GOAL: TO PERFORM SENTIMENT ANALYSIS ON AMAZON FOOD REVIEW.

In [37]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from wordcloud import WordCloud

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report,confusion_matrix
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Embedding,Flatten,SimpleRNN,Bidirectional,

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.linear_model import LogisticRegression
from sklearn.tree import  DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC

import warnings
warnings.filterwarnings("ignore")
```

In [38]:
```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/food_review.csv")
```

In [39]:
```
df.head()
```

Out[39]:

	Unnamed: 0	Text	Score
0	0	I bought these from a large chain pet store. a...	1
1	1	This soup is incredibly good! But honestly, I...	5
2	2	Our family loves these tasty and healthy sesam...	5
3	3	The local auto shop offers this free to it cus...	4
4	4	I brought 2 bottles. One I carry in my pocket...	5

In [40]:
```
df.drop(["Unnamed: 0"], axis = 1, inplace= True)
```

In [41]:
```
df.shape
```

Out[41]:
```
(40500, 2)
```

In [42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40500 entries, 0 to 40499
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
```

```
---  ------  --------------  -----
 0   Text    40500 non-null  object
 1   Score   40500 non-null  int64
dtypes: int64(1), object(1)
memory usage: 632.9+ KB
```

In [43]:
```
df.isnull().sum()
```

Out[43]:
```
Text     0
Score    0
dtype: int64
```

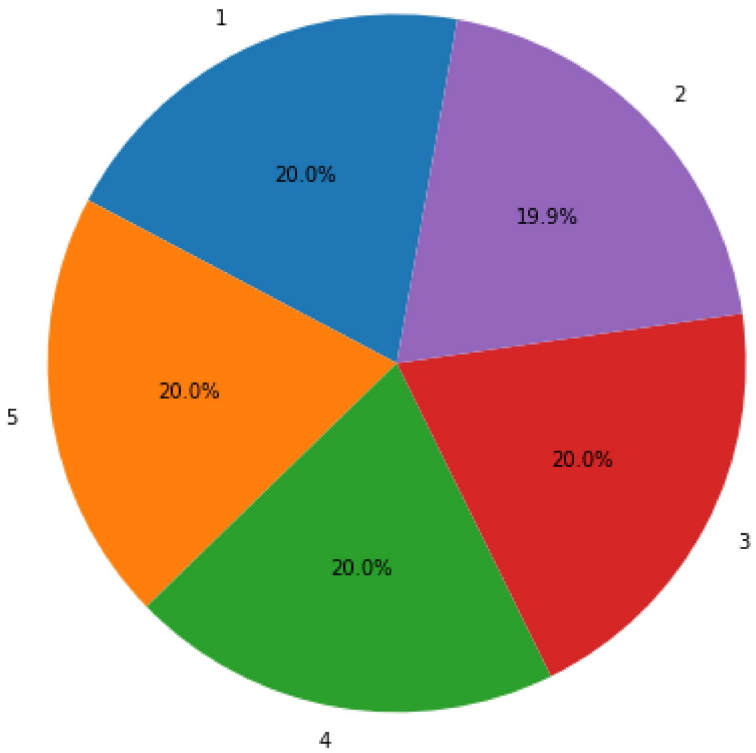
In [44]:
```
df.isin(['?']).sum(axis=0)
```

Out[44]:
```
Text     0
Score    0
dtype: int64
```

In [ ]:
```
df['Score'].value_counts()
```

Out[ ]:
```
4    8117
5    8106
3    8103
2    8095
1    8079
Name: Score, dtype: int64
```

In [ ]:
```
plt.figure(figsize=(8,8))
plt.pie(df['Score'].value_counts(),labels=df['Score'].unique(), autopct="%.1f%%", st
plt.show()
```



Based on the value counts and pie plot we can conclude that we do not have imbalanced data.

VISUALIZING EACH LABLE USING

WORDCLOUD:

In [ ]:
```
#LABEL 1
wc = WordCloud(width = 800,
               height = 800,
               background_color ='white',
               min_font_size = 10)

wc.generate(''.join(df[df["Score"]==1]["Text"]))

plt.figure(figsize = (8,8))
plt.imshow(wc)
plt.axis("off")
plt.show()
```

In [ ]:
```
#LABEL 2
wc = WordCloud(width = 800,
               height = 800,
               background_color ='white',
               min_font_size = 10)

wc.generate(''.join(df[df["Score"]==2]["Text"]))

plt.figure(figsize = (8,8))
plt.imshow(wc)
plt.axis("off")
plt.show()
```

In [ ]:

```
#LABEL 3
wc = WordCloud(width = 800,
               height = 800,
               background_color ='white',
               min_font_size = 10)

wc.generate(''.join(df[df["Score"]==3]["Text"]))

plt.figure(figsize = (8,8))
plt.imshow(wc)
plt.axis("off")
plt.show()
```

In [ ]:

```
#LABEL 4:
wc = WordCloud(width = 800,
               height = 800,
               background_color ='white',
               min_font_size = 10)

wc.generate(''.join(df[df["Score"]==4]["Text"]))

plt.figure(figsize = (8,8))
plt.imshow(wc)
plt.axis("off")
plt.show()
```

In [ ]:

```
#LABEL5
wc = WordCloud(width = 800,
               height = 800,
               background_color ='white',
               min_font_size = 10)

wc.generate(''.join(df[df["Score"]==5]["Text"]))

plt.figure(figsize = (8,8))
plt.imshow(wc)
plt.axis("off")
plt.show()
```

In [10]:
```
X = df["Text"]
y = df["Score"]
```

In [11]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)
```

Vectorization:

Count vectorization

In [47]:
```
cv = CountVectorizer(stop_words="english")
```

In [48]:
```
X_train_cv = cv.fit_transform(X_train)
X_test_cv = cv.transform(X_test)
```

LogisticRegression

In [ ]:
```
lr = LogisticRegression()
```

In [ ]:
```
lr.fit(X_train_cv, y_train)
```

Out[ ]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [ ]:
```
y_pred = lr.predict(X_test_cv)
```

In [ ]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support
```

```
              1       0.60      0.61      0.60      2409
              2       0.44      0.44      0.44      2426
              3       0.44      0.41      0.43      2492
              4       0.47      0.46      0.47      2420
              5       0.61      0.64      0.63      2403

       accuracy                           0.51     12150
      macro avg       0.51      0.51      0.51     12150
   weighted avg       0.51      0.51      0.51     12150
```

We observe that using Logestic Regression with Count vector we get an accuracy of 51 percent

Support Vector Machine:

In [ ]:
```
lsv = LinearSVC(C=0.5,random_state=1)
```

In [ ]:
```
lsv.fit(X_train_cv,y_train)
```

Out[ ]:
```
LinearSVC(C=0.5, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=1, tol=0.0001,
          verbose=0)
```

In [ ]:
```
y_pred_lsv = lsv.predict(X_test_cv)
```

In [ ]:
```
print(classification_report(y_test,y_pred_lsv))
```

```
              precision    recall  f1-score   support

           1       0.60      0.60      0.60      2409
           2       0.46      0.44      0.45      2426
           3       0.45      0.42      0.43      2492
           4       0.45      0.45      0.45      2420
           5       0.59      0.64      0.61      2403

    accuracy                           0.51     12150
   macro avg       0.51      0.51      0.51     12150
weighted avg       0.51      0.51      0.51     12150
```

Its observed that using SVM with count vector we are getting an accuracy of 51 percent.

RandomForestClassifier:

In [45]:
```
rfc = RandomForestClassifier(n_estimators=100,random_state=1)
```

In [49]:
```
rfc.fit(X_train_cv,y_train)
```

Out[49]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
```

In [50]:
```
y_pred_rfc = rfc.predict(X_test_cv)
```

In [51]:
```
print(classification_report(y_test,y_pred_rfc))
```

```
                precision    recall  f1-score   support

           1       0.58      0.71      0.64      2409
           2       0.52      0.41      0.46      2426
           3       0.52      0.41      0.46      2492
           4       0.50      0.44      0.47      2420
           5       0.54      0.72      0.62      2403

    accuracy                           0.54     12150
   macro avg       0.53      0.54      0.53     12150
weighted avg       0.53      0.54      0.53     12150
```

Its observed that using RANDOMFORESTCLASSIFIER with Count Vector we are getting an accuracy of 54 percent

TF-IDF Vectorization:

In [12]:
```
tfidf = TfidfVectorizer(stop_words = "english")
```

In [13]:
```
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

LogisticRegression

In [ ]:
```
lr1 = LogisticRegression()
```

In [ ]:
```
lr1.fit(X_train_tfidf,y_train)
```

Out[ ]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [ ]:
```
y_pred1 = lr1.predict(X_test_tfidf)
```

In [ ]:
```
print(classification_report(y_test,y_pred1))
```

```
                precision    recall  f1-score   support

           1       0.60      0.66      0.63      2409
           2       0.46      0.41      0.44      2426
           3       0.45      0.40      0.43      2492
           4       0.48      0.48      0.48      2420
           5       0.62      0.69      0.65      2403

    accuracy                           0.53     12150
   macro avg       0.52      0.53      0.52     12150
weighted avg       0.52      0.53      0.52     12150
```

We observe that using Logestic Regression with TF-IDF vector we get an accuracy of 53 percent

Support Vector Machine:

In [14]:
```
lsv = LinearSVC(C=0.5,random_state=1)
```

In [15]:
```
lsv.fit(X_train_tfidf,y_train)
```

Out[15]:
```
LinearSVC(C=0.5, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=1, tol=0.0001,
          verbose=0)
```

In [16]:
```
y_pred_tf = lsv.predict(X_test_tfidf)
```

In [18]:
```
print(classification_report(y_test,y_pred_tf))
```

```
              precision    recall  f1-score   support

           1       0.60      0.66      0.63      2409
           2       0.47      0.42      0.44      2426
           3       0.46      0.40      0.43      2492
           4       0.48      0.48      0.48      2420
           5       0.61      0.70      0.65      2403

    accuracy                           0.53     12150
   macro avg       0.53      0.53      0.53     12150
weighted avg       0.52      0.53      0.53     12150
```

We observe that using Support Vector Machine with TF-IDF vector we get an accuracy of 53 percent

RandomForestClassifier

In [ ]:
```
rfc = RandomForestClassifier(n_estimators=100,random_state=1)
```

In [ ]:
```
 rfc.fit(X_train_tfidf,y_train)
```

Out[ ]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
```

In [ ]:
```
y_pred = rfc.predict(X_test_tfidf)
```

In [ ]:
```
print(classification_report(y_test,y_pred_tf))
```

```
              precision    recall  f1-score   support

           1       0.60      0.66      0.63      2409
           2       0.47      0.42      0.44      2426
           3       0.46      0.40      0.43      2492
           4       0.48      0.48      0.48      2420
           5       0.61      0.70      0.65      2403

    accuracy                           0.53     12150
   macro avg       0.53      0.53      0.53     12150
weighted avg       0.52      0.53      0.53     12150
```

We observe that using RANDOMFORESTCLASSIFIER with TF-IDF vector we get an accuracy of 53 percent

TOKENIZATION:

In [ ]:
```
tokenizer = Tokenizer(oov_token="<OOV>")
```

```
tokenizer.fit_on_texts(X_train)
train_sequences = tokenizer.texts_to_sequences(X_train)
```

In [ ]:
```
doc_length = []
for doc in train_sequences:
  doc_length.append(len(doc))
max(doc_length)
```

Out[ ]:
```
1761
```

In [ ]:
```
np.quantile(doc_length,0.98)
```

Out[ ]:
```
326.0
```

In [ ]:
```
max_len = 326
train_padded = pad_sequences(train_sequences, maxlen=max_len)
print(train_padded)
```

```
[[   0    0    0 ...   17  545 1502]
 [   0    0    0 ...    4  520  290]
 [   0    0    0 ...   86  394   11]
 ...
 [   0    0    0 ...   85 1948 1179]
 [   0    0    0 ...  771   54   11]
 [   0    0    0 ...  191 3017  288]]
```

Here max doc length is 1761 but as 98 percent of doc have a length of 326 later is considered as max length.

In [ ]:
```
vocab_len = len(tokenizer.index_word)+1
```

FEED FORWARD NEURAL NETWORK:

In [ ]:
```
model = Sequential()
model.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model.add(Flatten())
model.add(Dense(16, activation="relu"))
model.add(Dense(16, activation="relu"))
model.add(Dense(6,activation="softmax"))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 326, 10)           347330
_________________________________________________________________
flatten_1 (Flatten)          (None, 3260)              0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                52176
_________________________________________________________________
dense_4 (Dense)              (None, 16)                272
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 102
=================================================================
Total params: 399,880
Trainable params: 399,880
Non-trainable params: 0
_________________________________________________________________
```

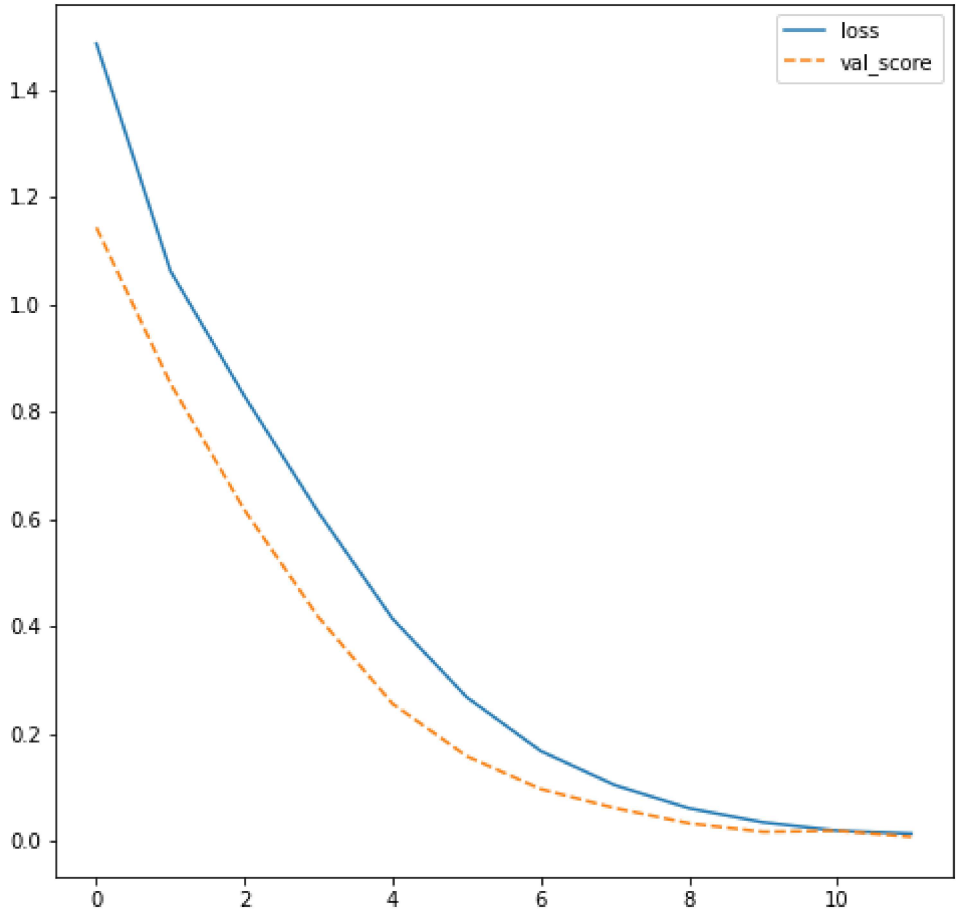
In [ ]:
```
model.compile(loss="sparse_categorical_crossentropy", optimizer = 'adam', metrics =
```

In [ ]:
```
history = model.fit(train_padded,y_train,epochs=12,batch_size=50,validation_data=(tr
```

```
Epoch 1/12
567/567 [==============================] - 6s 10ms/step - loss: 1.4852 - accuracy:
0.3240 - val_loss: 1.1427 - val_accuracy: 0.5347
Epoch 2/12
567/567 [==============================] - 6s 10ms/step - loss: 1.0617 - accuracy:
0.5520 - val_loss: 0.8521 - val_accuracy: 0.6724
Epoch 3/12
567/567 [==============================] - 5s 9ms/step - loss: 0.8282 - accuracy: 0.
6674 - val_loss: 0.6162 - val_accuracy: 0.7853
Epoch 4/12
567/567 [==============================] - 5s 10ms/step - loss: 0.6116 - accuracy:
0.7693 - val_loss: 0.4164 - val_accuracy: 0.8701
Epoch 5/12
567/567 [==============================] - 5s 9ms/step - loss: 0.4136 - accuracy: 0.
8586 - val_loss: 0.2555 - val_accuracy: 0.9322
Epoch 6/12
567/567 [==============================] - 5s 9ms/step - loss: 0.2677 - accuracy: 0.
9167 - val_loss: 0.1580 - val_accuracy: 0.9635
Epoch 7/12
567/567 [==============================] - 6s 10ms/step - loss: 0.1676 - accuracy:
0.9527 - val_loss: 0.0967 - val_accuracy: 0.9808
Epoch 8/12
567/567 [==============================] - 5s 9ms/step - loss: 0.1040 - accuracy: 0.
9728 - val_loss: 0.0613 - val_accuracy: 0.9879
Epoch 9/12
567/567 [==============================] - 5s 9ms/step - loss: 0.0610 - accuracy: 0.
9861 - val_loss: 0.0330 - val_accuracy: 0.9950
Epoch 10/12
567/567 [==============================] - 5s 9ms/step - loss: 0.0348 - accuracy: 0.
9933 - val_loss: 0.0172 - val_accuracy: 0.9980
Epoch 11/12
567/567 [==============================] - 5s 8ms/step - loss: 0.0193 - accuracy: 0.
9975 - val_loss: 0.0193 - val_accuracy: 0.9986
Epoch 12/12
567/567 [==============================] - 5s 9ms/step - loss: 0.0141 - accuracy: 0.
9984 - val_loss: 0.0080 - val_accuracy: 0.9991
```

In [ ]:
```
fig , ax = plt.subplots(figsize=(8,8))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'],label='val_score',linestyle='--')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fa947576510>
```



In [ ]:
```
test_sequences = tokenizer.texts_to_sequences(X_test)
test_padded = pad_sequences(test_sequences, maxlen=max_len)
```

In [ ]:
```
y_pred_ = model.predict(test_padded)
```

In [ ]:
```
y_pred
```

Out[ ]:
```
array([1, 2, 5, ..., 1, 5, 1])
```

In [ ]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       0.60      0.64      0.62      2409
           2       0.47      0.43      0.45      2426
           3       0.46      0.42      0.44      2492
           4       0.47      0.47      0.47      2420
           5       0.61      0.68      0.64      2403

    accuracy                           0.53     12150
   macro avg       0.52      0.53      0.52     12150
weighted avg       0.52      0.53      0.52     12150
```

In [ ]:
```
mat = confusion_matrix(y_test,y_pred)
```

Its observed that using Feed Forward neural network we are getting an accuracy of 53 percent.

RECURRENT NEURAL NETWORK:

In [ ]:
```
model2 = Sequential()
model2.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
```

```
model2.add(SimpleRNN(32, activation="relu",))
model2.add(Dropout(0.5))
model2.add(Dense(16, activation="relu"))
model2.add(Dropout(0.5))
model2.add(Dense(6,activation="sigmoid"))
```

In [ ]:
```
model2.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 326, 10)           347330
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 32)                1376
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_5 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_11 (Dense)             (None, 6)                 102
=================================================================
Total params: 349,336
Trainable params: 349,336
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model2.compile(loss="sparse_categorical_crossentropy", optimizer = 'adam')
```

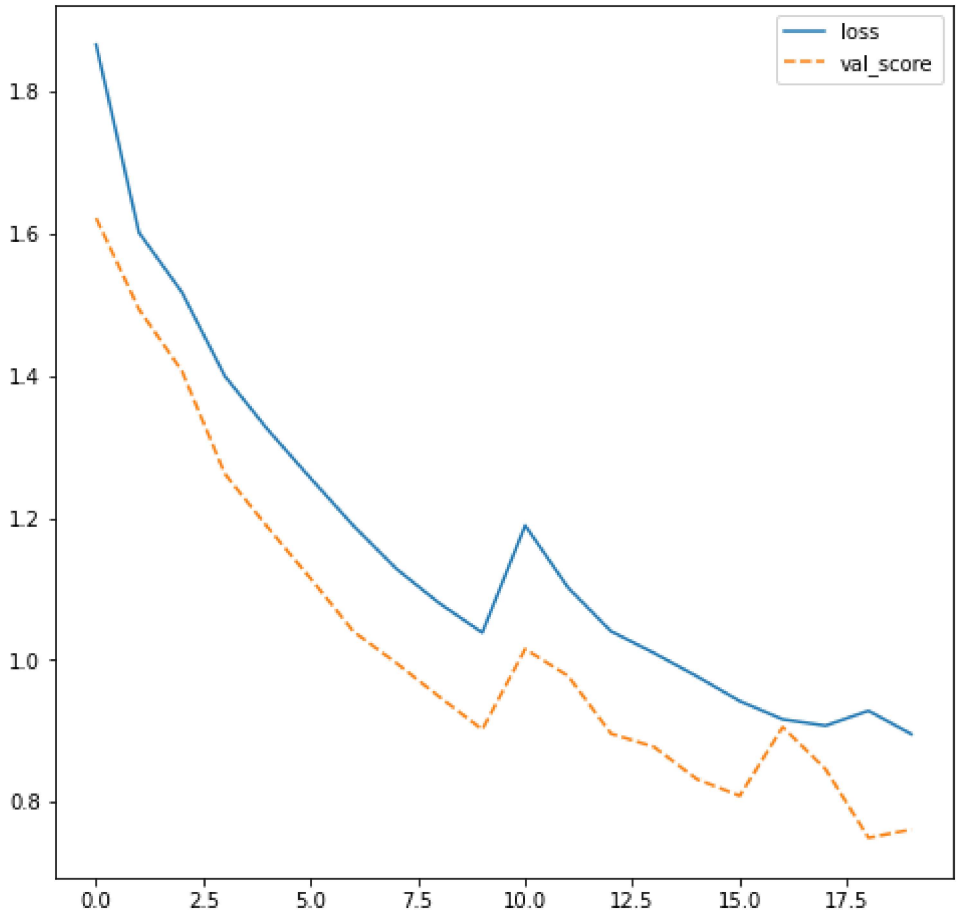
In [ ]:
```
history = model2.fit(train_padded,y_train,epochs=20,batch_size=50,validation_data=(t
```

```
Epoch 1/20
567/567 [==============================] - 66s 116ms/step - loss: 1.8665 - val_loss:
1.6221
Epoch 2/20
567/567 [==============================] - 65s 114ms/step - loss: 1.6015 - val_loss:
1.4935
Epoch 3/20
567/567 [==============================] - 64s 114ms/step - loss: 1.5176 - val_loss:
1.4067
Epoch 4/20
567/567 [==============================] - 65s 114ms/step - loss: 1.3995 - val_loss:
1.2618
Epoch 5/20
567/567 [==============================] - 65s 115ms/step - loss: 1.3241 - val_loss:
1.1864
Epoch 6/20
567/567 [==============================] - 64s 113ms/step - loss: 1.2557 - val_loss:
1.1143
Epoch 7/20
567/567 [==============================] - 64s 113ms/step - loss: 1.1885 - val_loss:
1.0396
Epoch 8/20
567/567 [==============================] - 65s 114ms/step - loss: 1.1280 - val_loss:
0.9953
Epoch 9/20
567/567 [==============================] - 66s 116ms/step - loss: 1.0795 - val_loss:
0.9477
Epoch 10/20
567/567 [==============================] - 64s 114ms/step - loss: 1.0382 - val_loss:
0.9016
Epoch 11/20
567/567 [==============================] - 64s 113ms/step - loss: 1.1890 - val_loss:
1.0151
Epoch 12/20
567/567 [==============================] - 65s 114ms/step - loss: 1.1014 - val_loss:
```

```
0.9771
Epoch 13/20
567/567 [==============================] - 66s 116ms/step - loss: 1.0402 - val_loss:
0.8959
Epoch 14/20
567/567 [==============================] - 64s 114ms/step - loss: 1.0095 - val_loss:
0.8772
Epoch 15/20
567/567 [==============================] - 65s 114ms/step - loss: 0.9767 - val_loss:
0.8319
Epoch 16/20
567/567 [==============================] - 65s 115ms/step - loss: 0.9418 - val_loss:
0.8083
Epoch 17/20
567/567 [==============================] - 65s 115ms/step - loss: 0.9160 - val_loss:
0.9052
Epoch 18/20
567/567 [==============================] - 65s 114ms/step - loss: 0.9071 - val_loss:
0.8459
Epoch 19/20
567/567 [==============================] - 65s 114ms/step - loss: 0.9281 - val_loss:
0.7492
Epoch 20/20
567/567 [==============================] - 65s 114ms/step - loss: 0.8952 - val_loss:
0.7609
```

In [ ]:
```
fig , ax = plt.subplots(figsize=(8,8))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'],label='val_score',linestyle='--')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fa9496b5810>
```



In [ ]:
```
y_pred_2 = model2.predict(test_padded)
```

In [ ]:
```
y_pred_2 = y_pred_2.argmax(axis=1)
```

In [ ]:
```
print(classification_report(y_test, y_pred_2))
```

```
              precision    recall  f1-score   support

           1       0.71      0.38      0.49      2409
           2       0.39      0.44      0.41      2426
           3       0.37      0.44      0.40      2492
           4       0.35      0.40      0.37      2420
           5       0.55      0.57      0.56      2403

    accuracy                           0.44     12150
   macro avg       0.47      0.44      0.45     12150
weighted avg       0.47      0.44      0.45     12150
```

Its observed that using RECURRENT NEURAL NETWORK we are getting an accuracy of 44 percent.

BIDIRECTIONAL RECURRENT NEURAL NETWORK:

In [ ]:
```
model3 = Sequential()
model3.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model3.add(Bidirectional(SimpleRNN(32, activation="relu")))
model3.add(Dropout(0.8))
model3.add(Dense(16, activation="relu"))
model3.add(Dropout(0.8))
model3.add(Dense(6,activation="softmax"))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 326, 10)           347330
_________________________________________________________________
flatten_1 (Flatten)          (None, 3260)              0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                52176
_________________________________________________________________
dense_4 (Dense)              (None, 16)                272
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 102
=================================================================
Total params: 399,880
Trainable params: 399,880
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model3.compile(loss="sparse_categorical_crossentropy", optimizer = 'adam')
```

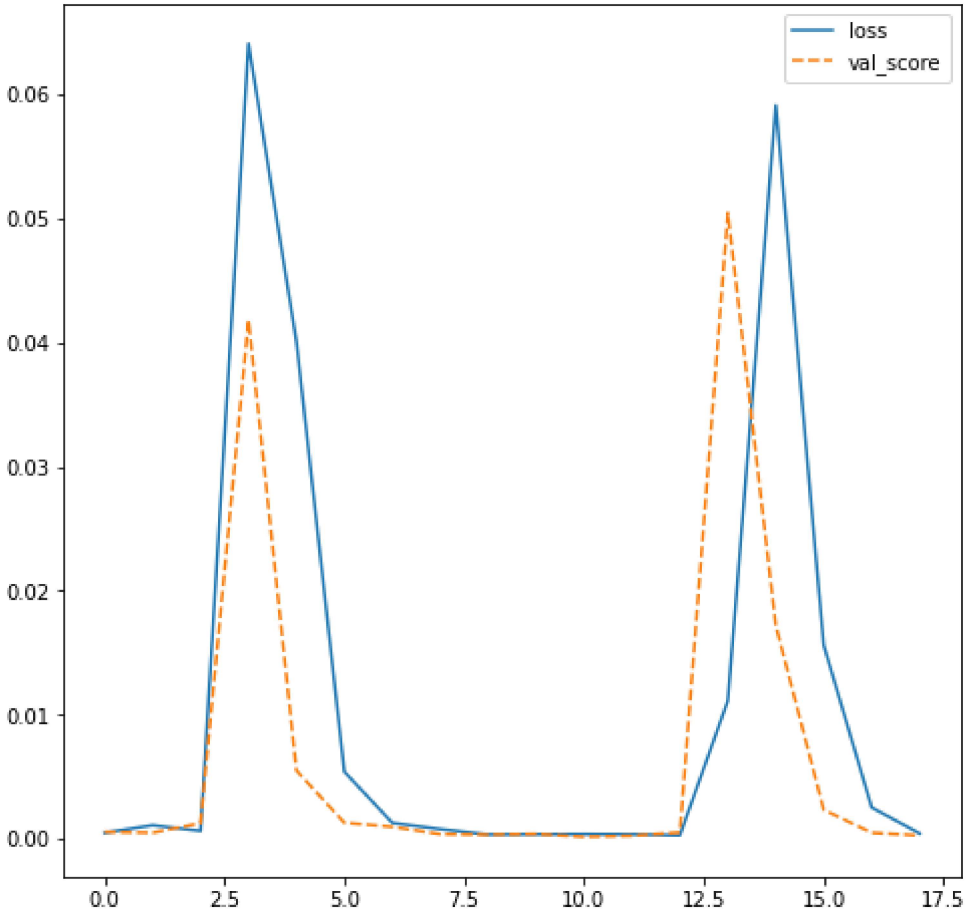
In [ ]:
```
history = model.fit(train_padded,y_train,epochs=18,batch_size=50,validation_data=(tr
```

```
Epoch 1/18
567/567 [==============================] - 5s 9ms/step - loss: 4.9776e-04 - accurac
y: 0.9999 - val_loss: 5.6763e-04 - val_accuracy: 0.9999
Epoch 2/18
567/567 [==============================] - 5s 8ms/step - loss: 0.0011 - accuracy: 0.
9999 - val_loss: 5.0477e-04 - val_accuracy: 0.9999
Epoch 3/18
567/567 [==============================] - 5s 8ms/step - loss: 6.4620e-04 - accurac
y: 0.9999 - val_loss: 0.0013 - val_accuracy: 0.9999
Epoch 4/18
```

```
567/567 [==============================] - 5s 9ms/step - loss: 0.0641 - accuracy: 0.
9795 - val_loss: 0.0418 - val_accuracy: 0.9875
Epoch 5/18
567/567 [==============================] - 5s 9ms/step - loss: 0.0401 - accuracy: 0.
9881 - val_loss: 0.0055 - val_accuracy: 0.9988
Epoch 6/18
567/567 [==============================] - 5s 8ms/step - loss: 0.0054 - accuracy: 0.
9987 - val_loss: 0.0013 - val_accuracy: 0.9999
Epoch 7/18
567/567 [==============================] - 5s 8ms/step - loss: 0.0013 - accuracy: 0.
9999 - val_loss: 9.7541e-04 - val_accuracy: 0.9999
Epoch 8/18
567/567 [==============================] - 5s 8ms/step - loss: 7.9048e-04 - accurac
y: 0.9999 - val_loss: 3.9096e-04 - val_accuracy: 1.0000
Epoch 9/18
567/567 [==============================] - 5s 9ms/step - loss: 3.5531e-04 - accurac
y: 1.0000 - val_loss: 3.4619e-04 - val_accuracy: 0.9999
Epoch 10/18
567/567 [==============================] - 5s 8ms/step - loss: 3.7553e-04 - accurac
y: 0.9999 - val_loss: 4.0386e-04 - val_accuracy: 0.9999
Epoch 11/18
567/567 [==============================] - 5s 9ms/step - loss: 3.8898e-04 - accurac
y: 0.9999 - val_loss: 1.7408e-04 - val_accuracy: 1.0000
Epoch 12/18
567/567 [==============================] - 5s 8ms/step - loss: 3.7015e-04 - accurac
y: 0.9999 - val_loss: 2.6007e-04 - val_accuracy: 1.0000
Epoch 13/18
567/567 [==============================] - 5s 9ms/step - loss: 2.9742e-04 - accurac
y: 1.0000 - val_loss: 5.2484e-04 - val_accuracy: 0.9999
Epoch 14/18
567/567 [==============================] - 5s 9ms/step - loss: 0.0111 - accuracy: 0.
9959 - val_loss: 0.0505 - val_accuracy: 0.9844
Epoch 15/18
567/567 [==============================] - 5s 9ms/step - loss: 0.0591 - accuracy: 0.
9819 - val_loss: 0.0172 - val_accuracy: 0.9958
Epoch 16/18
567/567 [==============================] - 5s 9ms/step - loss: 0.0156 - accuracy: 0.
9956 - val_loss: 0.0023 - val_accuracy: 0.9994
Epoch 17/18
567/567 [==============================] - 5s 9ms/step - loss: 0.0025 - accuracy: 0.
9994 - val_loss: 4.9839e-04 - val_accuracy: 1.0000
Epoch 18/18
567/567 [==============================] - 5s 8ms/step - loss: 4.4054e-04 - accurac
y: 1.0000 - val_loss: 2.8497e-04 - val_accuracy: 0.9999
```

In [ ]:
```
fig , ax = plt.subplots(figsize=(8,8))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'],label='val_score',linestyle='--')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fa945fc1950>
```



In [ ]:
```
y_pred_3 = model.predict(test_padded)
```

In [ ]:
```
y_pred_3 = y_pred_3.argmax(axis=1)
```

In [ ]:
```
print(classification_report(y_test, y_pred_3))
```

```
              precision    recall  f1-score   support

           1       0.59      0.53      0.56      2409
           2       0.45      0.46      0.46      2426
           3       0.44      0.46      0.45      2492
           4       0.43      0.46      0.44      2420
           5       0.56      0.55      0.56      2403

    accuracy                           0.49     12150
   macro avg       0.50      0.49      0.49     12150
weighted avg       0.50      0.49      0.49     12150
```

Its observed that using BIDIRECTIONAL RECURRENT NEURAL NETWORK we are getting an accuracy of 49 percent.

LONG SHORT TERM MEMORY:

In [ ]:
```
model4 = Sequential()
model4.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model4.add(LSTM(18, activation="relu"))
model4.add(Dropout(0.5))
model4.add(Dense(16, activation="relu"))
model4.add(Dropout(0.5))
model4.add(Dense(6,activation="softmax"))
```

In [ ]:
```
model4.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 326, 10)           347330
_________________________________________________________________
lstm (LSTM)                  (None, 32)                5504
_________________________________________________________________
dropout_8 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_14 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_9 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_15 (Dense)             (None, 6)                 102
=================================================================
Total params: 353,464
Trainable params: 353,464
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model4.compile(loss="sparse_categorical_crossentropy", optimizer = 'adam')
```

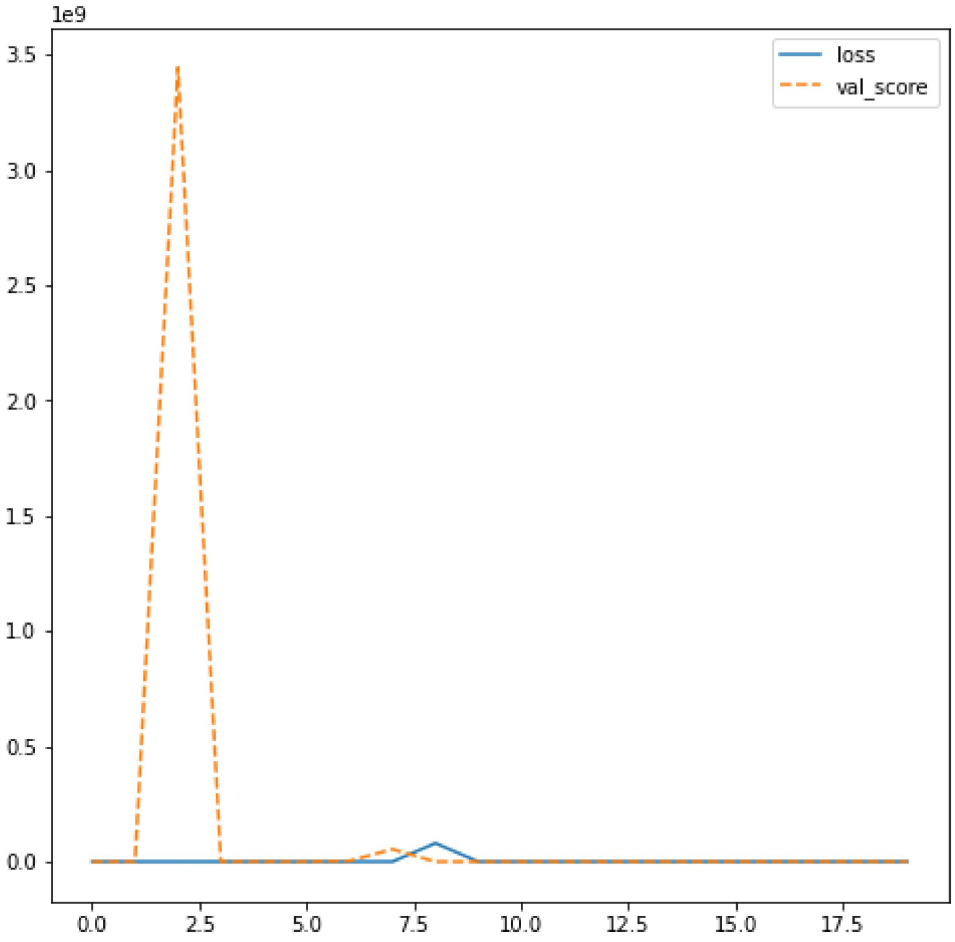
In [ ]:
```
history = model4.fit(train_padded,y_train,epochs=20,batch_size=50, validation_data=(
```

```
Epoch 1/20
567/567 [==============================] - 101s 179ms/step - loss: 19.5328 - val_los
s: 1.1704
Epoch 2/20
567/567 [==============================] - 102s 179ms/step - loss: 1.2586 - val_los
s: 2957.4468
Epoch 3/20
567/567 [==============================] - 105s 184ms/step - loss: 7622.1289 - val_l
oss: 3442531840.0000
Epoch 4/20
567/567 [==============================] - 105s 185ms/step - loss: 85795.3984 - val_
loss: 1.1722
Epoch 5/20
567/567 [==============================] - 101s 179ms/step - loss: 1.2522 - val_los
s: 1.1627
Epoch 6/20
567/567 [==============================] - 101s 179ms/step - loss: 1.2608 - val_los
s: 1.1566
Epoch 7/20
567/567 [==============================] - 101s 179ms/step - loss: 24.5810 - val_los
s: 3164371.2500
Epoch 8/20
567/567 [==============================] - 102s 180ms/step - loss: 10.2743 - val_los
s: 54015552.0000
Epoch 9/20
567/567 [==============================] - 102s 180ms/step - loss: 80073904.0000 - v
al_loss: 1.2786
Epoch 10/20
567/567 [==============================] - 101s 179ms/step - loss: 1.3059 - val_los
s: 1.1970
Epoch 11/20
567/567 [==============================] - 101s 179ms/step - loss: 1.2874 - val_los
s: 1.1850
Epoch 12/20
567/567 [==============================] - 102s 179ms/step - loss: 1.2889 - val_los
s: 1.1801
Epoch 13/20
567/567 [==============================] - 102s 180ms/step - loss: 1.2788 - val_los
s: 1.1736
Epoch 14/20
567/567 [==============================] - 101s 178ms/step - loss: 1.2730 - val_los
s: 1.1704
```

```
Epoch 15/20
567/567 [==============================] - 102s 180ms/step - loss: 1.2668 - val_los
s: 1.1651
Epoch 16/20
567/567 [==============================] - 102s 180ms/step - loss: 1.2664 - val_los
s: 1.1619
Epoch 17/20
567/567 [==============================] - 102s 179ms/step - loss: 1.2563 - val_los
s: 1.1555
Epoch 18/20
567/567 [==============================] - 103s 181ms/step - loss: 1.2566 - val_los
s: 1.1527
Epoch 19/20
567/567 [==============================] - 102s 180ms/step - loss: 1.2486 - val_los
s: 1.1467
Epoch 20/20
567/567 [==============================] - 102s 180ms/step - loss: 1.2515 - val_los
s: 1.1444
```

In [ ]:
```
fig , ax = plt.subplots(figsize=(8,8))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'],label='val_score',linestyle='--')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fa945d59c50>
```



In [ ]:
```
y_pred_4 = model.predict(test_padded)
```

In [ ]:
```
y_pred_4 = y_pred_4.argmax(axis=1)
```

In [ ]:
```
print(classification_report(y_test, y_pred_4))
```

```
              precision    recall  f1-score   support

           1       0.59      0.53      0.56      2409
```

```
           2       0.45      0.46      0.46      2426
           3       0.44      0.46      0.45      2492
           4       0.43      0.46      0.44      2420
           5       0.56      0.55      0.56      2403

    accuracy                           0.49     12150
   macro avg       0.50      0.49      0.49     12150
weighted avg       0.50      0.49      0.49     12150
```

Its observed that using LSTM NEURAL NETWORK we are getting an accuracy of 49 percent.

BIDIRECTIONAL LONG SHORT TERM MEMORY:

In [ ]:
```
model5 = Sequential()
model5.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model5.add(Bidirectional(LSTM(32, activation="relu")))
model5.add(Dropout(0.8))
model5.add(Dense(16, activation="relu"))
model5.add(Dropout(0.8))
model5.add(Dense(6,activation="softmax"))
```

In [ ]:
```
model5.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 326, 10)           347330
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                11008
_________________________________________________________________
dropout_10 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_16 (Dense)             (None, 16)                1040
_________________________________________________________________
dropout_11 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_17 (Dense)             (None, 6)                 102
=================================================================
Total params: 359,480
Trainable params: 359,480
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model5.compile(loss="sparse_categorical_crossentropy", optimizer = 'adam')
```

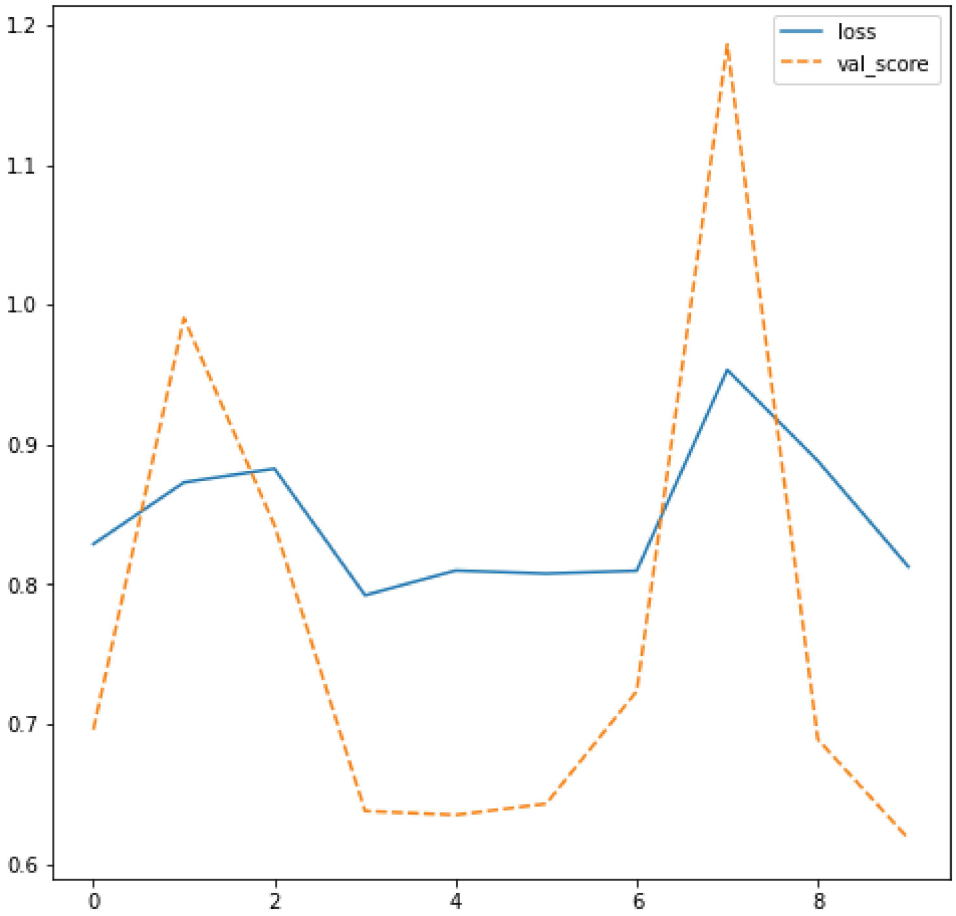
In [ ]:
```
history = model2.fit(train_padded,y_train,epochs=10,batch_size=50, validation_data=(
```

```
Epoch 1/10
567/567 [==============================] - 67s 118ms/step - loss: 0.8288 - val_loss:
0.6961
Epoch 2/10
567/567 [==============================] - 67s 118ms/step - loss: 0.8729 - val_loss:
0.9904
Epoch 3/10
567/567 [==============================] - 66s 117ms/step - loss: 0.8828 - val_loss:
0.8422
Epoch 4/10
567/567 [==============================] - 67s 119ms/step - loss: 0.7922 - val_loss:
0.6379
Epoch 5/10
567/567 [==============================] - 67s 118ms/step - loss: 0.8099 - val_loss:
0.6351
```

```
Epoch 6/10
567/567 [==============================] - 67s 118ms/step - loss: 0.8077 - val_loss:
0.6431
Epoch 7/10
567/567 [==============================] - 67s 118ms/step - loss: 0.8098 - val_loss:
0.7235
Epoch 8/10
567/567 [==============================] - 66s 116ms/step - loss: 0.9534 - val_loss:
1.1860
Epoch 9/10
567/567 [==============================] - 66s 116ms/step - loss: 0.8883 - val_loss:
0.6892
Epoch 10/10
567/567 [==============================] - 66s 117ms/step - loss: 0.8126 - val_loss:
0.6181
```

In [ ]:
```
fig , ax = plt.subplots(figsize=(8,8))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'],label='val_score',linestyle='--')
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x7fa945cfbd90>



In [ ]:
```
y_pred_5 = model.predict(test_padded)
```

In [ ]:
```
y_pred_5 = y_pred_5.argmax(axis=1)
```

In [ ]:
```
print(classification_report(y_test, y_pred_5))
```

```
              precision    recall  f1-score   support

           1       0.59      0.53      0.56      2409
           2       0.45      0.46      0.46      2426
           3       0.44      0.46      0.45      2492
           4       0.43      0.46      0.44      2420
           5       0.56      0.55      0.56      2403
```

```
    accuracy                           0.49     12150
   macro avg       0.50      0.49      0.49     12150
weighted avg       0.50      0.49      0.49     12150
```

Its observed that using BIDIRECTIONAL LSTM NEURAL NETWORK we are getting an accuracy of 49 percent.

GATED RECURRENT UNIT

In [ ]:
```
model6 = Sequential()
model6.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model6.add(Bidirectional(GRU(16, activation="relu")))
model6.add(Dropout(0.5))
model6.add(Dense(16, activation="relu"))
model6.add(Dropout(0.5))
model6.add(Dense(6,activation="softmax"))
```

In [ ]:
```
model6.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 326, 10)           347330
_________________________________________________________________
bidirectional_2 (Bidirection (None, 32)                2688
_________________________________________________________________
dropout_12 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_18 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_13 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_19 (Dense)             (None, 6)                 102
=================================================================
Total params: 350,648
Trainable params: 350,648
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model6.compile(loss="sparse_categorical_crossentropy", optimizer = 'adam')
```

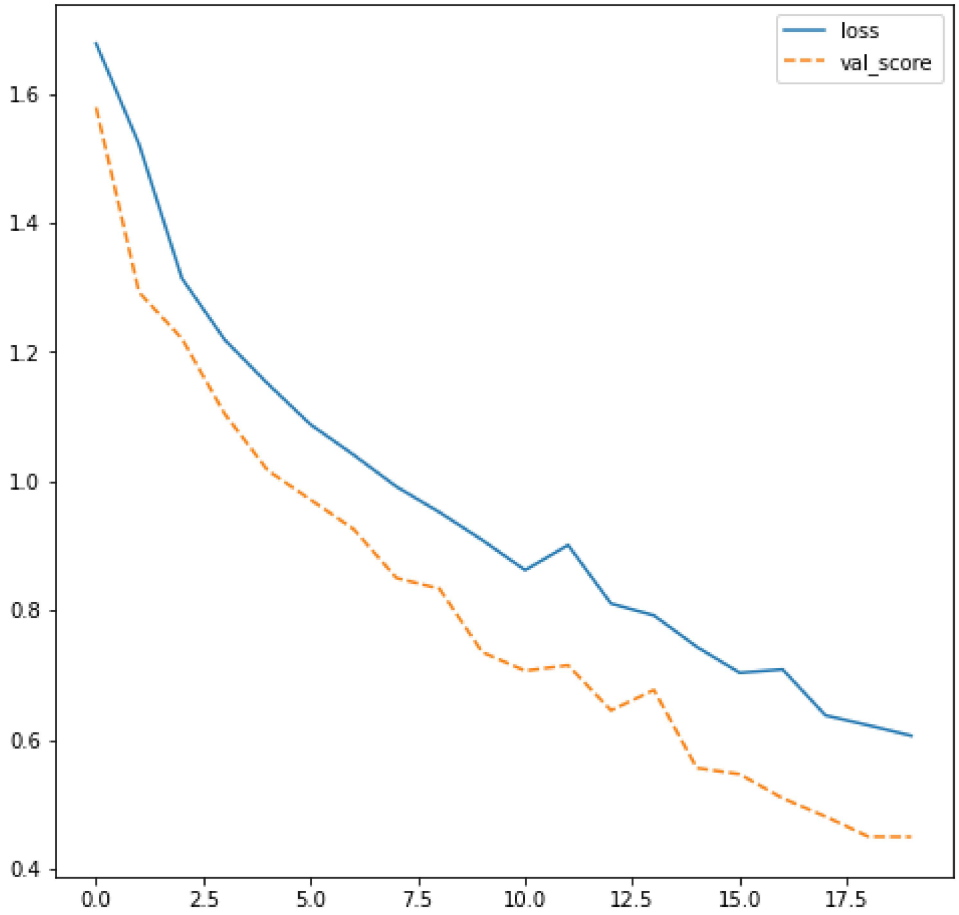
In [ ]:
```
history = model6.fit(train_padded,y_train,epochs=20,batch_size=50,validation_data=(t
```

```
Epoch 1/20
567/567 [==============================] - 138s 241ms/step - loss: 1.6771 - val_los
s: 1.5782
Epoch 2/20
567/567 [==============================] - 136s 241ms/step - loss: 1.5213 - val_los
s: 1.2922
Epoch 3/20
567/567 [==============================] - 137s 242ms/step - loss: 1.3138 - val_los
s: 1.2205
Epoch 4/20
567/567 [==============================] - 136s 239ms/step - loss: 1.2183 - val_los
s: 1.1036
Epoch 5/20
567/567 [==============================] - 137s 241ms/step - loss: 1.1508 - val_los
s: 1.0164
Epoch 6/20
567/567 [==============================] - 136s 240ms/step - loss: 1.0873 - val_los
s: 0.9711
Epoch 7/20
567/567 [==============================] - 136s 240ms/step - loss: 1.0406 - val_los
```

```
s: 0.9256
Epoch 8/20
567/567 [==============================] - 137s 241ms/step - loss: 0.9913 - val_los
s: 0.8500
Epoch 9/20
567/567 [==============================] - 135s 239ms/step - loss: 0.9518 - val_los
s: 0.8334
Epoch 10/20
567/567 [==============================] - 135s 238ms/step - loss: 0.9086 - val_los
s: 0.7351
Epoch 11/20
567/567 [==============================] - 135s 238ms/step - loss: 0.8622 - val_los
s: 0.7064
Epoch 12/20
567/567 [==============================] - 136s 240ms/step - loss: 0.9009 - val_los
s: 0.7149
Epoch 13/20
567/567 [==============================] - 136s 240ms/step - loss: 0.8104 - val_los
s: 0.6450
Epoch 14/20
567/567 [==============================] - 135s 239ms/step - loss: 0.7920 - val_los
s: 0.6765
Epoch 15/20
567/567 [==============================] - 136s 240ms/step - loss: 0.7434 - val_los
s: 0.5558
Epoch 16/20
567/567 [==============================] - 135s 239ms/step - loss: 0.7033 - val_los
s: 0.5463
Epoch 17/20
567/567 [==============================] - 135s 239ms/step - loss: 0.7084 - val_los
s: 0.5089
Epoch 18/20
567/567 [==============================] - 136s 240ms/step - loss: 0.6372 - val_los
s: 0.4808
Epoch 19/20
567/567 [==============================] - 135s 238ms/step - loss: 0.6219 - val_los
s: 0.4497
Epoch 20/20
567/567 [==============================] - 136s 240ms/step - loss: 0.6058 - val_los
s: 0.4494
```

In [ ]:
```
fig , ax = plt.subplots(figsize=(8,8))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'],label='val_score',linestyle='--')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fa945992290>
```



In [ ]:
```
y_pred_6 = model6.predict(test_padded)
```

In [ ]:
```
y_pred_6 = y_pred_6.argmax(axis=1)
```

In [ ]:
```
print(classification_report(y_test, y_pred_6))
```

```
              precision    recall  f1-score   support

           1       0.64      0.56      0.60      2409
           2       0.41      0.44      0.42      2426
           3       0.41      0.53      0.46      2492
           4       0.40      0.48      0.43      2420
           5       0.72      0.39      0.51      2403

    accuracy                           0.48     12150
   macro avg       0.52      0.48      0.48     12150
weighted avg       0.51      0.48      0.48     12150
```

Its observed that using BIDIRECTIONAL LSTM NEURAL NETWORK we are getting an accuracy of 48 percent.

Visualizing the Sentiments:

In [57]:
```
df_pred = pd.DataFrame({'lab':y_pred_rfc})
```

In [58]:
```
df_pred.head()
```

Out[58]:

	lab
0	1

	lab
1	2
2	5
3	5
4	2

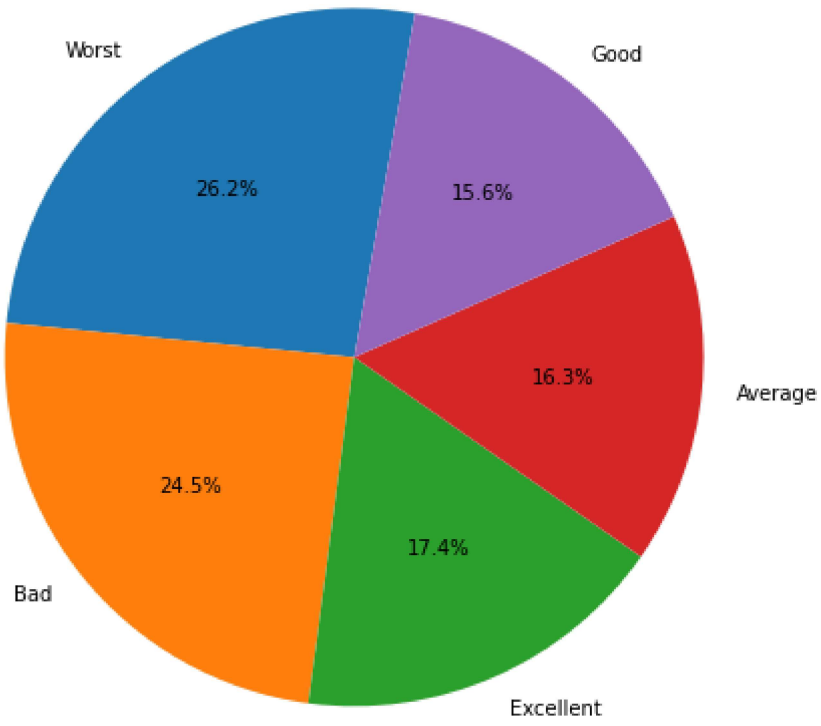
In [59]:
```
df_pred.loc[(df_pred['lab'] == 1, 'sentiment')] ='Worst'
df_pred.loc[(df_pred['lab'] == 2, 'sentiment')] = 'Bad'
df_pred.loc[(df_pred['lab'] == 3, 'sentiment')] = 'Average'
df_pred.loc[(df_pred['lab'] == 4, 'sentiment')] = 'Good'
df_pred.loc[(df_pred['lab'] == 5, 'sentiment')] ='Excellent'
```

In [60]:
```
df_pred.head()
```

Out[60]:

	lab	sentiment
0	1	Worst
1	2	Bad
2	5	Excellent
3	5	Excellent
4	2	Bad

In [61]:
```
plt.figure(figsize=(8,8))
plt.pie(df_pred["sentiment"].value_counts(), labels= df_pred["sentiment"].unique(),
plt.show()
```



CONCLUSION: After building various Machine learning and Deep learning model we observe

that- RNN is giving us an accuracy of 44 percent, BRNN, LSTM and GRU are all giving us an accuracy of 49 percent, while its strange that using Feed Forward Neural Network we are getting an accuracy of 53 percent. Talking about Machine Learning models we have build Logestic Regression, Support Vector Machine and Random Forest using Count and TF-IDF vectoriztion. Based on our models we can conclude that Random Forest with Count Vector is giving us the highest accuracy of 54 percent.