```
from google.colab import files
uploaded = files.upload()
```

Choose Files Dataset.csv

Dataset.csv(text/csv) - 1911016 bytes, last modified: 2/10/2026 - 100% done

```
Saving Dataset.csv to Dataset.csv
```

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('Dataset.csv',encoding='utf-8')
df
```

	Text	Language
0	Nature, in the broadest sense, is the natural...	English
1	"Nature" can refer to the phenomena of the phy...	English
2	The study of nature is a large, if not the onl...	English
3	Although humans are part of nature, human acti...	English
4	[1] The word nature is borrowed from the Old F...	English
...	...	...
10332	ನಿಮ್ಮ ತಪ್ಪು ಏನು ಬಂದಿದೆಯೆಂದರೆ ಆ ದಿನದಿಂದ ನಿಮಗೆ ಒ...	Kannada
10333	ನಾರ್ಸಿಸಾ ತಾನು ಮೊದಲಿಗೆ ಹೆಣಗಾಡುತ್ತಿದ್ದ ಮಾರ್ಗಗಳನ್...	Kannada
10334	ಹೇಗೆ ' ನಾರ್ಸಿಸಿಸಮ್ ಈಗ ಮರಿಯನ್ ಅವರಿಗೆ ಸಂಭವಿಸಿದ ಎ...	Kannada
10335	ಅವಳು ಈಗ ಹೆಚ್ಚು ಚಿನ್ನದ ಬೈಡ್ ಬಯಸುವುದಿಲ್ಲ ಎಂದು ...	Kannada
10336	ಟೆರ್ರಿ ನೀವು ನಿಜವಾಗಿಯೂ ಆ ದೇವದೂತನಂತೆ ಸ್ವಲ್ಪ ಕಾಣು...	Kannada

10337 rows × 2 columns

Next steps: Generate code with df | New interactive sheet

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Bidirectional, Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv("Dataset.csv")

texts = df['Text'].astype(str).values
labels = df['Language'].astype('category').cat.codes.values
```

```
df['Language'].unique()
```

```
array(['English', 'Malayalam', 'Hindi', 'Tamil', 'Portugeese', 'French',
       'Dutch', 'Spanish', 'Greek', 'Russian', 'Danish', 'Italian',
       'Turkish', 'Sweedish', 'Arabic', 'German', 'Kannada'], dtype=object)
```

```
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(texts)
seq = tokenizer.texts_to_sequences(texts)
padded_seq = pad_sequences(seq, maxlen=150)
```

```
X_train, X_test, y_train, y_test = train_test_split(
    padded_seq, labels, test_size=0.2, random_state=42
)
```

```
input_layer = Input(shape=(150,))   # increased length

embed = Embedding(
    input_dim=len(tokenizer.word_index)+1,
    output_dim=128
)(input_layer)

x = Bidirectional(LSTM(128, return_sequences=True))(embed)
x = Dropout(0.3)(x)

x = Bidirectional(LSTM(64))(x)
x = Dropout(0.3)(x)

x = Dense(64, activation='relu')(x)

output = Dense(len(np.unique(labels)), activation='softmax')(x)

model = Model(inputs=input_layer, outputs=output)
```

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

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)
```

```
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=32,
    validation_split=0.1,
    callbacks=[early_stop]
)
```

```
Epoch 1/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 184s 756ms/step - accuracy: 0.2440 - loss: 2.1936 - val_accuracy: 0.5091 -
Epoch 2/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 176s 757ms/step - accuracy: 0.5671 - loss: 1.1124 - val_accuracy: 0.5877 -
Epoch 3/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 168s 722ms/step - accuracy: 0.6647 - loss: 0.8615 - val_accuracy: 0.6542 -
Epoch 4/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 201s 716ms/step - accuracy: 0.7326 - loss: 0.6811 - val_accuracy: 0.7570 -
Epoch 5/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 209s 744ms/step - accuracy: 0.7914 - loss: 0.5617 - val_accuracy: 0.7400 -
Epoch 6/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 172s 740ms/step - accuracy: 0.8090 - loss: 0.5094 - val_accuracy: 0.8235 -
Epoch 7/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 177s 759ms/step - accuracy: 0.8479 - loss: 0.4220 - val_accuracy: 0.8537 -
Epoch 8/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 172s 737ms/step - accuracy: 0.8776 - loss: 0.3570 - val_accuracy: 0.8863 -
```

```
Epoch 9/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 172s 740ms/step - accuracy: 0.8914 - loss: 0.3071 - val_accuracy: 0.9081 -
Epoch 10/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 172s 739ms/step - accuracy: 0.9109 - loss: 0.2550 - val_accuracy: 0.9178 -
Epoch 11/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 171s 734ms/step - accuracy: 0.9306 - loss: 0.2111 - val_accuracy: 0.9238 -
Epoch 12/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 170s 729ms/step - accuracy: 0.9308 - loss: 0.2086 - val_accuracy: 0.9299 -
Epoch 13/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 170s 727ms/step - accuracy: 0.9462 - loss: 0.1584 - val_accuracy: 0.9093 -
Epoch 14/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 175s 754ms/step - accuracy: 0.9362 - loss: 0.1921 - val_accuracy: 0.9420 -
Epoch 15/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 170s 730ms/step - accuracy: 0.9495 - loss: 0.1471 - val_accuracy: 0.9383 -
Epoch 16/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 170s 731ms/step - accuracy: 0.9535 - loss: 0.1376 - val_accuracy: 0.9335 -
Epoch 17/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 175s 753ms/step - accuracy: 0.9580 - loss: 0.1212 - val_accuracy: 0.9432 -
Epoch 18/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 175s 752ms/step - accuracy: 0.9617 - loss: 0.1128 - val_accuracy: 0.9553 -
Epoch 19/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 170s 732ms/step - accuracy: 0.9664 - loss: 0.0905 - val_accuracy: 0.9504 -
Epoch 20/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 170s 730ms/step - accuracy: 0.9720 - loss: 0.0762 - val_accuracy: 0.9432 -
```

```
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)
```

```
Epoch 1/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 170s 731ms/step - accuracy: 0.9645 - loss: 0.0995 - val_accuracy: 0.9528 -
Epoch 2/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 170s 728ms/step - accuracy: 0.9713 - loss: 0.0835 - val_accuracy: 0.9347 -
Epoch 3/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 169s 727ms/step - accuracy: 0.9602 - loss: 0.1215 - val_accuracy: 0.9625 -
Epoch 4/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 171s 732ms/step - accuracy: 0.9686 - loss: 0.0860 - val_accuracy: 0.9613 -
Epoch 5/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 203s 738ms/step - accuracy: 0.9732 - loss: 0.0790 - val_accuracy: 0.9553 -
Epoch 6/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 172s 737ms/step - accuracy: 0.9796 - loss: 0.0567 - val_accuracy: 0.9637 -
Epoch 7/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 171s 733ms/step - accuracy: 0.9808 - loss: 0.0529 - val_accuracy: 0.9613 -
Epoch 8/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 173s 739ms/step - accuracy: 0.9867 - loss: 0.0365 - val_accuracy: 0.9649 -
Epoch 9/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 171s 735ms/step - accuracy: 0.9878 - loss: 0.0348 - val_accuracy: 0.9637 -
Epoch 10/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 172s 740ms/step - accuracy: 0.9846 - loss: 0.0505 - val_accuracy: 0.9480 -
<keras.src.callbacks.history.History at 0x7832a846f500>
```

```
def predict_language(text):
    seq = tokenizer.texts_to_sequences([text])
    pad = pad_sequences(seq, maxlen=150)
    pred = model.predict(pad)
    lang_index = np.argmax(pred)
    return df['Language'].astype('category').cat.categories[lang_index]

print(predict_language("Hola amigo"))
```

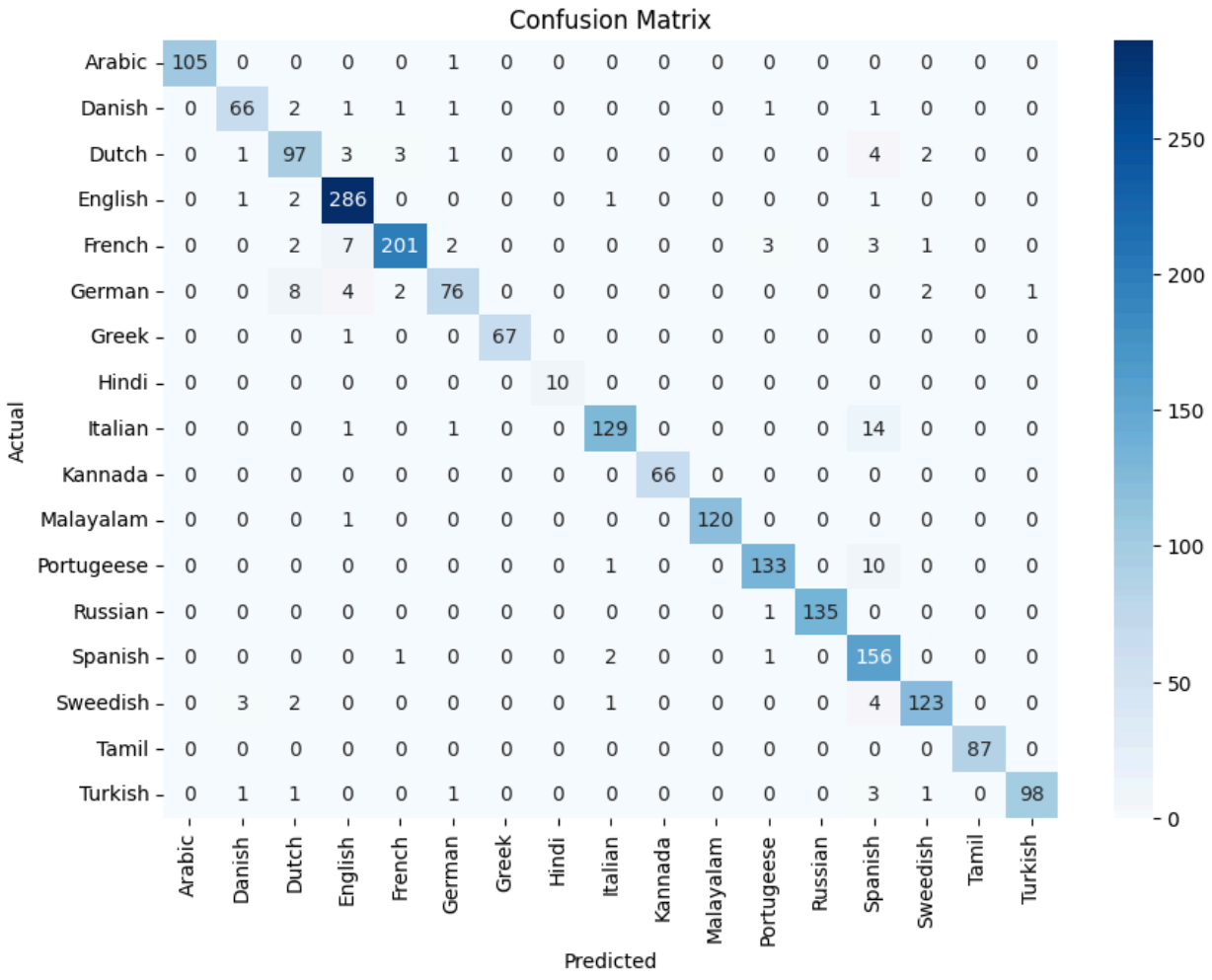
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 676ms/step
Spanish
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

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

# Accuracy
acc = accuracy_score(y_test, y_pred)
```

```
# Precision, Recall, F1 (macro because multi-class)
prec = precision_score(y_test, y_pred, average='macro')
rec = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print("Accuracy :", acc)
print("Precision:", prec)
print("Recall   :", rec)
print("F1 Score :", f1)
```

```
65/65 ━━━━━━━━━━━━━━━━━━━━ 13s 196ms/step
Accuracy : 0.945357833655706
Precision: 0.9558334686270075
Recall   : 0.9472040147596514
F1 Score : 0.9505761412196032
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=df['Language'].astype('category').cat.categories,
            yticklabels=df['Language'].astype('category').cat.categories)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



```
model.save("language_identification_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mo
```

```
import pickle

with open("tokenizer.pkl", "wb") as f:
    pickle.dump(tokenizer, f)
```

```
labels_list = list(df['Language'].astype('category').cat.categories)

with open("labels.pkl", "wb") as f:
    pickle.dump(labels_list, f)
```

```
from tensorflow.keras.models import load_model
import pickle

model = load_model("language_identification_model.h5")

with open("tokenizer.pkl", "rb") as f:
    tokenizer = pickle.load(f)

with open("labels.pkl", "rb") as f:
    labels_list = pickle.load(f)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_me
```

```
from google.colab import files

files.download("language_identification_model.h5")
files.download("tokenizer.pkl")
files.download("labels.pkl")
```

```

```

```
print(predict_language("नमस्ते दोस्त"))

print(predict_language("எப்படி இருக்கிறீர்கள்"))
print(predict_language("Hello World"))
print(predict_language("olá mundo"))
print(predict_language("Привет, мир"))
```

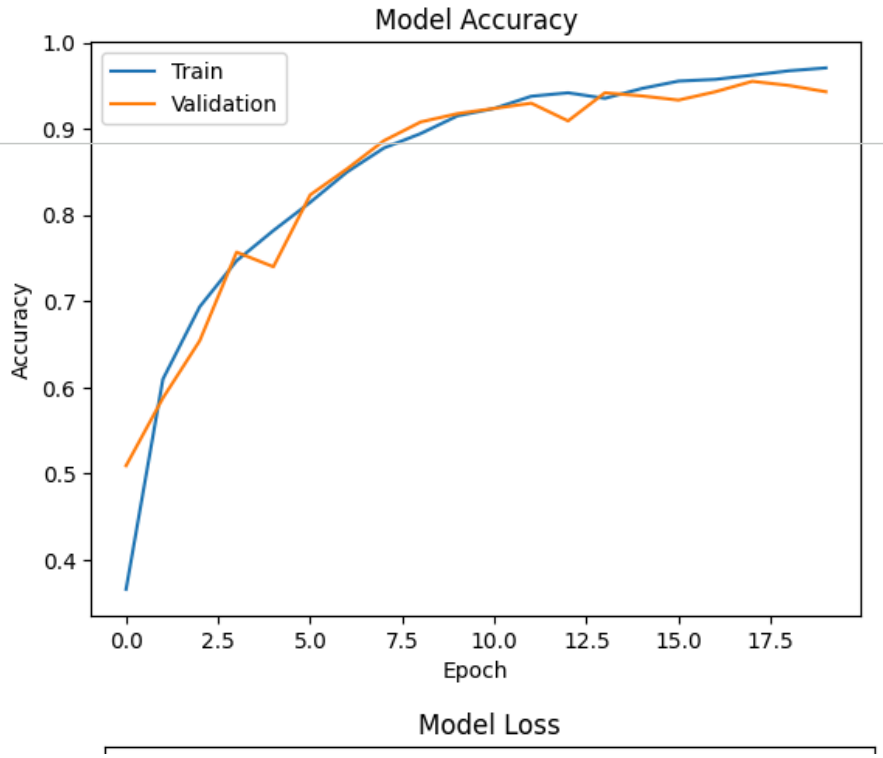
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 631ms/step
Malayalam
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
Tamil
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
English
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step
Spanish
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
Russian
```

```
def predict_lang(new_text):
    seq = tokenizer.texts_to_sequences([new_text])
    pad = pad_sequences(seq, maxlen=100)
    pred = model.predict(pad)
    idx = np.argmax(pred)
    return labels_list[idx]
```

```
print(predict_lang("Bonjour"))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
Spanish
```

```
 import matplotlib.pyplot as plt

# Accuracy Graph
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
plt.show()

# Loss Graph
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()
```


Model Accuracy
Train
Validation
Accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Model Loss