```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use("ggplot")

import re
import nltk

from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer, CountVectorizer
from nltk import sent_tokenize, word_tokenize
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, make_scorer

from time import time
import pickle
```

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

```
df.head()
```

	headline	label
0	cock suck before you piss around on my work	-1
1	you are gay or antisemmitian archangel white ...	-1
2	fuck your filthy mother in the ass dry	-1
3	get fuck ed up get fuck ed up got a drink t...	-1
4	stupid peace of shit stop deleting my stuff ...	-1

Next steps: Generate code with df | New interactive sheet

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

```
array([-1,  0])
```

```
def perform_data_manipulation():
    df = pd.read_csv("data.csv")

    for index in df.index:
        if df.loc[index, "label"]==-1:
            df.loc[index, "label"] = 1
    return df
```

```
df = perform_data_manipulation()
```

```
df.head()
```

	headline	label
0	cock suck before you piss around on my work	1
1	you are gay or antisemmitian archangel white ...	1
2	fuck your filthy mother in the ass dry	1
3	get fuck ed up get fuck ed up got a drink t...	1
4	stupid peace of shit stop deleting my stuff ...	1

Next steps: Generate code with df | New interactive sheet

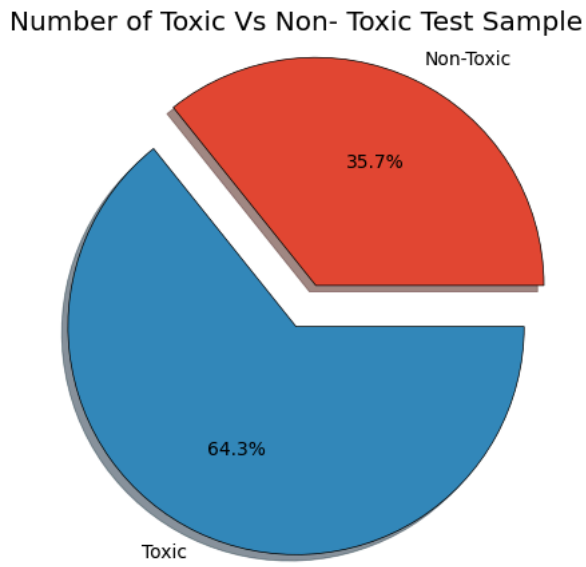
```
df['label'].unique()
```

```
array([1, 0])
```

```
def performdatadistribution(df):
    total = df.shape[0]
    num_non_toxic = df[df['label']==0].shape[0]

    slices = [num_non_toxic/total, (total-num_non_toxic)/total]

    labeling = ['Non-Toxic', 'Toxic']
```

```
    explode = [0.2, 0]

    plt.pie(slices, explode = explode, shadow=True, autopct="%1.1f%%", labels = labeling, wedgeprops={'edgecolor': 'black'}

    plt.title('Number of Toxic Vs Non- Toxic Test Sample')

    plt.tight_layout()

    plt.show()
```

```
performdatadistribution(df)
```




```
def remove_pattern(input_txt, pattern):
    if (type(input_txt)==str):
        r = re.findall(pattern, input_txt)
        for i in r:
            input_txt = re.sub(i, '', input_txt)
        return input_txt
    else:
        return ""
```

```
df.head(1)
```

	headline	label
0	cock suck before you piss around on my work	1

Next steps: Generate code with df | New interactive sheet

```
def datasetCleaning(df):
    df['length_headline'] = df['headline'].str.len()
    combined_df = df.append(df, ignore_index=True)
#     remove @ user
    combined_df['tidy_tweet'] = np.vectorize(remove_pattern)(combined_df['headline'], "@[\w]*")

#     remove extra letters
    combined_df['tidy_tweet'] = combined_df['tidy_tweet'].str.replace("[^a-zA-Z#]", " ")

#     remove all those words with size less than 3
    combined_df['tidy_tweet'] = combined_df['tidy_tweet'].apply(lambda x : ' '.join([w for w in x.split() if len(w)>3]))

    combined_df['length_tidy_tweet'] = combined_df['tidy_tweet'].str.len()

#     Tokenized
    tokenized_tweet = combined_df['tidy_tweet'].apply(lambda x : x.split())

    nltk.download('wordnet')

    lemmatizer = nltk.stem.WordNetLemmatizer()

    tokenized_tweet = tokenized_tweet.apply(lambda x :[lemmatizer.lemmatize(i) for i in x])

    for i in range(len(tokenized_tweet)):
```

```
        tokenized_tweet[i] = ' '.join(tokenized_tweet[i])
    combined_df['tidy_tweet'] = tokenized_tweet

    return combined_df, df
```

```
<>:5: SyntaxWarning: invalid escape sequence '\w'
<>:5: SyntaxWarning: invalid escape sequence '\w'
/tmp/ipython-input-3153128377.py:5: SyntaxWarning: invalid escape sequence '\w'
  combined_df['tidy_tweet'] = np.vectorize(remove_pattern)(combined_df['headline'], "@[\w]*")
```

```
def datasetCleaning(df):
    df['length_headline'] = df['headline'].str.len()
    combined_df = pd.concat([df, df], ignore_index=True)
#     remove @ user
    combined_df['tidy_tweet'] = np.vectorize(remove_pattern)(combined_df['headline'], "@[\\w]*")

#     remove extra letters
    combined_df['tidy_tweet'] = combined_df['tidy_tweet'].str.replace("[^a-zA-Z#]", " ")

#     remove all those words with size less than 3
    combined_df['tidy_tweet'] = combined_df['tidy_tweet'].apply(lambda x : ' '.join([w for w in x.split() if len(w)>3]))

    combined_df['length_tidy_tweet'] = combined_df['tidy_tweet'].str.len()

#     Tokenized
    tokenized_tweet = combined_df['tidy_tweet'].apply(lambda x : x.split())

    nltk.download('wordnet')

    lemmatizer = nltk.stem.WordNetLemmatizer()

    tokenized_tweet = tokenized_tweet.apply(lambda x :[lemmatizer.lemmatize(i) for i in x])

    for i in range(len(tokenized_tweet)):
        tokenized_tweet[i] = ' '.join(tokenized_tweet[i])
    combined_df['tidy_tweet'] = tokenized_tweet

    return combined_df, df

combined_df, df =datasetCleaning(df)
combined_df.head()
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

	headline	label	length_headline	tidy_tweet	length_tidy_tweet
0	cock suck before you piss around on my work	1	44	cock suck before piss around work	33
1	you are gay or antisemmitian archangel white ...	1	624	antisemmitian archangel white tiger meow greet...	400
2	fuck your filthy mother in the ass dry	1	39	fuck your filthy mother	23
3	get fuck ed up get fuck ed up got a drink	1	121	fuck fuck drink that cant down fuck fuck	51

Next steps: Generate code with combined_df | New interactive sheet

```
from sklearn.model_selection import train_test_split
```

```
def performdatasplit(x, y, combined_df, df):
    X_train, X_test, y_train, y_test = train_test_split(combined_df['tidy_tweet'], combined_df['label'], test_size = x, ran
    print(f"Number of rows in the total dataset: {combined_df.shape[0]}")
    print(f"Number of rows in the train dataset: {X_train.shape[0]}")
    print(f"Number of rows in the test dataset: {X_test.shape[0]}")

    files = open("stopwords.txt" , "r")
    content = files.read()
    content_list = content.split("\n")
    files.close()

    tfidfvector = TfidfVectorizer(stop_words=content_list, lowercase=True)

    training_data = tfidfvector.fit_transform(X_train.values.astype('U'))

    testing_data = tfidfvector.transform(X_test.values.astype('U'))

    filename = 'tfidfvectoizer.pkl'
```

```
    pickle.dump(tfidfvector.vocabulary_, open(filename, 'wb'))

    return X_train , X_test, y_train, y_test, testing_data, filename, training_data, content_list
```

```
X_train , X_test, y_train, y_test, testing_data, filename, training_data, content_list = performdatasplit(0.2, 42, combinec
```

```
Number of rows in the total dataset: 36296
Number of rows in the train dataset: 29036
Number of rows in the test dataset: 7260
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Dense, Conv2D, MaxPooling2D, Flatten,
    Dropout, BatchNormalization
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
import xgboost as xgb
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, BaggingClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
def pipeline(X_train, y_train, X_test, y_test):
    MODELS = [LinearSVC(), LogisticRegression(), MultinomialNB(), DecisionTreeClassifier(),
              AdaBoostClassifier(), BaggingClassifier(), SGDClassifier()]

    size = len(y_train)

    results = {}

    final_result = []

    for model in MODELS:
        results['Algorithm'] = model.__class__.__name__

        start = time()
        print(f"Training Time : {model.__class__.__name__}")
        model.fit(X_train, y_train)
        end = time()

        filename = model.__class__.__name__+".pkl"
        pickle.dump(model, open(filename, "wb"))

        results['Training Time'] = end - start

        start = time()
        prediction_test = model.predict(X_test)
        prediction_train = model.predict(X_train)
        end = time()

        results['Prediction Time'] = end - start

        results['Accuracy : Test'] = accuracy_score(y_test, prediction_test)
        results['Accuracy : Train'] = accuracy_score(y_train, prediction_train)

        results['F1 Score : Test'] = f1_score(y_test, prediction_test)
        results['F1 Score : Train'] = f1_score(y_train, prediction_train)

        results['Precision : Test'] = precision_score(y_test, prediction_test)
        results['Precision : Train'] = precision_score(y_train, prediction_train)

        results['Recall : Test'] = recall_score(y_test, prediction_test)
        results['Recall : Train'] = recall_score(y_train, prediction_train)

        print(f"Training {model.__class__.__name__} finished in {results['Training Time']} sec")

        final_result.append(results.copy())
    return final_result
```

```
#deep Learning
def deep_learning_pipeline(X_train, y_train, X_test, y_test):

    results = {}
```

```
    final_result = []

    # Define Deep Learning Model
    model = Sequential([
        Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    results['Algorithm'] = 'Artificial Neural Network (ANN)'

    # Training
    start = time.time()
    print("Training Deep Learning Model (ANN)")
    model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=0)
    end = time.time()

    model.save("ANN_model.h5")

    results['Training Time'] = end - start

    # Prediction
    start = time.time()
    y_test_pred = (model.predict(X_test) > 0.5).astype(int)
    y_train_pred = (model.predict(X_train) > 0.5).astype(int)
    end = time.time()

    results['Prediction Time'] = end - start

    # Metrics
    results['Accuracy : Test'] = accuracy_score(y_test, y_test_pred)
    results['Accuracy : Train'] = accuracy_score(y_train, y_train_pred)

    results['F1 Score : Test'] = f1_score(y_test, y_test_pred)
    results['F1 Score : Train'] = f1_score(y_train, y_train_pred)

    results['Precision : Test'] = precision_score(y_test, y_test_pred)
    results['Precision : Train'] = precision_score(y_train, y_train_pred)

    results['Recall : Test'] = recall_score(y_test, y_test_pred)
    results['Recall : Train'] = recall_score(y_train, y_train_pred)

    print("Training ANN finished in", results['Training Time'], "sec")

    final_result.append(results.copy())
    return final_result
```

```
final_result = pipeline(training_data, y_train, testing_data, y_test)
```

```
Training Time : LinearSVC
Training LinearSVC finished in 0.1796247959136963 sec
Training Time : LogisticRegression
Training LogisticRegression finished in 2.9588449001312256 sec
Training Time : MultinomialNB
Training MultinomialNB finished in 0.02796006202697754 sec
Training Time : DecisionTreeClassifier
Training DecisionTreeClassifier finished in 8.220506429672241 sec
Training Time : AdaBoostClassifier
Training AdaBoostClassifier finished in 5.546273231506348 sec
Training Time : BaggingClassifier
Training BaggingClassifier finished in 75.14079308509827 sec
Training Time : SGDClassifier
Training SGDClassifier finished in 0.09268426895141602 sec
```

```
def performfinalresult(final_results):
    results = pd.DataFrame(final_results)
    results.reindex(columns = ['Algorithm', 'Accuracy : Test', 'Precision : Test', 'Recall : Test', 'F1 Score : Test', 'Pre
                        'Accuracy : Train', 'Precision : Train', 'Recall : Train', 'F1 Score : Train', 'Training Time'])
    results.sort_values(by = 'F1 Score : Test', inplace=True, ascending=False)

    return results
```

```
results = performfinalresult(final_result)
results.reset_index(drop = True)
```

	Algorithm	Training Time	Prediction Time	Accuracy : Test	Accuracy : Train	F1 Score : Test	F1 Score : Train	Precision : Test	Precision : Train	Recall : Test	Recall : Train
0	DecisionTreeClassifier	8.220506	0.039331	0.973829	0.996832	0.979907	0.997534	0.972706	0.996146	0.987215	0.998926
1	BaggingClassifier	75.140793	0.475684	0.968871	0.994972	0.976008	0.996086	0.972498	0.994859	0.979544	0.997316
2	LinearSVC	0.179625	0.002424	0.964738	0.989048	0.972690	0.991459	0.973937	0.992152	0.971447	0.990767
3	LogisticRegression	2.958845	0.002992	0.936364	0.961152	0.950514	0.969637	0.955632	0.972464	0.945451	0.966826
4	SGDClassifier	0.092684	0.002825	0.936364	0.958190	0.950301	0.967221	0.959592	0.973054	0.941189	0.961458
5	MultinomialNB	0.027960	0.011166	0.899174	0.927676	0.925428	0.945438	0.886590	0.916159	0.967824	0.97664

```
print(type(X_train))
print(X_train.shape)
```

```
<class 'pandas.core.series.Series'>
(29036,)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(
    max_features=5000,
    ngram_range=(1,2),
    stop_words='english'
)

# Convert text series to numerical vectors
X_train = tfidf.fit_transform(X_train).toarray()
X_test = tfidf.transform(X_test).toarray()
```

```
import pickle
pickle.dump(tfidf, open("tfidf.pkl", "wb"))
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(29036, 5000)
(7260, 5000)
```

```
#deep learning
import numpy as np
import pandas as pd
import time
import pickle

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import (
    accuracy_score, precision_score,
    recall_score, f1_score, confusion_matrix
)

from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam
```

```
# Re-initialize X_train and X_test to their original text series form
# The variables X_train and X_test were overwritten by numpy arrays in a previous cell.
# We need to retrieve the original text versions before applying TfidfVectorizer again.
original_X_train_text, original_X_test_text, _, _, _, _, _ = performdatasplit(0.2, 42, combined_df, df)

tfidf = TfidfVectorizer(
    max_features=5000,
    ngram_range=(1, 2),
    stop_words="english"
)
```

```
X_train = tfidf.fit_transform(original_X_train_text).toarray()
X_test  = tfidf.transform(original_X_test_text).toarray()

# Save vectorizer for future prediction
pickle.dump(tfidf, open("tfidf.pkl", "wb"))

print(X_train.shape)
print(X_test.shape)
```

```
Number of rows in the total dataset: 36296
Number of rows in the train dataset: 29036
Number of rows in the test dataset: 7260
/usr/local/lib/python3.12/dist-packages/sklearn/feature_extraction/text.py:402: UserWarning: Your stop_words may be inconsis
  warnings.warn(
(29036, 5000)
(7260, 5000)
```

```
def deep_learning_pipeline(X_train, y_train, X_test, y_test):

    results = {}

    model = Sequential([
        Input(shape=(X_train.shape[1],)),
        Dense(128, activation="relu"),
        Dropout(0.3),
        Dense(64, activation="relu"),
        Dropout(0.2),
        Dense(1, activation="sigmoid")
    ])

    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss="binary_crossentropy",
        metrics=["accuracy"]
    )

    results["Algorithm"] = "Artificial Neural Network (ANN)"

    # Training
    start = time.time()
    print("Training ANN Model...")
    model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=1)
    end = time.time()

    model.save("ANN_model.h5")

    results["Training Time"] = end - start

    # Prediction
    start = time.time()
    y_test_pred  = (model.predict(X_test)  > 0.5).astype(int)
    y_train_pred = (model.predict(X_train) > 0.5).astype(int)
    end = time.time()

    results["Prediction Time"] = end - start

    # Metrics
    results["Accuracy Test"]  = accuracy_score(y_test, y_test_pred)
    results["Accuracy Train"] = accuracy_score(y_train, y_train_pred)

    results["Precision Test"] = precision_score(y_test, y_test_pred)
    results["Recall Test"]    = recall_score(y_test, y_test_pred)
    results["F1 Test"]        = f1_score(y_test, y_test_pred)

    print("Training finished in", results["Training Time"], "seconds")

    return results
```

```
dl_results = deep_learning_pipeline(X_train, y_train, X_test, y_test)
dl_results
```

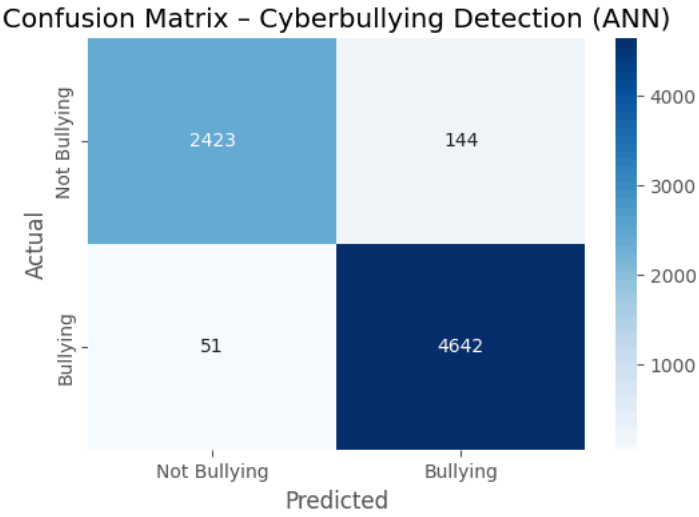
```
Training ANN Model...
Epoch 1/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 11s 10ms/step - accuracy: 0.8197 - loss: 0.3677
Epoch 2/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9475 - loss: 0.1365
Epoch 3/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9706 - loss: 0.0774
Epoch 4/20
```

```
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9828 - loss: 0.0479
Epoch 5/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9869 - loss: 0.0331
Epoch 6/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9894 - loss: 0.0265
Epoch 7/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9898 - loss: 0.0239
Epoch 8/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9909 - loss: 0.0217
Epoch 9/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9915 - loss: 0.0199
Epoch 10/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9918 - loss: 0.0188
Epoch 11/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - accuracy: 0.9923 - loss: 0.0181
Epoch 12/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9925 - loss: 0.0171
Epoch 13/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9921 - loss: 0.0166
Epoch 14/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9924 - loss: 0.0181
Epoch 15/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9919 - loss: 0.0162
Epoch 16/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9920 - loss: 0.0166
Epoch 17/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9934 - loss: 0.0146
Epoch 18/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9925 - loss: 0.0158
Epoch 19/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9929 - loss: 0.0163
Epoch 20/20
908/908 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9922 - loss: 0.0156
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
227/227 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
908/908 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step
Training finished in 191.48097491264343 seconds
{'Algorithm': 'Artificial Neural Network (ANN)',
 'Training Time': 191.48097491264343,
 'Prediction Time': 5.243460416793823,
 'Accuracy Test': 0.9731404958677686,
 'Accuracy Train': 0.9934219589475134,
 'Precision Test': 0.9699122440451317,
 'Recall Test': 0.989132750905604,
 'F1 Test': 0.9794282097267645}
```

```
#graphical Representation
def plot_confusion_matrix_dl(X_test, y_test):
    model = load_model("ANN_model.h5")

    y_pred = (model.predict(X_test) > 0.5).astype(int)
    cm = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(6,4))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=["Not Bullying", "Bullying"],
        yticklabels=["Not Bullying", "Bullying"]
    )
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix – Cyberbullying Detection (ANN)")
    plt.show()
```

```
plot_confusion_matrix_dl(X_test, y_test)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty
227/227 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```




```
import ipywidgets as widgets
from IPython.display import display, clear_output
```

Model Testing

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

model = load_model("ANN_model.h5")
tfidf = pickle.load(open("tfidf.pkl", "rb"))
```

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

```
def predict_text(text):
    vector = tfidf.transform([text]).toarray()
    prob = model.predict(vector)[0][0]

    if prob >= 0.5:
        return f"Cyberbullying Detected\nConfidence: {prob*100:.2f}%"
    else:
        return f" Not Cyberbullying\nConfidence: {(1-prob)*100:.2f}%"
```

```
# Text input box
text_input = widgets.Textarea(
    value='',
    placeholder='Enter text to check cyberbullying...',
    description='Input:',
    layout=widgets.Layout(width='80%', height='100px')
)

# Button
predict_button = widgets.Button(
    description='Check',
    button_style='primary'
)

# Output area
output = widgets.Output()
```

```
def on_button_clicked(b):
    with output:
        clear_output()
        if text_input.value.strip() == "":
            print(" Please enter some text")
        else:
            result = predict_text(text_input.value)
            print(result)
```

```
predict_button.on_click(on_button_clicked)
```

```
display(text_input, predict_button, output)
```

Input: hai fucks u

Check

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
🚨 Cyberbullying Detected
Confidence: 99.87%
```

Dashborad Analysis

```
!pip install gradio
```

Show hidden output

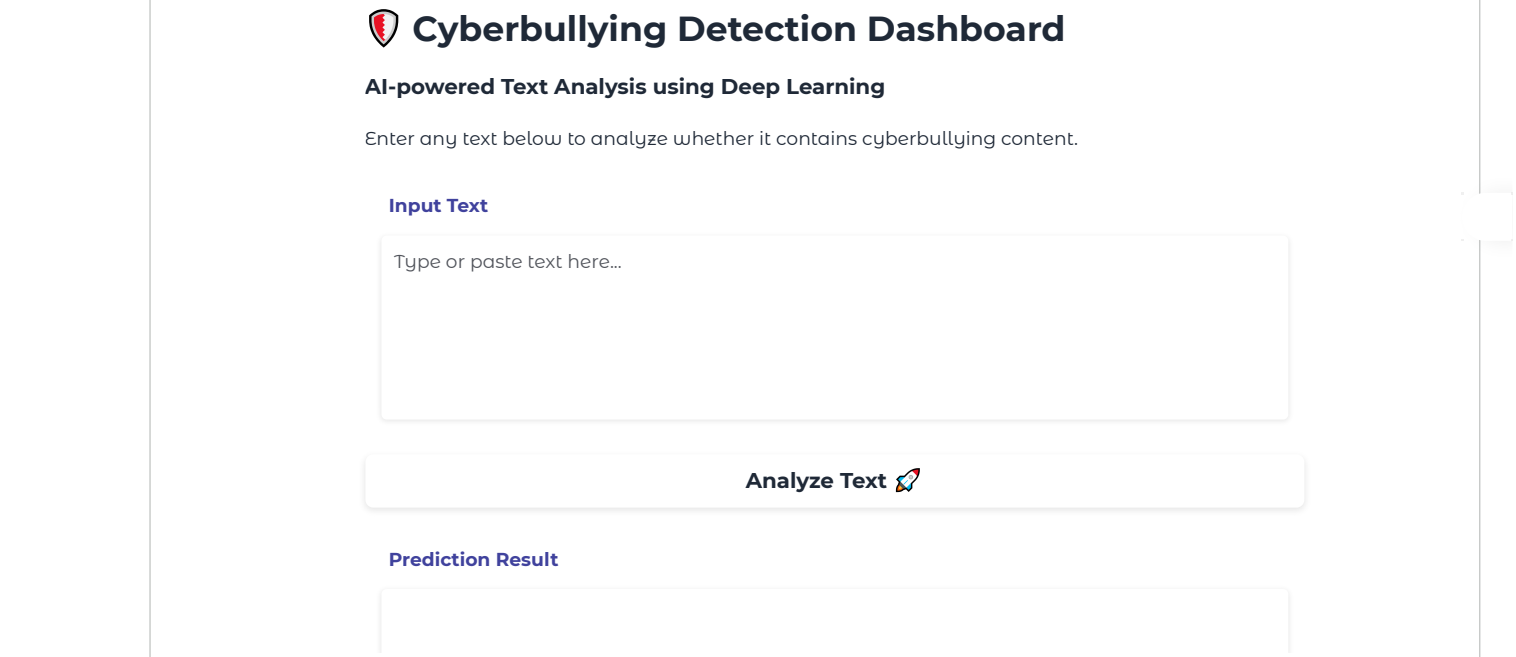
```
from tensorflow.keras.models import load_model
import pickle

model = load_model("ANN_model.h5")
tfidf = pickle.load(open("tfidf.pkl", "rb"))
```

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

```
def check_cyberbullying(text):
    vector = tfidf.transform([text]).toarray()
    prob = model.predict(vector)[0][0]

    if prob >= 0.5:
        return f" Cyberbullying Detected\n\nConfidence: {prob*100:.2f}%"
    else:
        return f"Not Cyberbullying\n\nConfidence: {(1-prob)*100:.2f}%"
```

```
import gradio as gr

with gr.Blocks(theme=gr.themes.Soft()) as demo:

    gr.Markdown(
        """
        # 🛡️ Cyberbullying Detection Dashboard
        ### AI-powered Text Analysis using Deep Learning

        Enter any text below to analyze whether it contains cyberbullying content.
        """
    )

    text_input = gr.Textbox(
        lines=6,
        placeholder="Type or paste text here...",
        label="Input Text",
        scale=2
    )

    check_btn = gr.Button("Analyze Text 🚀")

    output = gr.Textbox(
        label="Prediction Result",
        lines=4
    )

    check_btn.click(
        fn=check_cyberbullying,
        inputs=text_input,
        outputs=output
    )

demo.launch(share=True)
```

```
/tmp/ipython-input-3903926268.py:3: DeprecationWarning: The 'theme' parameter in the Blocks constructor will be removed in G
  with gr.Blocks(theme=gr.themes.Soft()) as demo:
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://9a7d9e49a101ff0590.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the
```



Next steps: Deploy to Cloud Run

Start coding or generate with AI.