Import Packages

```
import pandas as pd
import numpy as np

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
```

Part 1

The dataset is from https://www.kaggle.com/datasets/kaushiksuresh147/the-social-dilemma-tweets. After preprocessing, the dataset contains text from tweets regarding a Netflix documentary, "The Social Dilemma," and a label indicating whether the tweet should be classified as positive or negative sentiment.

```
google_sheet_id = "1x01AQ4PAsQpusZJDCNxCKwvxH3LWKyXAmhLi0M9e9fk"
sheet_name = "TheSocialDilemma"
google_sheet_url = "https://docs.google.com/spreadsheets/d/{}/gviz/tq?tqx=out:csv&sheet={}".f

df = pd.read_csv(google_sheet_url, header=0)
df = df[df['Sentiment'].isin(['Positive', 'Negative'])]
df = df[['text', 'Sentiment']]
df.head()
```

	text	Sentiment
2	Go watch “The Social Dilemma” on Netflix!\n\nI...	Positive
3	I watched #TheSocialDilemma last night. I’m sc...	Negative
4	The problem of me being on my phone most the t...	Positive
5	#TheSocialDilemma 😳 wow!! We need regulations ...	Positive
7	Erm #TheSocialDilemma makes me want to go off ...	Negative

```
# split df into train and test
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
```

```
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (10447, 2)
test data size:  (2639, 2)
```

```
# set up X and Y
num_labels = 2
vocab_size = 25000
batch_size = 100

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.text)

x_train = tokenizer.texts_to_matrix(train.text, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.text, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.Sentiment)
y_train = encoder.transform(train.Sentiment)
y_test = encoder.transform(test.Sentiment)

# check shape
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:", y_test[:5])
```

```
train shapes: (10447, 25000) (10447,)
test shapes: (2639, 25000) (2639,)
test first five labels: [1 0 1 1 1]
```

Part 2

```
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activation='rel
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=0,
                    validation_split=0.1)
```

```
pred = model.predict(x_test)
```

```
pred_labels = [1 if p>0.5 else 0 for p in pred]

print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
83/83 [==============================] - 0s 2ms/step
accuracy score:  0.8840469874952633
precision score:  0.9103869653767821
recall score:  0.9322210636079249
f1 score:  0.9211746522411128
```

Part 3

Fit RNN model

```
max_features = 10000

model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(1, activation='relu'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)


pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]

print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
83/83 [==============================] - 0s 2ms/step
accuracy score:  0.8901098901098901
precision score:  0.907
```

```
recall score:  0.9457768508863399
f1 score:  0.9259826442062276
```

Fit CNN model

```
model = models.Sequential()
model.add(layers.Embedding(max_features, 128, 32))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)


pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]

print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
83/83 [==============================] - 0s 2ms/step
accuracy score:  0.8863205759757484
precision score:  0.9123343527013251
recall score:  0.9332638164754953
f1 score:  0.9226804123711341
```

Part 4

```
model = models.Sequential()
model.add(layers.Embedding(max_features, 8, input_length=25000))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
```

```
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=0,
                    validation_split=0.1)


pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]

print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
83/83 [==============================] - 0s 3ms/step
accuracy score:  0.726790450928382
precision score:  0.726790450928382
recall score:  1.0
f1 score:  0.8417818740399385
```

Part 5

It appears that our RNN model has the best performance. This was expected, since RNNs are best known for how well they do on text datasets.