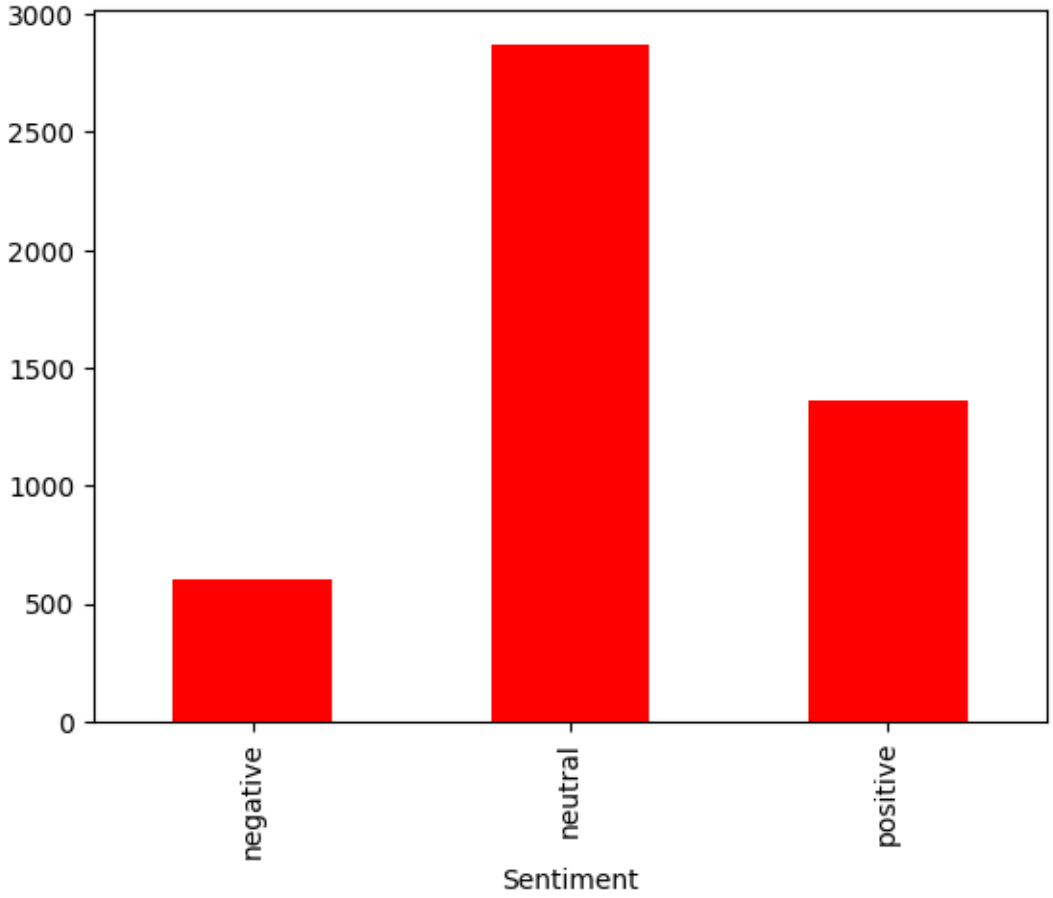
```
df['Cleaned_sentence'] = df['Sentence'].apply(preprocess_text)
```

در این قسمت ستون جدیدی تحت عنوان Cleaned_sentence ساخته میشود. سپس تابع preprocess_text روی هر رکورد از ستون sentence اجرا شده و نتیجه در سلول متناظر در ستون Cleaned_sentence ذخیره میشود. پس از این عملیات نظرات پیش‌پردازش شده و در ستون Cleaned_sentence در دسترس هستند.

```
df['Sentiment'].value_counts().reindex(('negative', 'neutral',
'positive')).plot(kind='bar', color='r')
```

در این قسمت تعداد هر نوع نظر (منفی، خنثی و مثبت) به دست آمده و به صورت نمودار میله‌ای نمایش داده میشود.

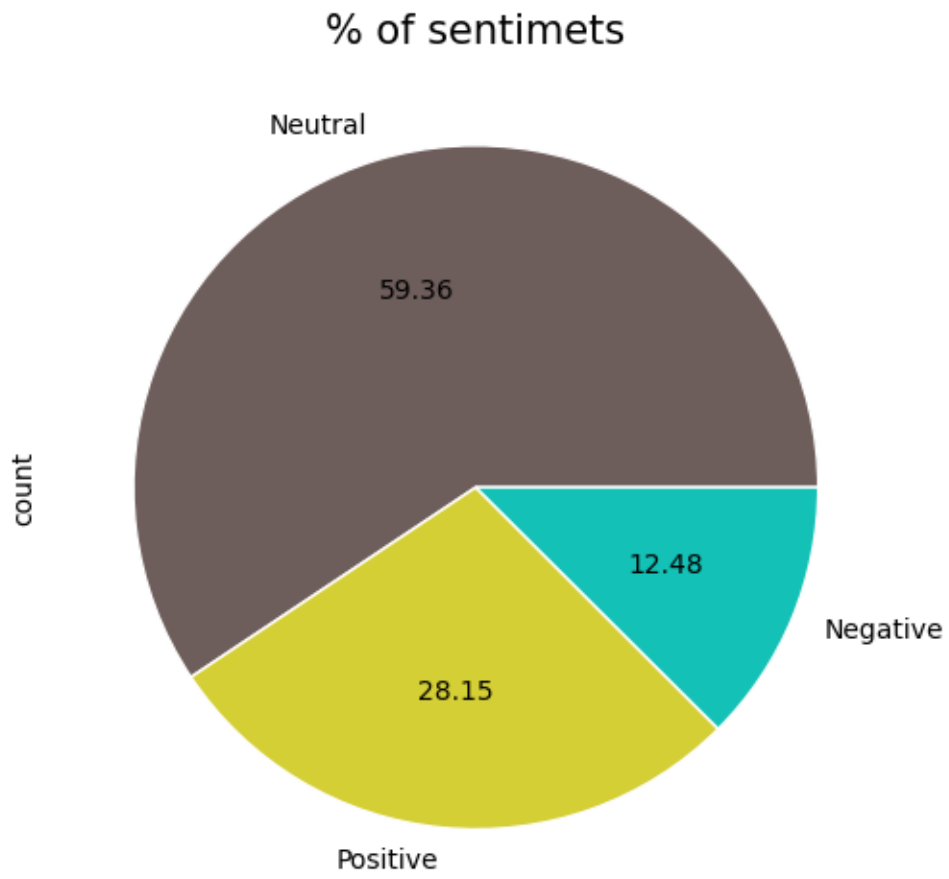
نتیجه این نمایش در زیر آمده‌است. تعداد نظرات خنثی از همه نظرات بیشتر بوده است و بعد از آن نظرات مثبت بیشترین نظرات را به خود اختصاص داده‌اند و در نهایت نظرات منفی کمترین نظرات بودند.




```
colors = ['#6E5E5B', '#D4CF34', '#14C1B6']
_, (ax1) = plt.subplots(ncols=1, figsize=(10, 5))
df.Sentiment.value_counts().plot(kind='pie', labels=('Neutral',
'Positive', 'Negative'), autopct='%.2f', ax=ax1, wedgeprops={ 'linewidth':
1, 'edgecolor': 'white' }, colors=colors)
plt.title('% of sentimets', size=15)
plt.tight_layout()
plt.show()
```

در این سلول با استفاده از کتابخانه matplotlib درصد هر نوع نظری را تحت یک نمودار دایره‌ای رسم میکنیم.

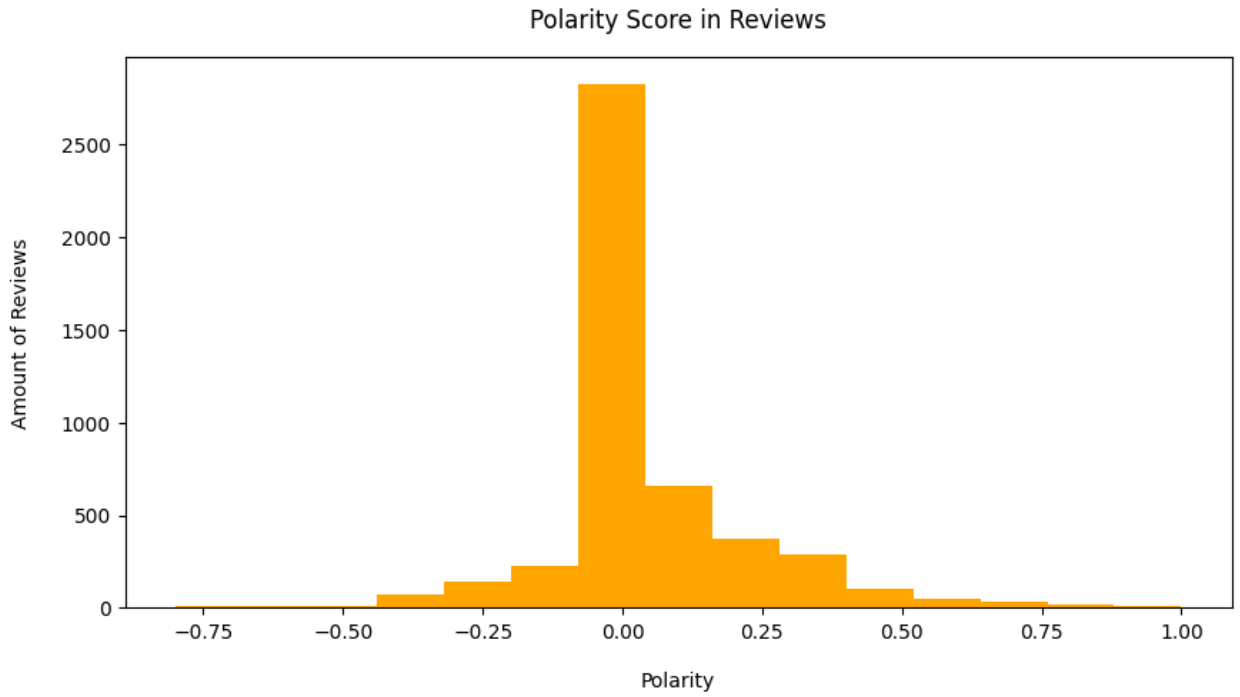
نتیجه این ترسیم به صورت زیر خواهد بود.



همانطور که از نمودار مشخص است ۵۹.۳۶ درصد از نظرات خنثی، ۲۸.۱۵ درصد از نظرات مثبت و ۱۲.۴۸ درصد از نظرات منفی بوده‌اند.

```
df['Polarity'] = df['Cleaned_sentence'].map(lambda Text:
TextBlob(Text).sentiment.polarity)
df['Polarity'].plot(kind='hist', bins=15, linewidth=1, color='orange',
figsize=(10, 5))
plt.title('Polarity Score in Reviews', pad=15)
plt.xlabel('Polarity', labelpad=15)
plt.ylabel('Amount of Reviews', labelpad=15)
plt.show()
```

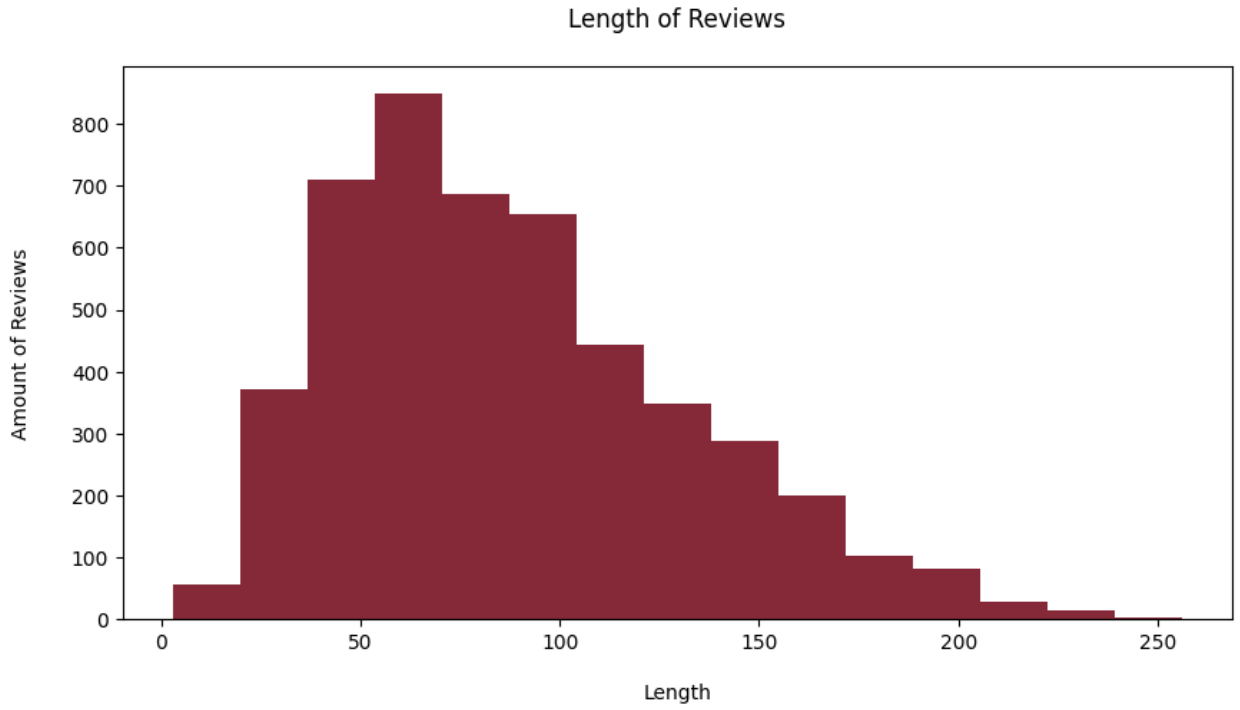
در این سلول ستون جدیدی تحت عنوان Polarity اضافه میشود و قطبیت هر نظر با استفاده از کتابخانه textblob محاسبه شده و در سلول متناظر هر نظر در این ستون قرار میگیرد. مقدار قطبیت از منفی یک تا مثبت یک خواهد بود. پس از آن نمودار هیستوگرام تعداد نظرات بر اساس قطبیت آنها رسم میشود. نتیجه این ترسیم به صورت زیر خوهد بود.



همانطور که در این نمودار هم مشاهده میشود تعداد نظرات با قطبیت خنثی یعنی صفر بیشتر از همه و بعد تعداد نظرات مثبت و بعد از آن تعداد نظرات منفی میباشد.

```
df['Length'] = df['Cleaned_sentence'].astype(str).apply(len)
df['Length'].plot(kind='hist', bins=15, linewidth=1, color='#852938',
figsize=(10, 5))
plt.title('Length of Reviews', pad=20)
plt.xlabel('Length', labelpad=15)
plt.ylabel('Amount of Reviews', labelpad=20)
plt.show()
```

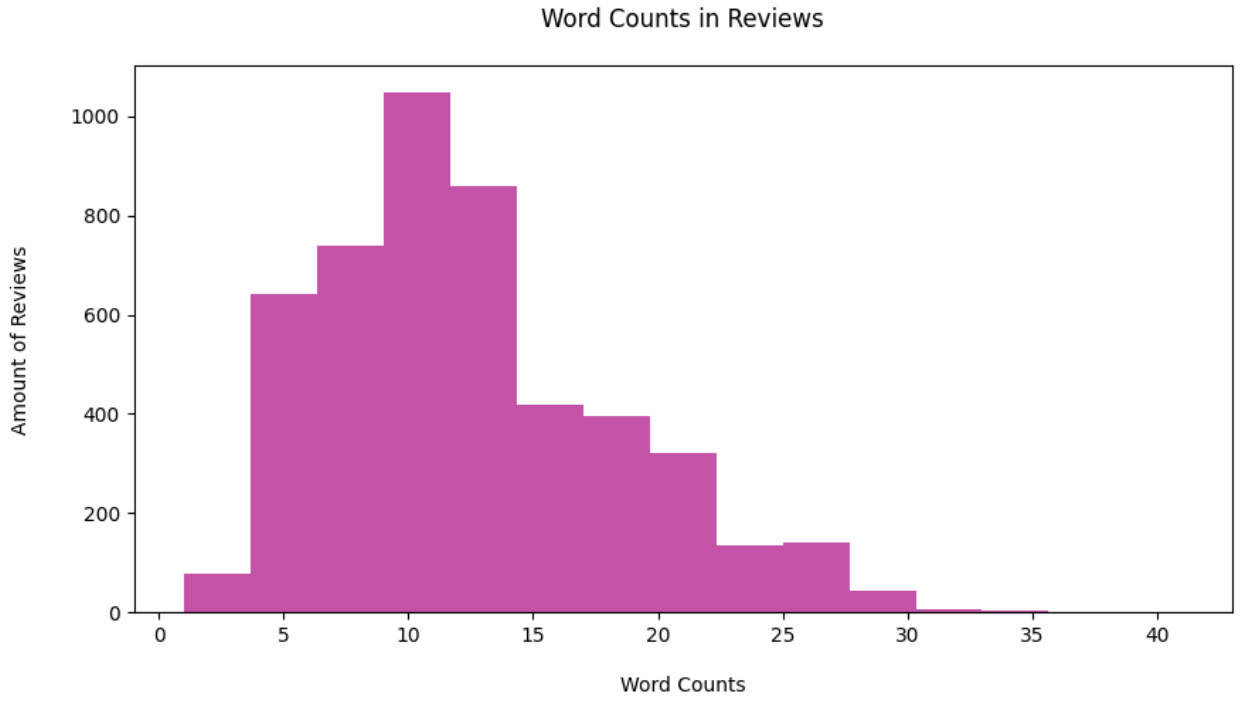
در این سلول طول هر یک از نظرات موجود در ستون Cleaned_sentence محاسبه میشود و در ستون جدیدی به نام Length ذخیره میگردد. در نهایت تعداد نظرات بر اساس طول آنها در یک نمودار رسم میشود.



همانطور که مشاهده میشود بیشتر نظرات طولی میان پنجاه تا صد کاراکتر دارند.

```
df['Word Counts'] = df['Cleaned_sentence'].apply(lambda x:
len(str(x).split()))
df['Word Counts'].plot(kind='hist', bins=15, linewidth=1, color='#C453A7',
figsize=(10, 5))
plt.title('Word Counts in Reviews', pad=20)
plt.xlabel('Word Counts', labelpad=15)
plt.ylabel('Amount of Reviews', labelpad=20)
plt.show()
```

در این سلول تعداد کلمات هر جمله در ستون Cleaned_sentence محاسبه میشود و در ستون جدید به نام Word Counts ذخیره میگردد. سپس با استفاده از اطلاعات این ستون تعداد نظرات بر اساس تعداد کلمات آنها در یک نمودار به صورت زیر ترسیم میگردد.



همانطور که در این نمودار مشاهده میشود بیشتر نظرات تعداد کلماتی میان ۵ تا ۱۵ کلمه داشته‌اند.

```
wc= WordCloud(background_color='white', random_state=1, max_words=2000,
width=3000, height=1500,
stopwords=Stopwords).generate(str(df['Cleaned_sentence']))
plt.figure(figsize=(15, 15))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.show()
```

در این سلول با استفاده از کلاس WordCloud یک ابر کلمات ایجاد میشود. این ابر اندازه هر کلمه بر اساس تکرار آن مشخص میکند. کلماتی که بیشتر تکرار شده‌اند بزرگتر خواهند بود و کلماتی که کمتر تکرار شده‌اند کوچکتر خواهند بود. ابر کلمه تولید شده به شکل زیر است.

بر اساس این نمودار میتوانیم نتیجه بگیریم که کلماتی مانند company و sale تکرار های زیادی داشته‌اند.

مدل‌های دسته‌بندی

```
X_train, X_test, y_train, y_test =
train_test_split(np.array(df['Cleaned_sentence']), np.array(df['Label']),
test_size=0.20, random_state=42)
```

در این قسمت داده‌ها به دو قسمت تست و ترین تقسیم میشوند. مقدار test_size نشان میدهد که ۲۰ درصد داده‌ها به عنوان داده تست انتخاب خواهند شد و باقیمانده داده‌ها برای آموزش خواهند بود. همچنین random_state مخالف صفر موجب میشود تا این تقسیم همیشه به یک صورت انجام گیرد.

```
tfidf = TfidfVectorizer(use_idf=True, tokenizer=word_tokenize,
min_df=0.00002, max_df=0.70)
X_train_tf = tfidf.fit_transform(X_train.astype('U'))
X_test_tf = tfidf.transform(X_test.astype('U'))
```

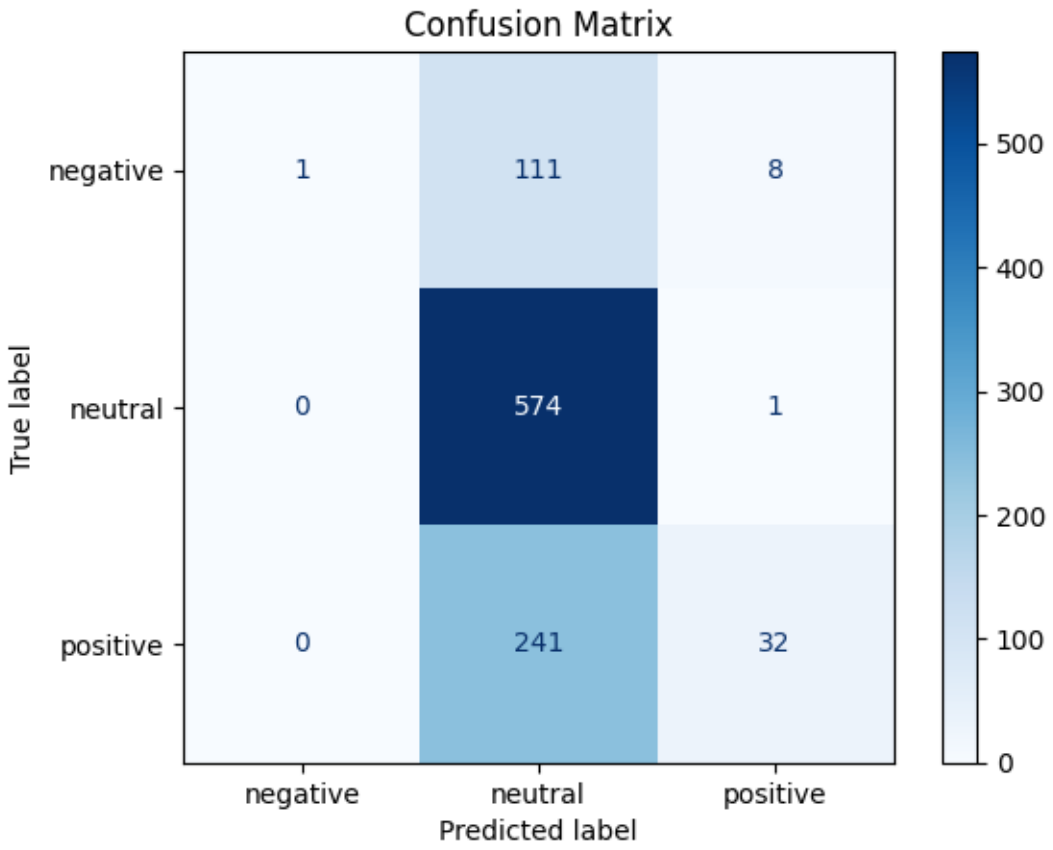
ویژگی‌های ما جملات و کلمات هستند اما برای پیاده‌سازی مدل‌های دسته‌بندی نیازمند داده‌های عددی هستیم. با استفاده از روش tf-idf هر نظر را تبدیل به یک بردار ویژگی عددی میکنیم. طبعا این عملیات باید برای هر دو داده تست و آموزش انجام شود.

```
rf = RandomForestClassifier(random_state=42)

parameters = {
    'n_estimators' : [100, 200, 300, 400],
    'max_depth' : [3, 5, 10],
    'min_samples_leaf' : [1, 2, 4]
}
gs = GridSearchCV(rf, parameters, cv=3)
gs.fit(X_train_tf, y_train)

rf.set_params(**gs.best_params_)

rf.fit(X_train_tf, y_train)

y_pred = rf.predict(X_test_tf)

cr = classification_report(y_test, y_pred, labels=[0, 1, 2],
target_names=('negative', 'neutral', 'positive'))
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=('negative', 'neutral', 'positive'))
print('With best parameters:')
print(pd.Series(gs.best_params_).to_string())
print(cr)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

در این قسمت ابتدا یک آبجکت از RandomForestClassifier ایجاد میشود. سپس در یک دیکشنری مقادیر متفاوتی از پارامتر های مورد نیاز این آبجکت تعریف میشود. با استفاده از GridSearchCV بهترین ترکیب مقادیر پارامتر ها از میان این پارامتر ها به دست می‌آید. سپس با استفاده از متود set_params این پارامترها روی مدل اصلی تنظیم میشوند. پس از آن مدل با استفاده از داده های آموزش، آموزش میبیند و در نهایت با متود predict و با استفاده از داده‌های تست مدل آموزش دیده را تست میکنیم. برای محاسبه مقادیری مانند precision، recall، accuracy و f1-score از تابع classification_report استفاده میکنیم. این تابع این مقادیر را به صورت خودکار محاسبه

میکند و تحت یک جدول خروجی را بازمیگرداند.همچنین با استفاده از تابع confusion_matrix، یک confusion matrix را محاسبه کرده و در نهایت به وسیله ConfusionMatrixDisplay به صورت زیبا شده تحت یک نمودار نمایش داده میشود. خروجی در زیر قابل مشاهده است.

```
With best parameters:
max_depth            10
min_samples_leaf      1
n_estimators        100
              precision    recall  f1-score   support

    negative       1.00      0.01      0.02       120
     neutral       0.62      1.00      0.76       575
    positive       0.78      0.12      0.20       273

    accuracy                           0.63       968
   macro avg       0.80      0.37      0.33       968
weighted avg       0.71      0.63      0.51       968
```

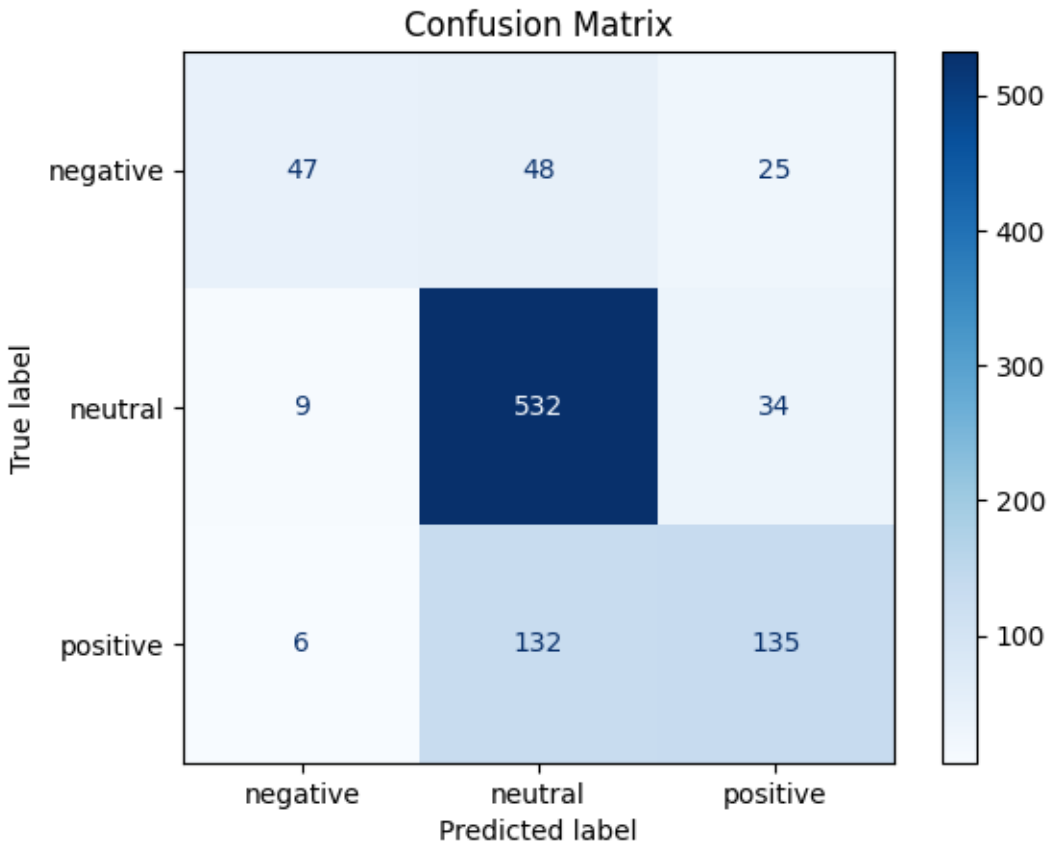


```
xgb = XGBClassifier(eval_metric='mlogloss', random_state=42)

parameters = { 'learning_rate': [0.001, 0.01, 0.1, 1] }
gs = GridSearchCV(xgb, parameters, cv=3)
gs.fit(X_train_tf, y_train)

xgb.set_params(**gs.best_params_)

xgb.fit(X_train_tf, y_train)

y_pred = xgb.predict(X_test_tf)

cr = classification_report(y_test, y_pred, labels=[0, 1, 2],
target_names=('negative', 'neutral', 'positive'))
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=('negative', 'neutral', 'positive'))
print('With best parameters:')
print(pd.Series(gs.best_params_).to_string())
print(cr)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

در این قسمت نیز مشابه مدل قبلی، ابتدا یک آبجکت از XGBClassifier ایجاد میشود. سپس در یک دیکشنری مقادیر متفاوتی از پارامتر learning_rate تعریف میشود. با استفاده از GridSearchCV بهترین مقدار این پارامتر از میان این پارامتر ها به دست می‌آید. سپس با استفاده از متود set_params این پارامترها روی مدل اصلی تنظیم میشوند. پس از آن مدل با استفاده از داده های آموزش، آموزش میبیند و در نهایت با متود predict و با استفاده از داده‌های تست مدل آموزش دیده را تست میکنیم. برای محاسبه مقادیری مانند precision، recall، accuracy و f1-score از تابع classification_report استفاده میکنیم. این تابع این مقادیر را به صورت خودکار محاسبه میکند و تحت یک جدول خروجی را بازمیگرداند.همچنین با استفاده از تابع confusion_matrix، یک confusion matrix را محاسبه کرده و در نهایت به وسیله ConfusionMatrixDisplay به صورت زیبا شده تحت یک نمودار نمایش داده میشود. خروجی در زیر قابل مشاهده است.

With best parameters:
learning_rate 0.1

	precision	recall	f1-score	support
negative	0.76	0.39	0.52	120
neutral	0.75	0.93	0.83	575
positive	0.70	0.49	0.58	273
accuracy			0.74	968
macro avg	0.73	0.60	0.64	968
weighted avg	0.73	0.74	0.72	968



شبکه عصبی

```
def plot_training_hist(history):
    fig, ax = plt.subplots(2, 2, figsize=(12, 8))
    # find precision and recall keys
    precision_key = list(filter(lambda v:
re.match(r'precision_\d+|precision', v), history.history.keys()))[0]
    recall_key = list(filter(lambda v: re.match(r'recall_\d+|recall', v),
history.history.keys()))[0]
    # first plot
    ax[0, 0].plot(history.history['accuracy'])
    ax[0, 0].plot(history.history['val_accuracy'])
    ax[0, 0].set_title('Model Accuracy')
    ax[0, 0].set_xlabel('epoch')
    ax[0, 0].set_ylabel('accuracy')
    ax[0, 0].legend(['train', 'validation'], loc='best')
    # second plot
    ax[0, 1].plot(history.history[precision_key])
    ax[0, 1].plot(history.history[f'val_{precision_key}'])
    ax[0, 1].set_title('Model Precision')
    ax[0, 1].set_xlabel('epoch')
    ax[0, 1].set_ylabel('precision')
    ax[0, 1].legend(['train', 'validation'], loc='best')
    #third plot
    ax[1, 0].plot(history.history[recall_key])
    ax[1, 0].plot(history.history[f'val_{recall_key}'])
    ax[1, 0].set_title('Model Recall')
    ax[1, 0].set_xlabel('epoch')
    ax[1, 0].set_ylabel('recall')
    ax[1, 0].legend(['train', 'validation'], loc='best')
    #fourth plot
    ax[1, 1].plot(history.history['loss'])
    ax[1, 1].plot(history.history['val_loss'])
    ax[1, 1].set_title('Model Loss')
    ax[1, 1].set_xlabel('epoch')
    ax[1, 1].set_ylabel('loss')
    ax[1, 1].legend(['train', 'validation'], loc='best')
    plt.tight_layout(h_pad=2.0, w_pad=2.0)
```

این تابع جهت رسم مقادیر precision، recall، accuracy و loss بر اساس تعداد epoch های انجام شده در فرآیند یادگیری شبکه عصبی پیاده‌سازی شده‌است و در ادامه کاربرد آن مشخص خواهد شد.

```
wv = KeyedVectors.load('/content/drive/MyDrive/University/word2vec-google-news-300.bin')
wv_dict = dict(zip(wv.key_to_index.keys(),  wv.vectors))
```

شرکت گوگل دیتاستی را طراحی کرده است و مدلی را آموزش داده است که در آن کلمات را با یک بردار سیصد تایی از اعداد متناظر کرده است. در این قسمت این دیتاست در کد بارگذاری شده‌است. این دیتاست حجم بالایی دارد و لینک گوگل درایو آن در این سند آمده‌است.

```
X_train, X_test, y_train, y_test = train_test_split(df.Cleaned_sentence, df.Label, test_size=0.2, random_state=42)

y_train_enc = to_categorical(y_train, 3)
y_test_enc = to_categorical(y_test, 3)
```

در این قسمت برای آموزش و تست مدل شبکه عصبی، داده‌ها به دو بخش تست و آموزش تقسیم شده‌اند. بیست درصد داده‌ها جهت تست و باقیمانده آنها برای آموزش خواهد بود. تابع to_categorical مقادیر y که اعدادی از بین صفر، یک یا دو بودند را مانند زیر تبدیل به اندیس آرایه های سه تایی میکنند.

y = 0 ➔ y = [1, 0, 0]
y = 1 ➔ y = [0, 1, 0]
y = 2 ➔ y = [0, 0, 1]

```
vectorizer = TextVectorization(output_mode='int', output_sequence_length=None)

vectorizer.adapt(list(X_train) + list(X_test))

xtrain_seq = vectorizer(X_train)
xtest_seq = vectorizer(X_test)

xtrain_pad = sequence.pad_sequences(xtrain_seq)
xtest_pad = sequence.pad_sequences(xtest_seq)

word_index = {word: idx for idx, word in enumerate(vectorizer.get_vocabulary())}
```

در این قسمت، در قسمت اول یک آبجکت از TextVectorization ساخته میشود. این آبجکت جملات را بر اساس الگوریتم‌هایی خاص به دنباله‌های عددی تبدیل میکند. در قسمت دوم پارامترهای مورد نیاز برای این تبدیل با

استفاده از داده‌های تست و آموزش به دست آورده میشوند. در قسمت سوم با استفاده از پارامترهای به دست آمده عملیات transform انجام میشود و جملات تبدیل به دنباله‌های عددی میشوند. در قسمت چهارم از آنجا که ممکن تعداد کلمات و طول هر جمله متفاوت باشد در نتیجه ممکن است طول دنباله‌های عددی متفاوت باشند در نتیجه این دنباله با استفاده از مقدار صفر طول‌های یکسان پیدا میکنند. و در قسمت آخر یک دیکشنری با کلید کمات و مقدار اندیس آنها در لغتنامه ساخته شده از دیتاست ساخته میشود. لغتنامه یک آرایه است شامل تمام کلمات منحصر به فرد تمام دیتاست.

```
embedding_matrix = np.zeros((len(word_index) + 1, 300))
for word, i in word_index.items():
    embedding_vector = wv_dict.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

در این قسمت یک ماتریس embedding ساخته میشود. یک آرایه دو بعدی با ابعاد اندازه تعداد کلمات به اضافه یک در سیصد متشکل از صفر ایجاد میشود. سپس هر کلمه و مقدار آن از دیکشنری word_index فراخوانی میشوند. پس از آن اگر آن کلمه در دیتابیس گوگل که پیشتر به آن اشاره شد موجود بود بردار سیصدتایی متناظر با آن کلمه در اندیس متناظر آن کلمه در ماتریس embedding ذخیره میشود. به عبارتی دیگر تمام کلمات لغتنامه را تبدیل به بردارهای عددی سیصدتایی متناظر میکند.

```
model = Sequential()

model.add(Embedding(len(word_index) + 1, 300, weights=[embedding_matrix],
trainable=False))

model.add(SpatialDropout1D(0.3))

model.add(LSTM(300, dropout = 0.3, recurrent_dropout = 0.3))

model.add(Dense(1024, activation = 'relu'))
model.add(Dropout(0.8))

model.add(Dense(1024, activation = 'relu'))
model.add(Dropout(0.8))

model.add(Dense(3))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy', Precision(), Recall()])
```

در این قسمت یک مدل توالی از شبکه عصبی ساخته میشود و لایه‌های متفاوتی به آن اضافه میشوند. این مدل از یک بردار تعبیه (embedding) از پیش آموزش دیده استفاده میکند. در ابتدا لایه Embedding به عنوان لایه اول به توالی اضافه میشود. این لایه کلمات ورودی را با استفاده از بردار تعبیه به dense vector به طول ثابت ۳۰۰ تبدیل میکند. پارامتر trainable برابر False میباشد و به این معناست که وزن‌های این لایه (بردار تعبیه) در طول آموزش بروزرسانی نخواهند شد.

سپس یک لایه SpatialDropout1D اضافه میشود. این لایه برای جلوگیری از وقوع overfitting اضافه میشود. این لایه به صورت تصادفی کسری از مقادیر ورودی را (در اینجا ۰.۳) در هر بروزرسانی در حین آموزش برابر صفر میکند. این موضوع به تقویت توانایی تعمیم مدل کمک میکند.

سپس یک لایه LSTM با سیصد واحد اضافه میگردد. لایه‌های LSTM برای داده‌های متوالی مانند متون بسیار پرکاربرد هستند. پارامترهای dropout و recurrent_dropout نیز برای جلوگیری از وقوع overfitting تنظیم شده‌اند.

در لایه‌های Dense هر نورون به نورون‌های لایه قبلی متصل میشود. در اینجا دو لایه Dense با ۱۰۲۴ نورون به مدل اضافه شده‌است. و میان این لایه‌ها نیز لایه‌های Dropout قرار گرفته است که به صورت تصادفی کسری از نورون‌ها را در حین آموزش برابر صفر میکند و با اطمینان از اینکه مدل به نورون‌های خاصی وابسته نیست از وقوع overfitting جلوگیری میکند.

در نهایت یک لایه Dense با سه نورون که به نورون‌های لایه قبل متصل است برای خروجی و یک لایه Activation اضافه میشود. خروجی سه کلاس مورد نظر خواهد بود و لایه Activation توزیع احتمال این سه کلاس متفاوت را محاسبه خواهد کرد.تابع softmax تابع مناسبی برای دسته‌بندی سه کلاسه میباشد.

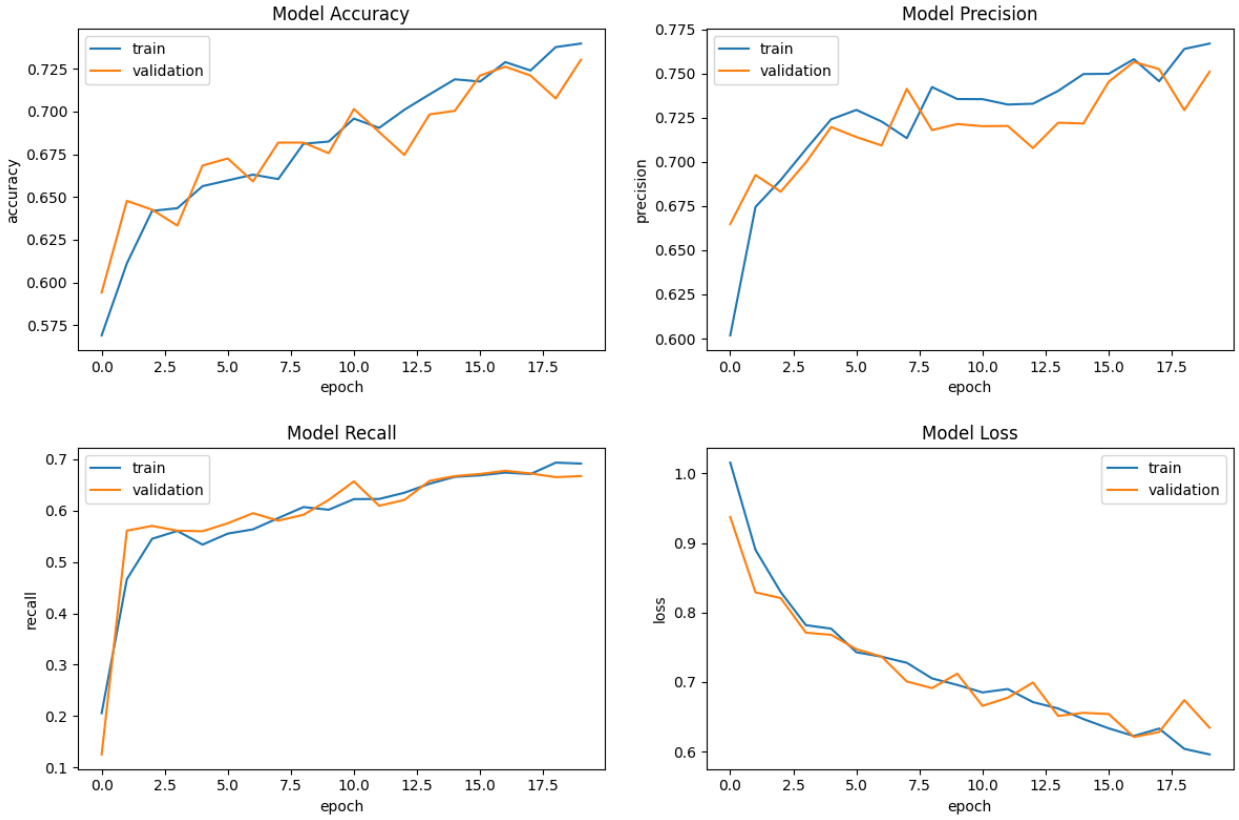
در نهایت مدل با استفاده از تابع هزینه categorical crossentropy، و بهینه ساز Adam و معیارهای ارزیابی precision، recall، accuracy کامپایل میشود. این معیارها کمک میکنند تا بازدهی مدل در حین آموزش و اعتبارسنجی ارزیابی شوند.

```
history = model.fit(xtrain_pad, y=y_train_enc, batch_size=512, epochs=20,
verbose=1, validation_data=(xtest_pad, y_test_enc))
```

در این مرحله مدل کامپایل شده با داده‌هایی که برای آموزش جدا شده بودند آموزش میبیند. Epoch ها در فایل کد قابل مشاهده هستند.

```
plot_training_hist(history)
```

در این قسمت همانطور که در پیش اشاره شد، تابع plot_training_hist فراخوانی میشود و نمودارهای معیارهای گوناگون بر اساس تعداد epoch ها رسم میشود که این نمودارها در زیر آمده‌اند.



با توجه به نمودارها مشاهده میشود که با افزایش epoch ها معیار های ارزیابی بهبود پیدا میکنند اما باید توجه داشت که اگر تعداد epoch بسیار زیاد هم باشند ممکن است overfitting رخ دهد.

```
y_pred = model.predict(xtest_pad)
cr = classification_report(np.argmax(y_test_enc, axis=1),
np.argmax(y_pred, axis=1), labels=[0,1,2], target_names=('negative',
'neutral', 'positive'))
cm = confusion_matrix(np.argmax(y_test_enc, axis=1), np.argmax(y_pred,
axis=1), labels=[0,1,2])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=('negative', 'neutral', 'positive'))
print(cr)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

در نهایت با استفاده از مدل آموزش دیده و داده‌هایی که برای تست جدا شده بودند مدل آموزش دیده تست میشود و نتایج آن و confusion matrix آن رسم میشود. این نتایج و confusion matrix در زیر قابل مشاهده است.

	precision	recall	f1-score	support
negative	0.54	0.65	0.59	120
neutral	0.76	0.93	0.84	575
positive	0.78	0.36	0.49	273
accuracy			0.73	968
macro avg	0.69	0.64	0.64	968
weighted avg	0.74	0.73	0.71	968