The dataset I used can be found HERE

In [ ]:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.random.seed(1234)

#input and clean up data
df = pd.read_csv('data.csv')

#creating train-test split
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]

#for some reason test.emails is being treated as floats so now its strings
test.email = test.email.astype(str)

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

#creating graph
pd.DataFrame(df["label"]).hist()
```

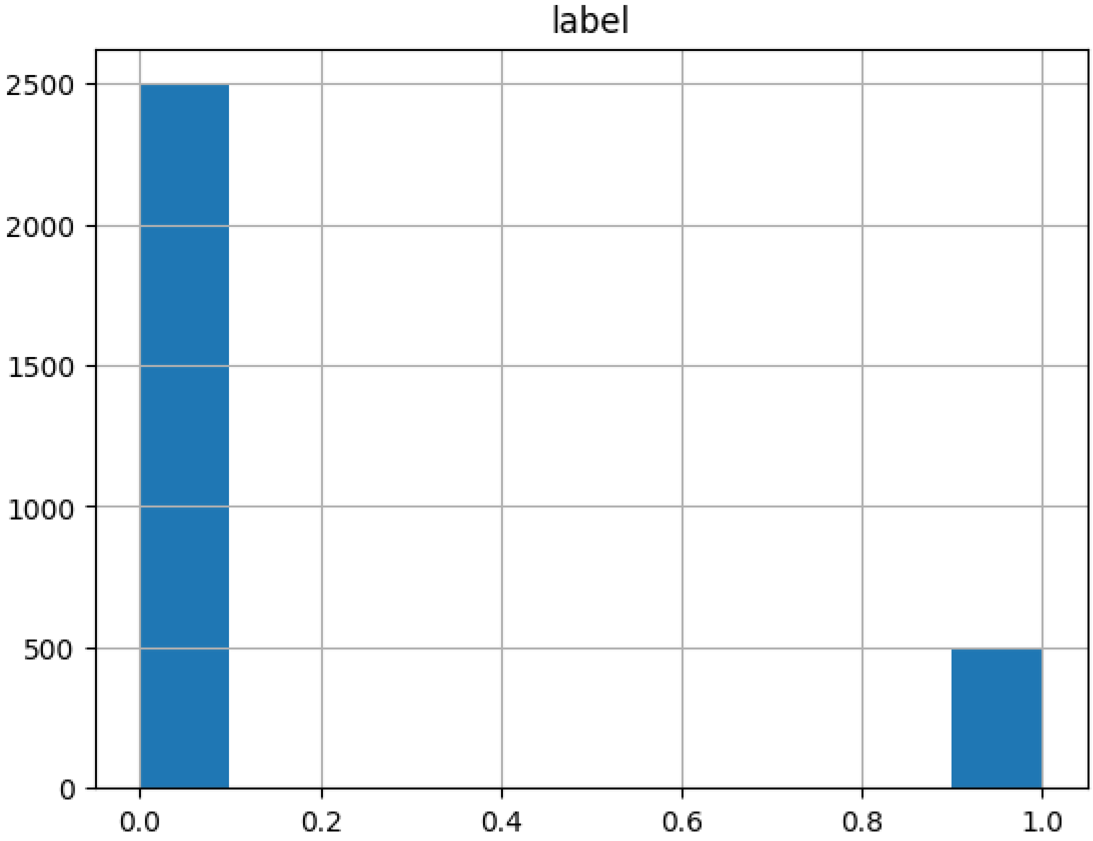
```
train data size:  (2379, 2)
test data size:  (621, 2)
```

```
C:\Users\antho\AppData\Local\Temp\ipykernel_6740\4206280936.py:21: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  test.email = test.email.astype(str)
```

Out[ ]:

```
array([[<AxesSubplot: title={'center': 'label'}>]], dtype=object)
```



Description The dataset delinates spam and non spam in email text

The model should be able to predict whether an email is spam or not

if an Email is spam then output 1 otherwise 0

In [ ]:

```
# 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.email)

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

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

In [ ]:

```
#simple Sequential
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activati
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=1,
                    validation_split=0.1)
```

```
Epoch 1/30
22/22 [==============================] - 1s 16ms/step - loss: 0.3831 - accuracy: 0.90
19 - val_loss: 3.4155 - val_accuracy: 0.0336
Epoch 2/30
22/22 [==============================] - 0s 8ms/step - loss: 0.1598 - accuracy: 0.957
5 - val_loss: 2.8691 - val_accuracy: 0.2185
Epoch 3/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0848 - accuracy: 0.986
9 - val_loss: 3.1195 - val_accuracy: 0.3739
Epoch 4/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0474 - accuracy: 0.996
7 - val_loss: 3.3445 - val_accuracy: 0.4286
Epoch 5/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0277 - accuracy: 0.999
1 - val_loss: 3.4705 - val_accuracy: 0.4874
Epoch 6/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0175 - accuracy: 0.999
5 - val_loss: 3.6668 - val_accuracy: 0.5420
Epoch 7/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0120 - accuracy: 0.999
5 - val_loss: 3.8902 - val_accuracy: 0.5756
Epoch 8/30
22/22 [==============================] - 0s 10ms/step - loss: 0.0090 - accuracy: 0.99
95 - val_loss: 4.0718 - val_accuracy: 0.5798
Epoch 9/30
22/22 [==============================] - 0s 11ms/step - loss: 0.0071 - accuracy: 0.99
95 - val_loss: 4.2361 - val_accuracy: 0.5924
Epoch 10/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0058 - accuracy: 0.999
5 - val_loss: 4.3929 - val_accuracy: 0.5924
Epoch 11/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0049 - accuracy: 0.999
5 - val_loss: 4.5250 - val_accuracy: 0.5924
Epoch 12/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0042 - accuracy: 0.999
5 - val_loss: 4.6421 - val_accuracy: 0.5924
Epoch 13/30
22/22 [==============================] - 0s 10ms/step - loss: 0.0037 - accuracy: 0.99
95 - val_loss: 4.7404 - val_accuracy: 0.5924
Epoch 14/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0032 - accuracy: 0.999
5 - val_loss: 4.8411 - val_accuracy: 0.5966
Epoch 15/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0029 - accuracy: 0.999
5 - val_loss: 4.9374 - val_accuracy: 0.6008
Epoch 16/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0026 - accuracy: 0.999
5 - val_loss: 5.0210 - val_accuracy: 0.6008
Epoch 17/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0024 - accuracy: 0.999
5 - val_loss: 5.0874 - val_accuracy: 0.6050
Epoch 18/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0022 - accuracy: 0.999
5 - val_loss: 5.1620 - val_accuracy: 0.6050
Epoch 19/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0020 - accuracy: 0.999
5 - val_loss: 5.2329 - val_accuracy: 0.6050
```

```
Epoch 20/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0019 - accuracy: 0.999
5 - val_loss: 5.2925 - val_accuracy: 0.6050
Epoch 21/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0018 - accuracy: 0.999
5 - val_loss: 5.3549 - val_accuracy: 0.6050
Epoch 22/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0017 - accuracy: 0.999
5 - val_loss: 5.4083 - val_accuracy: 0.6050
Epoch 23/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0016 - accuracy: 0.999
5 - val_loss: 5.4734 - val_accuracy: 0.6050
Epoch 24/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0015 - accuracy: 0.999
5 - val_loss: 5.5152 - val_accuracy: 0.6050
Epoch 25/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0014 - accuracy: 0.999
5 - val_loss: 5.5546 - val_accuracy: 0.6092
Epoch 26/30
22/22 [==============================] - 0s 8ms/step - loss: 0.0013 - accuracy: 0.999
5 - val_loss: 5.6060 - val_accuracy: 0.6176
Epoch 27/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0013 - accuracy: 0.999
5 - val_loss: 5.6457 - val_accuracy: 0.6176
Epoch 28/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0012 - accuracy: 0.999
5 - val_loss: 5.6901 - val_accuracy: 0.6176
Epoch 29/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0012 - accuracy: 0.999
5 - val_loss: 5.7332 - val_accuracy: 0.6218
Epoch 30/30
22/22 [==============================] - 0s 9ms/step - loss: 0.0011 - accuracy: 0.999
5 - val_loss: 5.7733 - val_accuracy: 0.6218
```

In [ ]:

```
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
print(score)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.4542 - accuracy: 0.9517
Accuracy:  0.9516907930374146
[0.45421674847602844, 0.9516907930374146]
```

In [ ]:

```
#attempting RNN
max_features = 10000
maxlen = 500
batch_size = 32

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

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

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
15/15 [==============================] - 632s 42s/step - loss: 0.2912 - accuracy: 0.9
327 - val_loss: 2.1462 - val_accuracy: 0.1744
Epoch 2/10
 3/15 [=====>........................] - ETA: 8:24 - loss: 0.0730 - accuracy: 1.0000
```

```
In [ ]: print(model.summary())
        score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
        print('Accuracy: ', score[1])
        print(score)
```

```
In [ ]: model.compile(optimizer='rmsprop',
                      loss='binary_crossentropy',
                      metrics=['accuracy'])

        history = model.fit(x_train, y_train,
                            epochs=10,
                            batch_size=128,
                            validation_split=0.2)
```

```
In [ ]: print(model.summary())
        score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
        print('Accuracy: ', score[1])
        print(score)
```

Analysis:
Overall the RNN took wayyy too long to run for even for the accuracy improvments, with my relatively small dataset the sequential seemed to have very good accuracy relative to the amount of time that it took to run. Embeddings seem to increase the accuracy a bit, however, again the sequential run got most of the way there for a fraction of the runtime.