<u>SENTIMENT ANALYSIS FOR MARKETING</u>

<u>PHASE 4</u>

NAME	SRIRAM V
TEAM ID	proj_212173_Team_2

DEVELOPING MODEL PART 2

TOPIC: *Continue building the sentiment analysis solution by Employing NLP techniques, Generating insights.*

<u>Overview of the process:</u>

Sentiment analysis in marketing is a process that involves the use of natural language processing (NLP) techniques to assess and understand the sentiment or emotions expressed in customer feedback, comments, reviews, and other textual data. Here's an overview of the sentiment analysis process for marketing.

Prepare the data: Clean and prepare the data by removing noise, irrelevant information, and special characters.

Tokenize the text into words or phrases for analysis.

Normalize the text (e.g., converting to lowercase) to ensure consistency.

Perform feature selection: This can be done using a variety of methods, such as correlation analysis, information gain, and recursive feature elimination.

Train the model: There are many different machine learning algorithms that can be used for sentiment analysis. Some popular choices include linear regression, random forests, and gradient boostingmachines.

1. **Evaluate the model:** This can be done by calculating the meansquared error (MSE) or the root mean squared error (RMSE) of the model's predictions on the held-out test set.
2. **Deploy the model:** Once the model has been evaluated and found to be performing well, it can be deployed to production so that it can beused to predict the house prices of new houses.

PROCEDURE:

Feature selection:

1. **Identify the target variable.** This is the variable that you want topredict, such as house price.
2. **Explore the data.** This will help you to understand the relationships between the different features and the target variable. Youcan use data visualization and correlation analysis to identify features that are highly correlated with the target variable.
3. **Remove redundant features.** If two features are highly correlatedwith each other, then you can remove one of the features, as they are likely to contain redundant information.
4. **Remove irrelevant features.** If a feature is not correlated with thetarget variable, then you can remove it, as it is unlikely to be useful for prediction.

Model training:

1. **Choose a machine learning algorithm.** There are a number of different machine learning algorithms that can be used for sentiement analysis, such as BERT, RoBERTa, Linear regression, random forests,etc.,

Using linear regression:

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.feature_extraction.text import
    TfidfTransformer

from sklearn.feature_extraction.text import
    CountVectorizer

#splitting the data

train_data,test_data,train_labels,test_labels=train_test_spl
    it(df['text'],df['airline_sentiment'],test_size=0.2,
    random_state=42)



vectorizer = CountVectorizer()

train_data_counts = vectorizer.fit_transform(train_data)

test_data_counts = vectorizer.transform(test_data)
```

```
vec =TfidfTransformer()

train_vec=vec.fit_transform(train_data_counts)

test_vec=vec.transform(test_data_counts)

model=LogisticRegression(max_iter=10000)

model.fit(train_vec,train_labels)
```

OUT:

```
LogisticRegression(max_iter=10000)
```

IN:

```
#  Evaluate the model on the test set

accuracy = model.score(test_vec, test_labels)

print(f'Test accuracy: {accuracy:.4f}')
```

OUT:

```
Test accuracy: 0.6282
```

Using Pretrained model BERT

```
import torch

from transformers import
TFBertForSequenceClassification,BertTokenizer,AdamW,
get_linear_schedule_with_warmup,AutoModel,AutoToken
izer,BertModel
```

```
train_data, val_data = train_test_split(df, test_size=0.2,
random_state=42)

tokenizer = BertTokenizer.from_pretrained('bert-base-
uncased')

train_encoding=tokenizer(list(train_data['text']),truncation
=True,padding=True)

valid_encoding=tokenizer(list(val_data['text']),

truncation=True,padding=True)

sentiment_dict = {'positive': 0, 'negative': 1, 'neutral': 2}

train_labels =
train_data['airline_sentiment'].map(sentiment_dict).values.astype
('int64')

valid_labels =
val_data['airline_sentiment'].map(sentiment_dict).values.astype('
int64')

print("train label ",len(train_labels))

print("train label",len(valid_labels))

print("train_encoding  ",len(train_encoding))

print("valid_encoding ",len(valid_encoding))
```

OUT:

```
train label 11588

train label 2897

train_encoding  3
```

valid_encoding 3

IN:

```
# Split the data into training and validation sets

    tokenizer = BertTokenizer.from_pretrained('bert-base -
    buncased')

# Create TensorFlow datasets

train_dataset =
tf.data.Dataset.from_tensor_slices((dict(train_encoding),
train_labels)).shuffle(len(train_labels)).batch(32)

val_dataset =
tf.data.Dataset.from_tensor_slices((dict(valid_encoding),
valid_labels)).batch(32)

# Load the pre-trained BERT model for sequence classification

model =
TFBertForSequenceClassification.from_pretrained('bert-base-
uncased', num_labels=3)

# Fine-tune the model

optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5,
epsilon=1e-08, clipnorm=1.0)

loss = tf.keras.losses.SparseCategoricalCrossentropy

(from_logits=True)

metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

model.compile(optimizer=optimizer, loss=loss,
metrics=[metric])
```

```
history = model.fit(train_dataset, epochs=10,
validation_data=val_dataset)
```

OUT:

Downloading tf_model.h5: 100%

536M/536M [00:02<00:00, 214MB/s]

```
All model checkpoint layers were used when initializing
TFBertForSequenceClassification.

Some layers of TFBertForSequenceClassification were not
initialized from the model checkpoint at bert-base-uncased and
are newly initialized: ['classifier']
You should probably TRAIN this model on a down-stream task
to be able to use it for predictions and inference.

Epoch 1/10
363/363 [==============================] - 183s
354ms/step - loss: 0.7615 - accuracy: 0.6855 - val_loss: 0.6550 -
val_accuracy: 0.7349
Epoch 2/10
363/363 [==============================] - 117s
321ms/step - loss: 0.5929 - accuracy: 0.7590 - val_loss: 0.6105 -
val_accuracy: 0.7432
Epoch 3/10
363/363 [==============================] - 116s
319ms/step - loss: 0.4703 - accuracy: 0.8182 - val_loss: 0.6843 -
val_accuracy: 0.7528
Epoch 4/10
363/363 [==============================] - 115s
317ms/step - loss: 0.3387 - accuracy: 0.8768 - val_loss: 0.7497 -
val_accuracy: 0.7439
Epoch 5/10
363/363 [==============================] - 115s
```

```
315ms/step - loss: 0.2372 - accuracy: 0.9152 - val_loss: 0.8090 -
val_accuracy: 0.7539
Epoch 6/10
363/363 [==============================] - 117s
322ms/step - loss: 0.1711 - accuracy: 0.9396 - val_loss: 0.9858 -
val_accuracy: 0.7273
Epoch 7/10
363/363 [==============================] - 115s
316ms/step - loss: 0.1409 - accuracy: 0.9508 - val_loss: 1.0430 -
val_accuracy: 0.7335
Epoch 8/10
363/363 [==============================] - 115s
318ms/step - loss: 0.0993 - accuracy: 0.9669 - val_loss: 1.1365 -
val_accuracy: 0.7366
Epoch 9/10
363/363 [==============================] - 114s
315ms/step - loss: 0.0841 - accuracy: 0.9716 - val_loss: 1.2648 -
val_accuracy: 0.7432
Epoch 10/10
363/363 [==============================] - 115s
316ms/step - loss: 0.0684 - accuracy: 0.9789 - val_loss: 1.1900 -
val_accuracy: 0.7508
```

IN:

```
df['airline_sentiment'].value_counts()
```

OUT:

```
negative    9082
neutral     3069
positive    2334
```

Name: airline_sentiment, dtype: int64

IN:

import seaborn as sns

sns.countplot(data=df, x='airline_sentiment')

OUT:

<AxesSubplot:xlabel='airline_sentiment', ylabel='count'>

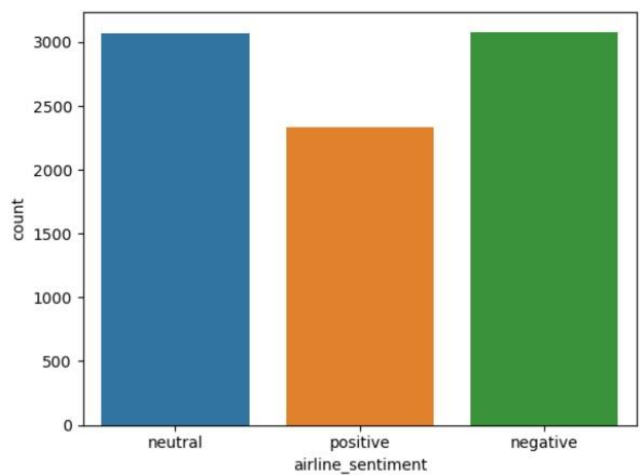
IN:

df_new=df.drop(df[df.airline_sentiment =='negative'].iloc[:6000].index)

sns.countplot(data=df_new, x='airline_sentiment')

OUT:

<AxesSubplot:xlabel='airline_sentiment', ylabel='count'>



After Dropping the 6000 data of negative sentiment the datasets seems to be balance

IN:

```
df_new['text']=df_new['text'].apply(preprocess_text)

train_data, test_data = train_test_split(df_new, test_size=0.2,
random_state=42)

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

train_encoding=tokenizer(list(train_data['text']),truncation=True,
padding=True)

test_encoding=tokenizer(list(test_data['text']),truncation=True,pa
dding=True)

sentiment_dict = {'positive': 0, 'negative': 1, 'neutral': 2}

train_labels =
train_data['airline_sentiment'].map(sentiment_dict).values.astype
('int64')

test_labels =
test_data['airline_sentiment'].map(sentiment_dict).values.astype(
'int64')

print(len(train_labels))

print(len(test_labels))

print(len(train_encoding))

print(len(test_encoding))
```

OUT:

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
# Create TensorFlow datasets
train_dataset =
tf.data.Dataset.from_tensor_slices((dict(train_encoding),
train_labels)).shuffle(len(train_labels)).batch(32)
val_dataset =
tf.data.Dataset.from_tensor_slices((dict(valid_encoding),
valid_labels)).batch(32)

# Load the pre-trained BERT model for sequence classification
model =
TFBertForSequenceClassification.from_pretrained('bert-base-
uncased', num_labels=3)

# Fine-tune the model
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5,
epsilon=1e-08, clipnorm=1.0)
loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True
)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss,
metrics=[metric])
history = model.fit(train_dataset, epochs=10,
validation_data=val_dataset)
```

OUT:

All model checkpoint layers were used when initializing TFBertForSequenceClassification.

Some layers of TFBertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Epoch 1/10
213/213 [==============================] - 125s 369ms/step - loss: 0.9499 - accuracy: 0.5514 - val_loss: 0.8267 - val_accuracy: 0.6458
Epoch 2/10
213/213 [==============================] - 74s 349ms/step - loss: 0.7917 - accuracy: 0.6690 - val_loss: 0.7899 - val_accuracy: 0.6631
Epoch 3/10
213/213 [==============================] - 72s 336ms/step - loss: 0.6574 - accuracy: 0.7335 - val_loss: 0.6111 - val_accuracy: 0.7528
Epoch 4/10
213/213 [==============================] - 71s 331ms/step - loss: 0.5206 - accuracy: 0.8005 - val_loss: 0.6245 - val_accuracy: 0.7677
Epoch 5/10
213/213 [==============================] - 71s 331ms/step - loss: 0.3808 - accuracy: 0.8631 - val_loss: 0.6166 - val_accuracy: 0.8001
Epoch 6/10
213/213 [==============================] - 71s 332ms/step - loss: 0.2793 - accuracy: 0.9023 - val_loss: 0.8184 - val_accuracy: 0.7718
Epoch 7/10
213/213 [==============================] - 71s 332ms/step - loss: 0.2128 - accuracy: 0.9262 - val_loss: 0.6901 - val_accuracy: 0.8064
Epoch 8/10
213/213 [==============================] - 70s 331ms/step - loss: 0.1604 - accuracy: 0.9445 - val_loss: 0.9027 - val_accuracy: 0.7829

Epoch 9/10
213/213 [==============================] - 71s 332ms/step - loss: 0.1439 - accuracy: 0.9517 - val_loss: 1.0076 - val_accuracy: 0.7736
Epoch 10/10
213/213 [==============================] - 70s 331ms/step - loss: 0.1116 - accuracy: 0.9613 - val_loss: 0.9026 - val_accuracy: 0.8136

Simple models

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(X_train_tfv,y_train)
```

MultinomialNB()

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(max_iter=1000)
log.fit(X_train_tfv,y_train)
```

LogisticRegression(max_iter=1000)

```
from sklearn.svm import LinearSVC
svc = LinearSVC()
svc.fit(X_train_tfv,y_train)
```

LinearSVC()

```
from sklearn.metrics import plot_confusion_matrix,classification_report
def report(model):
    preds = model.predict(X_test_tfv)
    print(classification_report(y_test,preds))
    plot_confusion_matrix(model,X_test_tfv,y_test)
```

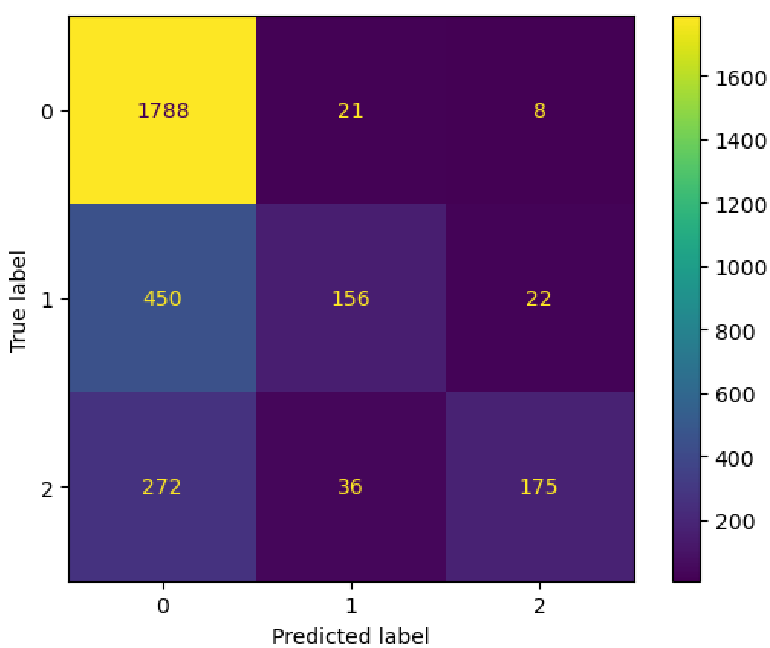
```
print("NB MODEL")
report(nb)
```

```
NB MODEL
              precision    recall  f1-score   support

           0       0.71      0.98      0.83      1817
           1       0.73      0.25      0.37       628
           2       0.85      0.36      0.51       483

    accuracy                           0.72      2928
   macro avg       0.77      0.53      0.57      2928
weighted avg       0.74      0.72      0.68      2928


/opt/conda/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



```
print("Logistic Regression")
report(log)
```

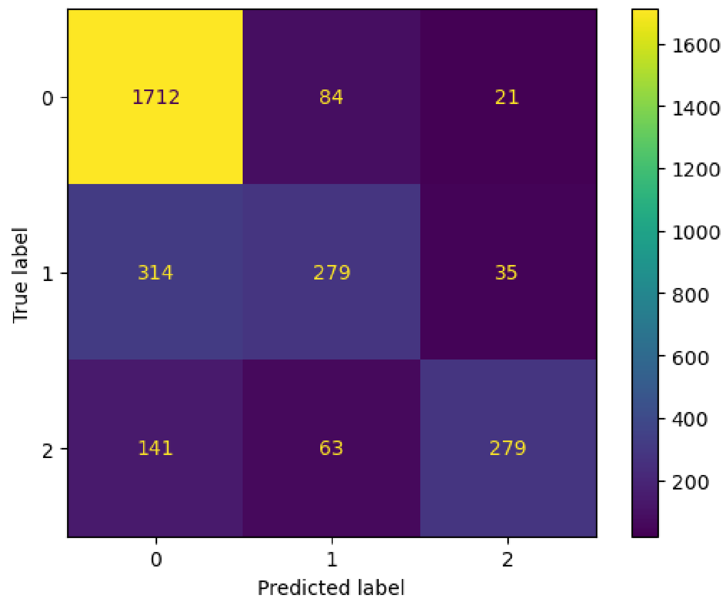
OUT:

```
Logistic Regression
              precision    recall  f1-score   support

           0       0.79      0.94      0.86      1817
           1       0.65      0.44      0.53       628
           2       0.83      0.58      0.68       483

    accuracy                           0.78      2928
   macro avg       0.76      0.65      0.69      2928
weighted avg       0.77      0.78      0.76      2928


/opt/conda/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



print('SVC')
report(svc)

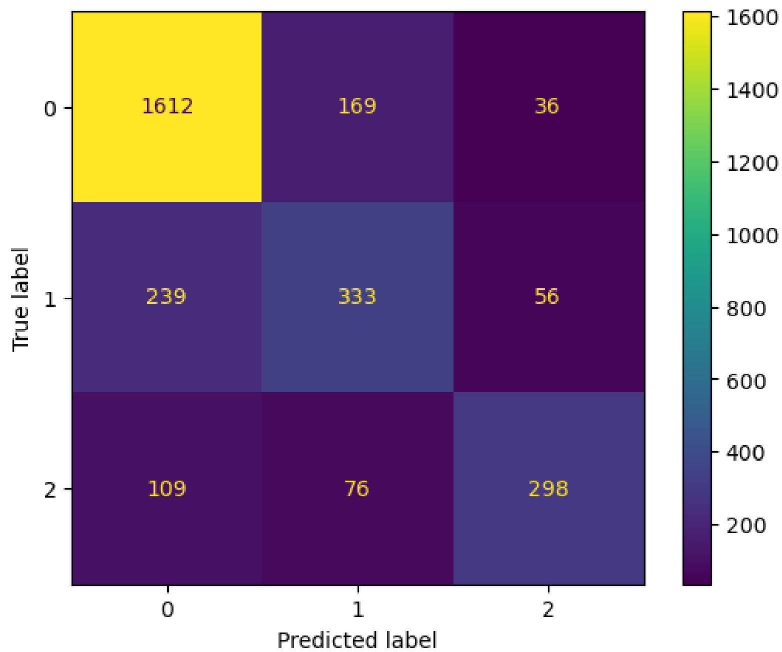
OUT:

SVC

	precision	recall	f1-score	support
0	0.82	0.89	0.85	1817
1	0.58	0.53	0.55	628
2	0.76	0.62	0.68	483
accuracy			0.77	2928
macro avg	0.72	0.68	0.70	2928
weighted avg	0.76	0.77	0.76	2928

/opt/conda/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:

ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)



Pipeline

from sklearn.pipeline **import** Pipeline

```
pipe = Pipeline([('tfidf',TfidfVectorizer()),
                 ('svc',LinearSVC())])
pipe.fit(raw_data['text'],raw_data['labels'])
```

Pipeline(steps=[('tfidf', TfidfVectorizer()), ('svc', LinearSVC())])

```
new_positive_tweet = ['good flight']
pipe.predict(new_positive_tweet)
```

array([2])

```
new_negative_tweet = ['bad flight']
pipe.predict(new_negative_tweet)
```

```
array([0])
```

```
new_neutral_tweet = ['ok flight']
pipe.predict(new_neutral_tweet)
```

```
array([1])
```

```
##pandasDF --> Hugging Face dataset
from datasets import Dataset
dataset = {"text": raw_data["text"].tolist(), "labels":
raw_data["labels"].tolist()}
dataset = Dataset.from_dict(dataset)
dataset = dataset.train_test_split(train_size=0.8, seed=101)
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'labels'],
        num_rows: 11712
    })
    test: Dataset({
        features: ['text', 'labels'],
        num_rows: 2928
    })
})
```

IN:

```
import tensorflow as tf
from transformers import
TFAutoModelForSequenceClassification, AutoTokenizer,
AutoConfig,DataCollatorWithPadding
from scipy.special import softmax


#cardiffnlp/twitter-roberta-base-sentiment
```

```
checkpoint = 'cardiffnlp/twitter-roberta-base-sentiment-latest'

batch_size = 16
num_epochs = 5

config = AutoConfig.from_pretrained(checkpoint)
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model =
TFAutoModelForSequenceClassification.from_pretrained(check
point, num_labels=3)
```

OUT:

{"model_id":"c68fd6d69d594f61ba400da869640125","version_
major":2,"version_minor":0}

{"model_id":"0defd486c4dd465ea5ad6a7f166867f0","version_
major":2,"version_minor":0}

{"model_id":"29bda555fd4f4ebc8fc2a88d262c7dce","version_m
ajor":2,"version_minor":0}

{"model_id":"e8f512df42c446eaa5f1e87f84ff18f2","version_ma
jor":2,"version_minor":0}

{"model_id":"e521fb77e7364bf4af34b516f5790788","version_
major":2,"version_minor":0}

All model checkpoint layers were used when initializing TFRobertaForSequenceClassification.

Some layers of TFRobertaForSequenceClassification were not initialized from the model checkpoint at cardiffnlp/twitter-roberta-base-sentiment-latest and are newly initialized: ['classifier']
You should probably TRAIN this model on a down-stream task

to be able to use it for predictions and inference.

```
def tokenize_function(example):

    return tokenizer(example['text'], truncation=True,
max_length = 35)


tokenized_datasets = dataset.map(tokenize_function,
batched=True,)

data_collator = DataCollatorWithPadding(tokenizer=tokenizer,
return_tensors="tf")
```

OUT:

{"model_id":"01f5e00

46d6f42f7a15365b6bd246b32","version_major":2,"version_minor":0}

{"model_id":"c3e58a6367704ebb9ed92e43ae966e92","version_major":2,"version_minor":0}

```
tf_train_dataset = tokenized_datasets["train"].to_tf_dataset(
    columns=["attention_mask", "input_ids", "token_type_ids"],
    label_cols=["labels"],
    shuffle=True,
    collate_fn=data_collator,
    batch_size=batch_size
)
tf_validation_dataset = tokenized_datasets["test"].to_tf_dataset(
    columns=["attention_mask", "input_ids", "token_type_ids"],
    label_cols=["labels"],
    shuffle=False,
    collate_fn=data_collator,
```

```
    batch_size=batch_size
)
```

You're using a RobertaTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using a method to encode the text followed by a call to the `pad` method to get a padded encoding.

```
from tensorflow.keras.optimizers.schedules import PolynomialDecay
from tensorflow.keras.optimizers import Adam

num_train_steps = len(tf_train_dataset) * num_epochs
lr_scheduler = PolynomialDecay(
    initial_learning_rate=5e-5, end_learning_rate=0.0, decay_steps=num_train_steps
)

opt = Adam(learning_rate=lr_scheduler)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=opt, loss=loss, metrics=['accuracy'])
model.fit(tf_train_dataset,validation_data=tf_validation_dataset, epochs=3, batch_size=batch_size)
```

```
Epoch 1/3
732/732 [==============================] - 145s 135ms/step - loss: 0.4544 - accuracy: 0.8277 - val_loss: 0.4202 - val_accuracy: 0.8361
Epoch 2/3
732/732 [==============================] - 80s 109ms/step - loss: 0.2816 - accuracy: 0.8984 - val_loss: 0.4345 -
```

val_accuracy: 0.8545
Epoch 3/3
732/732 [==============================] - 82s
112ms/step - loss: 0.1579 - accuracy: 0.9468 - val_loss: 0.5486 -
val_accuracy: 0.8446

<keras.callbacks.History at 0x79e6d69a04d0>

IN:

```
from transformers import pipeline

classifier = pipeline("sentiment-
analysis",tokenizer=tokenizer,model=model)

predicted_labels = []
for text in X_test:
    result = classifier(text)
    predicted_label = result[0]['label']
    predicted_labels.append(predicted_label)

df = pd.DataFrame(X_test)
df['predictions'] = predicted_labels
df['labels'] = df["predictions"].apply(lambda x: 0 if x ==
"negative" else 1 if x == "neutral" else 2)

df.head()
```

OUT:

	text	predictions	labels
4814	thanks very excited to see it d	positive	2
150	does that mean you don t have a policy for des...	negative	0
5322	any official word whether flight from bwi to m...	neutral	1
4885	i miss mine terribly a for my th anniversary w...	positive	2
7504	at what time all these passengers were sitting...	negative	0

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
print(classification_report(y_test,df['labels']))
```

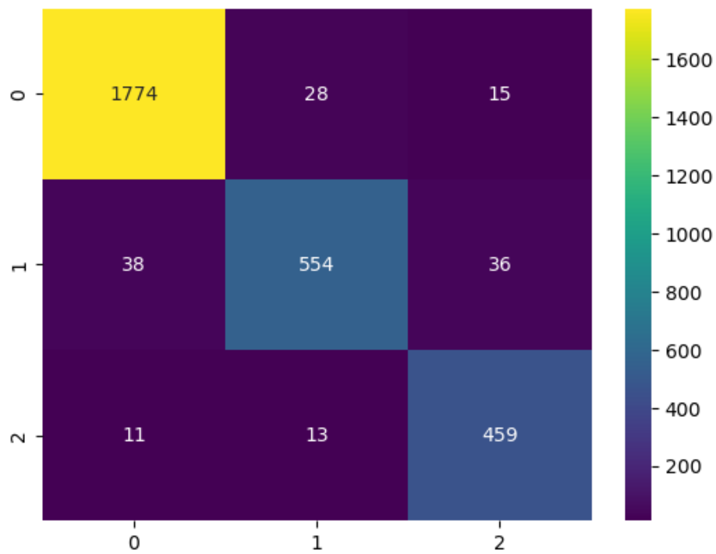
```
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      1817
           1       0.93      0.88      0.91       628
           2       0.90      0.95      0.92       483

    accuracy                           0.95      2928
   macro avg       0.93      0.94      0.94      2928
weighted avg       0.95      0.95      0.95      2928
```

```
sns.heatmap(confusion_matrix(y_test,df['labels']),cmap='viridis',
annot=True,fmt='d')
```

```
<AxesSubplot:>
```



new_positive_tweet = ['good flight']
classifier(new_positive_tweet)

OUT:

[{'label': 'positive', 'score': 0.9946355223655701}]

new_negative_tweet = ['bad flight']
classifier(new_negative_tweet)

OUT:

[{'label': 'negative', 'score': 0.9965829253196716}]

new_neutral_tweet = ['ok flight']
classifier(new_neutral_tweet)

OUT:

[{'label': 'neutral', 'score': 0.951370358467102}]

CONCLUSION:

Sentiment analysis of Twitter US airline data using the BERT model is a powerful and effective tool for understanding customer opinions and emotions in the airline industry. This approach allows airlines to gain valuable insights into passenger sentiment, which can be pivotal for various aspects of their operations and customer service:

1. Improved Customer Service: By monitoring sentiment, airlines can proactively address customer concerns and issues, leading to better customer experiences.

2. Crisis Management: Sentiment analysis using BERT can help airlines identify and respond to potential PR crises quickly.

3. Marketing and Campaigns: Airlines can fine-tune their marketing strategies based on the sentiments expressed by customers on social media, enabling more targeted and resonant campaigns.

4. Product and Service Enhancement: Understanding customer sentiment provides valuable feedback for improving in-flight services, amenities, and operational aspects.

5. Real-time Feedback Loop: The use of BERT in sentiment analysis ensures that airlines have access to real-time feedback, enabling them to adapt swiftly to customer preferences and concerns.

In essence, sentiment analysis using BERT is a vital tool for airlines to gauge and react to customer sentiment, thereby enhancing customer satisfaction, refining marketing strategies, and ultimately improving their overall services. It demonstrates the power of NLP and machine learning in gaining insights from vast social media data.