Introduction

- Abusive relationships are a critical issue that affect people worldwide.
- With the rise of social media, more people are sharing personal stories.
- Using NLP, we can detect signs of abuse and identify key risk factors.
- Allows for efficient identification of abusive patterns and better support for victims.

Data

Dataset of 10k Reddit posts which has been processed to:

- Post Metadata
- Relationship and Demographic data generated by Gemini pro
- Contexual Risk Factors generated by Gemini pro notice

Installations & Imports

```
# Installations
!pip install matplotlib seaborn
!pip install upgrade pandas
!pip install openpyxl
!pip install lime
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: numpy<2,>=1.21 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-packages (from seaborn) (2.2.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
ERROR: Could not find a version that satisfies the requirement upgrade (from versions: none)
ERROR: No matching distribution found for upgrade
Requirement already satisfied: openpyxl in /usr/local/lib/python3.10/dist-packages (3.1.5)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.10/dist-packages (from openpyxl) (2.0.0)
Requirement already satisfied: lime in /usr/local/lib/python3.10/dist-packages (0.2.0.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from lime) (3.8.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from lime) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from lime) (1.13.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from lime) (4.66.6)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.10/dist-packages (from lime) (1.5.2)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.10/dist-packages (from lime) (0.24.0)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (3.4.2)
Requirement already satisfied: pillow>=9.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (11.0.0)
Requirement already satisfied: imageio>=2.33 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (2.36.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (2024
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (3.5
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.4.7)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->lime) (1
```

```
# Imports
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib.colors import LogNorm
import matplotlib.cm as cm
```

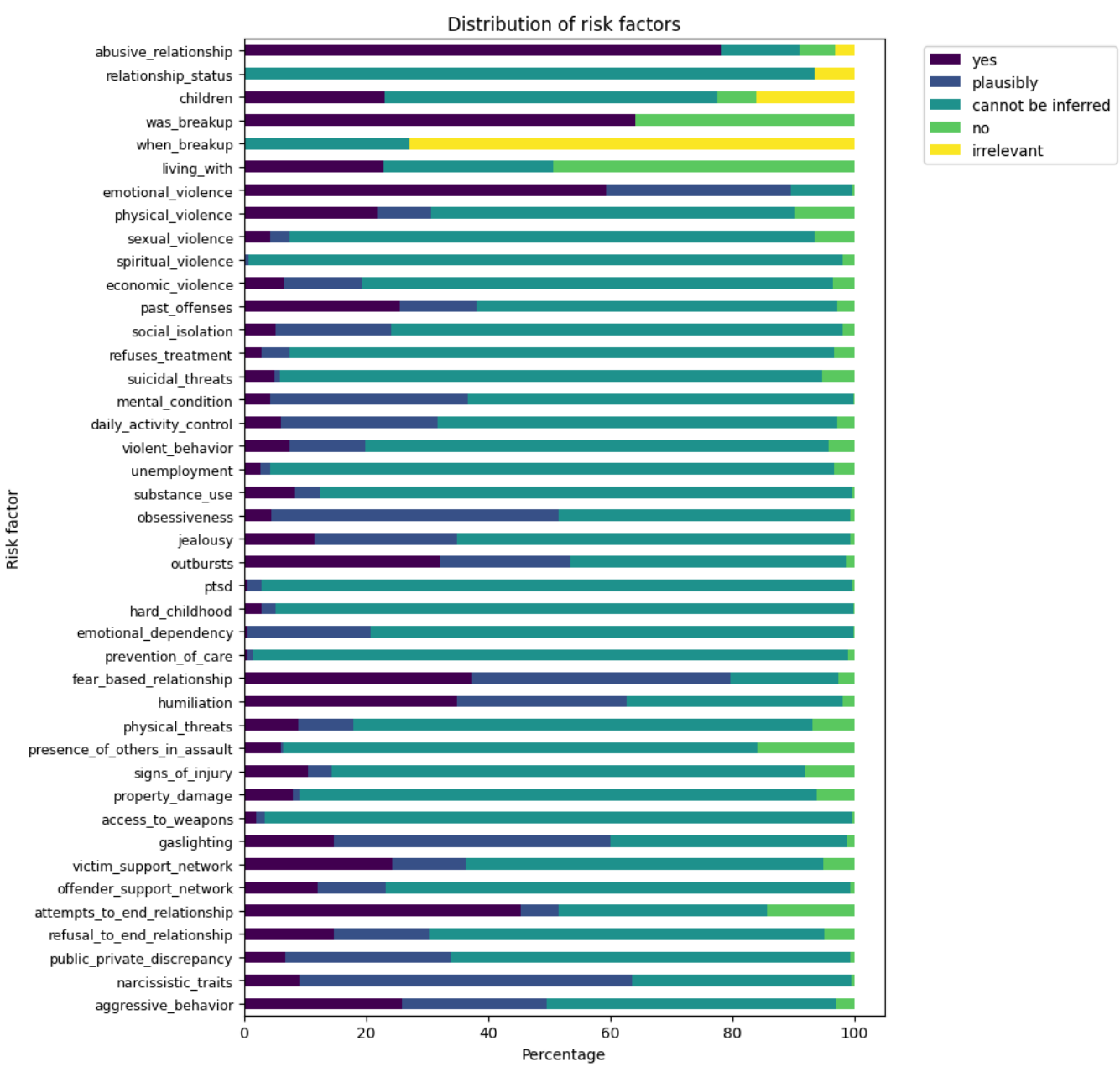
```
import plotly.express as px
from plotly.subplots import make_subplots

import nltk
from nltk.corpus import words

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, \
  precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
from sklearn.svm import SVC

import torch
from torch.nn import BCEWithLogitsLoss
from torch.utils.data import DataLoader, TensorDataset

from collections import Counter, defaultdict
import seaborn as sns
from tabulate import tabulate
import statsmodels.api as sm
from tqdm import tqdm
import zipfile
import random

from transformers import RobertaTokenizer, DistilBertTokenizer, RobertaForSequenceClassification, \
    DistilBertForSequenceClassification, AdamW

import gc
from lime.lime_text import LimeTextExplainer
```

Part 1 - Data Validation

Zoom out on the distributions of the data

```
#
# Load the Excel file, using the first row as headers
df = pd.read_excel('Labeled.xlsx', sheet_name='All', engine='openpyxl')

# Select the specified columns range
columns_range = df.columns[23:70]

# Define columns to exclude - columns with a different scale (shown in the next cell)
exclude_columns = ['author_gender', 'age_female', 'age_male', 'author_role', 'relationship_type']

# Filter out the columns to exclude
selected_columns = [col for col in columns_range if col not in exclude_columns]

# Create a new DataFrame with only the selected columns
df_selected = df[selected_columns]

# Define the possible labels
labels = ['yes', 'plausibly', 'cannot be inferred', 'no', 'irrelevant']

# Count the occurrences of each label in each column
label_counts = pd.DataFrame({label: (df_selected == label).sum() for label in labels}, index=selected_columns)

# Convert counts to percentages
label_percentages = label_counts.div(label_counts.sum(axis=1), axis=0) * 100

# Plot the stacked bar plot
fig, ax = plt.subplots(figsize=(12, 10))
label_percentages.plot(kind='barh', stacked=True, ax=ax, colormap='viridis')

# Add labels and title
ax.set_xlabel('Percentage')
ax.set_ylabel('Risk factor')
ax.set_title("Distribution of risk factors")

# Adjust spacing and reverse the Y-axis order
ax.set_yticks(range(len(selected_columns)))
ax.set_yticklabels(selected_columns, fontsize=9)
ax.invert_yaxis()

# Show legend and adjust layout
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
```

```
plt.subplots_adjust(left=0.3)  # Adjust margin for better readability
plt.show()
```

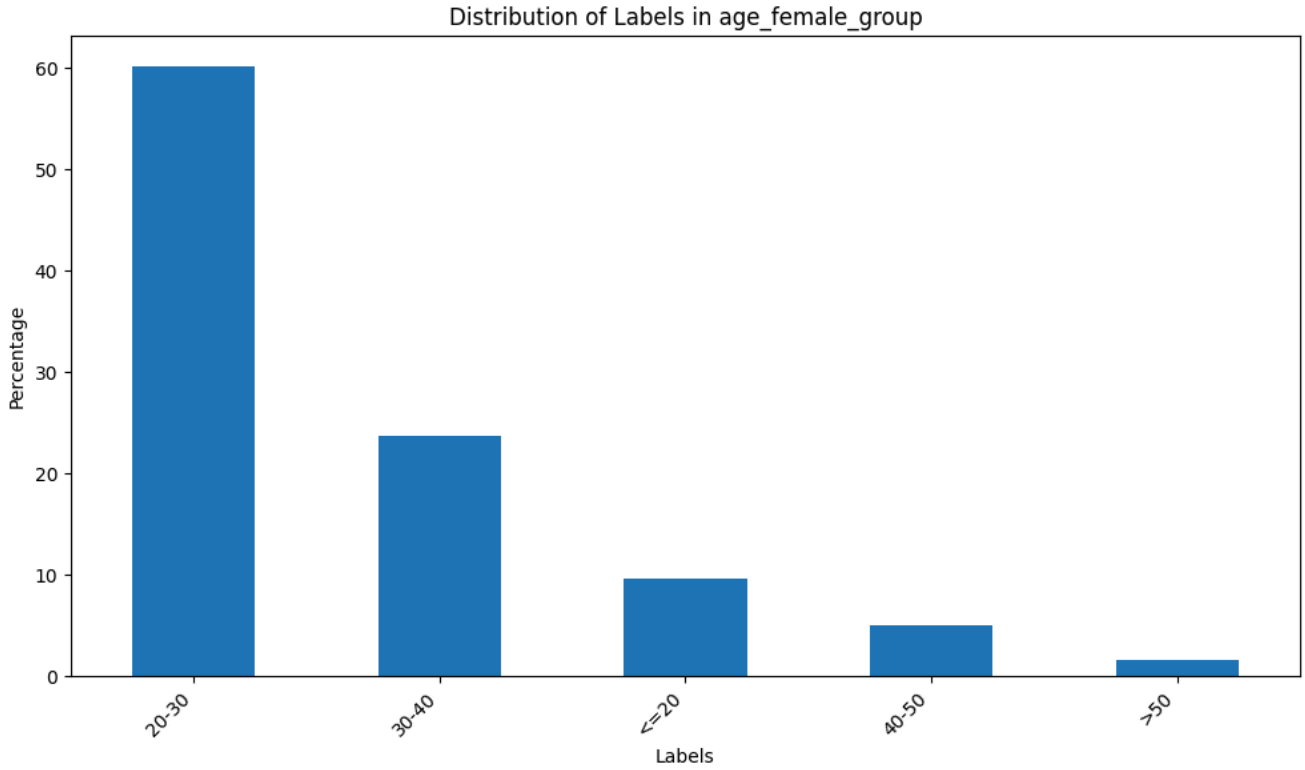


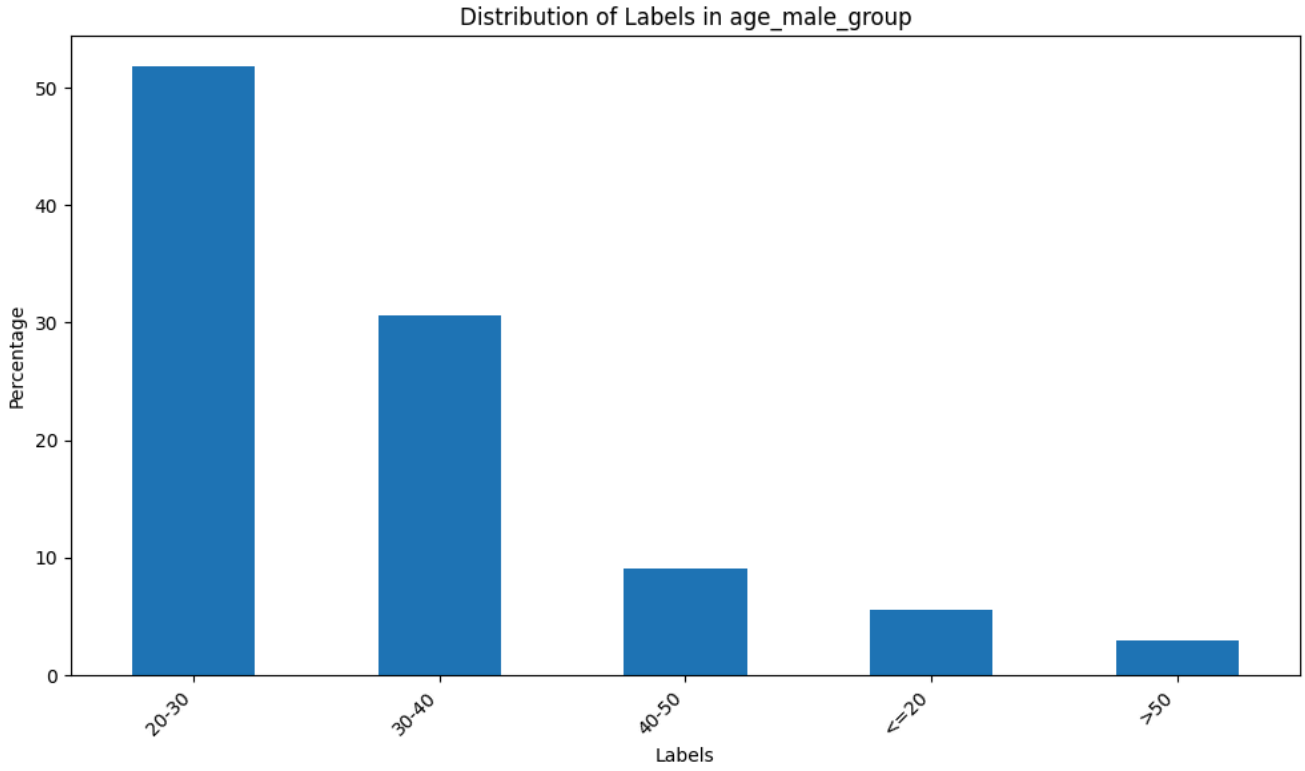
Ditribution of columns with different range

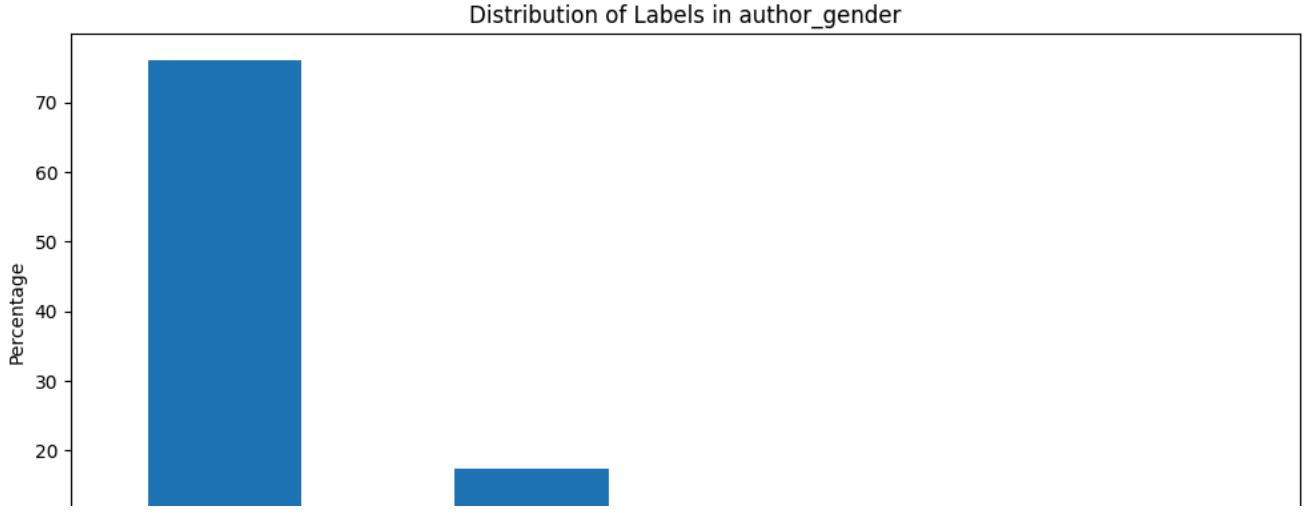
```
# Load the Excel file
df_excluded = pd.read_excel('Labeled.xlsx', sheet_name='All', engine='openpyxl')

# Define age bins and labels for grouping ages
age_bins = [0, 20, 30, 40, 50, float('inf')]
age_labels = ['<=20', '20-30', '30-40', '40-50', '>50']

# Process age-related columns
if 'age_female' in exclude_columns:
    df_excluded['age_female_group'] = pd.cut(df_excluded['age_female'], bins=age_bins, labels=age_labels)
if 'age_male' in exclude_columns:
    df_excluded['age_male_group'] = pd.cut(df_excluded['age_male'], bins=age_bins, labels=age_labels)

# Replace age columns with grouped versions in exclude_columns if present
exclude_columns = [
    'age_female_group' if 'age_female' in exclude_columns else 'age_female',
    'age_male_group' if 'age_male' in exclude_columns else 'age_male',
    *[col for col in exclude_columns if col not in ['age_female', 'age_male']]
]

# Iterate through each column in exclude_columns
for column in exclude_columns:
```

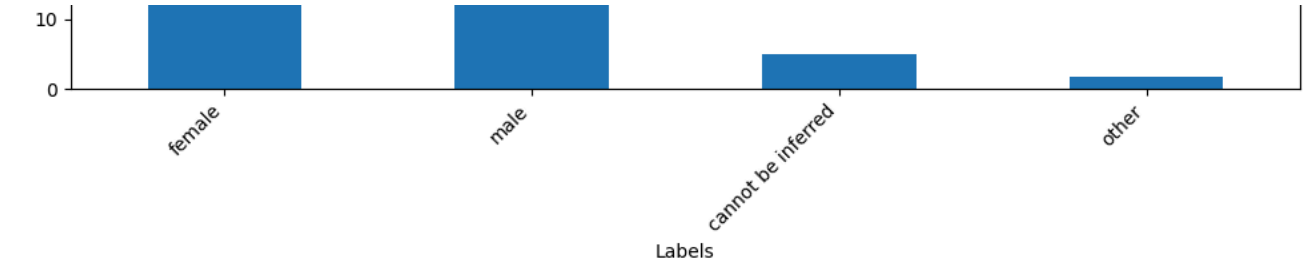
```
        # Skip columns that don't exist in the group
        if column not in df_excluded:
            continue

        # Extract unique labels for the current column
        labels = df_excluded[column].dropna().unique()

        # Count occurrences of each label
        label_counts = df_excluded[column].value_counts(normalize=True) * 100

        # Plot the distribution of labels for the current column
        fig, ax = plt.subplots(figsize=(10, 6))
        label_counts.plot(kind='bar', ax=ax)  # Changed to 'bar' for vertical bars

        # Add labels and title
        ax.set_xlabel('Labels')        # Switched x-axis label
        ax.set_ylabel('Percentage')    # Switched y-axis label
        ax.set_title(f"Distribution of Labels in {column}")

        # Rotate x-axis labels if they are too long
        plt.xticks(rotation=45, ha='right')

        # Adjust layout and display the plot
        plt.tight_layout()
        plt.show()
```

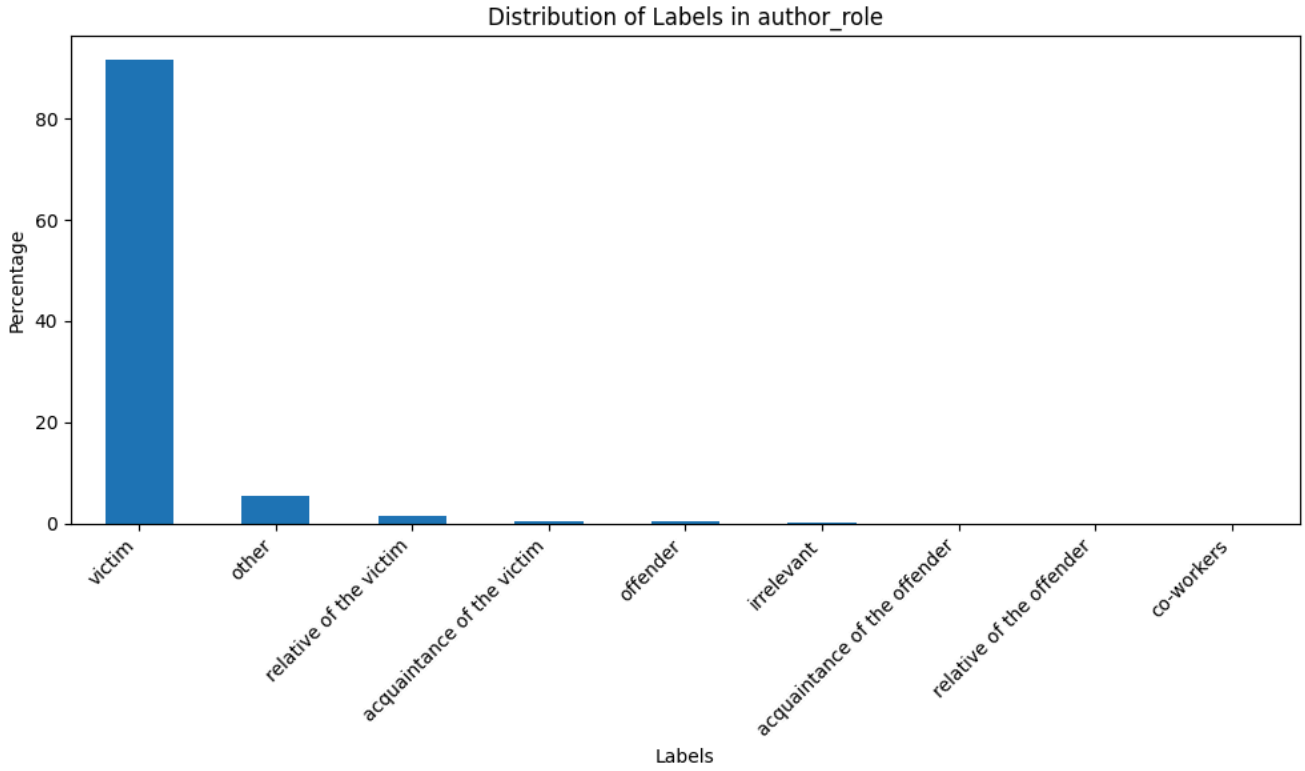
[+]

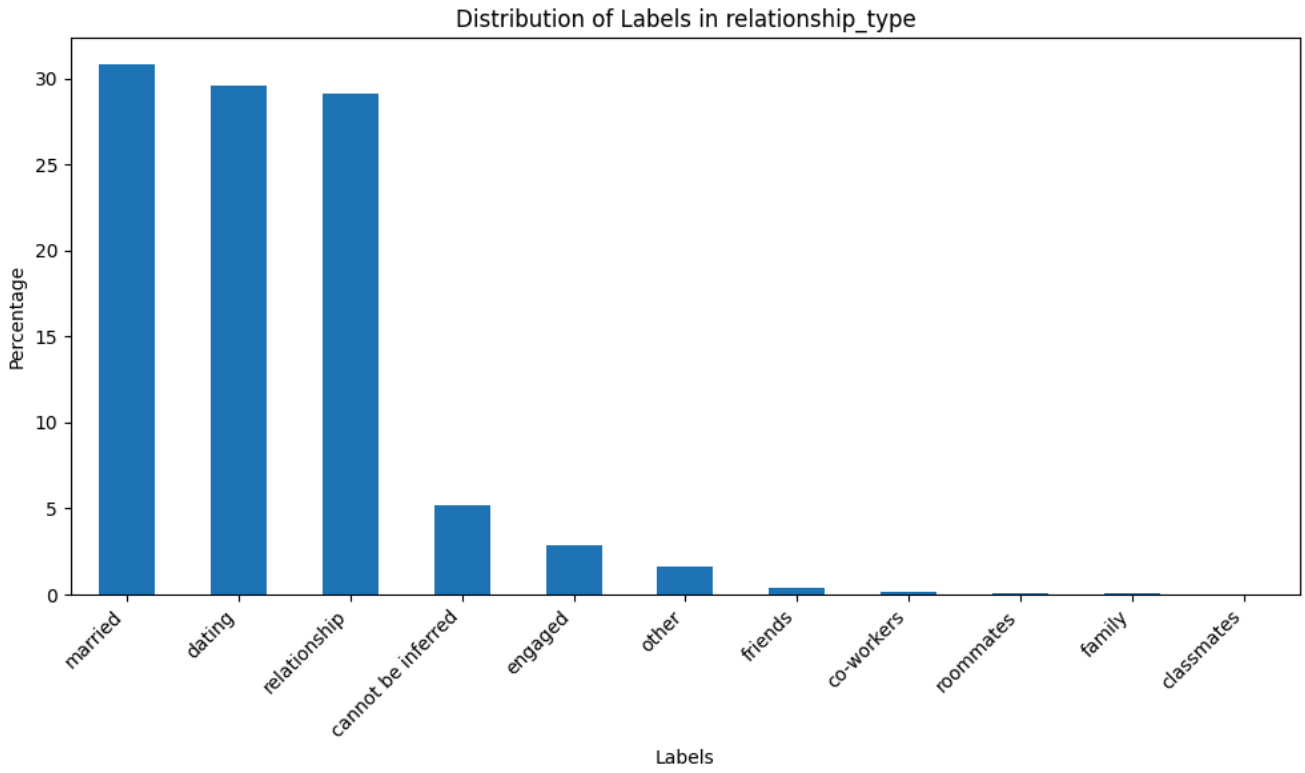

Distribution of Labels in age_female_group
Percentage
0
10
20
30
40
50
60
20-30
30-40
<=20
40-50
>50
Labels


Distribution of Labels in age_male_group
Percentage
0
10
20
30
40
50
20-30
30-40
40-50
<=20
>50
Labels


Distribution of Labels in author_gender
Percentage
20
30
40
50
60
70


10
0
female
male
cannot be inferred
other
Labels


Distribution of Labels in author_role
Percentage
80
60
40
20
0
victim
other
relative of the victim
acquaintance of the victim
offender
irrelevant
acquaintance of the offender
relative of the offender
co-workers
Labels


Distribution of Labels in relationship_type
Percentage
30
25
20
15
10
5
0
married
dating
relationship
cannot be inferred
engaged
other
friends
co-workers
roommates
family
classmates
Labels

Notice that there are no missing values in the columns of Relationship, Demographic and risk factore, as expected, since the data was generated.

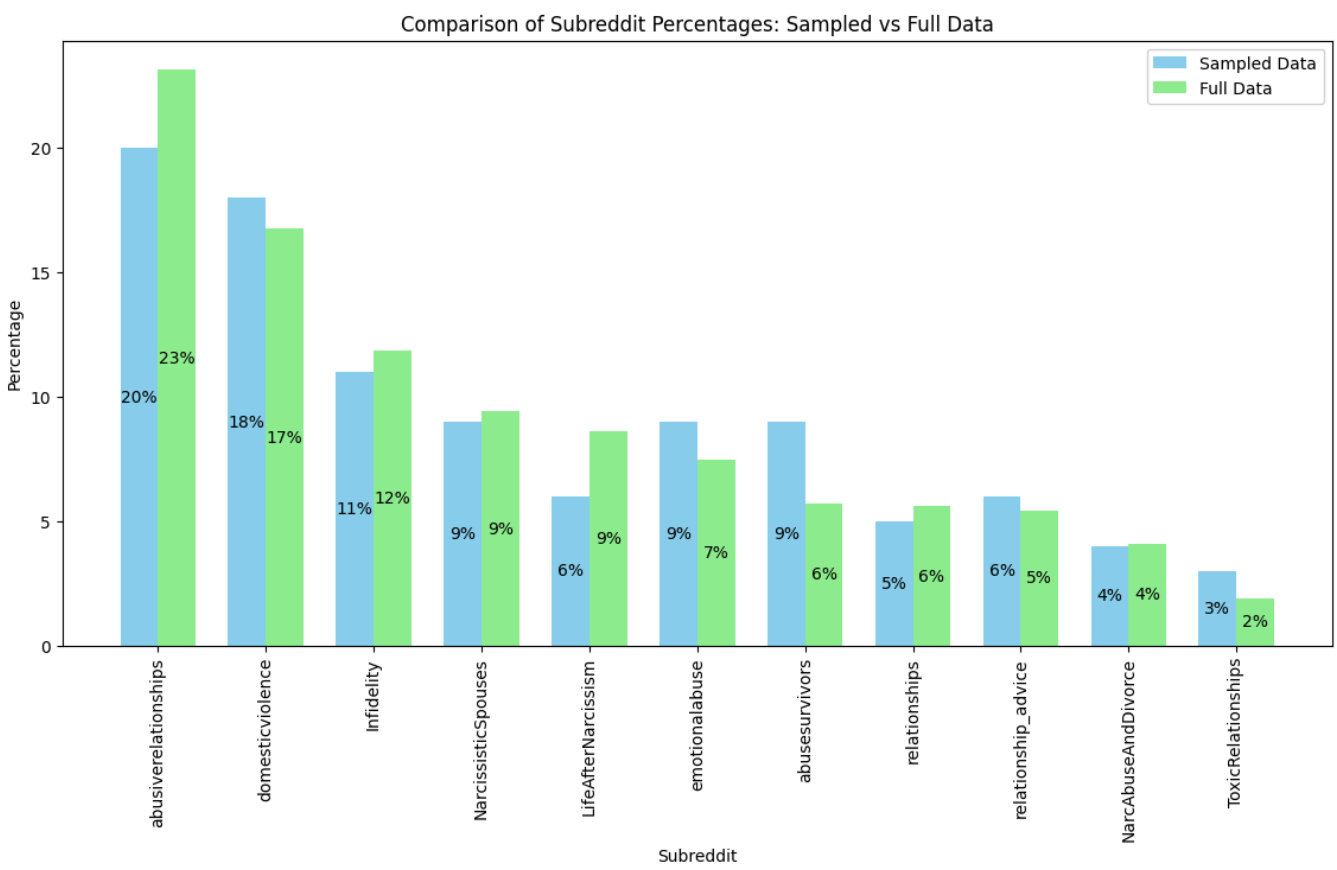
Part 2 - Data Exploration

- Sampling 100 examples.
- Verifying that the data is representative.
- Manually classifying based on the stories.

2.1 We randomly selected 100 examples and examined their representation based on subreddits.

```
# Define the sheets to process and the fixed row count for the sampled data
sheets = ['Test', 'All']  # Test = sample, All = full dataset
fixed_row_counts = {'Test': 100}  # Fixed row count for the sampled dataset

# Dictionary to store extracted subreddit values from each sheet
subreddit_data = {}

# Load the Excel file once
excel_file = pd.ExcelFile('Labeled.xlsx')

# Process each sheet
for sheet_name in sheets:
    # Load the sheet into a DataFrame
    df = excel_file.parse(sheet_name=sheet_name)

    # Specify the columns to process (column at index 2)
    subreddit_column = df.columns[2:3]

    # Determine the number of rows to process
    if sheet_name in fixed_row_counts:
        num_rows = fixed_row_counts[sheet_name]
    else:
        num_rows = len(df)  # Dynamically calculate for the full dataset

    # Extract subreddit values from the specified column
    subreddit_values = []
    for i in range(num_rows):
        subreddit_values.extend(df.loc[i, subreddit_column].tolist())

    # Store the subreddit values in the dictionary
    subreddit_data[sheet_name] = subreddit_values

# Access extracted subreddit values
sampled_subreddits = subreddit_data['Test']
full_subreddits = subreddit_data['All']

# Count the occurrences of each unique subreddit
sampled_counts = Counter(sampled_subreddits)
full_counts = Counter(full_subreddits)

# Combine "survivinginfidelity" into the key "Infidelity" for consistency
sampled_counts['Infidelity'] += sampled_counts.pop('survivinginfidelity', 0)
full_counts['Infidelity'] += full_counts.pop('survivinginfidelity', 0)

# Calculate the total counts for normalization
total_sampled = sum(sampled_counts.values())
total_full = sum(full_counts.values())

# Convert counts to percentages
sampled_percentages = {subreddit: (count / total_sampled) * 100 for subreddit, count in sampled_counts.items()}
full_percentages = {subreddit: (count / total_full) * 100 for subreddit, count in full_counts.items()}

# Get a combined list of all unique subreddits from both datasets
all_subreddits = set(sampled_percentages.keys()).union(set(full_percentages.keys()))

# Sort subreddits by their percentage in the full dataset (descending order)
sorted_subreddits = sorted(all_subreddits, key=lambda sub: full_percentages.get(sub, 0), reverse=True)

# Prepare data for the grouped bar chart
sampled_percentages_sorted = [sampled_percentages.get(sub, 0) for sub in sorted_subreddits]
full_percentages_sorted = [full_percentages.get(sub, 0) for sub in sorted_subreddits]

# Plot the grouped bar chart
x_positions = range(len(sorted_subreddits))
bar_width = 0.35  # Width of the bars
```

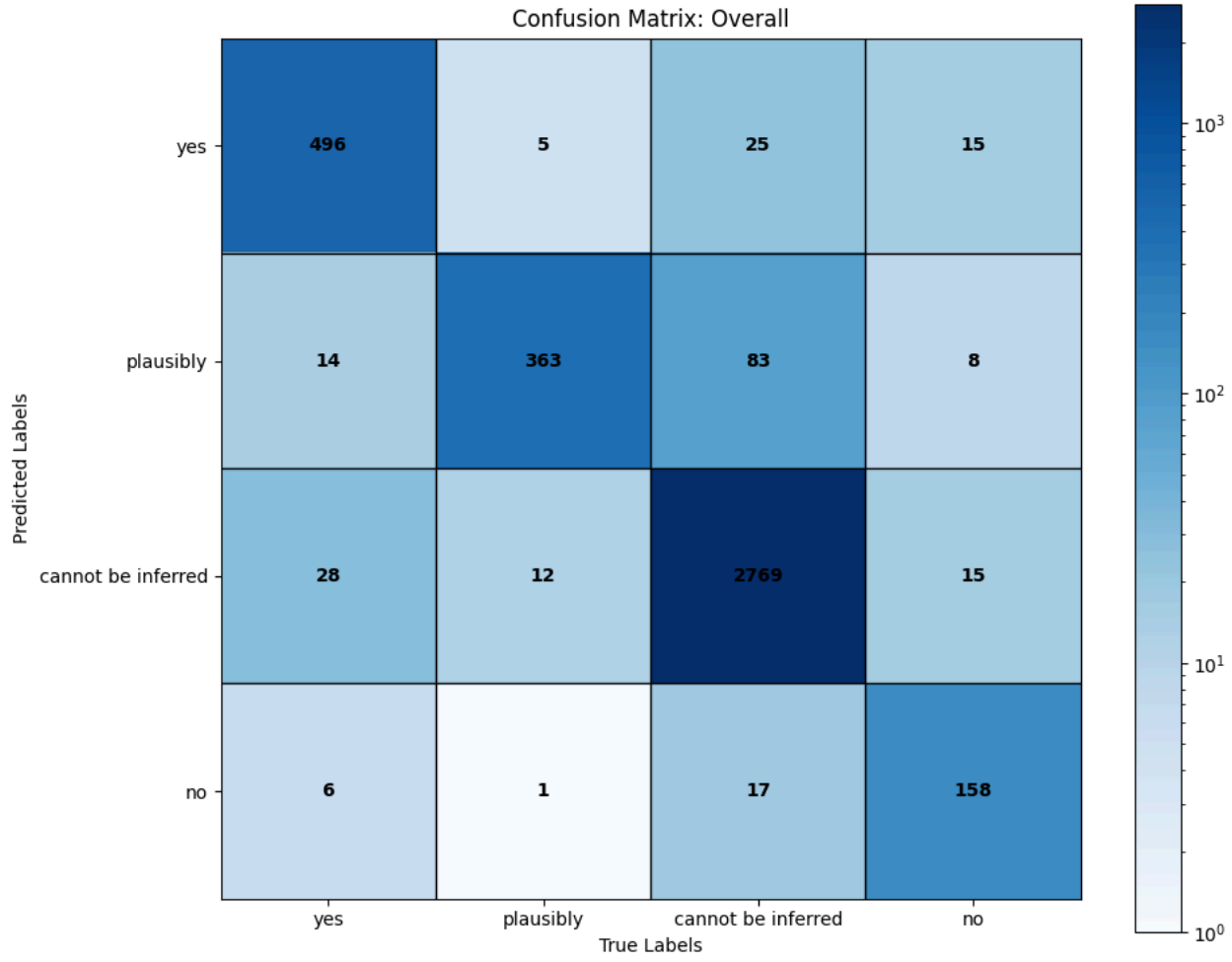
```
fig, ax = plt.subplots(figsize=(12, 8))
bars_sampled = ax.bar([pos - bar_width/2 for pos in x_positions], sampled_percentages_sorted, bar_width, label='Sampled Data', color='sl
bars_full = ax.bar([pos + bar_width/2 for pos in x_positions], full_percentages_sorted, bar_width, label='Full Data', color='lightgreen

# Add labels, title, and legend
ax.set_xlabel('Subreddit')
ax.set_ylabel('Percentage')
ax.set_title('Comparison of Subreddit Percentages: Sampled vs Full Data')
ax.set_xticks(x_positions)
ax.set_xticklabels(sorted_subreddits, rotation=90)
ax.legend()

# Annotate the bars with the percentage values
for bars in [bars_sampled, bars_full]:
    for bar in bars:
        y_value = bar.get_height()
        if y_value > 0:  # Only annotate bars with a value greater than 0
            ax.text(
                bar.get_x() + bar.get_width() / 2, y_value / 2, f'{round(y_value)}%',
                ha='center', va='center', color='black'
            )

# Adjust layout for better spacing
plt.tight_layout(pad=3)
plt.show()
```



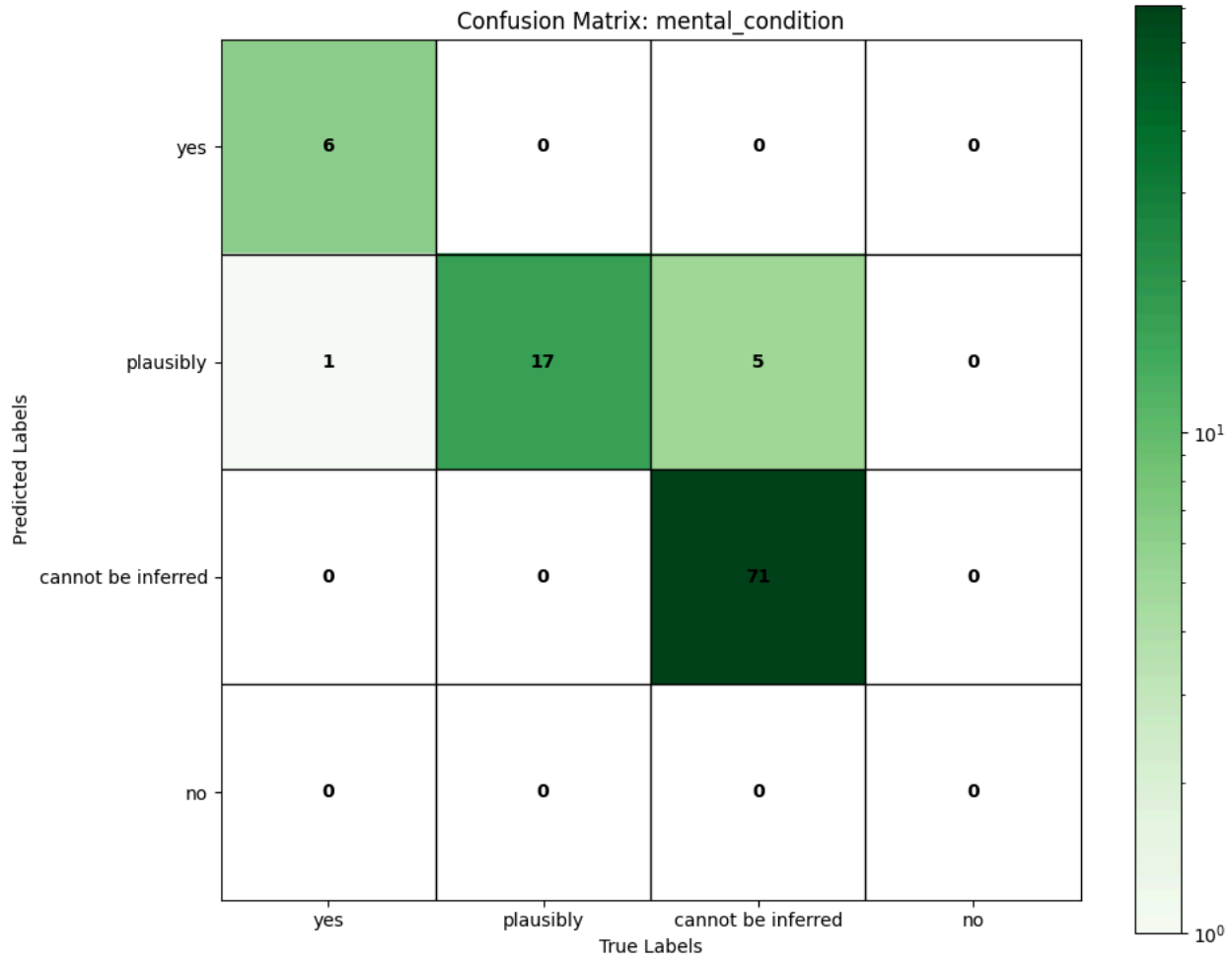
2.2 We compared Gemini Pro predictions with the true labels manually annotated by us.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, f1_score
from matplotlib.colors import LogNorm

# Load the Excel file
df = pd.read_excel('Labeled.xlsx', sheet_name='Test', engine='openpyxl')

# Define the contexts to search for in the column headers
contexts_to_find = ["Overall", "mental_condition", "jealousy", "emotional_violence"]
```

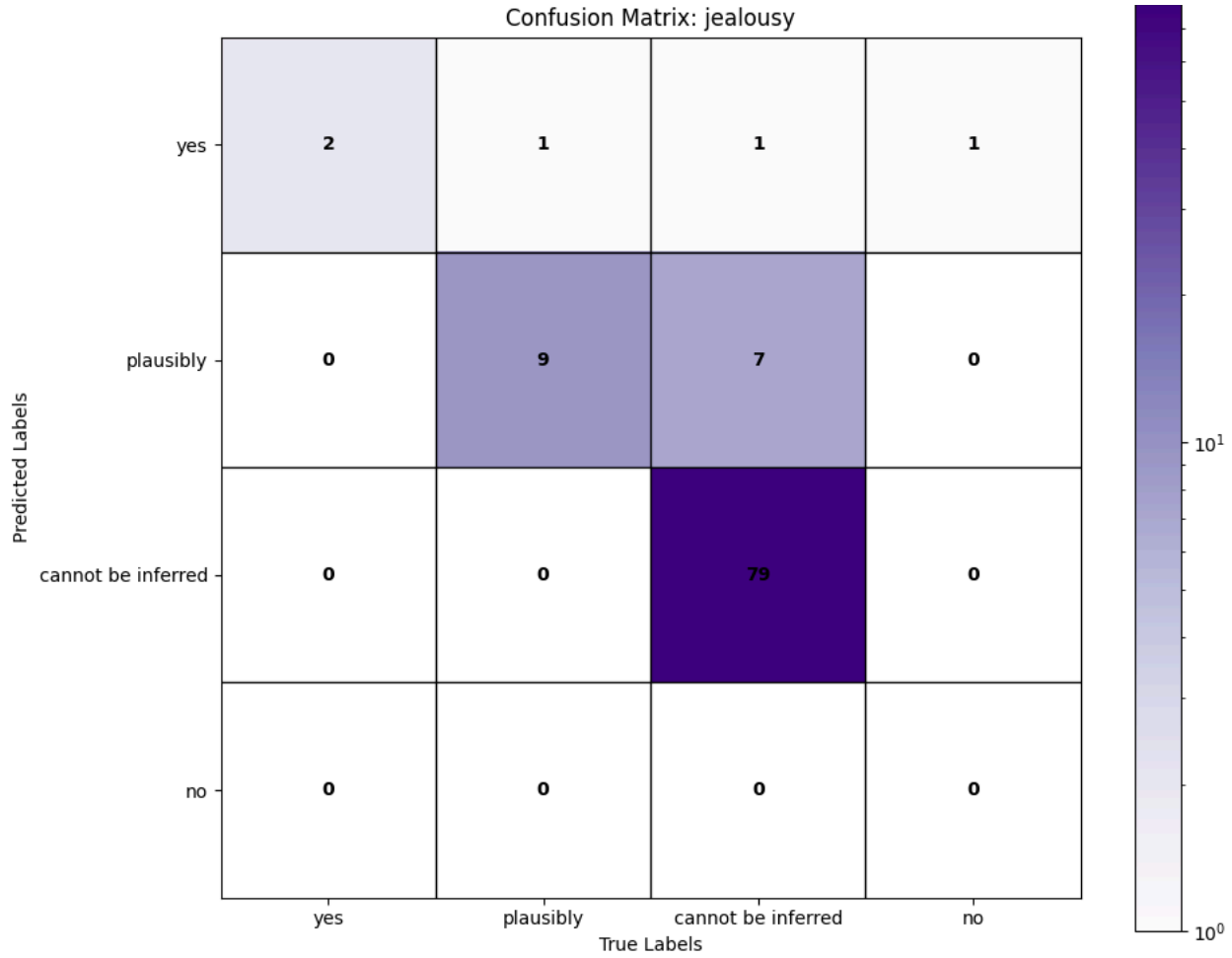
```
# Dynamically find column ranges or indices for the contexts
contexts = {}
for context in contexts_to_find:
    if context == "Overall":
        contexts[context] = df.columns[23:70]  # Overall is a range of columns
    else:
        contexts[context] = [col for col in df.columns if context in col]

# Define number of rows for predicted and true values
num_rows = 100

# Define the labels for the confusion matrix
labels = ['yes', 'plausibly', 'cannot be inferred', 'no']

# Define colors for each plot
plot_colors = {
    "Overall": "Blues",
    "mental_condition": "Greens",
    "jealousy": "Purples",
    "emotional_violence": "Oranges"
}

# Loop through each context and generate a plot
for context_name, columns_range in contexts.items():
    # Extract predicted values (first 100 rows)
    Predicted_values = []
    for i in range(num_rows):
        Predicted_values.extend(df.loc[i, columns_range].tolist())

    # Extract true values (rows 110-209)
    True_values = []
    for i in range(num_rows):
        True_values.extend(df.loc[i + 109, columns_range].tolist())

    # Compute the confusion matrix
    cm = confusion_matrix(Predicted_values, True_values, labels=labels)

    # Plot the confusion matrix
    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=plot_colors[context_name], norm=LogNorm(vmin=1, vmax=cm.max()))
    plt.colorbar()

    # Annotate the cells with counts
    for i in range(len(labels)):
        for j in range(len(labels)):
            plt.text(j, i, cm[i, j], ha='center', va='center', color='black', fontweight='bold')

    # Add gridlines
    for i in range(len(labels) - 1):
        plt.axhline(i + 0.5, color='black', linestyle='-', linewidth=1)
        plt.axvline(i + 0.5, color='black', linestyle='-', linewidth=1)

    # Add title and axis labels
    plt.title(f'Confusion Matrix: {context_name}')
    plt.xlabel('True Labels')
    plt.ylabel('Predicted Labels')
    plt.xticks(ticks=np.arange(len(labels)), labels=labels)
    plt.yticks(ticks=np.arange(len(labels)), labels=labels)

    # Adjust layout and show the plot
    plt.tight_layout()
    plt.show()

    # Calculate and print F1 scores
    f1_weighted = f1_score(True_values, Predicted_values, average='weighted')
    f1_macro = f1_score(True_values, Predicted_values, average='macro')
    print(f"F1 Score (Weighted) for {context_name}: {f1_weighted:.2f}")
    print(f"F1 Score (Macro) for {context_name}: {f1_macro:.2f}")
```



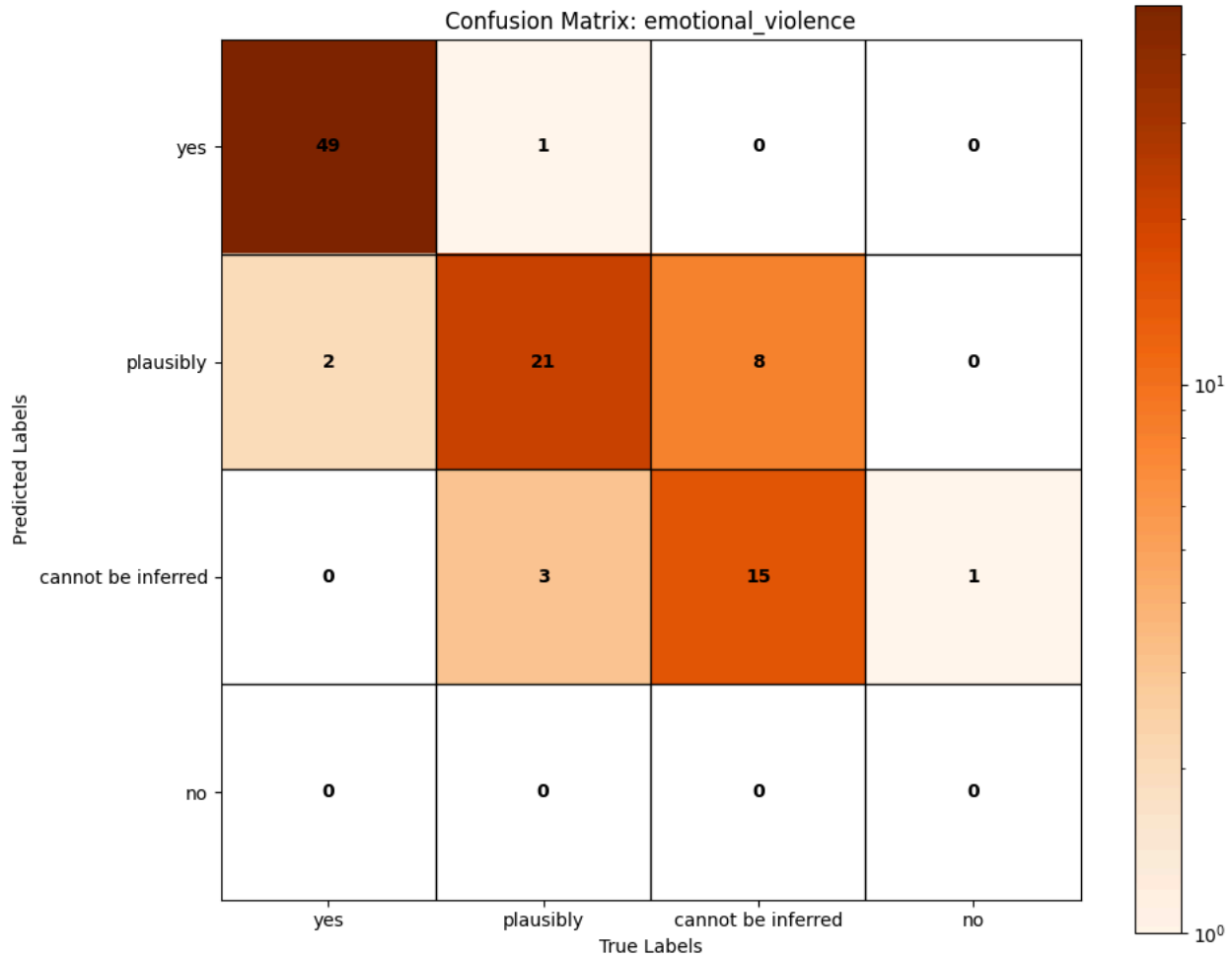

```
F1 Score (Weighted) for Overall: 0.94
F1 Score (Macro) for Overall: 0.83
```




```
F1 Score (Weighted) for mental_condition: 0.94
F1 Score (Macro) for mental_condition: 0.91
```




```
F1 Score (Weighted) for jealousy: 0.91
F1 Score (Macro) for jealousy: 0.55
```




```
F1 Score (Weighted) for emotional_violence: 0.85
F1 Score (Macro) for emotional_violence: 0.61
```

In the comparison, we evaluated two metrics: weighted F1 and unweighted F1. The blue graph shows a general trend, while the following graphs are examples where the metrics evaluate similarly or differently.

Part 3 - Data Preprocessing for models' prediction

- Comparing differences in evaluation metrics.
- Merging label categories into binary classifications.
- Selecting the most reliable risk factors for the following parts.

3.1 Comparison between weighted and unweighted F1

```
# Read the Excel file
df = pd.read_excel('Labeled.xlsx', sheet_name='Test')

# List to store the results
results = []

# Iterate over the column ranges from index 23 to 69
for j in range(47):
    column_name = df.columns[23 + j]
    if column_name in ['age_female', 'age_male', 'author_gender']:
        continue

    num_rows = 100
    Predicted_values = df.iloc[:num_rows][column_name].tolist()
    True_values = df.iloc[109:109 + num_rows][column_name].tolist()

    # Calculate the F1 scores
    f1_weighted = f1_score(True_values, Predicted_values, average='weighted')
    f1_macro = f1_score(True_values, Predicted_values, average='macro')

    # Append the results to the list
    results.append({
        'Risk Factor': column_name,
        'F1 Score - weighted': round(f1_weighted, 2),
        'F1 Score - macro': round(f1_macro, 2)
    })

# Create a DataFrame from the results
results_df = pd.DataFrame(results)

# Sort the DataFrame by F1 Score - weighted in descending order
results_df = results_df.sort_values(by='F1 Score - weighted', ascending=False)

# Split the DataFrame into two parts
midpoint = len(results_df) // 2
df1 = results_df.iloc[:midpoint]
df2 = results_df.iloc[midpoint:]

# Display the two tables side by side with tabulate
print("Table 1:")
print(tabulate(df1, headers='keys', tablefmt='pretty', showindex=False))
print("\nTable 2:")
print(tabulate(df2, headers='keys', tablefmt='pretty', showindex=False))
```

```
Table 1:
+--------------------------------+---------------------+------------------+
|          Risk Factor           | F1 Score - weighted | F1 Score - macro |
+--------------------------------+---------------------+------------------+
|         substance_use          |         1.0         |       1.0        |
|       access_to_weapons        |         1.0         |       1.0        |
|      prevention_of_care        |        0.99         |       0.5        |
|        hard_childhood          |        0.99         |       0.9        |
|             ptsd               |        0.99         |       0.82       |
|     emotional_dependency       |        0.98         |       0.65       |
|          author_role           |        0.98         |       0.9        |
|         unemployment           |        0.98         |       0.86       |
|        property_damage         |        0.98         |       0.91       |
|       economic_violence        |        0.97         |       0.83       |
|        signs_of_injury         |        0.97         |       0.94       |
|       refuses_treatment        |        0.97         |       0.62       |
|       social_isolation         |        0.97         |       0.96       |
|        sexual_violence         |        0.97         |       0.93       |
|       physical_violence        |        0.97         |       0.89       |
| presence_of_others_in_assault  |        0.96         |       0.88       |
|    daily_activity_control      |        0.95         |       0.88       |
|      spiritual_violence        |        0.94         |       0.25       |
```

```
|       physical_threats        |        0.94         |       0.88       |
|       suicidal_threats        |        0.94         |       0.78       |
|       mental_condition        |        0.94         |       0.91       |
|          gaslighting          |        0.94         |       0.93       |
+-------------------------------+---------------------+------------------+

Table 2:
+-------------------------------+---------------------+------------------+
|          Risk Factor          | F1 Score - weighted | F1 Score - macro |
+-------------------------------+---------------------+------------------+
| refusal_to_end_relationship   |        0.94         |       0.88       |
|  public_private_discrepancy   |        0.94         |       0.91       |
|           children            |        0.94         |       0.91       |
|   offender_support_network    |        0.93         |       0.85       |
|          outbursts            |        0.93         |       0.91       |
|     abusive_relationship      |        0.93         |       0.58       |
| attempts_to_end_relationship  |        0.92         |       0.86       |
|     aggressive_behavior       |        0.92         |       0.8        |
|     relationship_status       |        0.92         |       0.73       |
|           jealousy            |        0.91         |       0.55       |
|    victim_support_network     |        0.91         |       0.83       |
|         living_with           |        0.91         |       0.91       |
|         was_breakup           |        0.91         |       0.61       |
|     narcissistic_traits       |        0.91         |       0.88       |
|         humiliation           |         0.9         |       0.82       |
|        obsessiveness          |         0.9         |       0.7        |
|       violent_behavior        |        0.89         |       0.76       |
|        past_offenses          |        0.89         |       0.81       |
|   fear_based_relationship     |        0.86         |       0.79       |
|      relationship_type        |        0.86         |       0.6        |
|      emotional_violence       |        0.85         |       0.61       |
|         when_breakup          |        0.78         |       0.49       |
+-------------------------------+---------------------+------------------+
```

3.2 Comparison of unweighted F1 (macro) before and after merging labels

```
# Read the Excel file
df = pd.read_excel('Labeled.xlsx', sheet_name='Test')

# First table: F1 Score - weighted and F1 Score - macro
results = []

# Iterate over the column ranges from index 23 to 69
for j in range(47):
    column_name = df.columns[23 + j]
    if column_name in ['age_female', 'age_male', 'author_gender']:
        continue

    num_rows = 100
    Predicted_values = df.iloc[:num_rows][column_name].tolist()
    True_values = df.iloc[109:109 + num_rows][column_name].tolist()

    # Calculate the F1 scores
    f1_weighted = f1_score(True_values, Predicted_values, average='weighted')
    f1_macro = f1_score(True_values, Predicted_values, average='macro')

    # Append the results to the list
    results.append({
        'Risk Factor': column_name,
        'F1 Score - weighted': round(f1_weighted, 2),
        'F1 Score - macro old': round(f1_macro, 2)
    })

# Create a DataFrame from the results
results_df = pd.DataFrame(results)

# Second table: F1 Score - macro after replacing labels
columns_range = df.columns[34:70]
num_rows_replace = 211
df.loc[:num_rows_replace-1, columns_range] = df.loc[:num_rows_replace-1, columns_range].replace({'no': 'cannot be inferred', 'yes': 'plau

new_results = []

# Recalculate F1 scores after label replacement
for j in range(47):
    column_name = df.columns[23 + j]
    if column_name in ['age_female', 'age_male', 'author_gender']:
        continue

    num_rows = 100
    Predicted_values_new = df.iloc[:num_rows][column_name].tolist()
    True_values_new = df.iloc[109:109 + num_rows][column_name].tolist()

    # Calculate the new F1 score - macro
```

```
        f1_macro_new = f1_score(True_values_new, Predicted_values_new, average='macro')

        # Append the updated results to the list
        new_results.append({
            'Risk Factor': column_name,
            'F1 Score - macro new': round(f1_macro_new, 2)
        })

    # Create DataFrame from the new results
    results_new_df = pd.DataFrame(new_results)

    # Merge the old and new DataFrames on 'Risk Factor'
    final_df = pd.merge(results_df[['Risk Factor', 'F1 Score - macro old']], results_new_df, on='Risk Factor')

    # Rename the columns
    final_df.rename(columns={
        'F1 Score - macro old': 'F1 - old',
        'F1 Score - macro new': 'F1 - new'
    }, inplace=True)

    # Calculate the difference with two decimal places
    final_df['Difference'] = (final_df['F1 - new'] - final_df['F1 - old']).round(2)

    # Sort the final DataFrame by 'F1 - new'
    final_df = final_df.sort_values(by='F1 - new', ascending=False)

    # Reorder columns: F1 - new first, then F1 - old, then Difference
    final_df = final_df[['Risk Factor', 'F1 - new', 'F1 - old', 'Difference']]

    # Split the DataFrame into two parts
    midpoint = len(final_df) // 2
    df1 = final_df.iloc[:midpoint].reset_index(drop=True)
    df2 = final_df.iloc[midpoint:].reset_index(drop=True)

    # Display the two tables
    print("Table 1:")
    print(tabulate(df1, headers='keys', tablefmt='pretty', showindex=False))
    print("\nTable 2:")
    print(tabulate(df2, headers='keys', tablefmt='pretty', showindex=False))
```

```
Table 1:
+------------------------------+----------+----------+------------+
|          Risk Factor         | F1 - new | F1 - old | Difference |
+------------------------------+----------+----------+------------+
|             ptsd             |   1.0    |   0.82   |    0.18    |
|         substance_use        |   1.0    |   1.0    |    0.0     |
|       access_to_weapons      |   1.0    |   1.0    |    0.0     |
|       economic_violence      |   1.0    |   0.83   |    0.17    |
|        signs_of_injury       |   0.97   |   0.94   |    0.03    |
|     emotional_dependency     |   0.97   |   0.65   |    0.32    |
|       physical_violence      |   0.96   |   0.89   |    0.07    |
|        property_damage       |   0.96   |   0.91   |    0.05    |
|       social_isolation       |   0.96   |   0.96   |    0.0     |
|      aggressive_behavior     |   0.96   |   0.8    |    0.16    |
| attempts_to_end_relationship |   0.95   |   0.86   |    0.09    |
|          gaslighting         |   0.95   |   0.93   |    0.02    |
|        sexual_violence       |   0.94   |   0.93   |    0.01    |
|       mental_condition       |   0.94   |   0.91   |    0.03    |
|    daily_activity_control    |   0.94   |   0.88   |    0.06    |
|  refusal_to_end_relationship |   0.93   |   0.88   |    0.05    |
|  public_private_discrepancy  |   0.93   |   0.91   |    0.02    |
| presence_of_others_in_assault |   0.93   |   0.88   |    0.05    |
|          outbursts           |   0.93   |   0.91   |    0.02    |
|      narcissistic_traits     |   0.92   |   0.88   |    0.04    |
|          humiliation         |   0.92   |   0.82   |    0.1     |
|          living_with         |   0.91   |   0.91   |    0.0     |
+------------------------------+----------+----------+------------+

Table 2:
+-------------------------+----------+----------+------------+
|       Risk Factor       | F1 - new | F1 - old | Difference |
+-------------------------+----------+----------+------------+
|         children        |   0.91   |   0.91   |    0.0     |
|     physical_threats    |   0.91   |   0.88   |    0.03    |
|  victim_support_network |   0.9    |   0.83   |    0.07    |
| offender_support_network |   0.9    |   0.85   |    0.05    |
|      obsessiveness      |   0.9    |   0.7    |    0.2     |
|      hard_childhood     |   0.9    |   0.9    |    0.0     |
|       unemployment      |   0.9    |   0.86   |    0.04    |
|       author_role       |   0.9    |   0.9    |    0.0     |
| fear_based_relationship |   0.86   |   0.79   |    0.07    |
|      past_offenses      |   0.86   |   0.81   |    0.05    |
|    emotional_violence   |   0.84   |   0.61   |    0.23    |
|         jealousy        |   0.84   |   0.55   |    0.29    |
|     violent_behavior    |   0.81   |   0.76   |    0.05    |
```

```
|     suicidal_threats     |   0.79   |   0.78   |    0.01    |
|    spiritual_violence    |   0.74   |   0.25   |    0.49    |
|   relationship_status    |   0.73   |   0.73   |    0.0     |
|    refuses_treatment     |   0.69   |   0.62   |    0.07    |
|       was_breakup        |   0.61   |   0.61   |    0.0     |
|    relationship_type     |   0.6    |   0.6    |    0.0     |
|   abusive_relationship   |   0.58   |   0.58   |    0.0     |
|    prevention_of_care    |   0.5    |   0.5    |    0.0     |
|       when_breakup       |   0.49   |   0.49   |    0.0     |
+--------------------------+----------+----------+------------+
```

3.3 The final matric display - F1 macro after marging labels

```
# Load the Excel file
df = pd.read_excel('Labeled.xlsx', sheet_name='Test', engine='openpyxl')

columns_range = df.columns[34:70]
# Extract the first row values from the specified columns
num_rows = 211

# Replace 'no' with 'cannot be inferred' and 'yes' with 'plausibly' in the specified columns and rows
df.loc[:num_rows-1, columns_range] = df.loc[:num_rows-1, columns_range].replace({'no': 0, 'cannot be inferred': 0,'irrelevant':0, 'yes'
results = []
num_rows = 100
# Iterate over specified columns
for j in range(47):
    column_name = df.columns[23 + j]
    if column_name not in ['age_female', 'age_male', 'author_gender']:
        Predicted_values = df.iloc[:num_rows][column_name].tolist()
        True_values = df.iloc[109:109 + num_rows][column_name].tolist()
        results.append({
            'Risk Factor': column_name,
            'F1 Score - macro': round(f1_score(True_values, Predicted_values, average='macro'), 2),
        })
results_df = pd.DataFrame(results).sort_values(by='F1 Score - macro', ascending=False)

print(tabulate(results_df, headers='keys', tablefmt='pretty', showindex=False))
```

```
+--------------------------------+------------------+
|          Risk Factor           | F1 Score - macro |
+--------------------------------+------------------+
|              ptsd              |       1.0        |
|         substance_use          |       1.0        |
|       access_to_weapons        |       1.0        |
|       economic_violence        |       1.0        |
|        signs_of_injury         |       0.97       |
|     emotional_dependency       |       0.97       |
|       physical_violence        |       0.96       |
|        property_damage         |       0.96       |
|       social_isolation         |       0.96       |
|      aggressive_behavior       |       0.96       |
| attempts_to_end_relationship   |       0.95       |
|          gaslighting           |       0.95       |
|        sexual_violence         |       0.94       |
|       mental_condition         |       0.94       |
|    daily_activity_control      |       0.94       |
|  refusal_to_end_relationship   |       0.93       |
|  public_private_discrepancy    |       0.93       |
| presence_of_others_in_assault  |       0.93       |
|           outbursts            |       0.93       |
|      narcissistic_traits       |       0.92       |
|          humiliation           |       0.92       |
|          living_with           |       0.91       |
|           children             |       0.91       |
|       physical_threats         |       0.91       |
|    victim_support_network      |       0.9        |
|   offender_support_network     |       0.9        |
|         obsessiveness          |       0.9        |
|        hard_childhood          |       0.9        |
|         unemployment           |       0.9        |
|          author_role           |       0.9        |
|    fear_based_relationship     |       0.86       |
|         past_offenses          |       0.86       |
|      emotional_violence        |       0.84       |
|           jealousy             |       0.84       |
|       violent_behavior         |       0.81       |
|       suicidal_threats         |       0.79       |
|      spiritual_violence        |       0.74       |
|      relationship_status       |       0.73       |
|       refuses_treatment        |       0.69       |
|          was_breakup           |       0.61       |
|       relationship_type        |       0.6        |
|     abusive_relationship       |       0.58       |
```

```
|      prevention_of_care       |       0.5        |
|         when_breakup          |       0.49       |
+-------------------------------+------------------+
```

3.4 Keep the reliable risk factors (>0.9) for the models

```
# Filter for rows with F1 Score - macro >= 0.9
high_f1_scores = results_df[results_df['F1 Score - macro'] >= 0.9]

# Exclude the 'author_role' column
high_f1_scores_filtered = high_f1_scores[high_f1_scores['Risk Factor'] != 'author_role']

# Create a list of risk factors with F1 scores above 0.9, excluding 'author_role'
high_f1_risk_factors_list = high_f1_scores_filtered['Risk Factor'].tolist()

# Output the list
high_f1_risk_factors_list
```

```
['ptsd',
 'substance_use',
 'access_to_weapons',
 'economic_violence',
 'signs_of_injury',
 'emotional_dependency',
 'physical_violence',
 'property_damage',
 'social_isolation',
 'aggressive_behavior',
 'attempts_to_end_relationship',
 'gaslighting',
 'sexual_violence',
 'mental_condition',
 'daily_activity_control',
 'refusal_to_end_relationship',
 'public_private_discrepancy',
 'presence_of_others_in_assault',
 'outbursts',
 'narcissistic_traits',
 'humiliation',
 'living_with',
 'children',
 'physical_threats',
 'victim_support_network',
 'offender_support_network',
 'obsessiveness',
 'hard_childhood',
 'unemployment']
```

Corr of selected columns with new labels

Part 4 - Baselines

We predicted the risk factor: physical violence.

Goals:

- Establish a baseline for score.
- Data Insights: Identify key features and important words.

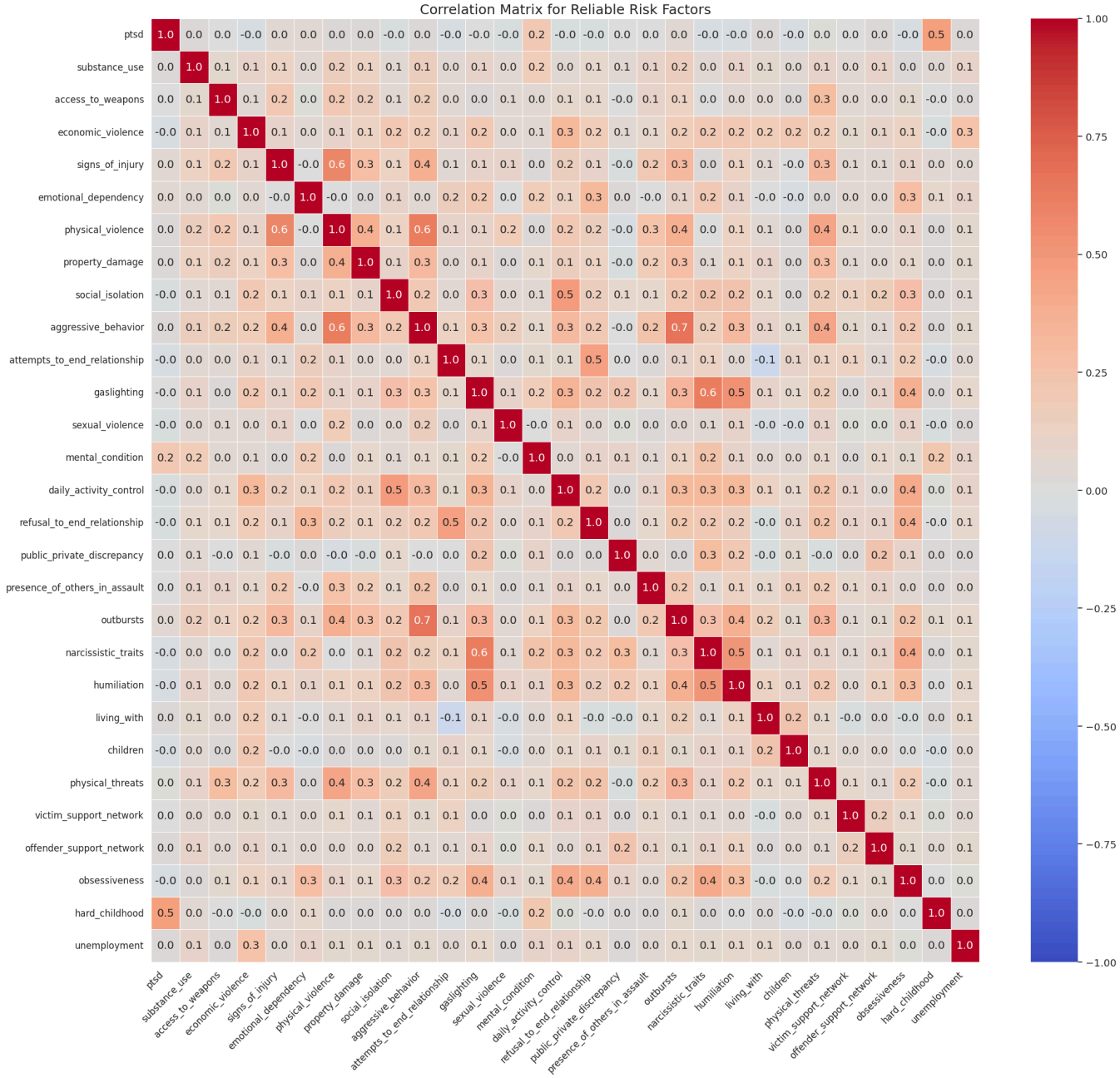
4.1 Logistic regression

Calculated correlation

```
# Load the Excel file
df_all = pd.read_excel('Labeled.xlsx', sheet_name='All', engine='openpyxl')

# Extract the columns corresponding to the top column names and merge labels
df_top_columns = df_all[high_f1_risk_factors_list].replace({'no': 0, 'cannot be inferred': 0,'irrelevant':0, 'yes': 1, 'plausibly': 1})
# Create a correlation matrix between these columns
correlation_matrix = df_top_columns.corr()
# Set up the matplotlib figure with a larger size and adjust font size
plt.figure(figsize=(22, 20))
sns.set(font_scale=1.2)
# Draw the heatmap without the mask and correct aspect ratio
heatmap = sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1, center=0, fmt='.1f', linewidths=0.5)
# Rotate the x and y axis labels for better readability
plt.xticks(rotation=45, ha='right', fontsize=12)
```

```
plt.yticks(rotation=0, fontsize=12)

# Add a title to the heatmap
plt.title('Correlation Matrix for Reliable Risk Factors', fontsize=18)

# Adjust the layout to fit everything within the figure area
plt.tight_layout()

# Save the plot as an image and display it
plt.savefig('correlation_matrix.png')
plt.show()
```

```
<ipython-input-13-15ff64ce499f>:5: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future ve
  df_top_columns = df_all[high_f1_risk_factors_list].replace({'no': 0, 'cannot be inferred': 0,'irrelevant':0, 'yes': 1, 'plausibly
```



```
# Load the Excel file
df_all = pd.read_excel('Labeled.xlsx', sheet_name='All', engine='openpyxl')

# Replace specific values in the entire DataFrame
replacement_dict = {
    'no': 0,
    'cannot be inferred': 0,
    'irrelevant': 0,
    'yes': 1,
    'plausibly': 1
}
df_all = df_all[high_f1_risk_factors_list].replace(replacement_dict)

# List of target columns to predict
#Target_to_predict = ['physical_violence', 'aggressive_behavior', 'gaslighting', 'narcissistic_traits']
Target_to_predict = ['physical_violence']

# Iterate over each target column
for target_column in Target_to_predict:
    print(f"\nThe target risk factor is: {target_column}")

    # Separate features and target variable
    X = df_all.drop(columns=[target_column])
    y = df_all[target_column]

    # Ensure the target variable is binary
    if y.nunique() != 2:
        print(f"Skipping '{target_column}' as it is not binary.")
        continue

    # Add a constant to the features (intercept)
    X = sm.add_constant(X)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # Initialize the logistic regression model using statsmodels
    model = sm.Logit(y_train, X_train)

    # Fit the model
    result = model.fit()

    # Print the summary which includes coefficients, z-values, and p-values
    print(result.summary2())

    # Remove columns with p-value > 0.05
    significant_columns = result.pvalues[result.pvalues <= 0.05].index.tolist()
    if 'const' in significant_columns:
        significant_columns.remove('const')

    # Print the significant columns
    print(f"Significant columns for {target_column}: {significant_columns}")

    # Recreate the feature set with only significant columns
    X_significant = X[significant_columns]
    X_significant = sm.add_constant(X_significant)
```

```
# Split the data again with significant columns
X_train, X_test, y_train, y_test = train_test_split(X_significant, y, test_size=0.3, random_state=42)

# Initialize the logistic regression model using statsmodels with significant columns
model_significant = sm.Logit(y_train, X_train)

# Fit the model with significant columns
result_significant = model_significant.fit()

# Print the summary for the model with significant columns
print(result_significant.summary2())

# Make predictions on the test set with significant columns
y_pred_significant = result_significant.predict(X_test)
y_pred_binary_significant = [1 if x > 0.5 else 0 for x in y_pred_significant]

# Evaluate the model with significant columns
accuracy_significant = accuracy_score(y_test, y_pred_binary_significant)
report_significant = classification_report(y_test, y_pred_binary_significant, target_names=['0', '1'])
print(f"\n The target risk factor is: {target_column}")
print(report_significant)
print(f"Accuracy with significant columns: {accuracy_significant:.2f}")
print("\n" + "="*60 + "\n")
```

```
mental_condition              -0.0937   0.0981  -0.9551 0.3395 -0.2861  0.0986
daily_activity_control         0.3876   0.1116   3.4715 0.0005  0.1688  0.6064
refusal_to_end_relationship   -0.1196   0.1121  -1.0674 0.2858 -0.3393  0.1000
public_private_discrepancy     0.0097   0.0988   0.0981 0.9219 -0.1839  0.2033
presence_of_others_in_assault  0.8079   0.1575   5.1279 0.0000  0.4991  1.1167
outbursts                     -0.7416   0.1311  -5.6572 0.0000 -0.9985 -0.4847
narcissistic_traits           -0.8284   0.1374  -6.0299 0.0000 -1.0976 -0.5591
humiliation                   -0.3642   0.1229  -2.9638 0.0030 -0.6050 -0.1234
living_with                   -0.2253   0.1094  -2.0598 0.0394 -0.4397 -0.0109
children                       0.0046   0.1063   0.0435 0.9653 -0.2037  0.2129
physical_threats               0.9771   0.1052   9.2878 0.0000  0.7709  1.1833
victim_support_network         0.1181   0.0909   1.2989 0.1940 -0.0601  0.2962
offender_support_network      -0.2299   0.1088  -2.1139 0.0345 -0.4431 -0.0167
obsessiveness                 -0.0844   0.1069  -0.7891 0.4300 -0.2939  0.1252
hard_childhood                 0.3299   0.2246   1.4685 0.1420 -0.1104  0.7702
unemployment                   0.2677   0.2120   1.2623 0.2068 -0.1479  0.6832
================================================================================

Significant columns for physical_violence: ['substance_use', 'access_to_weapons', 'signs_of_injury', 'property_damage', 'aggressi
Optimization terminated successfully.
         Current function value: 0.264570
         Iterations 8
                            Results: Logit
================================================================================
Model:                  Logit                Method:               MLE
Dependent Variable:     physical_violence    Pseudo R-squared:     0.572
Date:                   2024-12-01 20:35     AIC:                  3530.6753
No. Observations:       6612                 BIC:                  3639.4216
Df Model:               15                   Log-Likelihood:       -1749.3
Df Residuals:           6596                 LL-Null:              -4088.1
Converged:              1.0000               LLR p-value:          0.0000
No. Iterations:         8.0000               Scale:                1.0000
--------------------------------------------------------------------------------
                            Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
const                      -3.5620   0.1384 -25.7365 0.0000 -3.8332 -3.2907
substance_use               0.4126   0.1231   3.3529 0.0008  0.1714  0.6538
access_to_weapons           0.7638   0.2550   2.9956 0.0027  0.2641  1.2636
signs_of_injury             4.0842   0.2185  18.6897 0.0000  3.6559  4.5125
property_damage             1.6642   0.1455  11.4415 0.0000  1.3792  1.9493
aggressive_behavior         3.9821   0.1640  24.2806 0.0000  3.6607  4.3036
attempts_to_end_relationship   0.2508   0.0885   2.8330 0.0046  0.0773  0.4243
sexual_violence             0.8257   0.1446   5.7096 0.0000  0.5422  1.1091
daily_activity_control      0.2813   0.0962   2.9230 0.0035  0.0927  0.4699
presence_of_others_in_assault  0.8195   0.1541   5.3185 0.0000  0.5175  1.1215
outbursts                  -0.7429   0.1287  -5.7725 0.0000 -0.9952 -0.4907
narcissistic_traits        -0.8694   0.1132  -7.6807 0.0000 -1.0912 -0.6475
humiliation                -0.3483   0.1182  -2.9465 0.0032 -0.5800 -0.1166
living_with                -0.2327   0.1044  -2.2288 0.0258 -0.4373 -0.0281
physical_threats            0.9371   0.1023   9.1621 0.0000  0.7366  1.1376
offender_support_network   -0.2476   0.1042  -2.3769 0.0175 -0.4518 -0.0434
================================================================================


 The target risk factor is: physical_violence
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      1973
```

4.2 TF-IDF + SVM

```
# read the excel file
df = pd.read_excel('Labeled.xlsx', sheet_name='All', engine='openpyxl')

# keep only relevant columns (columns that are interesting to predict and Yoni got high f1 score on them)
feature_column = ['post_body']
abusive_types_columns = ['physical_violence','sexual_violence']
risk_factors_columns = ['social_isolation', 'gaslighting', 'mental_condition', 'daily_activity_control', 'aggressive_behavior', 'narciss

# labels_columns = ['abusive_relationship','emotional_violence',
#                     'physical_violence','sexual_violence','economic_violence']
df = df[feature_column + abusive_types_columns + risk_factors_columns]

# re-label the data and unite labels - ('no' + 'cannot be inferred' + 'irrelevant')  and  ('yes' + 'plausibly')
converts_dict = {
    'no': 0,
    'cannot be inferred': 0,
    'irrelevant': 0,
    'yes': 1,
    'plausibly': 1
}

df = df.replace(converts_dict)

# preprocess the 'post_body' column - convert to TF-IDF vector representation
df['post_body'] = df['post_body'].str.lower()

# Calculate Word Frequencies
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['post_body'])

# Sum up the counts of each vocabulary word
word_counts = X.toarray().sum(axis=0)
vocab = vectorizer.get_feature_names_out()

# Count the number of unique words
vocabulary_size = len(vocab)
print(f'There are {vocabulary_size} different words in the vocabulary')

# Create a dictionary with word frequencies
word_freq = dict(zip(vocab, word_counts))

# Filter Words by Frequency Threshold=200
frequency_threshold = 200
filtered_words = {word: freq for word, freq in word_freq.items() if freq >= frequency_threshold}

print(len(filtered_words))

# Set max_features Based on Filtered Words
max_features = len(filtered_words)
```

```
<ipython-input-15-a1fa6e6ad570>:22: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future v
  df = df.replace(converts_dict)
There are 29791 different words in the vocabulary
1191
```

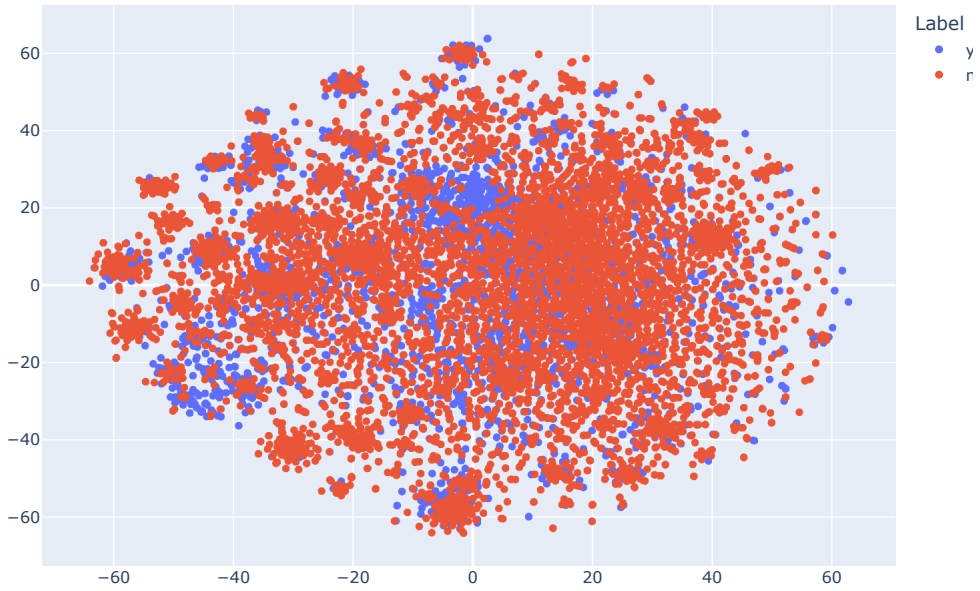
```
# max_features is the embedding dim
vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
post_body_vectors = vectorizer.fit_transform(df['post_body'])


X_train_dict = {}
X_test_dict = {}
y_train_dict = {}
y_test_dict = {}

# Perform stratified train-test split for each label
labels_columns = abusive_types_columns + risk_factors_columns
for label in labels_columns:
    sss = StratifiedShuffleSplit(n_splits=1, test_size=0.3, random_state=42)
    train_indices, test_indices = next(sss.split(post_body_vectors, df[label]))
    X_train_dict[label], X_test_dict[label] = post_body_vectors[train_indices], post_body_vectors[test_indices]
    y_train_dict[label], y_test_dict[label] = df[label].iloc[train_indices].values, df[label].iloc[test_indices].values
```

Plotting the data distribution in a 2D space

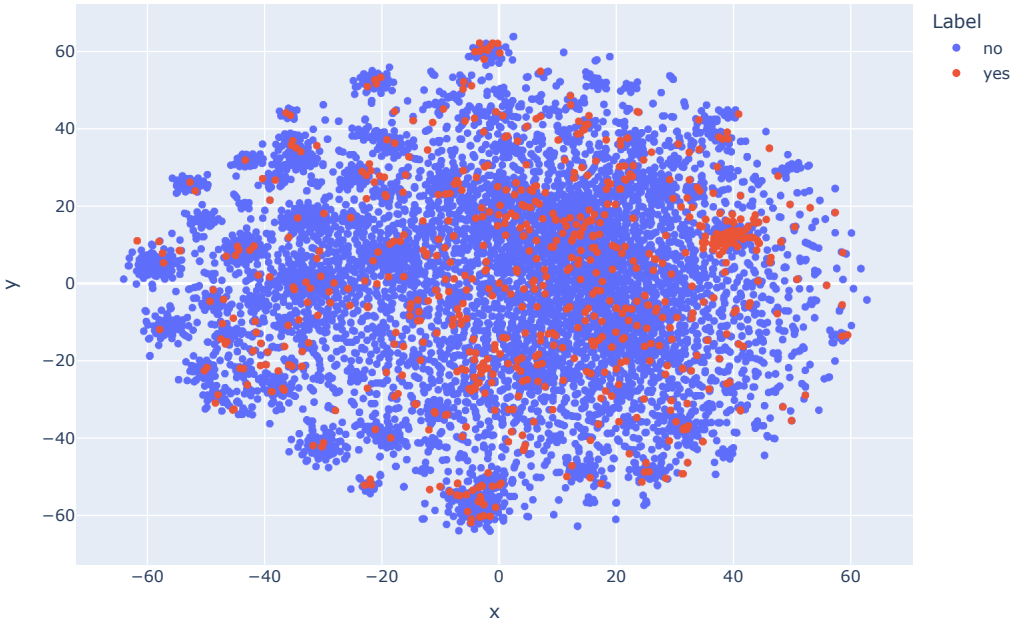
```
# perform t-SNE to reduce to 2D
tsne = TSNE(n_components=2, random_state=42)
reduced_vectors = tsne.fit_transform(post_body_vectors.toarray())

plot_df = pd.DataFrame({
    'x': reduced_vectors[:, 0],
    'y': reduced_vectors[:, 1],
})


for column in labels_columns:
    plot_df['label'] = df[column].map({1: 'yes', 0: 'no'})

    # plot the reduced vectors using Plotly
    fig = px.scatter(
      plot_df, x='x', y='y', color=plot_df['label'].astype(str),
      title=f'2D representation of TF-IDF vectors, classified by {column}',
      labels={'color': 'Label'},
      width=800, height=600
    )
    fig.show()
```

x

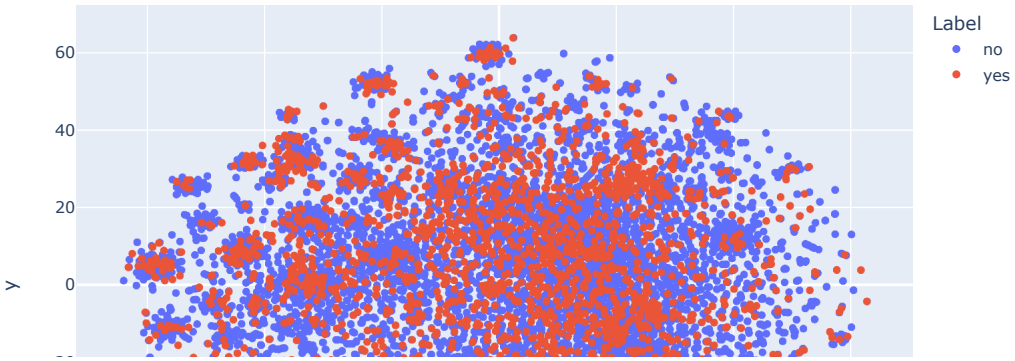
2D representation of TF-IDF vectors, classified by physical_violence

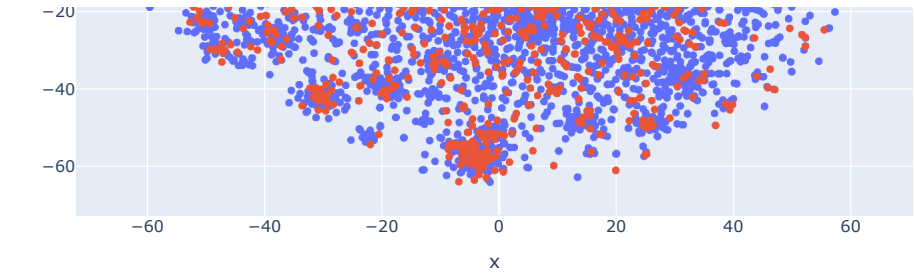


2D representation of TF-IDF vectors, classified by sexual_violence

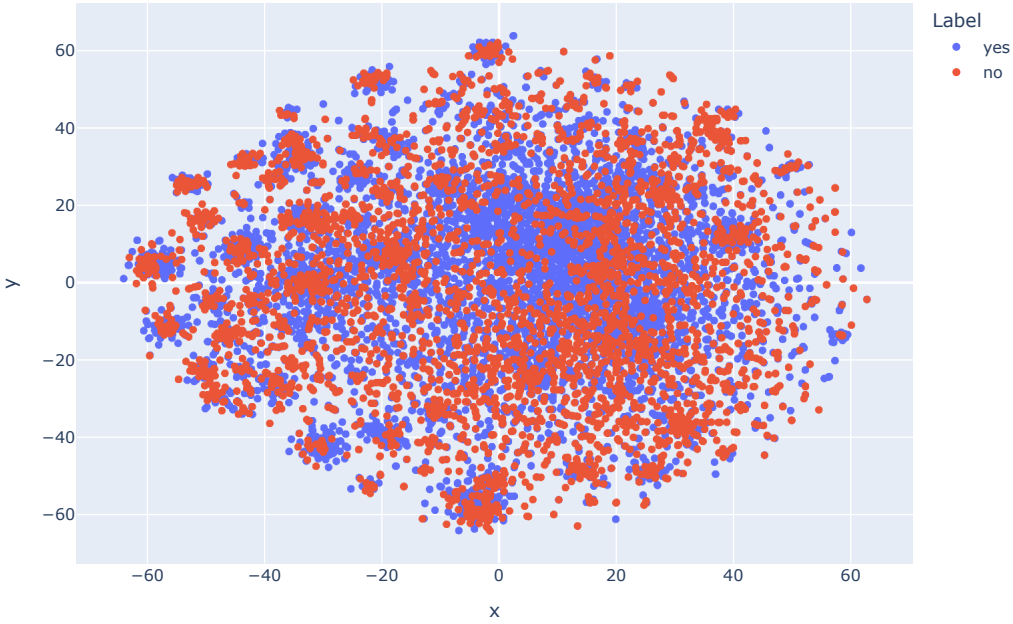


2D representation of TF-IDF vectors, classified by social_isolation

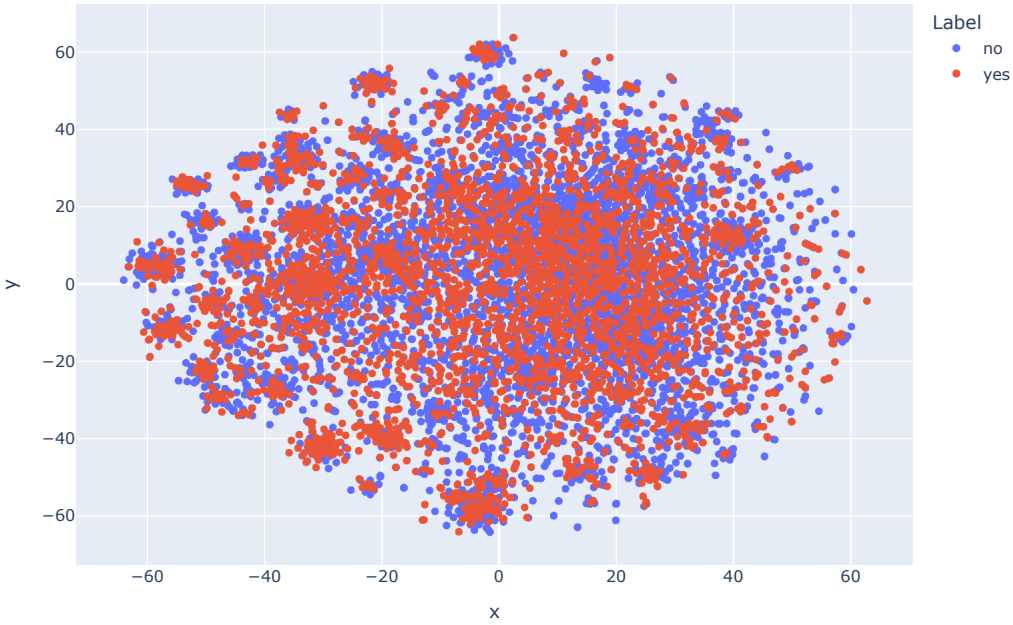




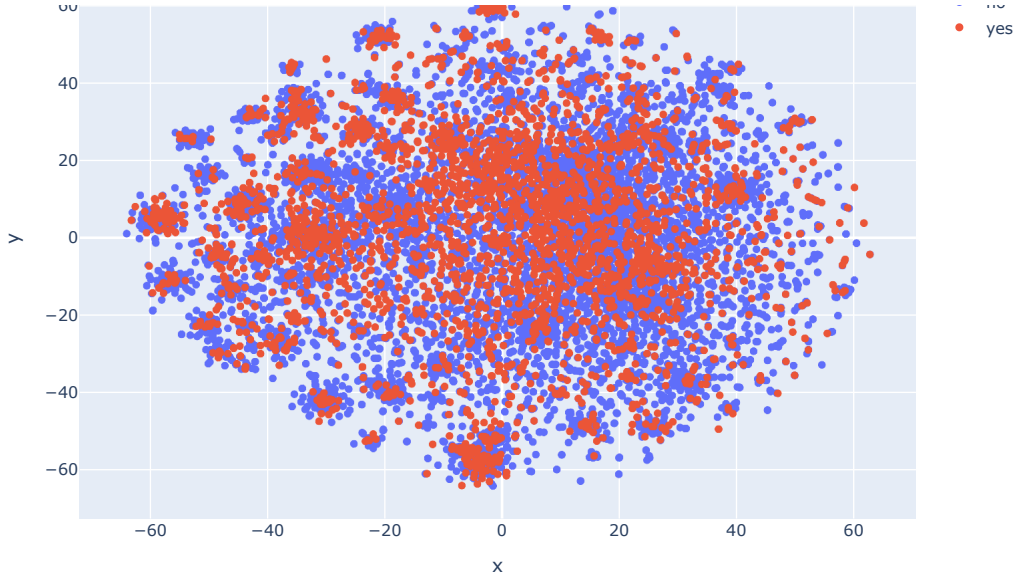
2D representation of TF-IDF vectors, classified by gaslighting



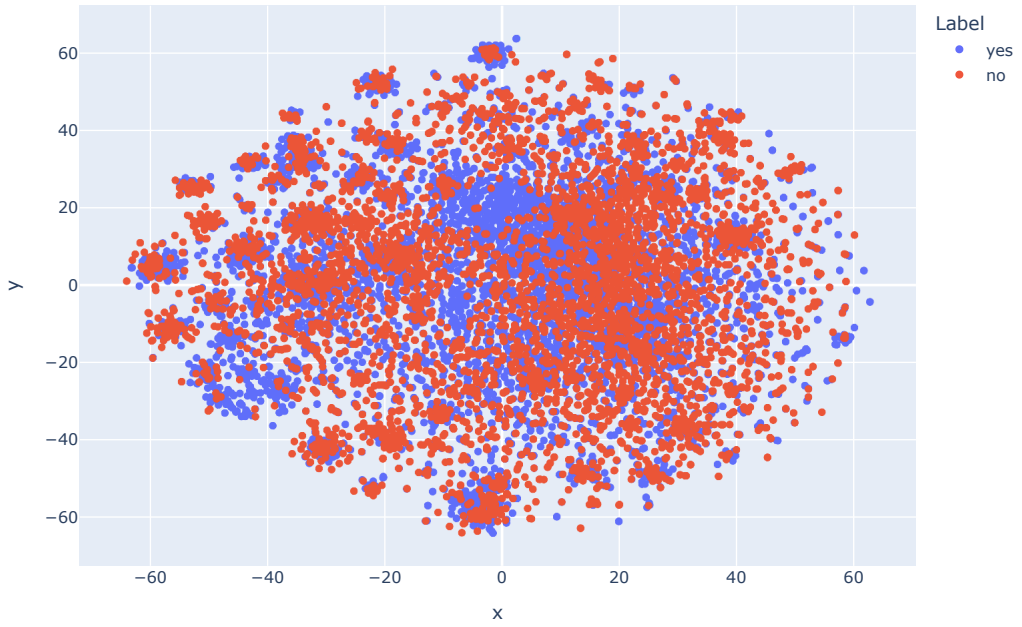
2D representation of TF-IDF vectors, classified by mental_condition



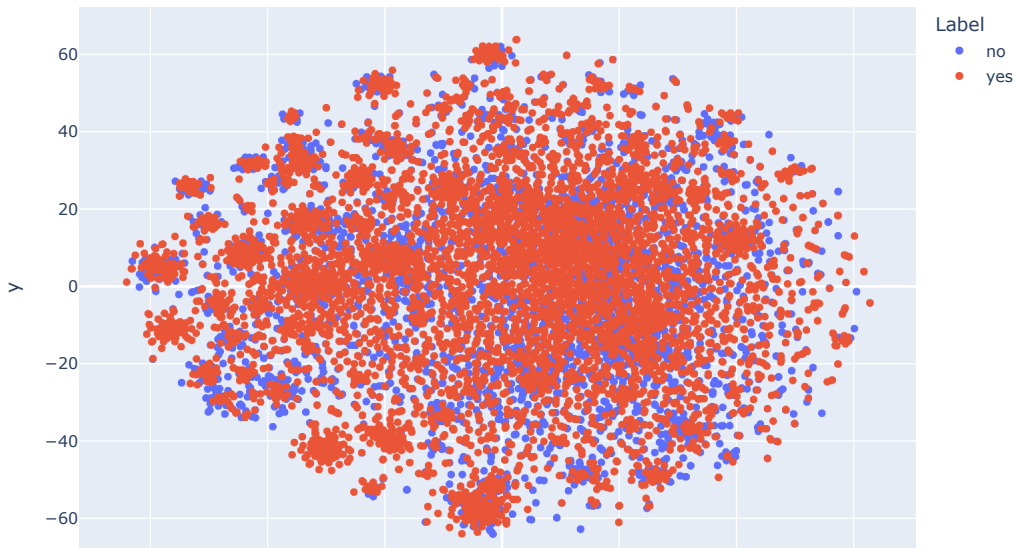
2D representation of TF-IDF vectors, classified by daily_activity_control

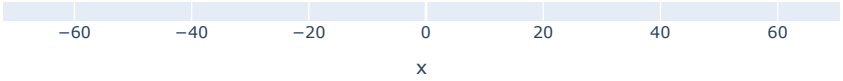


2D representation of TF-IDF vectors, classified by aggressive_behavior



2D representation of TF-IDF vectors, classified by narcissistic_traits




−60
−40
−20
0
20
40
60
x

Creating the SVM classifier

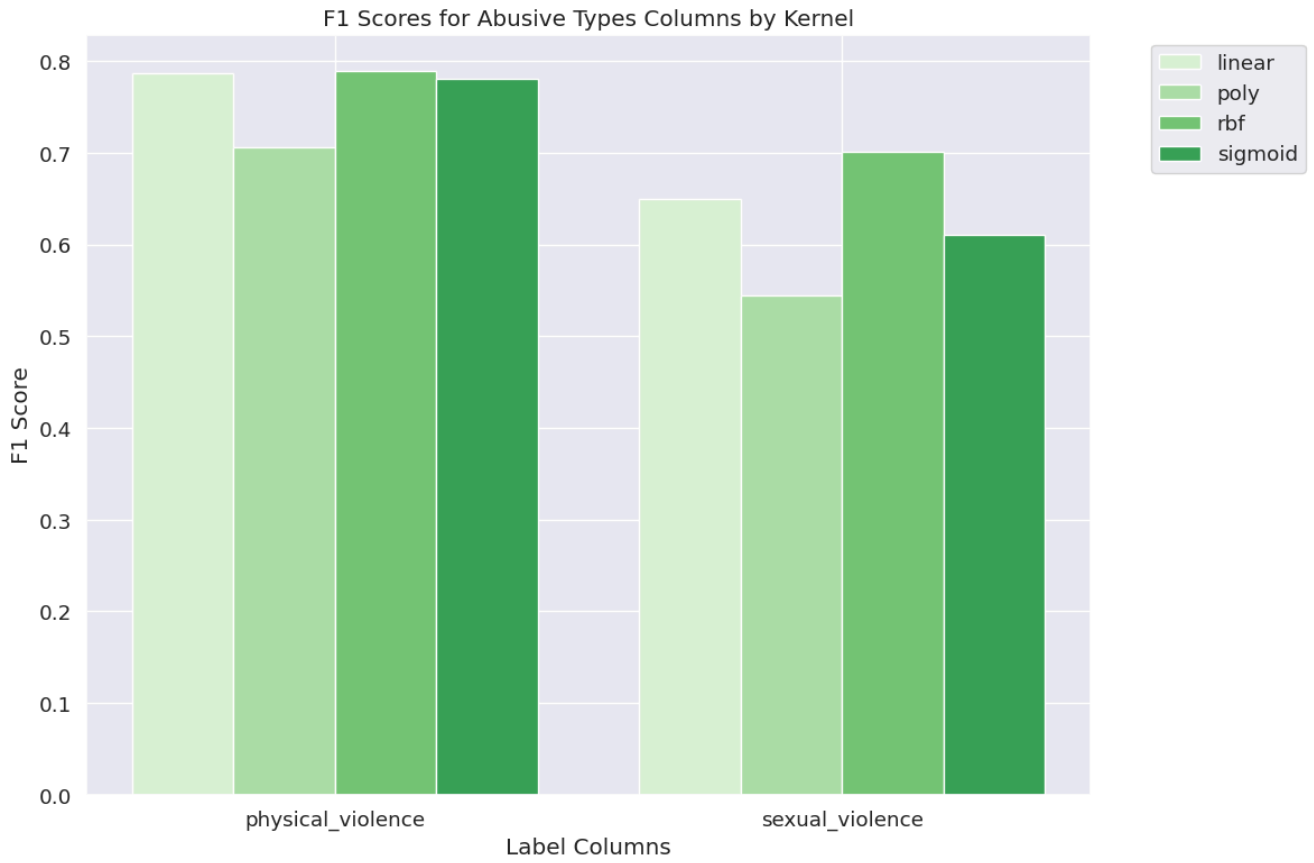
```
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
f1_per_kernel = {kernel: {} for kernel in kernels}

for kernel in kernels:
    svm_classifier = SVC(kernel=kernel, C=1.0, random_state=42, class_weight='balanced')

    f1_scores = {label: [] for label in labels_columns}

    for column in tqdm(labels_columns, desc=f"Processing label columns for {kernel} kernel"):
        svm_classifier.fit(X_train_dict[column], y_train_dict[column])
        y_pred = svm_classifier.predict(X_test_dict[column])
        f1 = round(f1_score(y_test_dict[column], y_pred, average='macro'), 4)
        f1_scores[column].append(f1)

    f1_per_kernel[kernel] = f1_scores
```

```
Processing label columns for linear kernel: 100%|████████| 8/8 [05:00<00:00, 37.57s/it]
Processing label columns for poly kernel: 100%|████████| 8/8 [08:04<00:00, 60.50s/it]
Processing label columns for rbf kernel: 100%|████████| 8/8 [06:33<00:00, 49.19s/it]
Processing label columns for sigmoid kernel: 100%|████████| 8/8 [05:11<00:00, 38.98s/it]
```

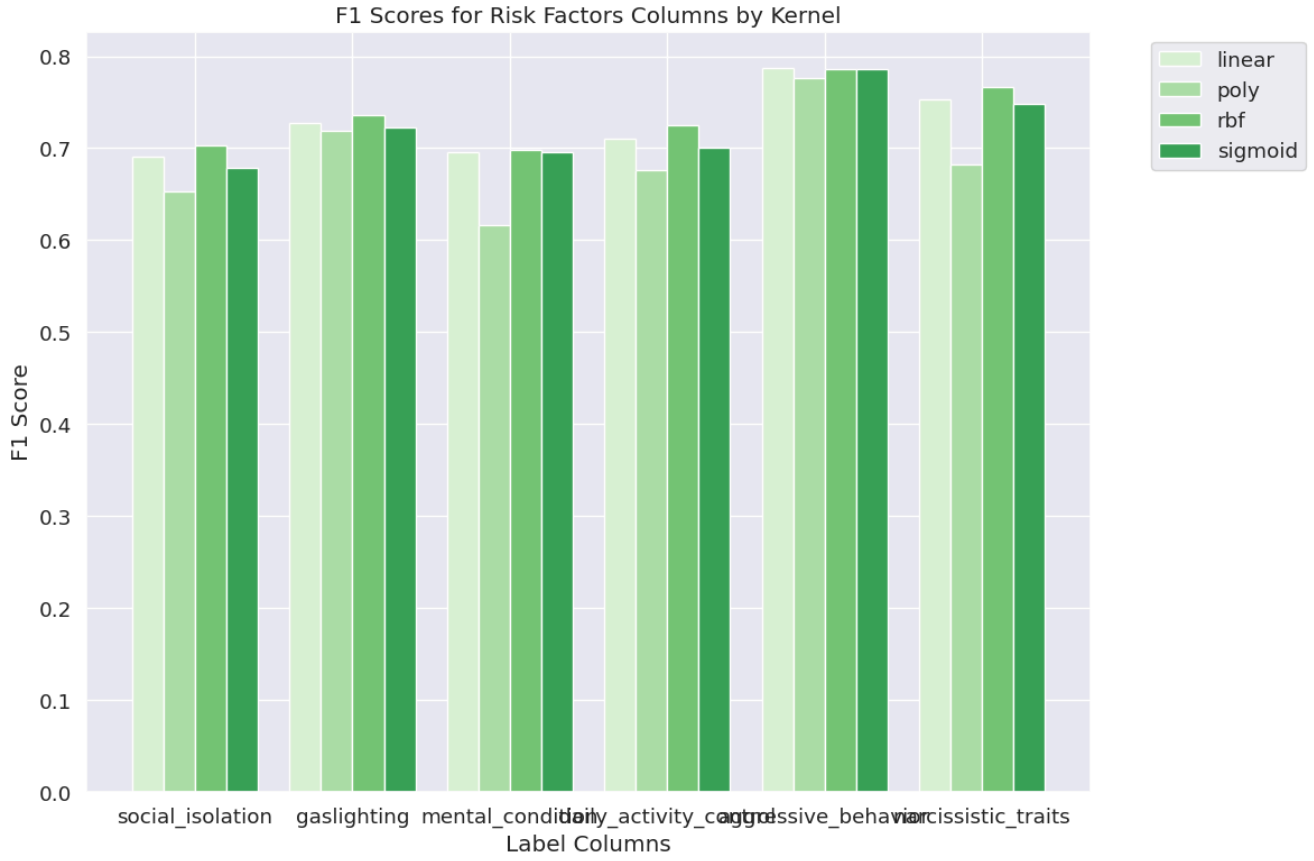
Adding plots for the different f1 scores per label and kernel

```
def plot_f1_scores(column_list, title):
    # Prepare data for plotting
    plot_data = {label: [f1_per_kernel[kernel][label][0] for kernel in kernels] for label in column_list}
    df_plot = pd.DataFrame(plot_data, index=kernels)

    # Plot the F1 scores
    fig, ax = plt.subplots(figsize=(12, 8))

    bar_width = 0.2
    index = np.arange(len(column_list))

    # Generate shades of green using a colormap
    colormap = cm.get_cmap('Greens', len(kernels) + 3)
    green_shades = [colormap(i + 1) for i in range(len(kernels))]

    for i, kernel in enumerate(kernels):
        ax.bar(index + i * bar_width, df_plot.loc[kernel], bar_width, color=green_shades[i], label=kernel)

    ax.set_xlabel('Label Columns')
    ax.set_ylabel('F1 Score')
    ax.set_title(title)
    ax.set_xticks(index + bar_width * (len(kernels) - 1) / 2)
    ax.set_xticklabels(column_list)

    # Position the legend outside the plot area
    ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

    plt.tight_layout()
    plt.show()

# Plot for abusive_types_columns
plot_f1_scores(abusive_types_columns, 'F1 Scores for Abusive Types Columns by Kernel')

# Plot for risk_factors_columns
plot_f1_scores(risk_factors_columns, 'F1 Scores for Risk Factors Columns by Kernel')
```

```
<ipython-input-20-5e55b2a864da>:13: MatplotlibDeprecationWarning:

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib.colormaps[
```




```
<ipython-input-20-5e55b2a864da>:13: MatplotlibDeprecationWarning:

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib.colormaps[
```



Getting TF-IDF statistics - the scores of the words

```
nltk.download('words')

# here I removed all the non-English words. This might harm our reliability, but it can make the results more interpretable
english_words = set(words.words())
```

```
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
```

```
# initial the possible labels again, we might want different features for this part
labels_columns = abusive_types_columns
terms_score_dict = {label: None for label in labels_columns}

vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)

for label in labels_columns:
    # keeping posts that are positive ('yes') for the specific label
    positive_on_label_posts = df['post_body'][df[label] == 1].str.lower()

    tfidf_matrix = vectorizer.fit_transform(positive_on_label_posts)
    terms = vectorizer.get_feature_names_out()

    term_scores = defaultdict(float)
    term_counts = defaultdict(int)

    for row in tfidf_matrix.toarray():
        for term_idx, score in enumerate(row):
            if score > 0:
                term = terms[term_idx]
                term_scores[term] += score
                term_counts[term] += 1

    mean_term_scores = {term: round(term_scores[term] / term_counts[term],3) for term in term_scores}

    # Filter out non-real words
    mean_term_scores = {term: score for term, score in mean_term_scores.items() if term in english_words}

    term_score_df = pd.DataFrame(list(mean_term_scores.items()), columns=['Term', 'Mean TF-IDF Score'])

    terms_score_dict[label] = term_score_df.sort_values(by='Mean TF-IDF Score', ascending=False)


for label in terms_score_dict:
    print(f"Top 10 terms for {label}:")
    print(terms_score_dict[label][['Term', 'Mean TF-IDF Score']].head(10))
    print()
```

```
Top 10 terms for physical_violence:
            Term  Mean TF-IDF Score
378         mum              0.207
182         son              0.183
404    daughter              0.178
254         cat              0.175
261        dont              0.168
669      lawyer              0.159
675    birthday              0.156
510  protection              0.153
123        baby              0.150
593         gun              0.148

Top 10 terms for sexual_violence:
            Term  Mean TF-IDF Score
656         dog              0.196
767      affair              0.194
259         mum              0.186
563         son              0.176
483     wedding              0.176
110        dont              0.174
606    daughter              0.172
760      suffer              0.169
718  disgusting              0.165
734       power              0.164
```

```
# extracting the most significant features from the tf-ids vectorizer
feature_names = vectorizer.get_feature_names_out()
idf_scores = vectorizer.idf_

idf_df = pd.DataFrame({'term': feature_names, 'score': idf_scores})

print(idf_df.sort_values(by='score', ascending=False).head(10))
```

```
             term     score
0              00  5.964242
318       fiancé  5.271095
964       whilst  5.088774
592          mum  4.934623
732  restraining  4.801091
660         porn  4.801091
307          fat  4.801091
735     romantic  4.740467
943        wanna  4.740467
677    proceeded  4.740467
```

Part 5 - LIME analysis

```
with zipfile.ZipFile("distilbert_model.zip", "r") as zip_ref:
    zip_ref.extractall("/content/")
```

0. Hyper-parameters and models

Here you need to select the model you want to use. Modify this cell before using the code.

```
# Initiating the device
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# Insert the path to the xlsx file and the name of the relevant sheet
file_path = '../content/Abusive Relationship Stories.xlsx'
sheet_name = 'Abusive Relationship Stories'

### If using RoBerta ###
##tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
##model = RobertaForSequenceClassification.from_pretrained('model', output_attentions=True, num_labels=1)

### If using DistilBERT ###
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained('distilbert_model',  output_attentions=True, num_labels=1)
```

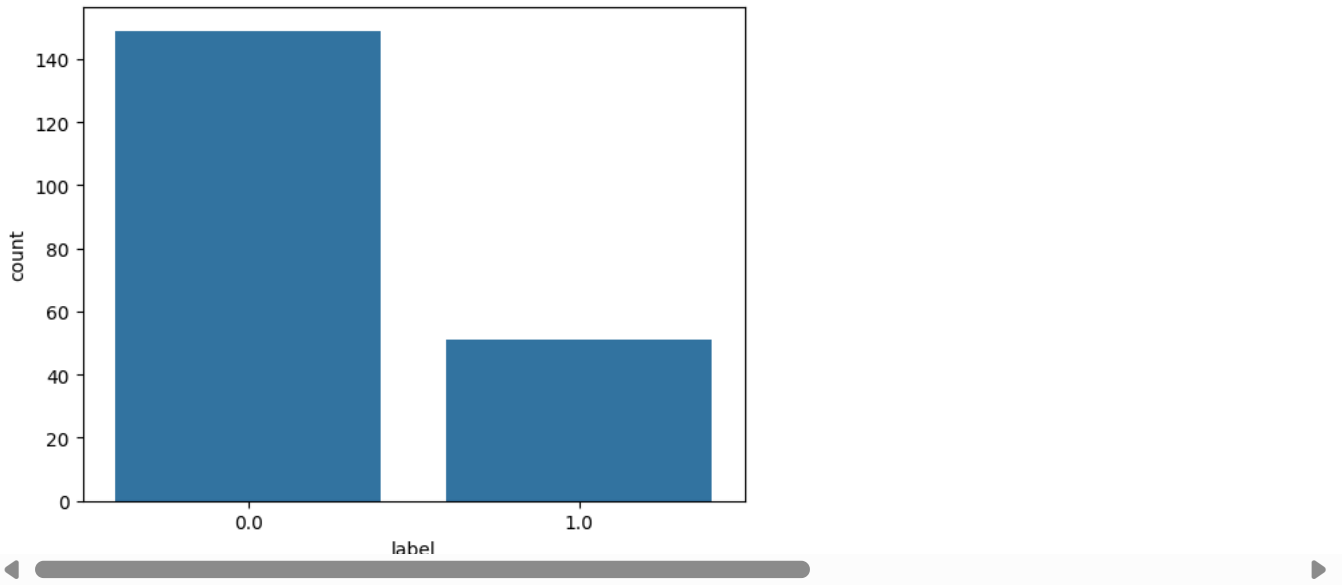
1. Load Dataset and Preprocessing

We will be focusing on classifying the posts to positive and negative physical violence.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load data and select columns
df = pd.read_excel('Labeled.xlsx', sheet_name='Test', engine='openpyxl')
df = df[['title', 'body', 'physical_violence']]

# Replace target labels
df['physical_violence'] = df['physical_violence'].map({
    'no': 0,
    'cannot be inferred': 0,
    'irrelevant': 0,
    'yes': 1,
    'plausibly': 1
})

# Rename columns
df.columns = ['title', 'body', 'label']

# Remove missing values
df = df.dropna()

# Convert to string
df['title'] = df['title'].astype(str)
df['body'] = df['body'].astype(str)
```

```
# Plot the label distribution
sns.countplot(x='label', data=df)
plt.show()
```

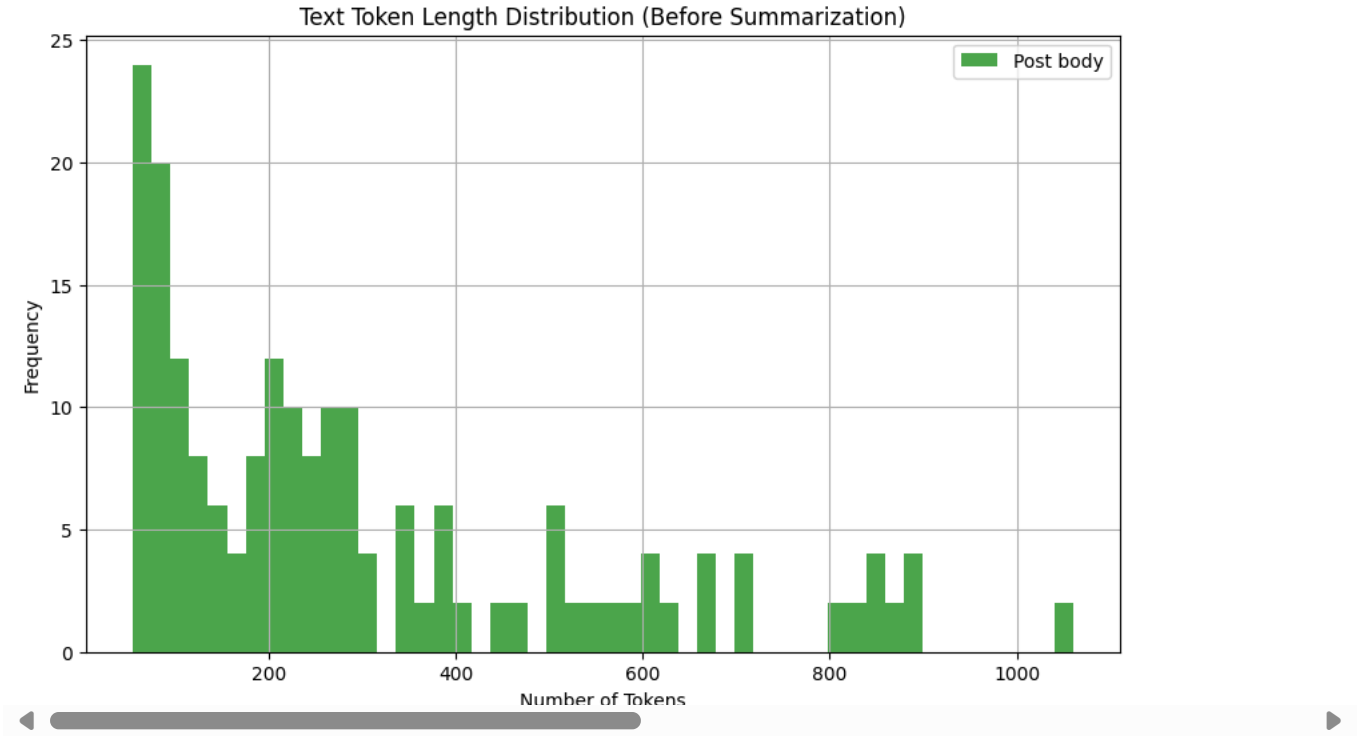
```
<ipython-input-22-1986ebf45b95>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df['title'] = df['title'].astype(str)
<ipython-input-22-1986ebf45b95>:25: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus
  df['body'] = df['body'].astype(str)
```



2. Tokenization and Length Distribution Analysis

```
# Tokenize the texts and calculate lengths (in tokens)
token_lengths = [len(tokenizer.encode(text, truncation=False)) for text in df['body']]

# Count how many texts are longer than 512 tokens (the limit of the tokenizer)
long_texts_count = sum(1 for length in token_lengths if length > 512)
long_texts_percentage = round(100 * long_texts_count / len(df['body']), 3)

# Print the number of texts longer than 512 tokens with percentage
print(f'Number of texts longer than 512 tokens: {long_texts_count}, which are {long_texts_percentage}% of the data.')

# Plot token length distribution
plt.figure(figsize=(10, 6))
plt.hist(token_lengths, bins=50, alpha=0.7, label='Post body', color='green')
plt.xlabel('Number of Tokens')
plt.ylabel('Frequency')
plt.title('Text Token Length Distribution (Before Summarization)')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```

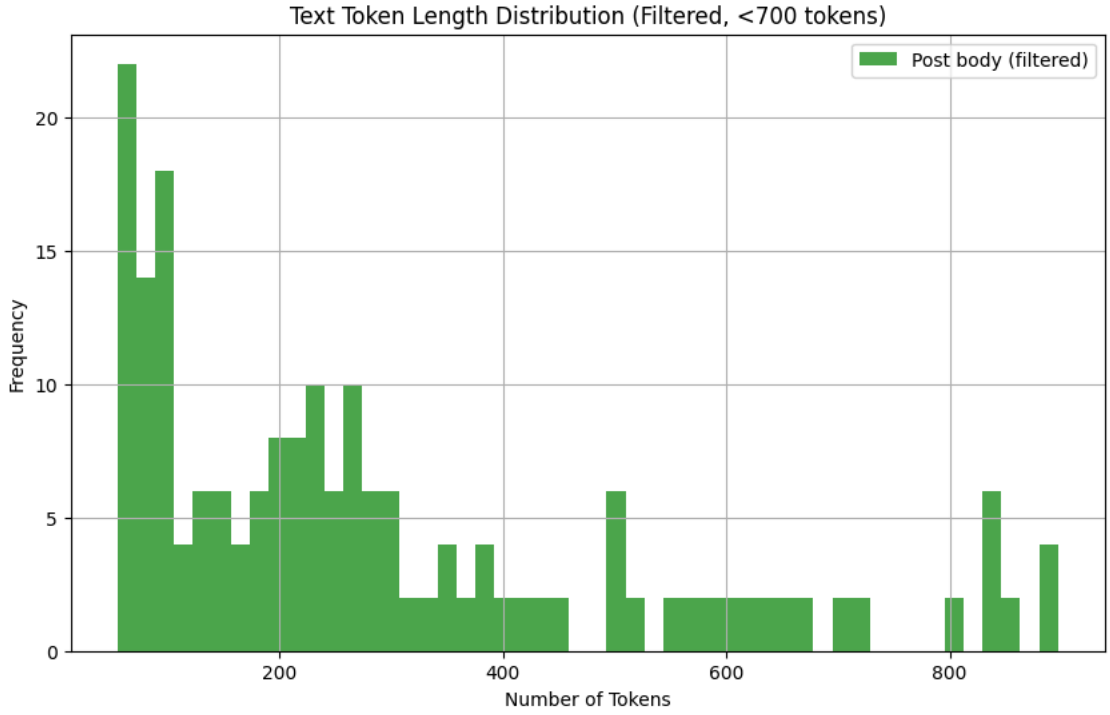
```
Token indices sequence length is longer than the specified maximum sequence length for this model (859 > 512). Running this sequence
Number of texts longer than 512 tokens: 38, which are 19.0% of the data.
```



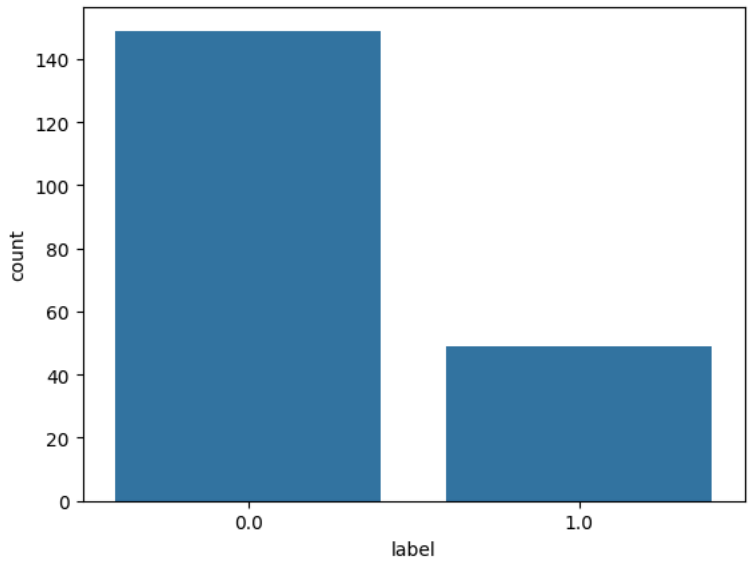
3. Filter Texts by Token Length

```
# Filter the dataset to only include rows where the length of the text is less than 512 tokens (or 700, then truncate to 512)
filtered_df = df[[len(tokenizer.encode(text, truncation=False)) < 1000 for text in df['body']]]

# Show the number of rows in the new filtered dataset
print(f'Number of texts with less than 1000 tokens: {len(filtered_df)}')

# Recalculate the token lengths for the filtered dataset
filtered_token_lengths = [len(tokenizer.encode(text, truncation=False)) for text in filtered_df['body']]

# Plot token length distribution for the filtered dataset
plt.figure(figsize=(10, 6))
plt.hist(filtered_token_lengths, bins=50, alpha=0.7, label='Post body (filtered)', color='green')
plt.xlabel('Number of Tokens')
plt.ylabel('Frequency')
plt.title('Text Token Length Distribution (Filtered, <700 tokens)')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()

# Check the effect of the data filtering on the label distribution
sns.countplot(x='label', data=filtered_df)
```

```
Number of texts with less than 1000 tokens: 198
```




```
<Axes: xlabel='label', ylabel='count'>
```



As we can see, dropping the long posts (more than 700 tokens) helps a bit to balance the labels

4. Hyperparameters and Data Split

- We will use the train set solely for training the model.
- We will use the test sets both as validations set in training, and to analyze the model with LIME.

Note: The GPU we used had low vRAM, so we limited the size of samples in the test set to be up to 150 tokens (will be later used during LIME). If you have a bigger GPU, don't run this cell. Instead, activate the next cell for regular train-test split.

```
# Define hyperparameters
hyperparameters = {
    'batch_size': 16,
    'epochs': 2,
    'tokenizer_max_length': 512,  # Max length for training samples
    'test_max_length': 512,       # Max length for test samples
    'class_1_weight': 3.0,
    'learning_rate': 1e-5,
    'test_size': 0.2
}

# Set the random seed for reproducibility
random.seed(42)

# Filter the dataset to separate samples that fit the test max length
```

```
filtered_texts = filtered_df['body'].tolist()
filtered_labels = filtered_df['label'].tolist()

short_texts, short_labels = [], []
long_texts, long_labels = [], []

for text, label in zip(filtered_texts, filtered_labels):
    encoding = tokenizer(text, truncation=True, max_length=hyperparameters['tokenizer_max_length'], return_tensors='pt')
    if encoding['input_ids'].shape[1] <= hyperparameters['test_max_length']:
        short_texts.append(text)
        short_labels.append(label)
    else:
        long_texts.append(text)
        long_labels.append(label)

# Determine the number of samples needed for the test set (20% of the data)
test_size = int(hyperparameters['test_size'] * len(filtered_texts))
actual_test_size = min(test_size, len(short_texts))

# Randomly sample from short texts for the test set
test_indices = random.sample(range(len(short_texts)), actual_test_size)
test_texts = [short_texts[i] for i in test_indices]
test_labels = [short_labels[i] for i in test_indices]

# Use remaining samples for training
train_texts = long_texts + [short_texts[i] for i in range(len(short_texts)) if i not in test_indices]
train_labels = long_labels + [short_labels[i] for i in range(len(short_labels)) if i not in test_indices]

# Display statistics
print(f'train_size: {len(train_texts)}')
print(f'train 0 label count: {train_labels.count(0)}')
print(f'train 1 label count: {train_labels.count(1)}')
print()
print(f'test_size: {len(test_texts)}')
print(f'test 0 label count: {test_labels.count(0)}')
print(f'test 1 label count: {test_labels.count(1)}')
print()
```

```
train_size: 159
train 0 label count: 122
train 1 label count: 37

test_size: 39
test 0 label count: 27
test 1 label count: 12
```

Activate this cell to run a regular train-test split

```
"""
from sklearn.model_selection import train_test_split
# Define hyperparameters
 hyperparameters = {
     'batch_size': 16,
     'epochs': 2,
     'tokenizer_max_length': 512,
     'class_1_weight': 3.0,
     'learning_rate': 1e-5
 }

 # Split the filtered data into training and testing sets
 train_texts, test_texts, train_labels, test_labels = train_test_split(
     filtered_df['body'].tolist(),
     filtered_df['label'].tolist(),
     test_size=0.2,
     random_state=42
 )

 print(f'train_size: {len(train_texts)}')
 print(f'train 0 label count: {train_labels.count(0)}')
 print(f'train 1 label count: {train_labels.count(1)}')
 print()

 print(f'test_size: {len(test_texts)}')
 print(f'test 0 label count: {test_labels.count(0)}')
 print(f'test 1 label count: {test_labels.count(1)}')
 print()
"""
```

```
  File "<ipython-input-72-55709597c99e>", line 29
    """
       ^
SyntaxError: unterminated string literal (detected at line 29)
```

Next steps: Fix error

5. Tokenization and DataLoader Setup

```
# Tokenization function
def tokenize_data(texts, labels,tokenizer):
    encodings = tokenizer(texts, truncation=True, padding=True, max_length=hyperparameters['tokenizer_max_length'])
    inputs = torch.tensor(encodings['input_ids'])
    attention_masks = torch.tensor(encodings['attention_mask'])
    labels = torch.tensor(labels, dtype=torch.float32).unsqueeze(1)
    return TensorDataset(inputs, attention_masks, labels)

# Tokenize the data
train_dataset = tokenize_data(train_texts, train_labels,tokenizer)
test_dataset = tokenize_data(test_texts, test_labels,tokenizer)

# Create DataLoader
train_loader = DataLoader(train_dataset, batch_size=hyperparameters['batch_size'], shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=hyperparameters['batch_size'])

# Initiating the device
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
```

6. Model Training and Evaluations, and saving

(Activating the training is not in this cell, but in the next one)

```
# Training function
def train_model(model, train_loader, test_loader, device, epochs=hyperparameters['epochs']):
    model.to(device)

    # Define loss function and optimizer
    pos_weight = torch.tensor([hyperparameters['class_1_weight']]).to(device)
    loss_fn = BCEWithLogitsLoss(pos_weight=pos_weight)
    optimizer = AdamW(model.parameters(), lr=hyperparameters['learning_rate'])

    model.train()

    for epoch in range(epochs):
        total_loss = 0
        progress_bar = tqdm(train_loader, desc=f'Epoch {epoch+1}/{epochs}', leave=False)

        for batch in progress_bar:
            optimizer.zero_grad()

            input_ids, attention_mask, labels = [x.to(device) for x in batch]
            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs.logits

            loss = loss_fn(logits, labels)
            total_loss += loss.item()

            loss.backward()
            optimizer.step()

            progress_bar.set_postfix({'Batch Loss': f'{loss.item():.4f}'})

        avg_train_loss = total_loss / len(train_loader)
        print(f'Epoch {epoch+1}/{epochs}, Training Loss: {avg_train_loss:.4f}')

        # Evaluate on test set after each epoch
        evaluate_model(model, test_loader,loss_fn)

    # Move model back to CPU after training
    model.to('cpu')


# Model evaluation function
def evaluate_model(model, val_loader, loss_fn):
    model.eval()
    val_loss = 0
    correct = 0
```

```
    total = 0
    all_labels = []
    all_preds = []

    with torch.no_grad():
        for batch in val_loader:
            input_ids, attention_mask, labels = [x.to(device) for x in batch]
            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs.logits

            loss = loss_fn(logits, labels)
            val_loss += loss.item()

            # Apply sigmoid activation and threshold to get binary predictions
            preds = torch.round(torch.sigmoid(logits))

            all_labels.extend(labels.cpu().numpy())
            all_preds.extend(preds.cpu().numpy())

            correct += (preds == labels).sum().item()
            total += labels.size(0)

    avg_val_loss = val_loss / len(val_loader)
    accuracy = correct / total

    # Calculate Precision, Recall, F1
    precision = precision_score(all_labels, all_preds)
    recall = recall_score(all_labels, all_preds)
    f1 = f1_score(all_labels, all_preds)

    print(f'Validation Loss: {avg_val_loss:.4f}, Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {
```

Activate the two cells below to activate the model training and save it

```
# train_model(model, train_loader, test_loader, device)
# torch.cuda.empty_cache()
```

```
"""
import os

 # Save model and tokenizer
 output_dir = 'distilbert_model'
 if not os.path.exists(output_dir):
     os.makedirs(output_dir)

 model.save_pretrained(output_dir)
 tokenizer.save_pretrained(output_dir)

 print(f"Model and tokenizer saved to {output_dir}")
"""
```

7. Preparing the test set dataframe for LIME analysis

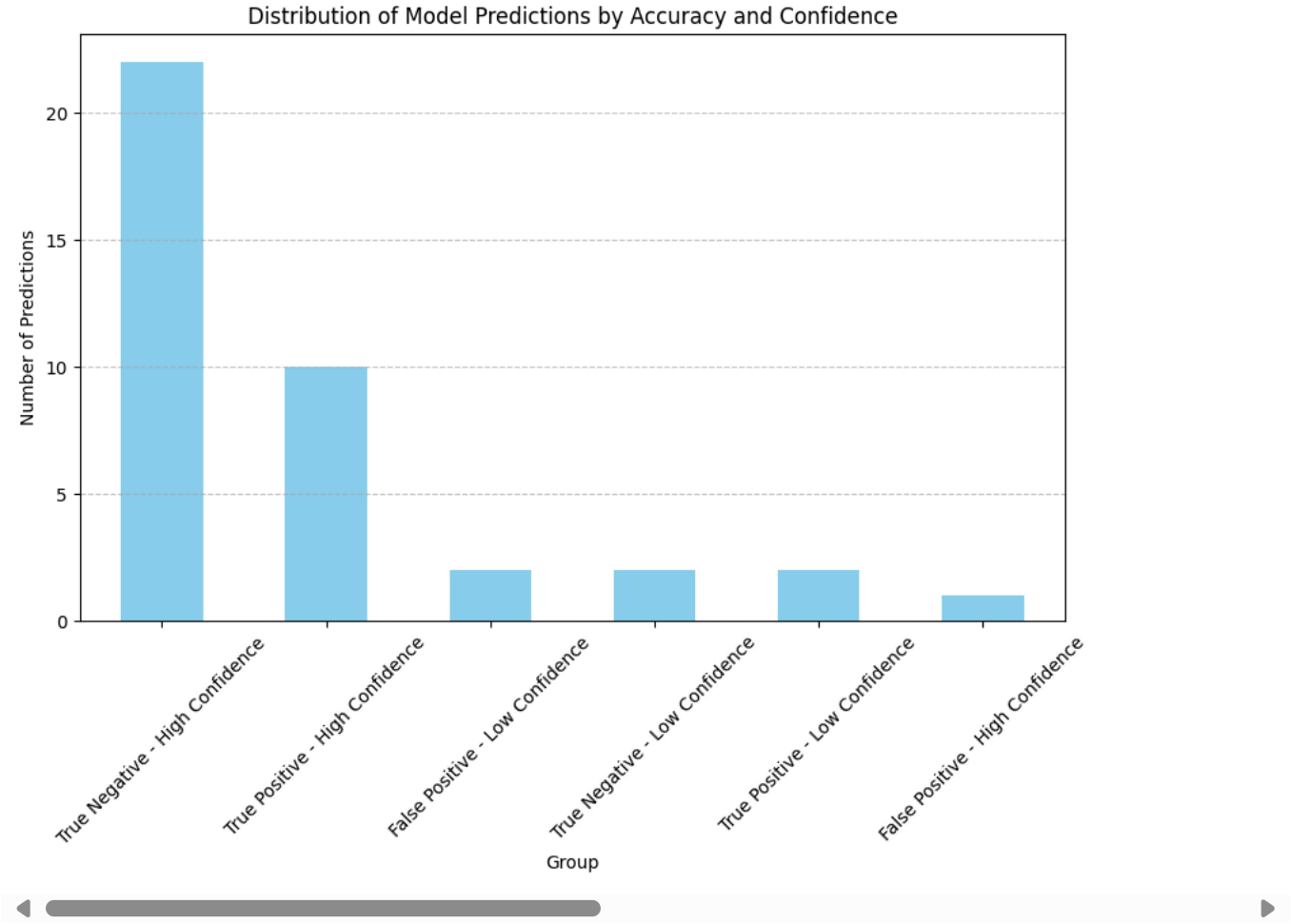
Splitting the test_set into eight groups, based on the accuracy and confidence:

- True Positive (TP) with High Confidence.
- True Positive (TP) with Low Confidence.
- True Negative (TN) with High Confidence.
- True Negative (TN) with Low Confidence.
- False Positive (FP) with High Confidence.
- False Positive (FP) with Low Confidence.
- False Negative (FN) with High Confidence.
- False Negative (FN) with Low Confidence.

```
# Prediction function (using probabilities for confidence)
def predict_with_confidence(model, dataloader):
    model.to(device)  # Move model to GPU
    model.eval()
    predictions, confidences_class_1, true_labels = [], [], []

    with torch.no_grad():
        # Use tqdm to add a progress bar to the dataloader loop
        for batch in tqdm(dataloader, desc="Predicting", leave=False):
            input_ids, attention_mask, labels = [x.to(device) for x in batch]
            outputs = model(input_ids, attention_mask=attention_mask)
```

```
            logits = outputs.logits
            probs = torch.sigmoid(logits).cpu().numpy()  # Sigmoid output, gives probability for class 1
            preds = np.round(probs)

            predictions.extend(preds.flatten())  # Flatten predictions
            confidences_class_1.extend(probs.flatten())  # Confidence scores for class 1
            true_labels.extend(labels.cpu().numpy().flatten())  # Flatten true labels

    model.to('cpu')  # Move model back to CPU
    torch.cuda.empty_cache()

    return np.array(predictions), np.array(confidences_class_1), np.array(true_labels)

# Define thresholds for high and low confidence
high_conf_thresh = 0.8  # High confidence for class 1
low_conf_thresh = 0.2   # High confidence for class 0

# Make predictions on the test set and get confidence scores
predictions, confidences_class_1, true_labels = predict_with_confidence(model, test_loader)
torch.cuda.empty_cache()

# Create a DataFrame to store predictions, true labels, confidence values, and confidence scores
df_test = pd.DataFrame({
    'body': test_texts,       # Assuming 'test_texts' contains the original text data
    'y_true': true_labels,    # True labels
    'y_pred': predictions,    # Model predictions
    'confidence_class_1': confidences_class_1, # Probability for class 1 (sigmoid output)
    'confidence_in_predicted_class': np.where(predictions == 1, confidences_class_1, 1 - confidences_class_1)  # Confidence in the predi
})

# Categorize into True Positive, True Negative, False Positive, False Negative
df_test['group'] = np.where((df_test['y_true'] == 1) & (df_test['y_pred'] == 1), 'True Positive',
                    np.where((df_test['y_true'] == 0) & (df_test['y_pred'] == 0), 'True Negative',
                    np.where((df_test['y_true'] == 0) & (df_test['y_pred'] == 1), 'False Positive', 'False Negative')))

# Define high and low confidence
df_test['confidence_level'] = np.where((df_test['confidence_in_predicted_class'] >= high_conf_thresh) |
                                       (df_test['confidence_in_predicted_class'] <= low_conf_thresh),
                                       'High Confidence', 'Low Confidence')

# Combine group (TP, TN, FP, FN) and confidence level
df_test['final_group'] = df_test['group'] + ' - ' + df_test['confidence_level']

# View distribution of the final groups
group_counts = df_test['final_group'].value_counts()
print(group_counts)

# Plot the distribution of the final groups as a bar chart
plt.figure(figsize=(10, 6))
group_counts.plot(kind='bar', color='skyblue')
plt.title('Distribution of Model Predictions by Accuracy and Confidence')
plt.xlabel('Group')
plt.ylabel('Number of Predictions')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

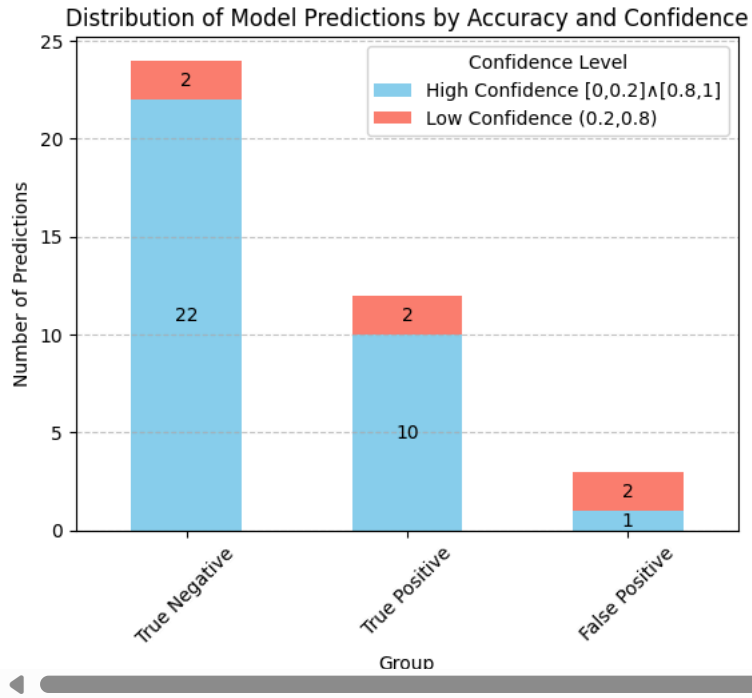
```
Predicting:   0%|          | 0/3 [00:00<?, ?it/s]DistilBertSdpaAttention is used but `torch.nn.functional.scaled_dot_product_attenti
final_group
True Negative - High Confidence     22
True Positive - High Confidence     10
False Positive - Low Confidence      2
True Negative - Low Confidence       2
True Positive - Low Confidence       2
False Positive - High Confidence     1
Name: count, dtype: int64
```



Stacked bar

```
# Count occurrences of each group and confidence level
group_confidence_counts = df_test.groupby(['group', 'confidence_level']).size().unstack(fill_value=0)

# Sort groups by total count (sum of high and low confidence)
group_confidence_counts = group_confidence_counts.loc[group_confidence_counts.sum(axis=1).sort_values(ascending=False).index]

# Plot the stacked bar chart
plt.figure(figsize=(10, 6))
ax = group_confidence_counts.plot(kind='bar', stacked=True, color=['skyblue', 'salmon'])

# Add a custom legend with confidence ranges
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, ['High Confidence [0,0.2]∧[0.8,1]', 'Low Confidence (0.2,0.8)'], title='Confidence Level', loc='upper right')

# Add annotations for each segment
for container in ax.containers:
    ax.bar_label(container, label_type='center', fontsize=10)

# Customize plot appearance
plt.title('Distribution of Model Predictions by Accuracy and Confidence')
plt.xlabel('Group')
plt.ylabel('Number of Predictions')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Column Descriptions for df_test

- **body**: The text content of each sample.
- **y_true**: The actual label (ground truth) for each sample.
- **y_pred**: The predicted label from the model for each sample.
- **confidence_class_1**: Model's confidence score for classifying the sample as Class 1 (abusive).
- **confidence_in_predicted_class**: Model's confidence score for the predicted label.
- **group**: Category based on the correctness and confidence of the prediction (TP, TN, FP, FN).
- **confidence_level**: Confidence level categorization (e.g., Low, High).
- **final_group**: Combined 'group' and 'confidence_level.'

8. LIME Analysis

Watch this video to understand LIME:

- https://www.youtube.com/watch?v=qQvC6FWlc-E

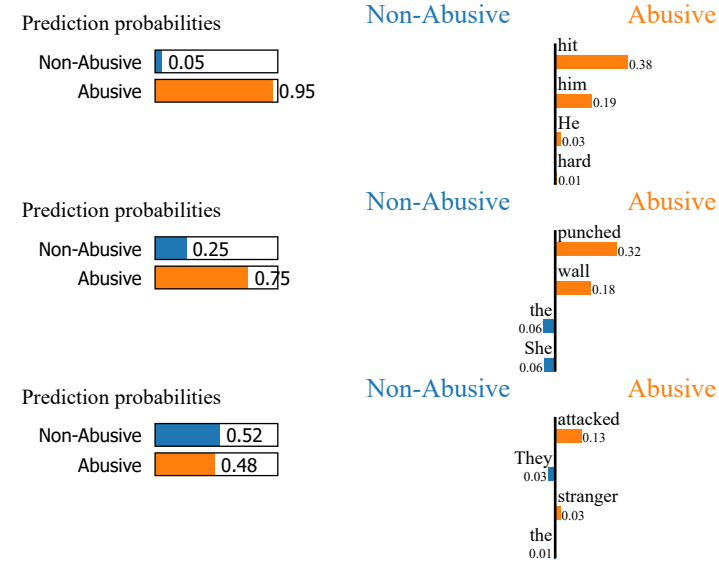
Note: From now on, we will conduct various analyses on the model to understand how it makes predictions. This will enable us to improve it with data that reflects our insights.

8.1. Simple LIME examples on dummy samples

```
# Initialize the LIME explainer with fixed class names
explainer = LimeTextExplainer(class_names=['Non-Abusive', 'Abusive'])

# Efficient Prediction Function for LIME
def predict_fn(text, model, tokenizer, max_tokens_length):
    encodings = tokenizer(text, truncation=True, padding=True, max_length=max_tokens_length, return_tensors="pt")
    input_ids = encodings['input_ids'].to(device)
    attention_mask = encodings['attention_mask'].to(device)

    with torch.no_grad():
        prob = torch.sigmoid(model(input_ids, attention_mask=attention_mask).logits).cpu().numpy()

    # Clear GPU memory
    del input_ids, attention_mask
    torch.cuda.empty_cache(); gc.collect()
    return np.hstack([1 - prob, prob])

# --- Per-Sample LIME Analysis (Optimized for GPU Memory) ---
def lime_analysis_per_sample(model, tokenizer, categorized_texts, analysis_params):
    model.to(device)

    for category, texts in categorized_texts.items():
        # Display header for the category with bold effect
```

```
            print("\n" + "="*60)
            print(f"\033[1m--- LIME Analysis for '{category}' ---\033[0m")
            print("="*60)

            for text in texts:
                # Perform LIME explanation for each sample
                explanation = explainer.explain_instance(
                    text,
                    lambda x: predict_fn(x, model, tokenizer, analysis_params['max_tokens_length']),
                    num_features=analysis_params['top_k_features'],
                    num_samples=analysis_params['num_of_perturbations_in_lime']
                )

                # Display explanation
                explanation.show_in_notebook()  # Viewable in notebooks, or use explanation.as_pyplot_figure() in scripts

                # Clear GPU memory after each sample
                torch.cuda.empty_cache()
                gc.collect()

        # Move model back to CPU after processing
        model.to('cpu')
        torch.cuda.empty_cache()
        gc.collect()


    # Dummy sample categorized texts for LIME analysis
    categorized_texts = {
        "Direct Violence": ["He hit him hard.", "She punched the wall.", "They attacked the stranger."],
        "Indirect Violence": ["They were shouting loudly.", "He threw the chair across the room.", "The situation was getting tense."],
        "Non-Violent Conflict": ["They argued for hours.", "He left without saying a word.", "She was upset, but kept calm."],
        "Neutral": ["They went for a walk.", "He smiled and waved.", "She worked at her desk."],
        "Ambiguous": ["He pushed him gently.", "They were playing rough.", "She slapped him jokingly."],
        "Sarcasm/Metaphor": ["She killed it at the presentation.", "He was beating the competition."],
        "Threats": ["I'll hurt you if you don't listen.", "She threatened to hit him."]
    }


    # Define analysis parameters for LIME
    analysis_params = {
        'max_samples_per_category': 20,      # Max samples per category for analysis
        'top_k_features': 10,               # Top k features to display in LIME analysis
        'max_tokens_length': 200,           # Max token length for the tokenizer
        'num_of_perturbations_in_lime': 300 # Number of perturbations for LIME
    }

    lime_analysis_per_sample(model, tokenizer, categorized_texts, analysis_params)
```

```
============================================================
--- LIME Analysis for 'Direct Violence' ---
============================================================
```

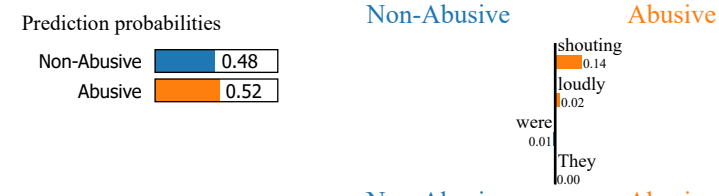


Text with highlighted words
He hit him hard.

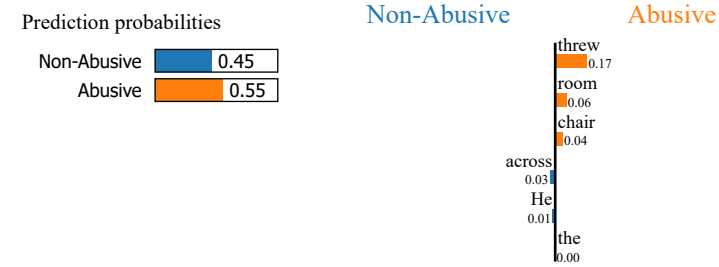
Text with highlighted words
She punched the wall.

Text with highlighted words
They attacked the stranger.

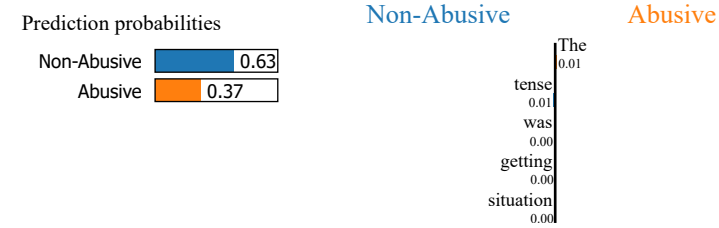
```
============================================================
--- LIME Analysis for 'Indirect Violence' ---
============================================================
```



Text with highlighted words
They were shouting loudly.

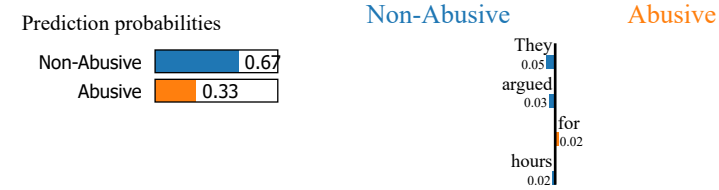


Text with highlighted words
He threw the chair across the room.

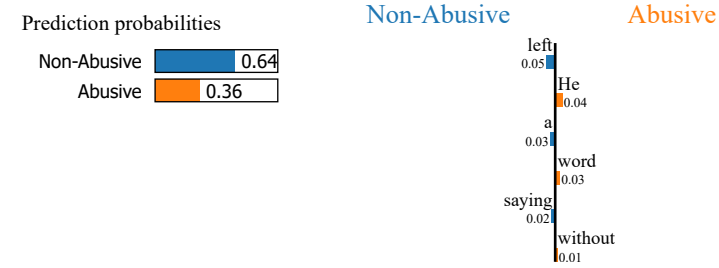


Text with highlighted words
The situation was getting tense.

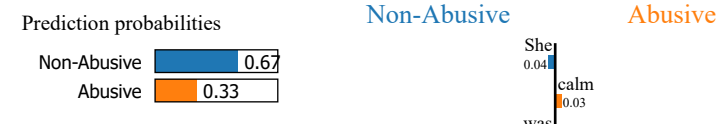
```
============================================================
--- LIME Analysis for 'Non-Violent Conflict' ---
============================================================
```



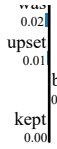
Text with highlighted words
They argued for hours.



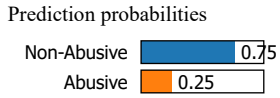
Text with highlighted words
He left without saying a word.

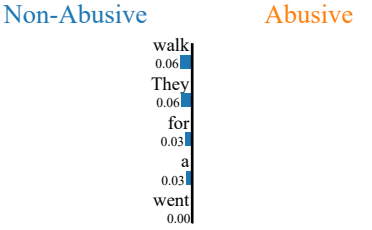


Text with highlighted words
She was upset, but kept calm.



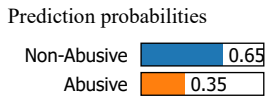

```
============================================================
--- LIME Analysis for 'Neutral' ---
============================================================
```

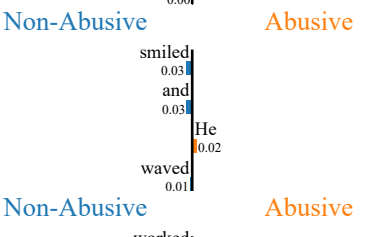




Text with highlighted words

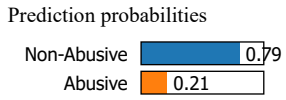
They went for a walk.

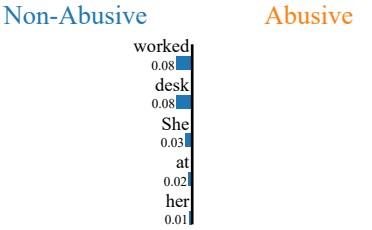




Text with highlighted words

He smiled and waved.

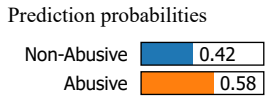


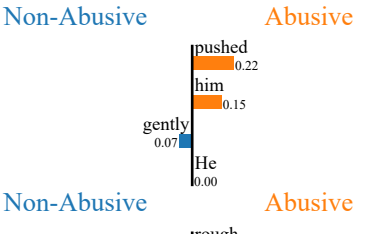


Text with highlighted words

She worked at her desk.

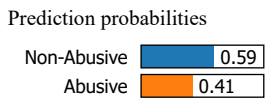
```
============================================================
--- LIME Analysis for 'Ambiguous' ---
============================================================
```

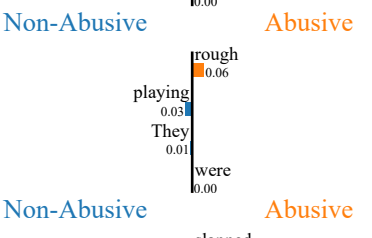




Text with highlighted words

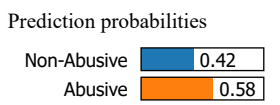
He pushed him gently.

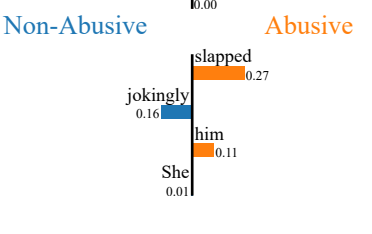




Text with highlighted words

They were playing rough.

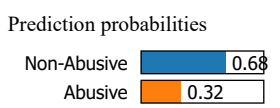


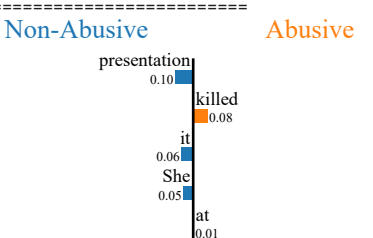


Text with highlighted words

She slapped him jokingly.

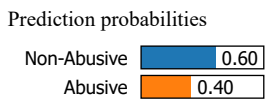
```
============================================================
--- LIME Analysis for 'Sarcasm/Metaphor' ---
============================================================
```

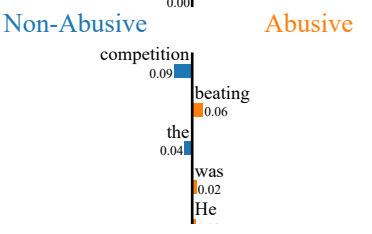




Text with highlighted words

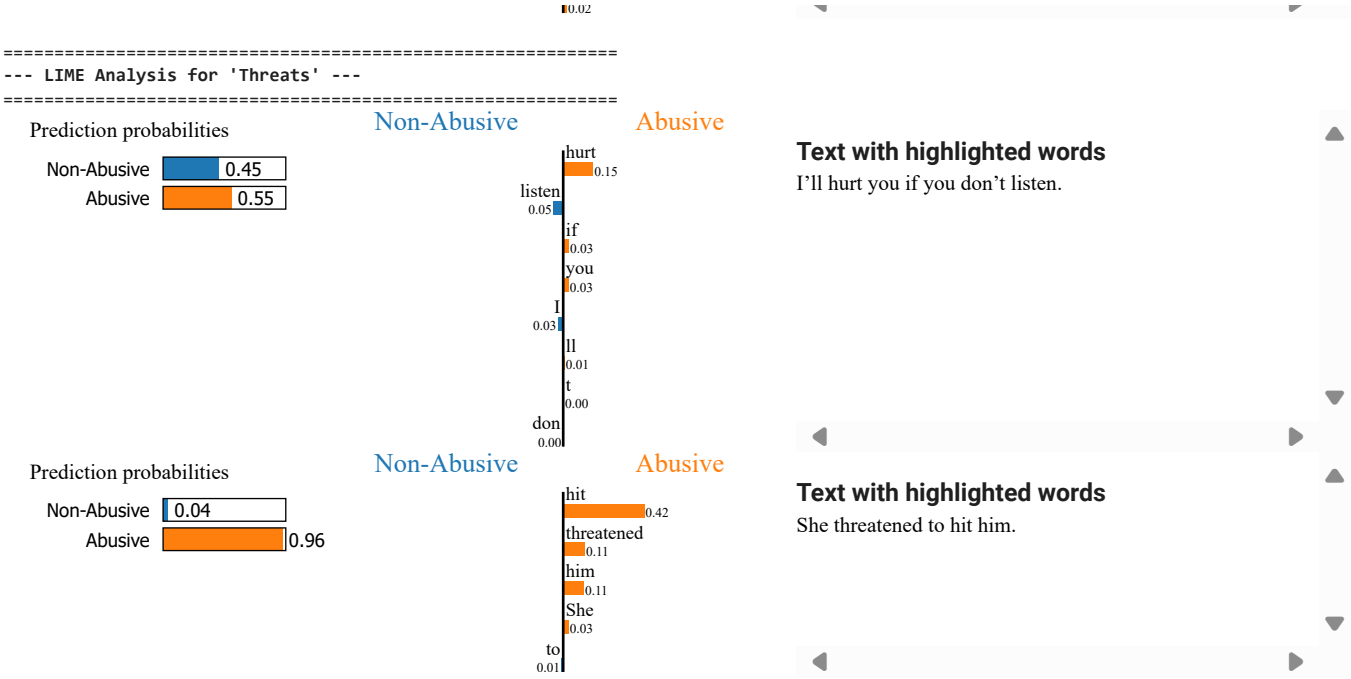
She killed it at the presentation.





Text with highlighted words

He was beating the competition.



8.2. Low-Confidence Analysis: Identifying Confusing Cases

This analysis focuses on finding cases where the model is most uncertain (confidence level near 0.5).

- We will identify the most confusing samples within each category: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).
- This is a per-sample analysis, aiming to understand the characteristics of samples that the model finds confusing.

```
# Prepare categorized texts for LIME analysis, selecting only groups that match the condition if specified
def select_extreme_samples(df, analysis_params, condition=None):
    categorized_texts = {}

    for group in df['final_group'].unique():
        # Process all groups if condition is None; otherwise, filter based on condition
        if condition is None or (condition is not None and condition in group):
            group_df = df[df['final_group'] == group].copy()
            group_df['distance_from_0.5'] = abs(group_df['confidence_in_predicted_class'] - 0.5)
            group_df = group_df.sort_values(by='distance_from_0.5')

            # Select up to max_samples_per_category samples
            categorized_texts[group] = group_df['body'].tolist()[:min(analysis_params['max_samples_per_category'], len(group_df))]

    return categorized_texts


# Run the low-confidence analysis
categorized_texts = select_extreme_samples(df_test, analysis_params,condition="Low Confidence")
lime_analysis_per_sample(model, tokenizer, categorized_texts, analysis_params)
```