Generate print hello world using rot13 Close

```
import os
import pandas as pd
import kagglehub
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

## download and read the data set

```
path = kagglehub.dataset_download("jp797498e/twitter-entity-sentiment-analysis")
csv_file = os.path.join(path, "twitter_training.csv")
df = pd.read_csv(csv_file, delimiter=",", names=["ID", "game", "sentiment", "text"], encoding="utf-8")
```

## Convert Sentiment labels to numbers and keep only Sentiment & Text

```
df = df[["sentiment", "text"]].dropna()
label_mapping = {"Positive": 2, "Neutral": 1, "Negative": 0}
df["sentiment"] = df["sentiment"].map(label_mapping).fillna(1).astype(int)
```

## Tokenization & Padding

Generate 10 random numbers using numpy Close

```
VOCAB_SIZE = 5000
MAX_LENGTH = 50

tokenizer = Tokenizer(num_words=VOCAB_SIZE, oov_token="<OOV>")
tokenizer.fit_on_texts(df["text"])
sequences = tokenizer.texts_to_sequences(df["text"])
padded_sequences = pad_sequences(sequences, maxlen=MAX_LENGTH, padding="post")
```

## Split Data into Train & Test

```
y = np.array(df["sentiment"])
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, y, test_size=0.2, random_state=42)
```

## Model Definition and Train

```
model = Sequential([
    Embedding(input_dim=VOCAB_SIZE, output_dim=32, input_length=MAX_LENGTH),
    LSTM(64, activation="tanh", return_sequences=False),
    Dense(3, activation="softmax")
])

model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model.summary()
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. J
  warnings.warn(
Model: "sequential_11"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding_11 (Embedding) | ? | 0 (unbuilt) |
| lstm_3 (LSTM) | ? | 0 (unbuilt) |
| dense_11 (Dense) | ? | 0 (unbuilt) |

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
1480/1480 ━━━━━━━━━━━━━━━━━━━━ 48s 31ms/step - accuracy: 0.4112 - loss: 1.0802 - val_accuracy: 0.4148 - val_loss: 1.0730
Epoch 2/5
1480/1480 ━━━━━━━━━━━━━━━━━━━━ 48s 32ms/step - accuracy: 0.4177 - loss: 1.0785 - val_accuracy: 0.4139 - val_loss: 1.0854
Epoch 3/5
1480/1480 ━━━━━━━━━━━━━━━━━━━━ 79s 30ms/step - accuracy: 0.4805 - loss: 1.0053 - val_accuracy: 0.6607 - val_loss: 0.7596
Epoch 4/5
1480/1480 ━━━━━━━━━━━━━━━━━━━━ 85s 33ms/step - accuracy: 0.7211 - loss: 0.6626 - val_accuracy: 0.7331 - val_loss: 0.6395
Epoch 5/5
1480/1480 ━━━━━━━━━━━━━━━━━━━━ 79s 31ms/step - accuracy: 0.7988 - loss: 0.5084 - val_accuracy: 0.7528 - val_loss: 0.6066
```

## Evaluate Model on Test Data

```
y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)
```

```
463/463 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step
```

## Calculate Accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Test Accuracy: 74.88%
```