Sentiment Analysis for Twitter US Airline Sentiment Dataset

Load Libraries

In [1]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
```

Initialize GPU

In [2]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_virtual_device_configuration(gpus[0], [tf.config.experimental.VirtualDeviceConfiguration(memory
    except RuntimeError as e:
        print(e)
```

Load Data

In [3]:
```
df = pd.read_csv('Tweets.csv')
df.head()
```

Out[3]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	na
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cai
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnarc
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnal

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	na
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnarc
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnarc

In [5]:
```
df = df[['text', 'airline_sentiment']]
df.head()
```

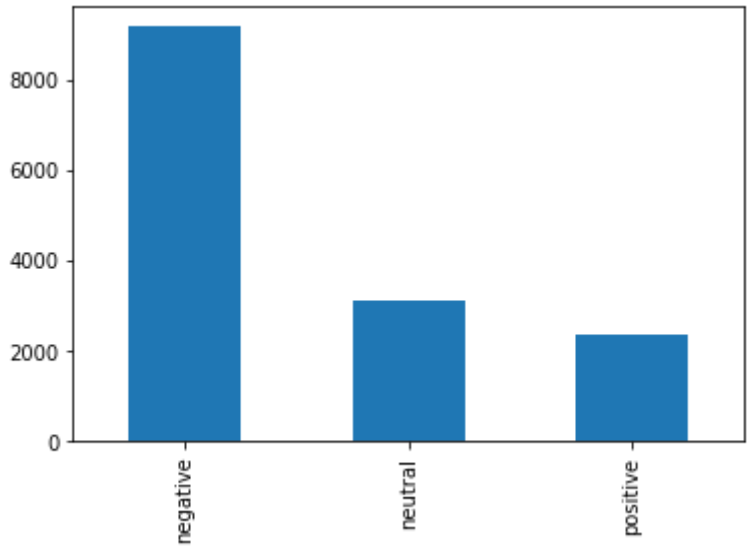
Out[5]:

	text	airline_sentiment
0	@VirginAmerica What @dhepburn said.	neutral
1	@VirginAmerica plus you've added commercials t...	positive
2	@VirginAmerica I didn't today... Must mean I n...	neutral
3	@VirginAmerica it's really aggressive to blast...	negative
4	@VirginAmerica and it's a really big bad thing...	negative

Data Visualization

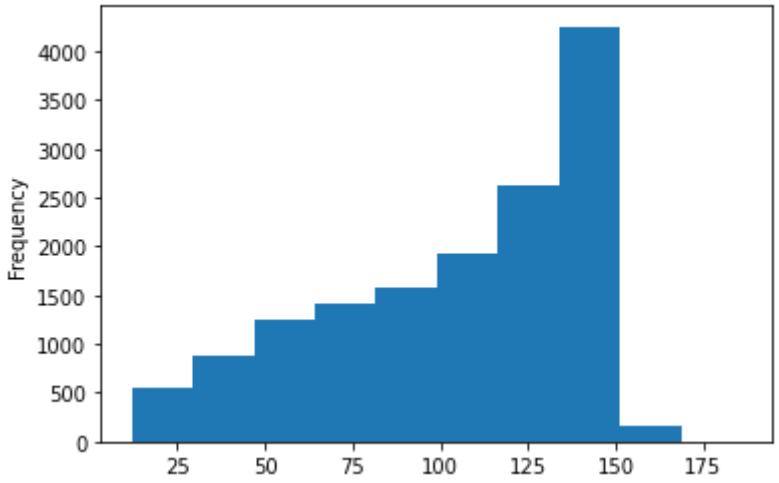
In [6]:
```
df['airline_sentiment'].value_counts().sort_index().plot.bar()
```

Out[6]:
```
<AxesSubplot:>
```



In [7]:
```
df['text'].str.len().plot.hist()
```

Out[7]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [8]:
```
df['text'].str.len().max()
```

Out[8]:
```
186
```

Text Preprocessing

In [9]:
```
import re

df['text'] = df['text'].str.replace('@VirginAmerica', '')
df['text'] = df['text'].apply(lambda x: x.lower())
df['text'] = df['text'].apply(lambda x: re.sub('[^a-zA-Z0-9\s]', '', x))
df.head()
```

Out[9]:

	text	airline_sentiment
0	what dhepburn said	neutral
1	plus youve added commercials to the experienc...	positive
2	i didnt today must mean i need to take anothe...	neutral
3	its really aggressive to blast obnoxious ente...	negative
4	and its a really big bad thing about it	negative

In [10]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer(num_words=500, split=" ")
tokenizer.fit_on_texts(df['text'].values)

X = tokenizer.texts_to_sequences(df['text'].values)
X = pad_sequences(X)
X[:5]
```

Out[10]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  57, 217],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   1,   2, 196],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   3, 184,  97,   3,  75,
          1, 147, 142, 190],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,  64, 131,   1,  15,  21,
```

```
  61,  54,  22, 486],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,  10,  64,   4, 131, 468,
 206, 487,  80,  20]])
```

In [11]:

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

model = Sequential()
model.add(Embedding(500, 256, input_length=X.shape[1]))
model.add(Dropout(0.3))
model.add(LSTM(256, return_sequences=True, dropout=0.3, recurrent_dropout=0.2))
model.add(LSTM(256, dropout=0.3, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU k
ernel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU
kernel as fallback when running on GPU
```

In [12]:

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 30, 256)           128000
_________________________________________________________________
dropout (Dropout)            (None, 30, 256)           0
_________________________________________________________________
lstm (LSTM)                  (None, 30, 256)           525312
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               525312
_________________________________________________________________
dense (Dense)                (None, 3)                 771
=================================================================
Total params: 1,179,395
Trainable params: 1,179,395
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

```
y = pd.get_dummies(df['airline_sentiment']).values
for i in range(5):
    print(df['airline_sentiment'][i], y[i])
```

```
neutral [0 1 0]
positive [0 0 1]
neutral [0 1 0]
negative [1 0 0]
negative [1 0 0]
```

In [14]:

```
from sklearn.model_selection import train_test_split

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(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(11712, 30)
(2928, 30)
(11712, 3)
(2928, 3)
```

In [18]:

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(
    monitor='accuracy',
    restore_best_weights=True,
    min_delta=0,
    patience=5,
    verbose=1,
    mode='auto'
)

history = model.fit(
    X_train,
    y_train,
    epochs=50,
    batch_size=64,
    callbacks=[early_stop]
)
```

```
Epoch 1/50
```

```
183/183 [==============================] - 38s 210ms/step - loss: 0.1992 - accuracy: 0.9208
Epoch 2/50
183/183 [==============================] - 38s 210ms/step - loss: 0.1798 - accuracy: 0.9276
Epoch 3/50
183/183 [==============================] - 39s 211ms/step - loss: 0.1666 - accuracy: 0.9352
Epoch 4/50
183/183 [==============================] - 39s 211ms/step - loss: 0.1489 - accuracy: 0.9419
Epoch 5/50
183/183 [==============================] - 39s 211ms/step - loss: 0.1477 - accuracy: 0.9422
Epoch 6/50
183/183 [==============================] - 39s 211ms/step - loss: 0.1442 - accuracy: 0.9450
Epoch 7/50
183/183 [==============================] - 38s 209ms/step - loss: 0.1259 - accuracy: 0.9549
Epoch 8/50
183/183 [==============================] - 38s 210ms/step - loss: 0.1162 - accuracy: 0.9563
Epoch 9/50
183/183 [==============================] - 39s 212ms/step - loss: 0.1088 - accuracy: 0.9616
Epoch 10/50
183/183 [==============================] - 39s 211ms/step - loss: 0.1092 - accuracy: 0.9593
Epoch 11/50
183/183 [==============================] - 40s 216ms/step - loss: 0.0996 - accuracy: 0.9635
Epoch 12/50
183/183 [==============================] - 39s 211ms/step - loss: 0.0967 - accuracy: 0.9639
Epoch 13/50
183/183 [==============================] - 39s 211ms/step - loss: 0.0879 - accuracy: 0.9678
Epoch 14/50
183/183 [==============================] - 38s 209ms/step - loss: 0.0855 - accuracy: 0.9682
Epoch 15/50
183/183 [==============================] - 39s 212ms/step - loss: 0.0854 - accuracy: 0.9685
Epoch 16/50
183/183 [==============================] - 39s 212ms/step - loss: 0.0746 - accuracy: 0.9732
Epoch 17/50
183/183 [==============================] - 39s 211ms/step - loss: 0.0759 - accuracy: 0.9719
Epoch 18/50
183/183 [==============================] - 39s 212ms/step - loss: 0.0782 - accuracy: 0.9728
Epoch 19/50
183/183 [==============================] - 38s 208ms/step - loss: 0.0749 - accuracy: 0.9735
Epoch 20/50
183/183 [==============================] - 39s 212ms/step - loss: 0.0743 - accuracy: 0.9731
Epoch 21/50
183/183 [==============================] - 39s 211ms/step - loss: 0.0626 - accuracy: 0.9786
Epoch 22/50
183/183 [==============================] - 39s 214ms/step - loss: 0.0652 - accuracy: 0.9770
Epoch 23/50
```

```
183/183 [==============================] - 38s 209ms/step - loss: 0.0611 - accuracy: 0.9785
Epoch 24/50
183/183 [==============================] - 38s 210ms/step - loss: 0.0609 - accuracy: 0.9777
Epoch 25/50
183/183 [==============================] - 39s 213ms/step - loss: 0.0614 - accuracy: 0.9766
Epoch 26/50
183/183 [==============================] - 39s 214ms/step - loss: 0.0624 - accuracy: 0.9784
Restoring model weights from the end of the best epoch.
Epoch 00026: early stopping
```

In [19]:

```
model.save_weights('sentiment_model_weights.h5')
model.save('sentiment_model.h5')
```

In [20]:

```
predictions = model.predict(X_test)

for i in range(5):
    print(df['text'][i], predictions[i], y_test[i])
```

```
 what dhepburn said [7.5271464e-07 4.0145472e-04 9.9959785e-01] [0 0 1]
 plus youve added commercials to the experience tacky [9.9997377e-01 2.4588608e-05 1.6238899e-06] [1 0 0]
 i didnt today must mean i need to take another trip [1.0000000e+00 5.6908078e-10 2.8734148e-09] [1 0 0]
 its really aggressive to blast obnoxious entertainment in your guests faces amp they have little recourse [6.3326021e-05 9.999266
9e-01 1.0030033e-05] [0 1 0]
 and its a really big bad thing about it [1.0358228e-04 9.9988639e-01 9.9740391e-06] [1 0 0]
```

In [24]:

```
pos_count, neu_count, neg_count = 0, 0, 0
real_pos, real_neu, real_neg = 0, 0, 0

for i, prediction in enumerate(predictions):
    if np.argmax(prediction)==2:
        pos_count+=1
    elif np.argmax(prediction)==1:
        neu_count+=1
    else:
        neg_count+=1

    if np.argmax(y_test[i])==2:
        real_pos+=1
    elif np.argmax(y_test[i])==1:
        real_neu+=1
    else:
        real_neg+=1
```

```
print('Positive Predictions:', pos_count)
print('Neutral Predictions:', neu_count)
print('Negative Predictions:', neg_count)
print('Real Positive:', real_pos)
print('Real Neutral:', real_neu)
print('Real Negative:', real_neg)
```

```
Positive Predictions: 440
Neutral Predictions: 579
Negative Predictions: 1909
Real Positive: 459
Real Neutral: 580
Real Negative: 1889
```