Compilation reports

```
data['cleaned_text'] = data['text'].apply(clean_text)

data.head()
```

	text	label	cleaned_text
0	So there is no way for me to plug it in here i...	0	way plug us unless go converter .
1	Good case, Excellent value.	1	good case , excellent value .
2	Great for the jawbone.	1	great jawbone .
3	Tied to charger for conversations lasting more...	0	tied charger conversations lasting 45 minutes....
4	The mic is great.	1	mic great .

```
# Implementar DummyClassifier
dummy_clf = DummyClassifier(strategy='most_frequent')

# Entrenar el modelo
dummy_clf.fit(X_train_vectorized, y_train)
```

```
DummyClassifier(strategy='most_frequent')
```

```
# Hacer predicciones
y_pred = dummy_clf.predict(X_test_vectorized)

# Calcular métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.5091
Precision: 0.5091
Recall: 1.0000
F1 Score: 0.6747
```

```python
y_pred = rnnModel.predict(X_test_padded)
y_pred = (y_pred > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
kappa = cohen_kappa_score(y_test, y_pred)

# Imprimir métricas
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"Cohen's Kappa: {kappa}")
```

```
26/26 [==============================] - 0s 9ms/step
Accuracy: 0.5915151515151515
Precision: 0.6009732360097324
Recall: 0.5880952380952381
F1 Score: 0.5944645006016847
Cohen's Kappa: 0.18308431398475022
```

```python
rnnModel = KerasClassifier(build_fn=create_model,dropout_rate=0.2,lstm_units=50,learning_rate=0.001)

param_grid = {
    'epochs': [3, 5],
    'batch_size': [32, 64],
    'learning_rate': [0.001, 0.01],
    'lstm_units': [50, 100],
    'dropout_rate': [0.2, 0.5]
}

grid = GridSearchCV(estimator=rnnModel, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train_padded, y_train)
print("Mejores parámetros: %s" % grid_result.best_params_)
```

```
/usr/local/lib/python3.10/dist-packages/scikeras/wrappers.py:915: UserWarning: ``build_fn`` will be renamed to ``mod
  X, y = self._initialize(X, y)
Epoch 1/5
61/61 [==============================] - 3s 21ms/step - loss: 0.6817 - accuracy: 0.5809
Epoch 2/5
61/61 [==============================] - 1s 21ms/step - loss: 0.5821 - accuracy: 0.8190
Epoch 3/5
61/61 [==============================] - 1s 21ms/step - loss: 0.4415 - accuracy: 0.8991
Epoch 4/5
61/61 [==============================] - 1s 21ms/step - loss: 0.3081 - accuracy: 0.9319
Epoch 5/5
61/61 [==============================] - 1s 21ms/step - loss: 0.2162 - accuracy: 0.9594
Mejores parámetros: {'batch_size': 32, 'dropout_rate': 0.5, 'epochs': 5, 'learning_rate': 0.001, 'lstm_units': 100}
```

```
modelLSTM.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 128)         640000

 lstm (LSTM)                 (None, 50)                35800

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 675851 (2.58 MB)
Trainable params: 675851 (2.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
y_pred = modelLSTM.predict(X_test_padded)
y_pred = (y_pred > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
kappa = cohen_kappa_score(y_test, y_pred)

# Imprimir métricas
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"Cohen's Kappa: {kappa}")
```

```
26/26 [==============================] - 1s 17ms/step
Accuracy: 0.7963636363636364
Precision: 0.7863636363636364
Recall: 0.8238095238095238
F1 Score: 0.8046511627906977
Cohen's Kappa: 0.5922330097087378
```

```
modelLSTM = KerasClassifier(build_fn=create_model,dropout_rate=0.2,lstm_units=50,learning_rate=0.001)

param_grid = {
    'epochs': [3,5],
    'batch_size': [32, 64],
    'learning_rate': [0.001, 0.01],
    'lstm_units': [50, 100],
    'dropout_rate': [0.2, 0.5]
}

grid = GridSearchCV(estimator=modelLSTM, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train_padded, y_train)
```

```
print("Mejores parámetros: %s" % grid_result.best_params_)
```

```
Mejores parámetros: {'batch_size': 32, 'dropout_rate': 0.2, 'epochs': 5, 'learning_rate': 0.001, 'lstm_units': 50}
```

```
y_pred = bestModelLSTM.predict(X_test_padded)
y_pred = (y_pred > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
kappa = cohen_kappa_score(y_test, y_pred)

# Imprimir métricas
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"Cohen's Kappa: {kappa}")
```

```
26/26 [==============================] - 1s 41ms/step
Accuracy: 0.7975757575757576
Precision: 0.801909307875895
Recall: 0.8
F1 Score: 0.8009535160905841
Cohen's Kappa: 0.595035492262246
```