main

May 12, 2023

1 Importing Libraries:

[ ]:
```
import numpy as np
import pandas as pd
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
import re
from wordcloud import WordCloud, STOPWORDS
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer, PorterStemmer
import math
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve, auc, mean_squared_error
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from xgboost import XGBRegressor
import gensim
import string
import tensorflow as tf
import keras
from keras.callbacks import ModelCheckpoint
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import CuDNNLSTM
from keras.layers import Dropout
from keras.layers import Embedding
import warnings
from keras import backend as K
warnings.filterwarnings("ignore")
```

The following block of code will show the detials about the system

[ ]:
```
import sys
import tensorflow.keras
import pandas as pd
import sklearn as sk
import scipy as sp
import tensorflow as tf
import platform
print(f"Python Platform: {platform.platform()}")
print(f"Tensor Flow Version: {tf.__version__}")
print(f"Keras Version: {tensorflow.keras.__version__}")
print()
print(f"Python {sys.version}")
print(f"Pandas {pd.__version__}")
print(f"Scikit-Learn {sk.__version__}")
print(f"SciPy {sp.__version__}")
gpu = len(tf.config.list_physical_devices('GPU'))>0
print("GPU is", "available" if gpu else "NOT AVAILABLE")
```

```
Python Platform: macOS-13.3.1-arm64-arm-64bit
Tensor Flow Version: 2.12.0
Keras Version: 2.12.0

Python 3.10.10 (main, Mar 21 2023, 13:41:05) [Clang 14.0.6 ]
Pandas 1.5.3
Scikit-Learn 1.2.2
SciPy 1.10.1
GPU is available
```

1.1 Reading Data:

[ ]:
```
train_df = pd.read_csv('train.csv', index_col='id', engine='python')
train_df.head()
```

[ ]:
```
         target                                       comment_text  \
id
59848  0.000000  This is so cool. It's like, 'would you want yo…
59849  0.000000  Thank you!! This would make my life a lot less…
59852  0.000000  This is such an urgent design problem; kudos t…
59855  0.000000  Is this something I'll be able to install on m…
59856  0.893617               haha you guys are a bunch of losers.

       severe_toxicity  obscene  identity_attack   insult  threat  asian  \
id
59848         0.000000      0.0         0.000000  0.00000     0.0    NaN
59849         0.000000      0.0         0.000000  0.00000     0.0    NaN
59852         0.000000      0.0         0.000000  0.00000     0.0    NaN
59855         0.000000      0.0         0.000000  0.00000     0.0    NaN
```

```
59856         0.021277      0.0         0.021277  0.87234     0.0    0.0

       atheist  bisexual  …  article_id    rating  funny  wow  sad  likes  \
id                        …
59848      NaN       NaN  …        2006  rejected      0    0    0      0
59849      NaN       NaN  …        2006  rejected      0    0    0      0
59852      NaN       NaN  …        2006  rejected      0    0    0      0
59855      NaN       NaN  …        2006  rejected      0    0    0      0
59856      0.0       0.0  …        2006  rejected      0    0    0      1

       disagree  sexual_explicit  identity_annotator_count  \
id
59848         0              0.0                         0
59849         0              0.0                         0
59852         0              0.0                         0
59855         0              0.0                         0
59856         0              0.0                         4

       toxicity_annotator_count
id
59848                         4
59849                         4
59852                         4
59855                         4
59856                        47

[5 rows x 44 columns]
```

[ ]:
```python
test_df = pd.read_csv('test.csv', index_col='id', engine='python')
test_df.head()
```

[ ]:
```
                                              comment_text
id
7097320  [ Integrity means that you pay your debts.]\n\…
7097321  This is malfeasance by the Administrator and t…
7097322  @Rmiller101 - Spoken like a true elitist. But …
7097323  Paul: Thank you for your kind words.  I do, in…
7097324  Sorry you missed high school. Eisenhower sent …
```

[ ]:
```python
train_df.describe()
```

[ ]:
```
             target  severe_toxicity       obscene  identity_attack  \
count  1.804874e+06     1.804874e+06  1.804874e+06     1.804874e+06
mean   1.030173e-01     4.582099e-03  1.387721e-02     2.263571e-02
std    1.970757e-01     2.286128e-02  6.460419e-02     7.873156e-02
min    0.000000e+00     0.000000e+00  0.000000e+00     0.000000e+00
25%    0.000000e+00     0.000000e+00  0.000000e+00     0.000000e+00
```

```
50%    0.000000e+00     0.000000e+00  0.000000e+00     0.000000e+00
75%    1.666667e-01     0.000000e+00  0.000000e+00     0.000000e+00
max    1.000000e+00     1.000000e+00  1.000000e+00     1.000000e+00

             insult        threat          asian        atheist  \
count  1.804874e+06  1.804874e+06  405130.000000  405130.000000
mean   8.115273e-02  9.311271e-03       0.011964       0.003205
std    1.760657e-01  4.942218e-02       0.087166       0.050193
min    0.000000e+00  0.000000e+00       0.000000       0.000000
25%    0.000000e+00  0.000000e+00       0.000000       0.000000
50%    0.000000e+00  0.000000e+00       0.000000       0.000000
75%    9.090909e-02  0.000000e+00       0.000000       0.000000
max    1.000000e+00  1.000000e+00       1.000000       1.000000

            bisexual          black  …     parent_id    article_id  \
count  405130.000000  405130.000000  …  1.026228e+06  1.804874e+06
mean        0.001884       0.034393  …  3.722687e+06  2.813597e+05
std         0.026077       0.167900  …  2.450261e+06  1.039293e+05
min         0.000000       0.000000  …  6.100600e+04  2.006000e+03
25%         0.000000       0.000000  …  7.960188e+05  1.601200e+05
50%         0.000000       0.000000  …  5.222993e+06  3.321260e+05
75%         0.000000       0.000000  …  5.775758e+06  3.662370e+05
max         1.000000       1.000000  …  6.333965e+06  3.995410e+05

              funny           wow           sad         likes      disagree  \
count  1.804874e+06  1.804874e+06  1.804874e+06  1.804874e+06  1.804874e+06
mean   2.779269e-01  4.420696e-02  1.091173e-01  2.446167e+00  5.843688e-01
std    1.055313e+00  2.449359e-01  4.555363e-01  4.727924e+00  1.866589e+00
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
50%    0.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00  0.000000e+00
75%    0.000000e+00  0.000000e+00  0.000000e+00  3.000000e+00  0.000000e+00
max    1.020000e+02  2.100000e+01  3.100000e+01  3.000000e+02  1.870000e+02

       sexual_explicit  identity_annotator_count  toxicity_annotator_count
count     1.804874e+06              1.804874e+06              1.804874e+06
mean      6.605974e-03              1.439019e+00              8.784694e+00
std       4.529782e-02              1.787041e+01              4.350086e+01
min       0.000000e+00              0.000000e+00              3.000000e+00
25%       0.000000e+00              0.000000e+00              4.000000e+00
50%       0.000000e+00              0.000000e+00              4.000000e+00
75%       0.000000e+00              0.000000e+00              6.000000e+00
max       1.000000e+00              1.866000e+03              4.936000e+03

[8 rows x 41 columns]
```

[ ]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1804874 entries, 59848 to 6334010
Data columns (total 44 columns):
 #   Column                               Dtype
---  ------                               -----
 0   target                               float64
 1   comment_text                         object
 2   severe_toxicity                      float64
 3   obscene                              float64
 4   identity_attack                      float64
 5   insult                               float64
 6   threat                               float64
 7   asian                                float64
 8   atheist                              float64
 9   bisexual                             float64
 10  black                                float64
 11  buddhist                             float64
 12  christian                            float64
 13  female                               float64
 14  heterosexual                         float64
 15  hindu                                float64
 16  homosexual_gay_or_lesbian            float64
 17  intellectual_or_learning_disability  float64
 18  jewish                               float64
 19  latino                               float64
 20  male                                 float64
 21  muslim                               float64
 22  other_disability                     float64
 23  other_gender                         float64
 24  other_race_or_ethnicity              float64
 25  other_religion                       float64
 26  other_sexual_orientation             float64
 27  physical_disability                  float64
 28  psychiatric_or_mental_illness        float64
 29  transgender                          float64
 30  white                                float64
 31  created_date                         object
 32  publication_id                       int64
 33  parent_id                            float64
 34  article_id                           int64
 35  rating                               object
 36  funny                                int64
 37  wow                                  int64
 38  sad                                  int64
 39  likes                                int64
 40  disagree                             int64
 41  sexual_explicit                      float64
 42  identity_annotator_count             int64
```

```
 43  toxicity_annotator_count            int64
dtypes: float64(32), int64(9), object(3)
memory usage: 619.7+ MB
```

[ ]:
```
train_df.isnull().sum()
```

[ ]:
```
target                                      0
comment_text                                0
severe_toxicity                             0
obscene                                     0
identity_attack                             0
insult                                      0
threat                                      0
asian                                 1399744
atheist                               1399744
bisexual                              1399744
black                                 1399744
buddhist                              1399744
christian                             1399744
female                                1399744
heterosexual                          1399744
hindu                                 1399744
homosexual_gay_or_lesbian             1399744
intellectual_or_learning_disability   1399744
jewish                                1399744
latino                                1399744
male                                  1399744
muslim                                1399744
other_disability                      1399744
other_gender                          1399744
other_race_or_ethnicity               1399744
other_religion                        1399744
other_sexual_orientation              1399744
physical_disability                   1399744
psychiatric_or_mental_illness         1399744
transgender                           1399744
white                                 1399744
created_date                                0
publication_id                              0
parent_id                              778646
article_id                                  0
rating                                      0
funny                                       0
wow                                         0
sad                                         0
likes                                       0
disagree                                    0
```

```
sexual_explicit                              0
identity_annotator_count                     0
toxicity_annotator_count                     0
dtype: int64
```

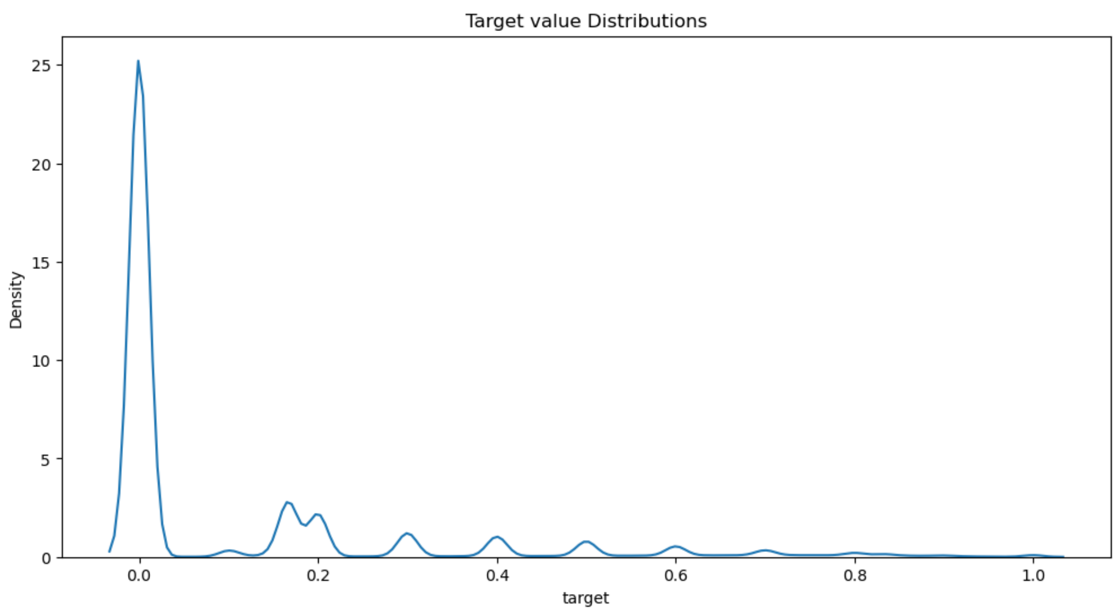
[ ]:
```python
print("Train and test shape: {} {}".format(train_df.shape, test_df.shape))
```

```
Train and test shape: (1804874, 44) (97320, 1)
```

1.2 Exploratory Data Analysis

1.3 1. Target Feature:

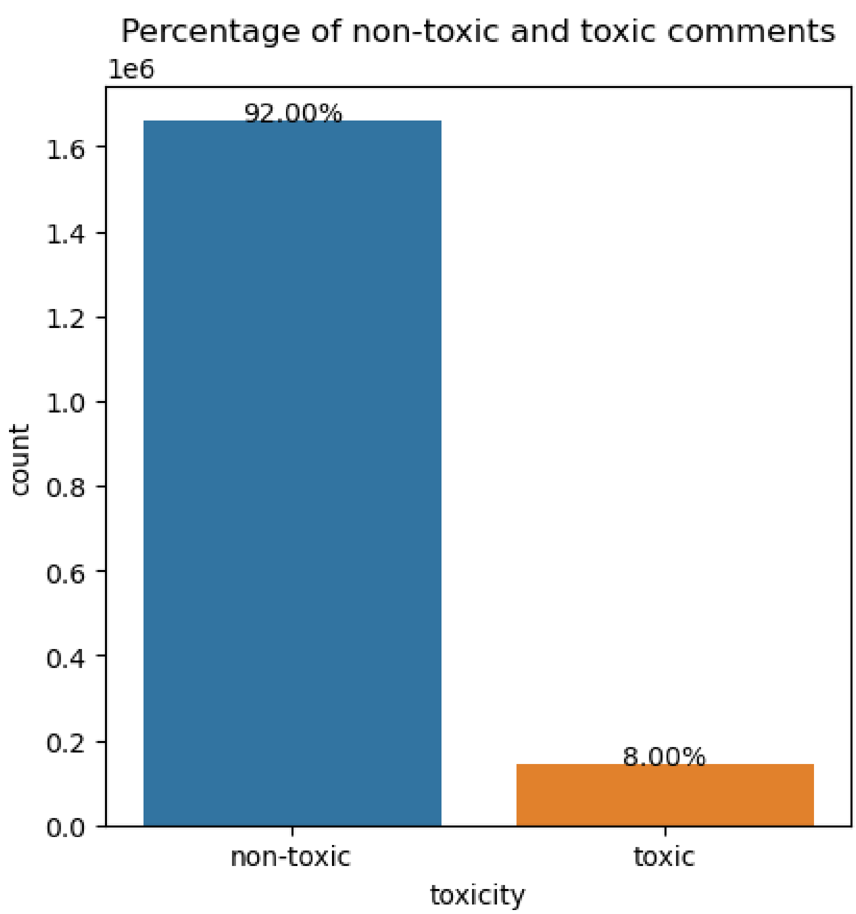
[ ]:
```python
plt.figure(figsize=(12,6))
plt.title("Target value Distributions")
sns.distplot(train_df['target'], kde=True, hist=False, bins=240, label='target')
plt.show()
```



We see that most of the comments present in the dataset are actually non-toxic (<0.5) and only a few of them are actually toxic (>0.5)

[ ]:
```python
# If toxicity rating < 0.5 then the comment is non-toxic else it is toxic.
# Get toxic and non-toxic comments.
temp = train_df['target'].apply(lambda x: "non-toxic" if x < 0.5 else "toxic")

# Convert to DataFrame and specify column name.
temp_df = temp.to_frame(name='toxicity')
```

```python
# Plot the number and percentage of toxic and non-toxic comments.
fig, ax = plt.subplots(1,1,figsize=(5,5))
total = float(len(temp))

# Plot the count plot.
cntplot = sns.countplot(data=temp_df, x='toxicity')
cntplot.set_title('Percentage of non-toxic and toxic comments')

# Get the height and calculate percentage then display it the plot itself.
for p in ax.patches:
    # Get height.
    height = p.get_height()
    # Plot at appropriate position.
    ax.text(p.get_x() + p.get_width()/2.0, height + 3, '{:1.2f}%'.
 ↪format(100*height/total), ha='center')

plt.show()
```



The dataset is imbalanced as 92% of the comments are non-toxic and only 8% are

toxic

1.4 2. Toxicity Subtype Features:

severe_toxicity

obscene

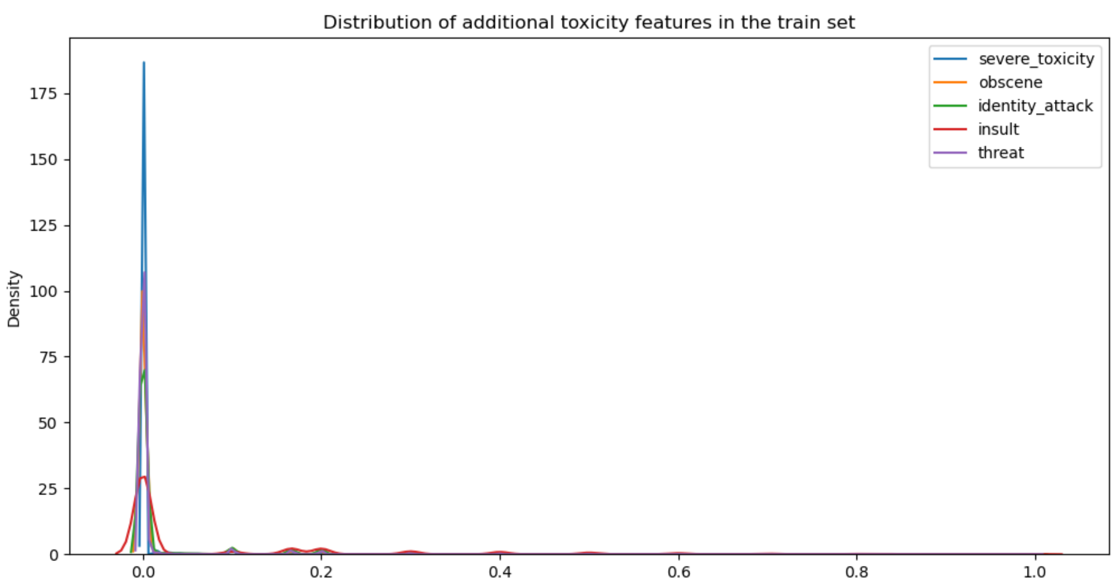
threat

insult

identity_attack

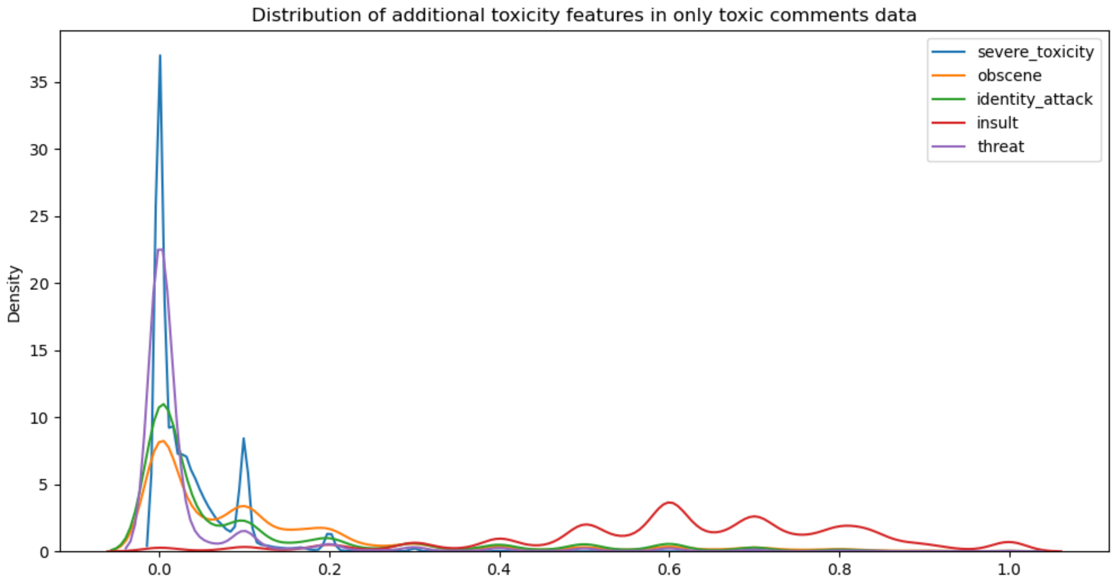
[ ]:
```
def plot_features_distribution(features, title, data):
    plt.figure(figsize=(12,6))
    plt.title(title)
    for feature in features:
        sns.distplot(data[feature],kde=True,hist=False, bins=240, label=feature)
    plt.xlabel('')
    plt.legend()
    plt.show()
```

[ ]:
```
features = ['severe_toxicity', 'obscene', 'identity_attack', 'insult', 'threat']
plot_features_distribution(features, "Distribution of additional toxicity␣
 ↪features in the train set", train_df)
```



[ ]:
```
# Looking at the distribution of additional toxicity features on the comments␣
 ↪that are actually considered toxic:
temp = train_df[train_df['target'] > 0.5]
```

```
plot_features_distribution(features, "Distribution of additional toxicity␣
↪features in only toxic comments data", temp)
```



We see that for toxic comments data, there are more insulting comments as compared to obscene comments

```
[ ]: # Getting the count of additonal toxicity features in toxic comments data(temp):
def get_comment_nature(row):
    # Extract type of toxic comment
    row = [row['severe_toxicity'], row['obscene'], row['identity_attack'],␣
↪row['insult'], row['threat']]

    maxarg = np.argmax(np.array(row)) # Get the max value index.

    if maxarg == 0: return 'severe_toxicity'
    elif maxarg == 1: return 'obscene'
    elif maxarg == 2: return 'identity_attack'
    elif maxarg == 3: return 'insult'
    else: return 'threat'
```

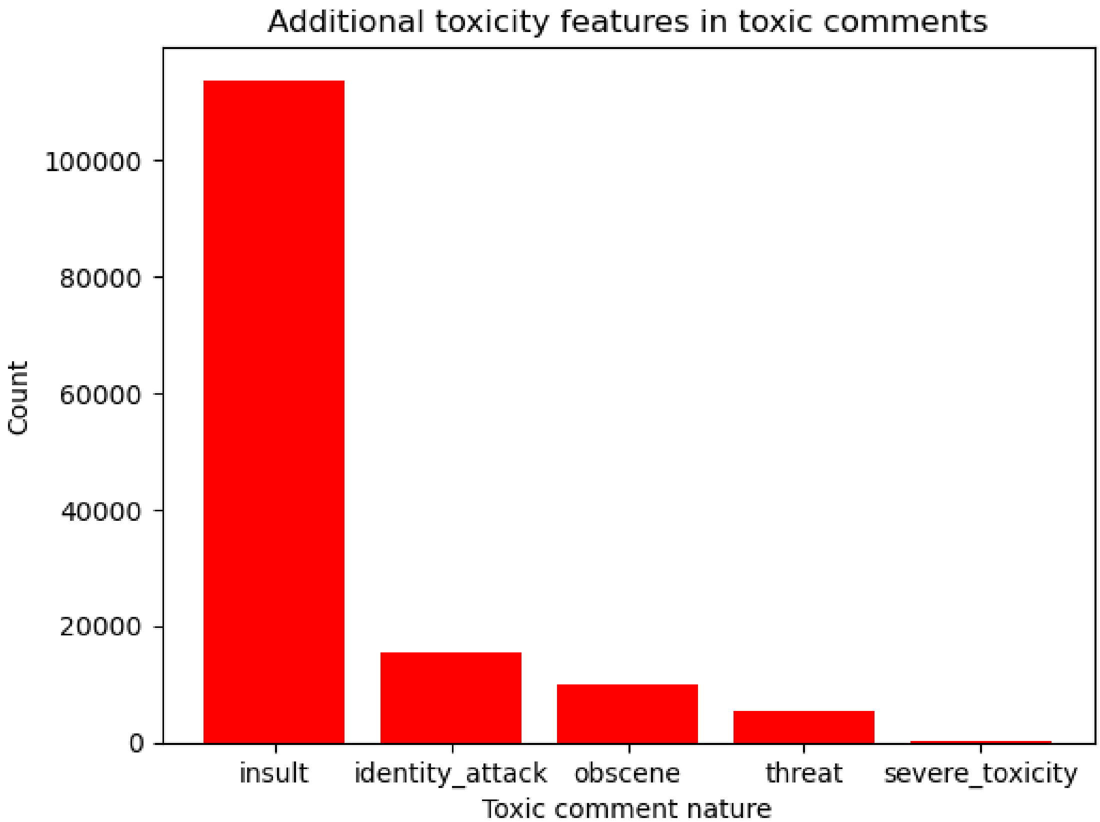
```
[ ]: # If toxicity rating < 0.5 then the comment is non-toxic else it is toxic.
# Get toxic and non-toxic comments.
temp = train_df['target'].apply(lambda x: "non-toxic" if x < 0.5 else "toxic")
print(temp)
```

```
id
59848      non-toxic
59849      non-toxic
```

```
59852      non-toxic
59855      non-toxic
59856          toxic
             …
6333967    non-toxic
6333969    non-toxic
6333982    non-toxic
6334009        toxic
6334010    non-toxic
Name: target, Length: 1804874, dtype: object
```

[ ]:
```
# Get nature of each toxic comment
#x = temp[temp == 'toxic'].index.map(lambda i: get_comment_nature(train_df.
 ↪iloc[i]))
#x
```

[ ]:
```
import matplotlib.pyplot as plt

# Get the count of each comment nature
comment_nature_counts = train_df[train_df['target'] >= 0.5].
 ↪apply(get_comment_nature, axis=1).value_counts()

# Plot the graph
plt.bar(comment_nature_counts.index, comment_nature_counts.values, color='red')

# Set the title and labels
plt.title("Additional toxicity features in toxic comments")
plt.xlabel("Toxic comment nature")
plt.ylabel("Count")

# Display the graph
plt.show()
```



In our train dataset only 8% of the data was toxic. Out of that 8%, 81% of the toxic comments made are insults, 8.37% are identity attacks, 7.20% are obscene, 3.35% are threats and a very small amount of toxic comments are severly toxic.

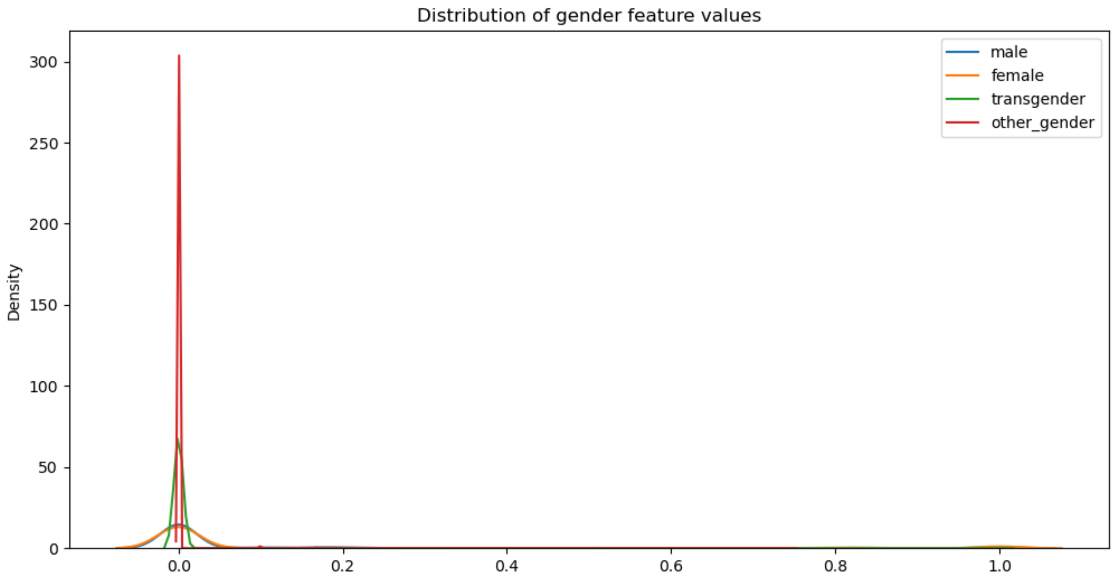
1.5 3. Identity Attributes:

Sensitive topics:

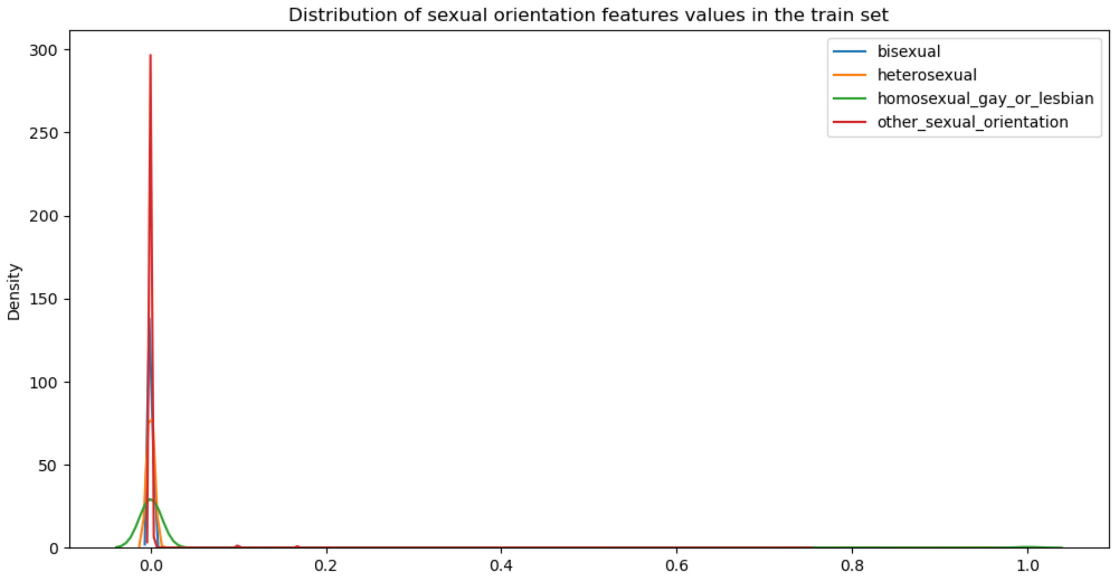
- male
- female
- homosexual_gay_or_lesbian
- bisexual
- heterosexual
- christian
- jewish
- muslim
- black
- white
- asian
- latino

[ ]:
```
temp = train_df.dropna(axis = 0, how = 'any')
```

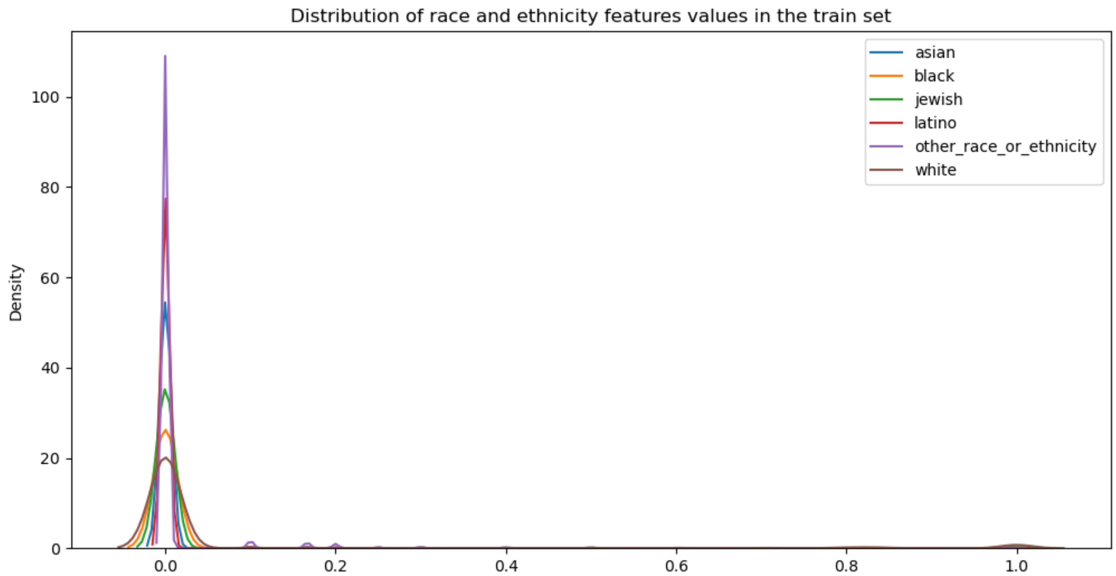
```
[ ]: features = ['male', 'female', 'transgender', 'other_gender']
     plot_features_distribution(features, "Distribution of gender feature values",␣
      ↪temp)
```




```
[ ]: features = ['bisexual', 'heterosexual', 'homosexual_gay_or_lesbian',␣
      ↪'other_sexual_orientation']
     plot_features_distribution(features, "Distribution of sexual orientation␣
      ↪features values in the train set", temp)
```

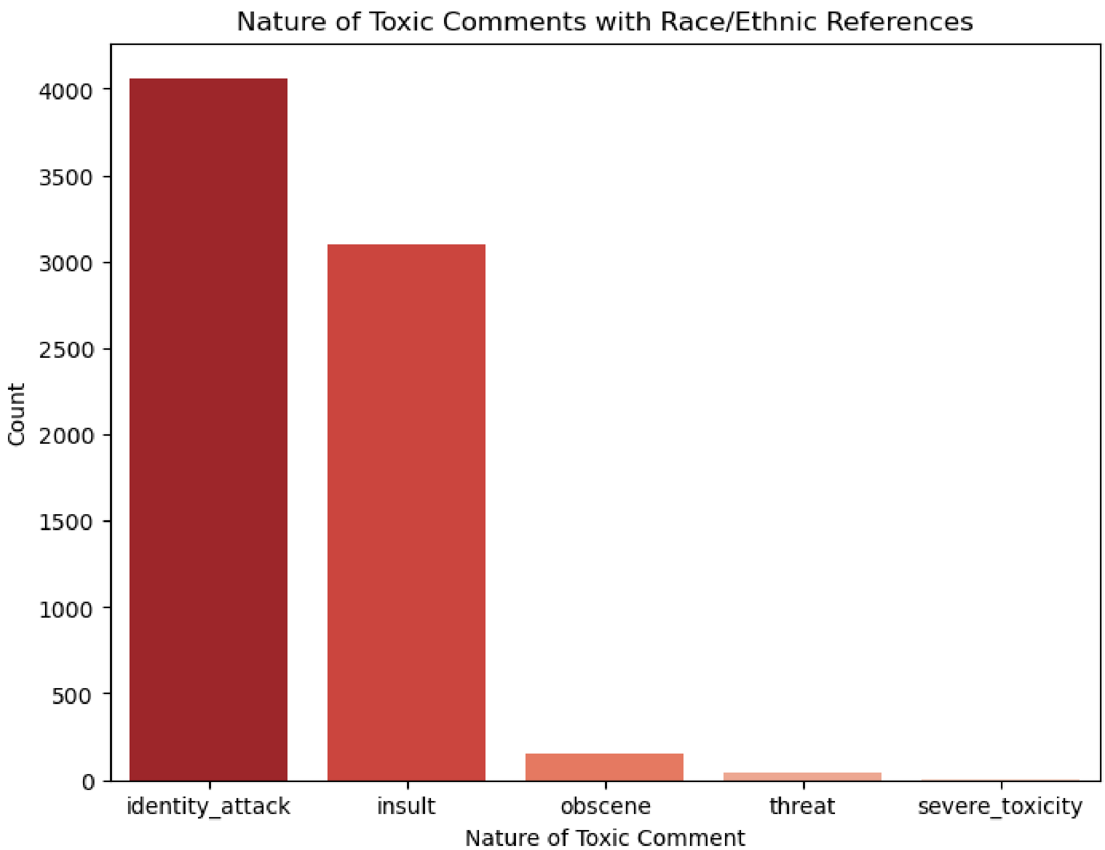


[ ]:
```
features = ['asian', 'black', 'jewish', 'latino', 'other_race_or_ethnicity',␣
 ↪'white']
plot_features_distribution(features, "Distribution of race and ethnicity␣
 ↪features values in the train set", temp)
```



[ ]:
```
# Get data where race/ethnic references are made.
cond = (train_df['asian'] > 0.5) | (train_df['black'] > 0.5) |␣
 ↪(train_df['jewish'] > 0.5) | (train_df['latino'] > 0.5) | (train_df['white']␣
 ↪> 0.5)
temp = train_df[cond] # Get data where race/ethnic references are made.
temp = temp[temp['target'] > 0.5] # Extract only toxic comments.

x = temp.apply(get_comment_nature, axis=1) # Get nature of each toxic comment
```

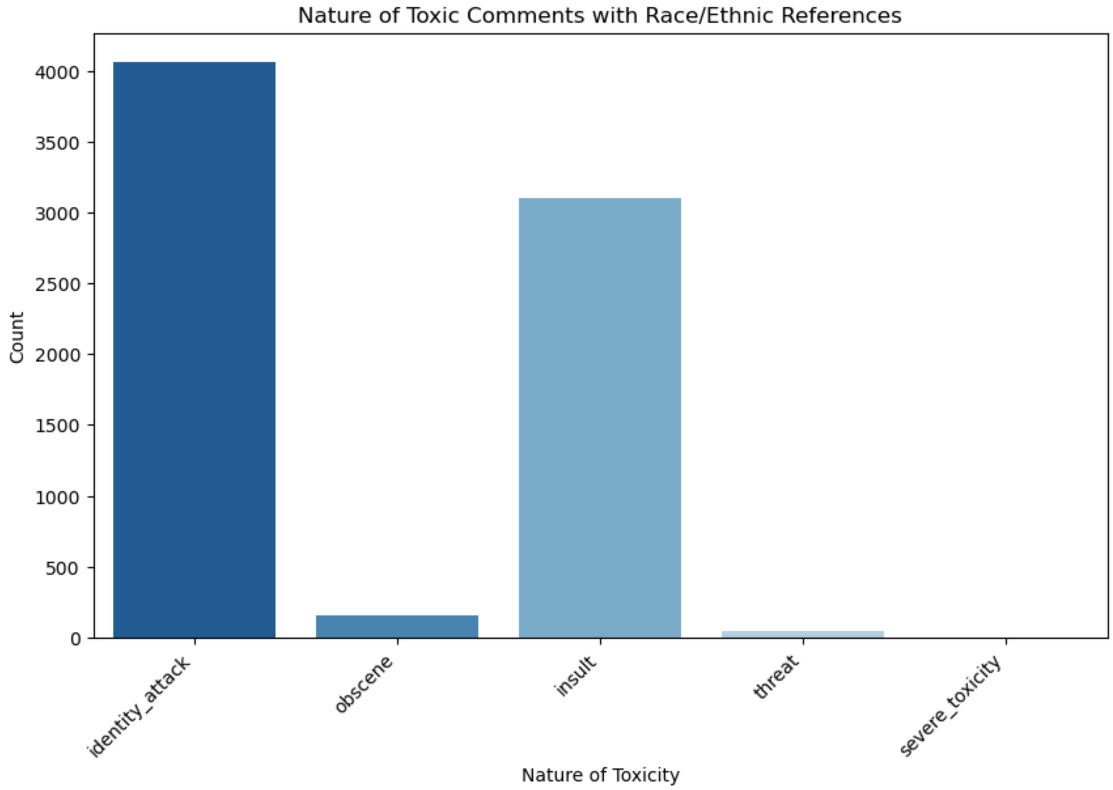
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Count the nature of each toxic comment
nature_count = x.value_counts()

# Plot the bar graph
plt.figure(figsize=(8,6))
sns.barplot(x=nature_count.index, y=nature_count.values, palette="Reds_r")
plt.title("Nature of Toxic Comments with Race/Ethnic References")
plt.xlabel("Nature of Toxic Comment")
plt.ylabel("Count")
plt.show()
```



We see that the toxic comments involving words like black, asian etc. are mainly used for identity attacks or insults.

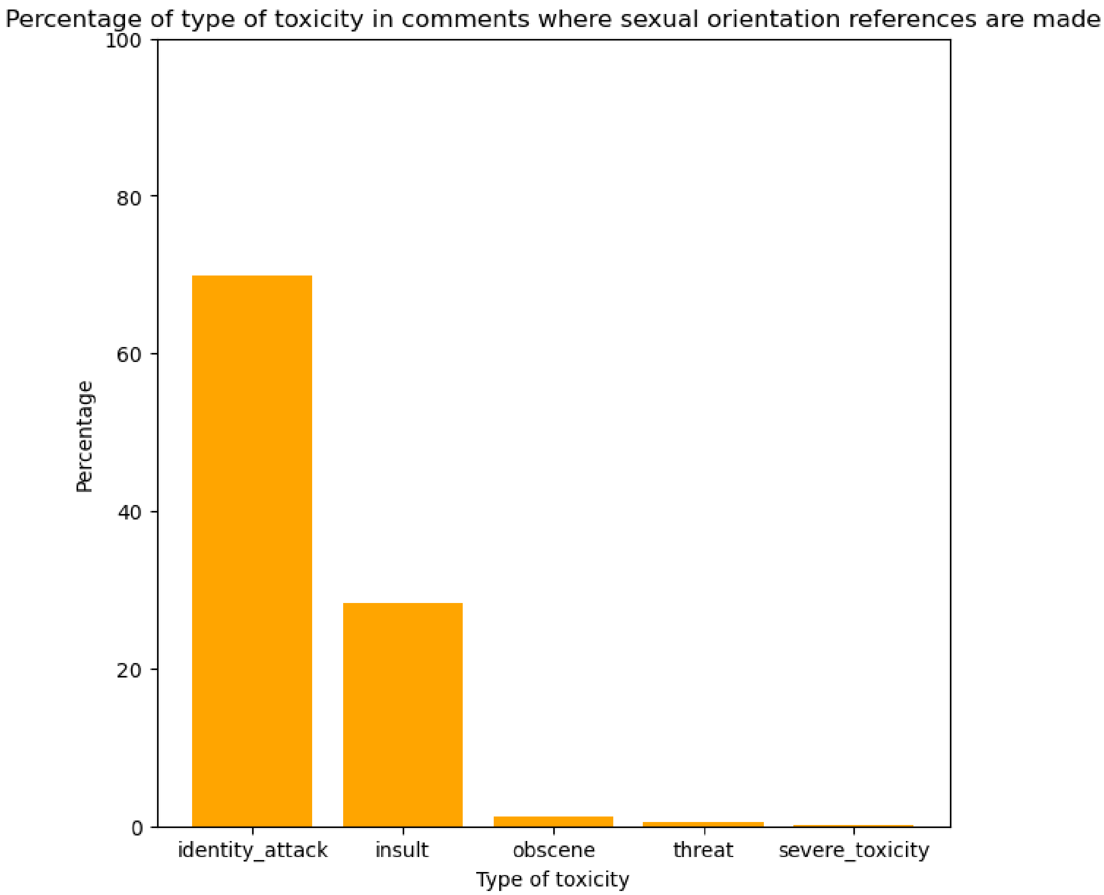
[ ]:
```
train_df2 = train_df
train_df2['insult'] = pd.to_numeric(train_df['insult'], errors='coerce')

cond = (train_df2['asian'] > 0.5) | (train_df2['black'] > 0.5) |␣
↪(train_df2['jewish'] > 0.5) | (train_df2['latino'] > 0.5) |␣
↪(train_df2['white'] > 0.5)
temp = train_df2[cond] # Get data where race/ethnic references are made.
temp = temp[temp['target'] > 0.5] # Extract only toxic comments.
temp = temp.reset_index(drop=True) # Reset index of temp DataFrame

x = temp.apply(get_comment_nature, axis=1) # Get nature of each toxic comment

# Plot the graph
fig, ax = plt.subplots(figsize=(10,6))
ax = sns.countplot(x=x, palette='Blues_r')
ax.set_title("Nature of Toxic Comments with Race/Ethnic References")
ax.set_xlabel("Nature of Toxicity")
```

```
ax.set_ylabel("Count")
plt.xticks(rotation=45, ha='right')
plt.show()
```



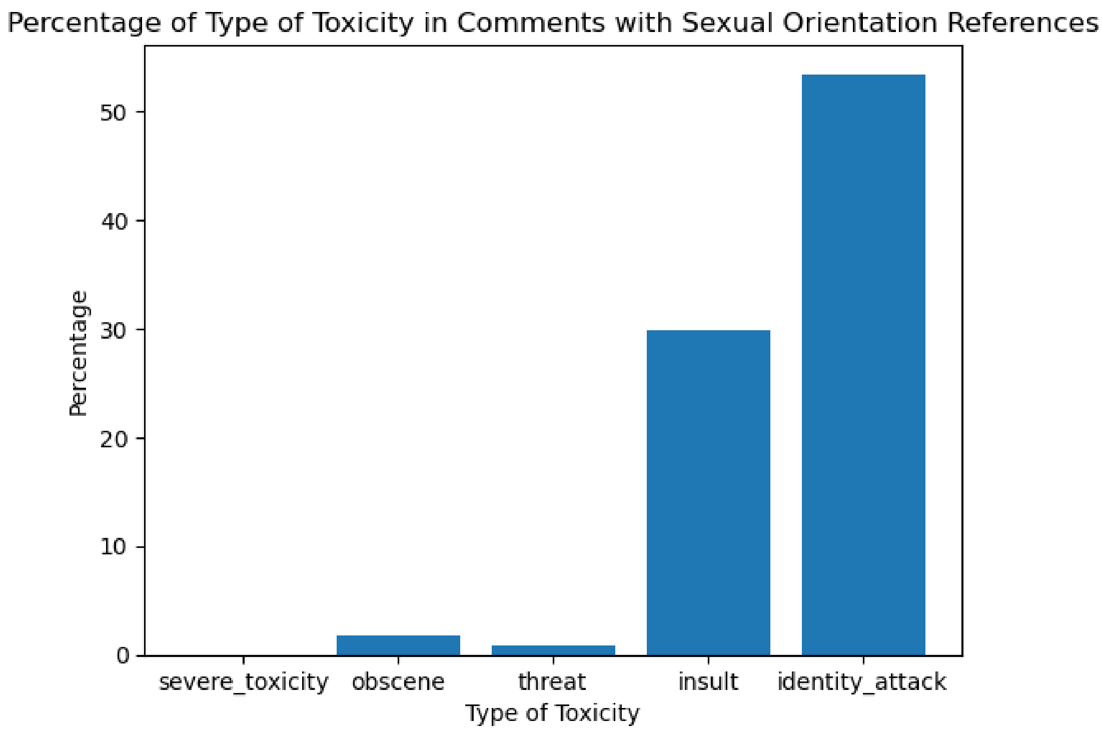

```
[ ]: # Get data where sexual orientation references are made.
cond = (train_df['bisexual'] > 0.5) | (train_df['heterosexual'] > 0.5) |␣
 ↪(train_df['homosexual_gay_or_lesbian'] > 0.5) |␣
 ↪(train_df['other_sexual_orientation'] > 0.5)
temp = train_df[cond]
temp = temp[temp['target'] > 0.5]

# Get the nature of each toxic comment.
x = temp.apply(get_comment_nature, axis=1)

# Calculate the percentage of each type of toxicity.
percentages = x.value_counts(normalize=True) * 100

# Plot the graph.
fig, ax = plt.subplots(1,1,figsize=(7,7))
ax.bar(percentages.index, percentages, color='orange')
```

```
ax.set_title("Percentage of type of toxicity in comments where sexual␣
↪orientation references are made")
ax.set_ylabel("Percentage")
ax.set_xlabel("Type of toxicity")
ax.set_ylim([0,100])
plt.show()
```



[ ]:
```
import matplotlib.pyplot as plt

# Define a function to get the percentage of each type of toxicity
def get_toxicity_percentages(df):
    num_comments = len(df)
    percentages = {}
    for toxicity_type in [ 'severe_toxicity', 'obscene', 'threat', 'insult',␣
↪'identity_attack']:
        num_toxic = len(df[df[toxicity_type] > 0.5])
        percentages[toxicity_type] = num_toxic / num_comments * 100
    return percentages
```

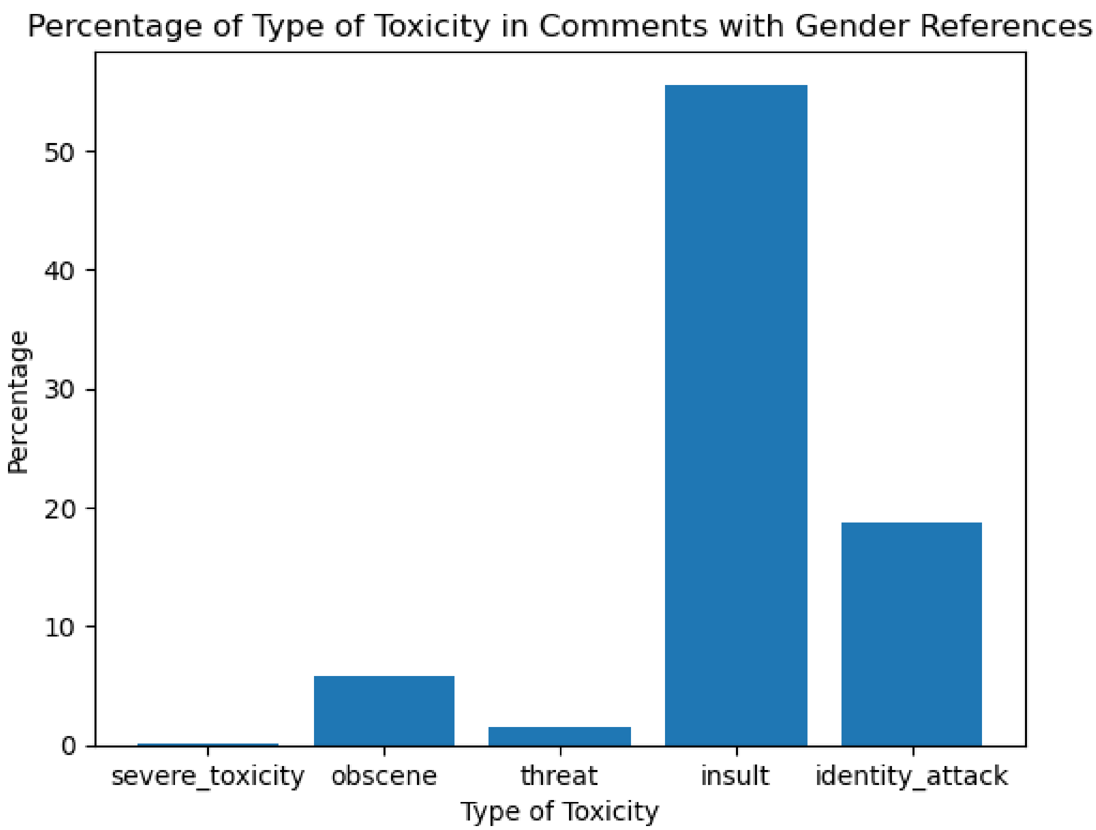
```
# Filter the data to only include comments with sexual orientation references␣
↪and that are toxic
cond = (train_df['bisexual'] > 0.5) | (train_df['heterosexual'] > 0.5) |␣
↪(train_df['homosexual_gay_or_lesbian'] > 0.5) |␣
↪(train_df['other_sexual_orientation'] > 0.5)
temp = train_df[cond]
temp = temp[temp['target'] > 0.5]

# Calculate the percentage of each type of toxicity in the filtered data
toxicity_percentages = get_toxicity_percentages(temp)

# Plot a bar chart showing the percentage of each type of toxicity
plt.bar(toxicity_percentages.keys(), toxicity_percentages.values())
plt.xlabel('Type of Toxicity')
plt.ylabel('Percentage')
plt.title('Percentage of Type of Toxicity in Comments with Sexual Orientation␣
↪References')
plt.show()
```



We see from the plot that the toxic comments where sexual orientation references are made are mostly used for identity attacks.

[ ]:

```python
import matplotlib.pyplot as plt

# Define a function to get the percentage of each type of toxicity
def get_toxicity_percentages(df):
    num_comments = len(df)
    percentages = {}
    for toxicity_type in [ 'severe_toxicity', 'obscene', 'threat', 'insult',␣
 ↪'identity_attack']:
        num_toxic = len(df[df[toxicity_type] > 0.5])
        percentages[toxicity_type] = num_toxic / num_comments * 100
    return percentages

# Filter the data to only include comments with gender references and that are␣
 ↪toxic
cond = (train_df['male'] > 0.5) | (train_df['female'] > 0.5) |␣
 ↪(train_df['transgender'] > 0.5) | (train_df['other_gender'] > 0.5)
temp = train_df[cond]
temp = temp[temp['target'] > 0.5]

# Calculate the percentage of each type of toxicity in the filtered data
toxicity_percentages = get_toxicity_percentages(temp)

# Plot a bar chart showing the percentage of each type of toxicity
plt.bar(toxicity_percentages.keys(), toxicity_percentages.values())
plt.xlabel('Type of Toxicity')
plt.ylabel('Percentage')
plt.title('Percentage of Type of Toxicity in Comments with Gender References')
plt.show()
```



From the plot we see that the toxic comments which involve words like male, female etc are insults.

1.6 4. Features generated by users feedback:

- funny
- sad
- wow
- likes
- disagree

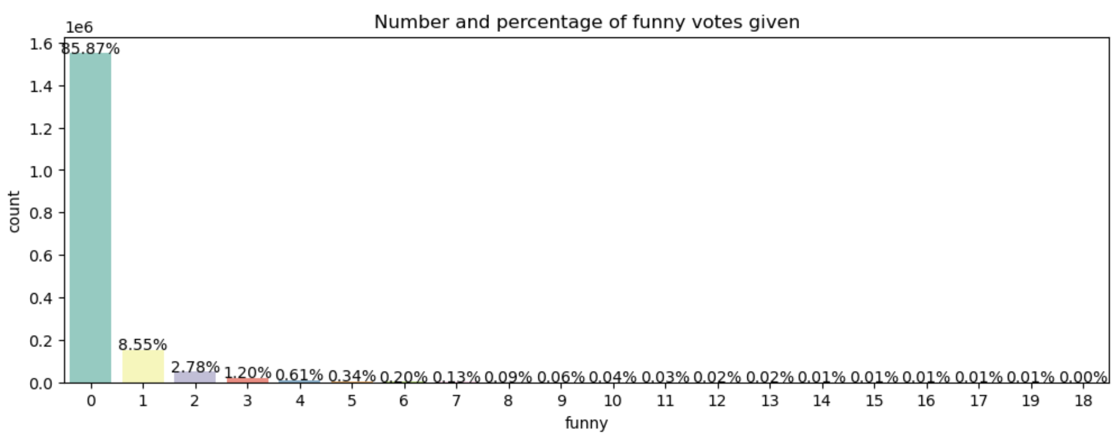
[ ]:
```
'''
This block of code will result in error for the following graphs

def plot_count(feature, title, data, size=1):
    f, ax = plt.subplots(1,1, figsize=(4*size,4))
    total = float(len(data))
    g = sns.countplot(data[feature], order = data[feature].value_counts().
 ↪index[:20], palette='Set3')
    g.set_title("Number and percentage of {}".format(title))
    for p in ax.patches:
```

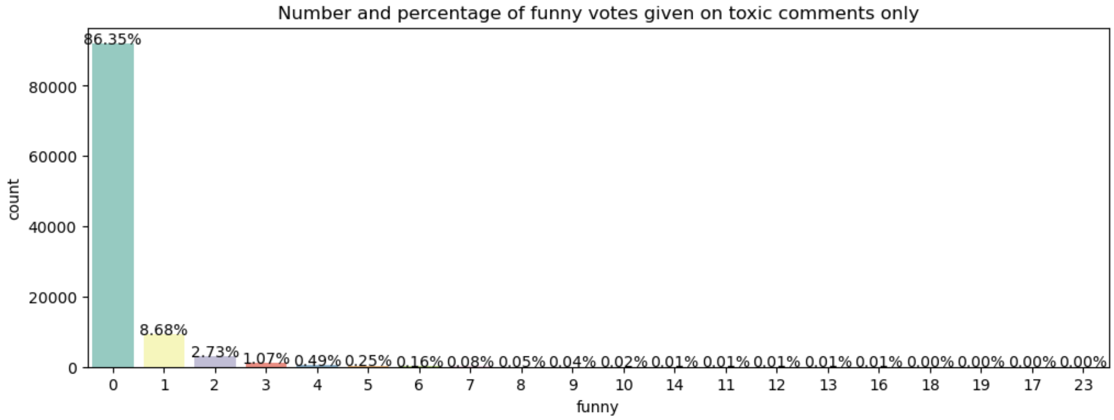
```
        height = p.get_height()
        ax.text(p.get_x()+p.get_width()/2.,
                height + 3,
                '{:1.2f}%'.format(100*height/total),
                ha="center")
    plt.show()   '''
```

[ ]: ' \nThis block of code will result in error for the following graphs\n\ndef plot_count(feature, title, data, size=1):\n   f, ax = plt.subplots(1,1, figsize=(4*size,4))\n   total = float(len(data))\n   g = sns.countplot(data[feature], order = data[feature].value_counts().index[:20], palette=\'Set3\')\n   g.set_title("Number and percentage of {}".format(title))\n   for p in ax.patches:\n   height = p.get_height()\n ax.text(p.get_x()+p.get_width()/2.,\n   height + 3,\n \'{:1.2f}%\'.format(100*height/total),\n   ha="center") \n plt.show()   '

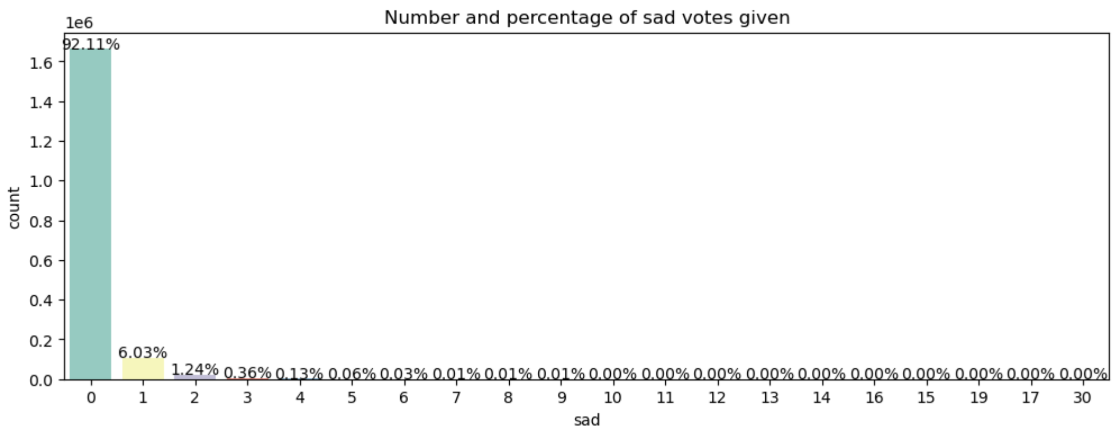
[ ]:
```
def plot_count(feature, title, data, size=1):
    f, ax = plt.subplots(1,1, figsize=(4*size,4))
    total = float(len(data))
    g = sns.countplot(x=feature, data=data, order=data[feature].value_counts().
 ↪index[:20], palette='Set3')
    g.set_title("Number and percentage of {}".format(title))
    for p in ax.patches:
        height = p.get_height()
        ax.text(p.get_x()+p.get_width()/2., height + 3, '{:1.2f}%'.
 ↪format(100*height/total),ha="center")
    plt.show()
```

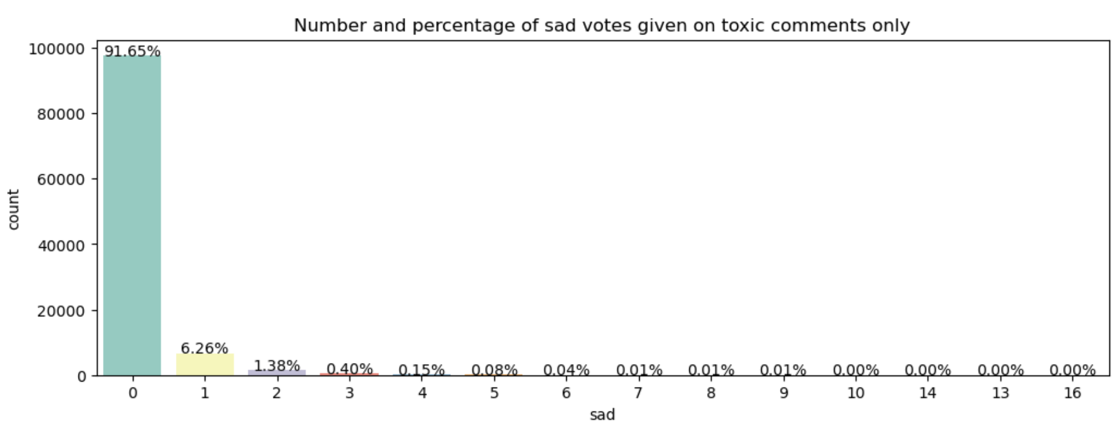
[ ]:
```
plot_count('funny','funny votes given', train_df, 3)
plot_count('funny', 'funny votes given on toxic comments only',␣
 ↪train_df[train_df['target'] > 0.5], 3)
```



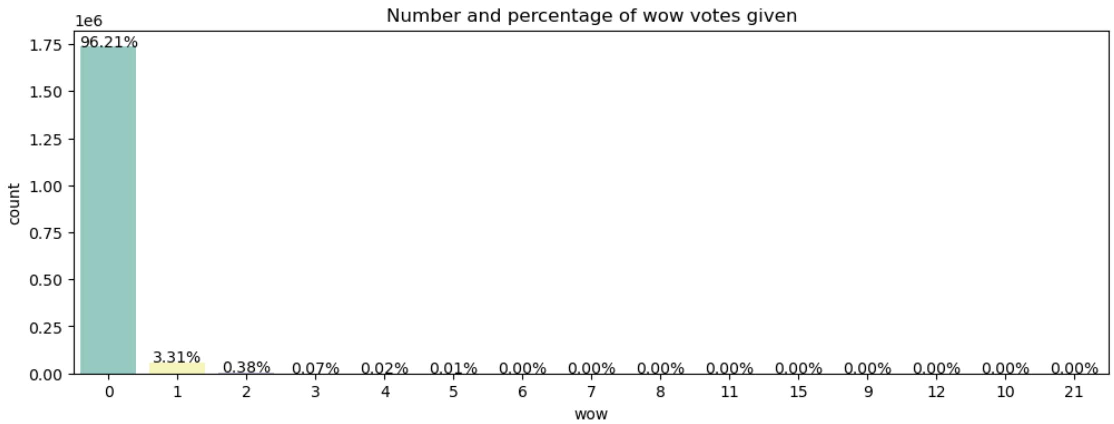


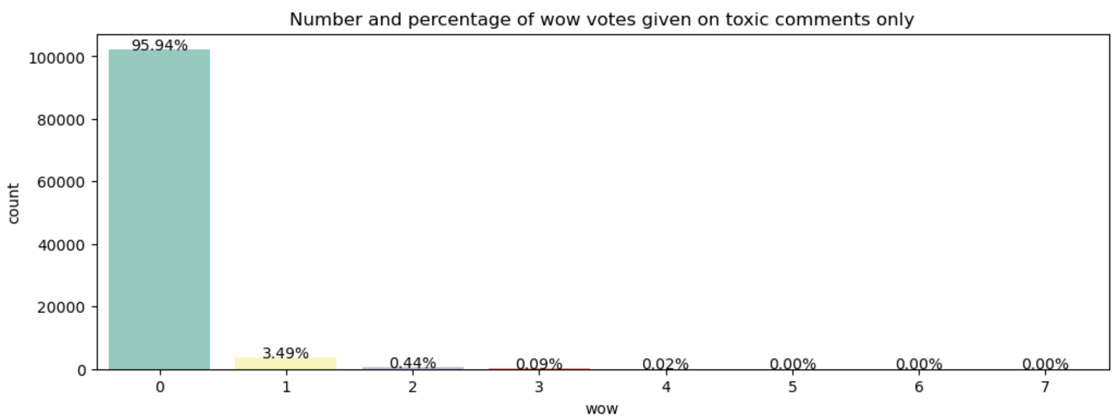
[ ]:
```
plot_count('sad','sad votes given', train_df, 3)
plot_count('sad', 'sad votes given on toxic comments only',␣
↪train_df[train_df['target'] > 0.5], 3)
```



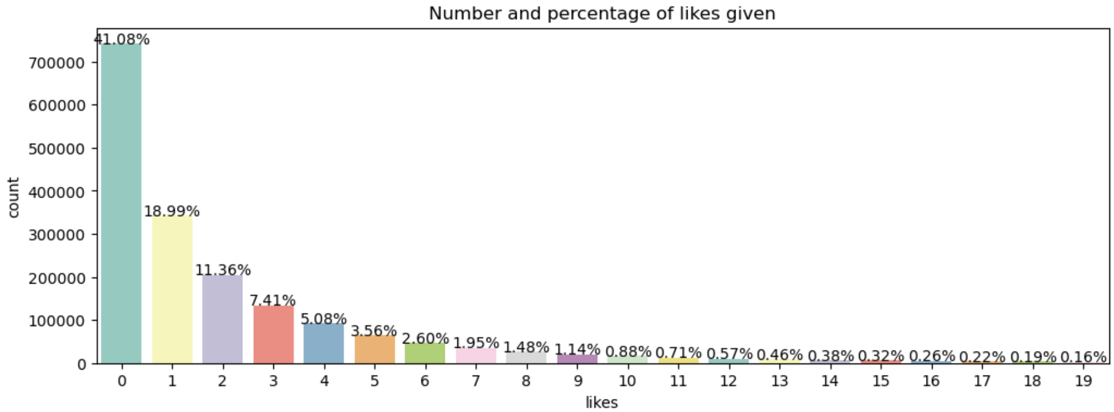


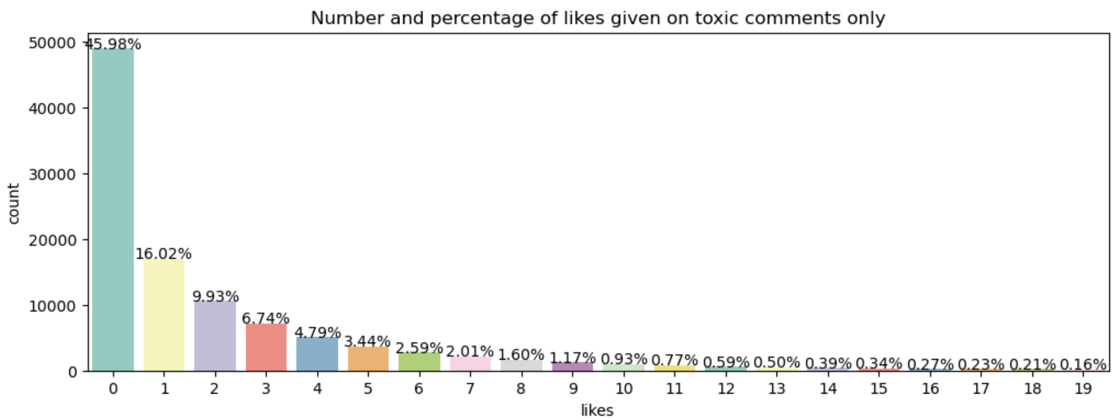
[ ]:
```
plot_count('wow','wow votes given', train_df, 3)
plot_count('wow', 'wow votes given on toxic comments only',␣
↪train_df[train_df['target'] > 0.5], 3)
```



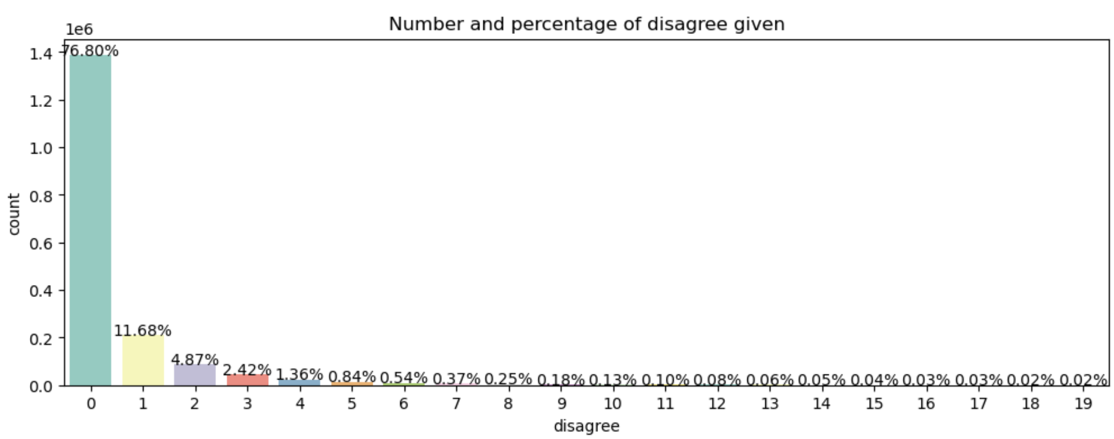


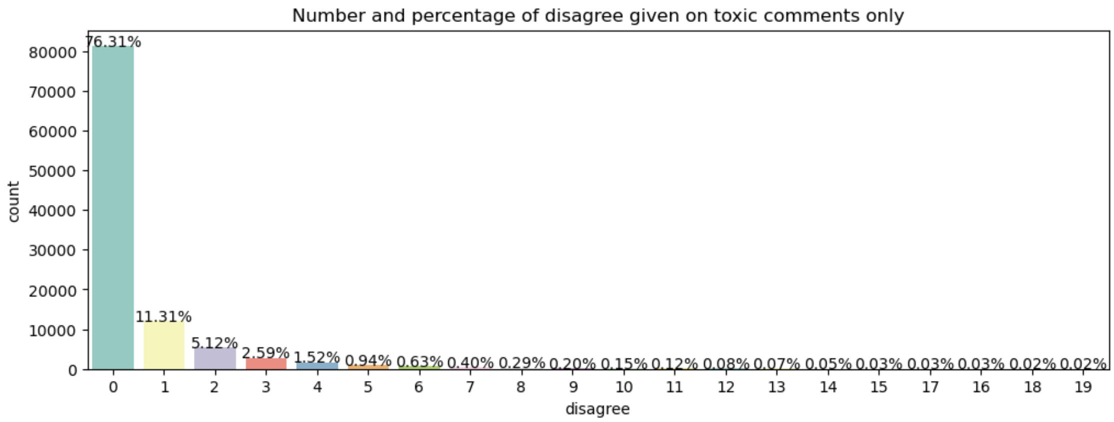
[ ]:
```
plot_count('likes','likes given', train_df, 3)
plot_count('likes', 'likes given on toxic comments only',␣
↪train_df[train_df['target'] > 0.5], 3)
```





[ ]:
```
plot_count('disagree','disagree given', train_df, 3)
plot_count('disagree', 'disagree given on toxic comments only',␣
 ↪train_df[train_df['target'] > 0.5], 3)
```

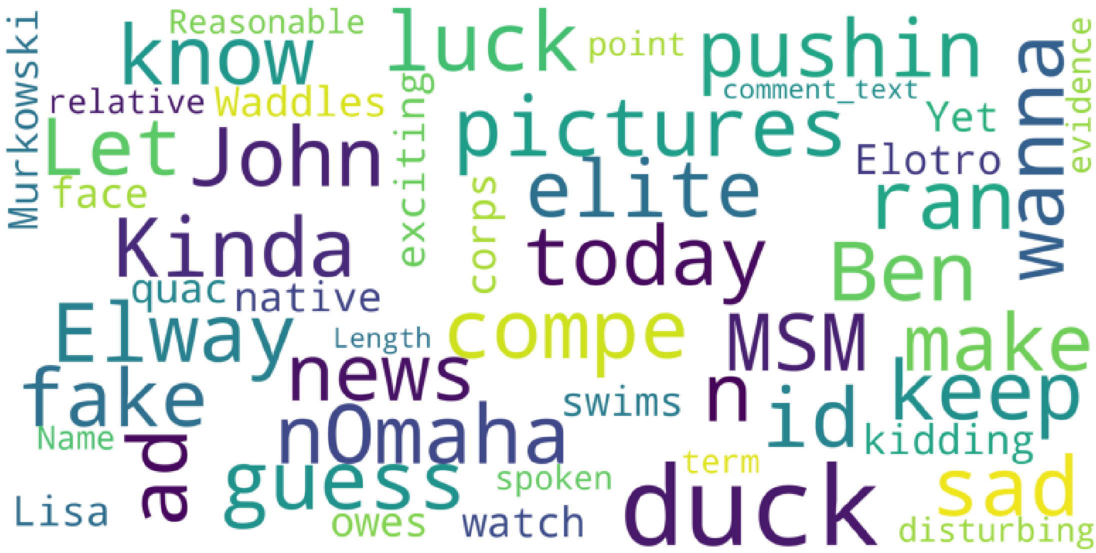




1.7 5. Comments_text Feature:

[ ]:
```
stpwrds = set(STOPWORDS)

def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stpwrds,
        max_words=50,
        max_font_size=40,
        scale=5,
        random_state=1
    ).generate(str(data))

    fig = plt.figure(1, figsize=(10,10))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()
```

[ ]:
```
show_wordcloud(train_df['comment_text'].sample(20000), title = 'Prevalent words␣
↪in comments - train data')
```



Prevalent words in comments - train data

```
[ ]: show_wordcloud(train_df.loc[train_df['insult'] > 0.75]['comment_text'].
     ↪sample(20000),
                   title = 'Prevalent comments with insult score > 0.75')
```

Prevalent comments with insult score > 0.75

[ ]:
```python
show_wordcloud(train_df.loc[train_df['threat'] > 0.75]['comment_text'],
               title = 'Prevalent words in comments with threat score > 0.75')
```

Prevalent words in comments with threat score > 0.75

[ ]:
```python
show_wordcloud(train_df.loc[train_df['obscene'] > 0.75]['comment_text'],
               title = 'Prevalent words in comments with obscene score > 0.75')
```

Prevalent words in comments with obscene score > 0.75

```
[ ]: show_wordcloud(train_df.loc[train_df['target'] > 0.75]['comment_text'],
                title = 'Prevalent words in comments with target score > 0.75')
```

Prevalent words in comments with target score > 0.75

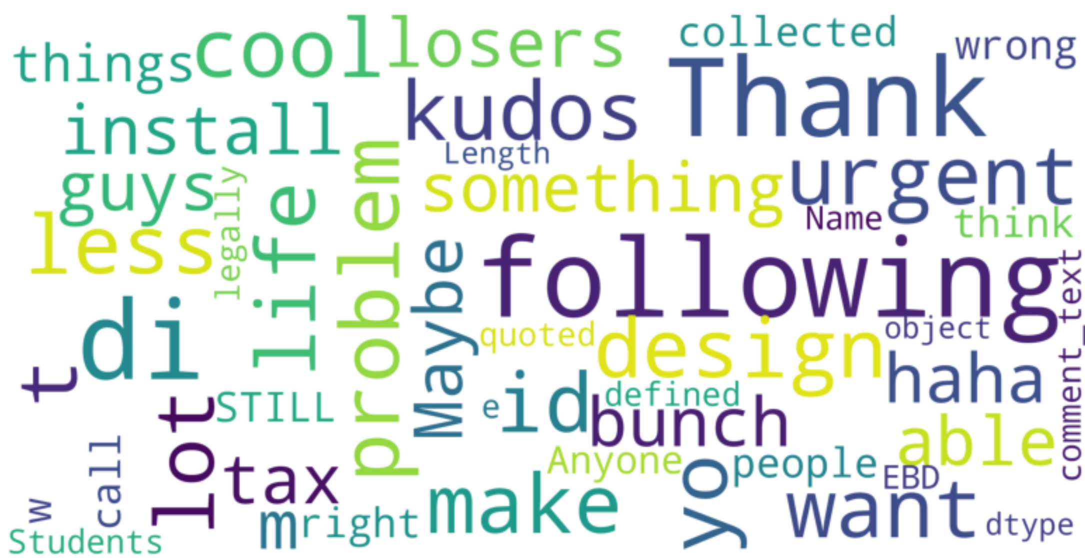
```
[ ]: show_wordcloud(train_df.loc[train_df['target'] < 0.25]['comment_text'],
                title = 'Prevalent words in comments with target score < 0.25')
```

Prevalent words in comments with target score < 0.25

```
[ ]: show_wordcloud(train_df.loc[train_df['obscene']< 0.25]['comment_text'],
                    title = 'Prevalent words in comments with obscene score < 0.25')
```

Prevalent words in comments with obscene score < 0.25

```
[ ]: show_wordcloud(train_df.loc[train_df['threat'] < 0.25]['comment_text'],
                    title = 'Prevalent words in comments with threat score < 0.25')
```



Prevalent words in comments with threat score < 0.25

```
[ ]: show_wordcloud(train_df.loc[train_df['insult'] < 0.25]['comment_text'].
      ↪sample(20000),
                    title = 'Prevalent comments with insult score < 0.25')
```

Prevalent comments with insult score < 0.25

1.8 Preprocessing Text and Train-Test Split:

```
[ ]: stemmer = SnowballStemmer("english")
     stop_words = set(stopwords.words('english'))
     def preprocess(text_string):
         text_string = text_string.lower() # Convert everything to lower case.
         text_string = re.sub('[^A-Za-z0-9]+', ' ', text_string) # Remove special␣
      ↪characters and punctuations

         x = text_string.split()
         new_text = []

         for word in x:
             if word not in stop_words:
                 new_text.append(stemmer.stem(word))

         text_string = ' '.join(new_text)
         return text_string
```

[ ]:
```
train_df['preprocessed_text'] = train_df['comment_text'].apply(preprocess)
```

[ ]:
```
train_df.head()
```

[ ]:
```
         target                                      comment_text  \
id
59848  0.000000  This is so cool. It's like, 'would you want yo…
59849  0.000000  Thank you!! This would make my life a lot less…
59852  0.000000  This is such an urgent design problem; kudos t…
59855  0.000000  Is this something I'll be able to install on m…
59856  0.893617              haha you guys are a bunch of losers.

       severe_toxicity  obscene  identity_attack   insult  threat  asian  \
id
59848         0.000000      0.0         0.000000  0.00000     0.0    NaN
59849         0.000000      0.0         0.000000  0.00000     0.0    NaN
59852         0.000000      0.0         0.000000  0.00000     0.0    NaN
59855         0.000000      0.0         0.000000  0.00000     0.0    NaN
59856         0.021277      0.0         0.021277  0.87234     0.0    0.0

       atheist  bisexual  …    rating  funny  wow  sad  likes  disagree  \
id                        …
59848      NaN       NaN  …  rejected      0    0    0      0         0
59849      NaN       NaN  …  rejected      0    0    0      0         0
59852      NaN       NaN  …  rejected      0    0    0      0         0
59855      NaN       NaN  …  rejected      0    0    0      0         0
59856      0.0       0.0  …  rejected      0    0    0      1         0

       sexual_explicit  identity_annotator_count  toxicity_annotator_count  \
id
59848              0.0                         0                         4
59849              0.0                         0                         4
59852              0.0                         0                         4
59855              0.0                         0                         4
59856              0.0                         4                        47

                                   preprocessed_text
id
59848  cool like would want mother read realli great …
59849  thank would make life lot less anxieti induc k…
59852            urgent design problem kudo take impress
59855                      someth abl instal site releas
59856                             haha guy bunch loser

[5 rows x 45 columns]
```

[ ]:
```
test_df['preprocessed_text'] = test_df['comment_text'].apply(preprocess)
```

[ ]:
```python
feature = train_df[['preprocessed_text']]
output = train_df[['target']]
X_train, X_cv, y_train, y_cv = train_test_split(feature, output)

print(X_train.shape)
print(X_cv.shape)
print(y_train.shape)
print(y_cv.shape)
```

```
(1353655, 1)
(451219, 1)
(1353655, 1)
(451219, 1)
```

[ ]:
```python
X_train.head()
```

[ ]:
```
                                          preprocessed_text
id
5142661  think peopl vote airport 3 better ever travel …
6330562                     redeem featur presid get real
883306             particular poster resid world altern fact
5172542    dept health handl rat problem downtown honolulu
775347               enjoy retir know extra firework tonight
```

[ ]:
```python
X_cv.head()
```

[ ]:
```
                                          preprocessed_text
id
6042743                                                weed
836923   harper govern approv northern gateway condit a…
5077104  mr troy payn pleas tell citizen best feloni ar…
868618   thought devalu currenc canadian peso play trad…
6274855  realli watch charad fair close seen one iota p…
```

[ ]:
```python
X_test = test_df[['preprocessed_text']]
X_test.head()
```

[ ]:
```
                                          preprocessed_text
id
7097320              integr mean pay debt appli presid trump
7097321                   malfeas administr board wast money
7097322  rmiller101 spoken like true elitist look bud a…
7097323  paul thank kind word inde strong belief hide b…
7097324  sorri miss high school eisenhow sent troop vie…
```

[ ]:
```python
# Saving the files to csv so that we dont need to preprocess again.
X_train.to_pickle('X_train.pkl')
X_cv.to_pickle('X_cv.pkl')
```

```
X_test.to_pickle('X_test.pkl')
y_train.to_pickle('y_train.pkl')
y_cv.to_pickle('y_cv.pkl')
```

1.9 Training Models:

[ ]:
```
# To load the csv files:
X_train = pd.read_pickle('X_train.pkl')
X_cv = pd.read_pickle('X_cv.pkl')
X_test = pd.read_pickle('X_test.pkl')
y_train = pd.read_pickle('y_train.pkl')
y_cv = pd.read_pickle('y_cv.pkl')
```

1.9.1 1. Bag of Words (BoW):

[ ]:
```
cnt_vec = CountVectorizer(ngram_range=(1,2), max_features=30000)
vectorizer = CountVectorizer()
bow_train = cnt_vec.fit_transform(X_train['preprocessed_text'])
bow_cv = cnt_vec.transform(X_cv['preprocessed_text'])
bow_test = cnt_vec.transform(X_test['preprocessed_text'])

print(bow_train.shape)
print(bow_cv.shape)
print(bow_test.shape)
```

```
(1353655, 30000)
(451219, 30000)
(97320, 30000)
```

1.1 SGDRegressor:

1.1.1 Hyperparameter Tuning:

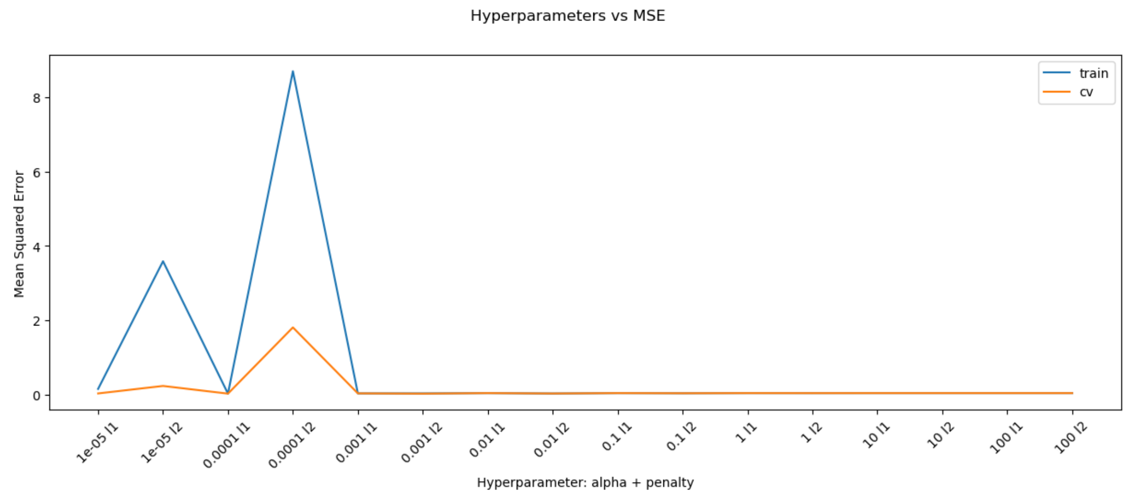
[ ]:
```
# Performing hyperparameter tuning:
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
penalty = ['l1', 'l2']
xticks = []
tr_errors = []
cv_errors = []
best_model = None
best_error = 100
for a in alpha:
    for p in penalty:
        xticks.append(str(a) + ' ' + p)
        print(str(a) + ' ' + p + " :")

        model = SGDRegressor(alpha=a, penalty=p)
        model.fit(bow_train, y_train) # Train
```

```
        preds = model.predict(bow_train) # Get predictions
        err = mean_squared_error(y_train['target'], preds) # Calculate error on␣
↪trainset
        tr_errors.append(err)
        print("Mean Squared Error on train set: ", err)

        preds = model.predict(bow_cv) # Get predictions on CV set
        err = mean_squared_error(y_cv['target'], preds) # Calculate error on cv␣
↪set
        cv_errors.append(err)
        print("Mean Squared Error on cv set: ", err)

        if err < best_error: # Get best model trained
            best_error = err
            best_model = model

        print("*"*50)
```

```
1e-05 l1 :
Mean Squared Error on train set:  0.15065175341880893
Mean Squared Error on cv set:  0.02957816498385944
**************************************************
1e-05 l2 :
Mean Squared Error on train set:  3.58620679928222
Mean Squared Error on cv set:  0.23109845447757335
**************************************************
0.0001 l1 :
Mean Squared Error on train set:  0.02451641343290528
Mean Squared Error on cv set:  0.024403824984518645
**************************************************
0.0001 l2 :
Mean Squared Error on train set:  8.698601247793816
Mean Squared Error on cv set:  1.805904495383908
**************************************************
0.001 l1 :
Mean Squared Error on train set:  0.03147213252585549
Mean Squared Error on cv set:  0.03132296778787472
**************************************************
0.001 l2 :
Mean Squared Error on train set:  0.030386499583444852
Mean Squared Error on cv set:  0.0239018925547874
**************************************************
0.01 l1 :
Mean Squared Error on train set:  0.038891616641619615
Mean Squared Error on cv set:  0.038680562200838355
**************************************************
```

```
0.01 l2 :
Mean Squared Error on train set:  0.02808646424847104
Mean Squared Error on cv set:  0.02788989381632082
**************************************************
0.1 l1 :
Mean Squared Error on train set:  0.038891628791575435
Mean Squared Error on cv set:  0.0386805911574697
**************************************************
0.1 l2 :
Mean Squared Error on train set:  0.034920839345738766
Mean Squared Error on cv set:  0.03473634121857183
**************************************************
1 l1 :
Mean Squared Error on train set:  0.03889160634138984
Mean Squared Error on cv set:  0.0386805279850375
**************************************************
1 l2 :
Mean Squared Error on train set:  0.03805769761265688
Mean Squared Error on cv set:  0.03784948514651869
**************************************************
10 l1 :
Mean Squared Error on train set:  0.038891606678717744
Mean Squared Error on cv set:  0.03868052958059795
**************************************************
10 l2 :
Mean Squared Error on train set:  0.03879215186100527
Mean Squared Error on cv set:  0.038580937423791054
**************************************************
100 l1 :
Mean Squared Error on train set:  0.038891622058010306
Mean Squared Error on cv set:  0.038680575810107205
**************************************************
100 l2 :
Mean Squared Error on train set:  0.038887526313509745
Mean Squared Error on cv set:  0.03867670652884471
**************************************************
```

[ ]:
```
plt.figure(figsize=(15,5))
plt.suptitle("Hyperparameters vs MSE")
plt.plot(range(len(alpha) * len(penalty)), tr_errors)
plt.plot(range(len(alpha) * len(penalty)), cv_errors)
plt.legend(['train', 'cv'])
plt.xticks(range(len(alpha) * len(penalty)), xticks, rotation=45)
plt.xlabel('Hyperparameter: alpha + penalty')
plt.ylabel('Mean Squared Error')
plt.show()
```



[ ]:
```
# Getting the best model parameters:
best_model.get_params()
```

[ ]:
```
{'alpha': 0.001,
 'average': False,
 'early_stopping': False,
 'epsilon': 0.1,
 'eta0': 0.01,
 'fit_intercept': True,
 'l1_ratio': 0.15,
 'learning_rate': 'invscaling',
 'loss': 'squared_error',
 'max_iter': 1000,
 'n_iter_no_change': 5,
 'penalty': 'l2',
 'power_t': 0.25,
 'random_state': None,
 'shuffle': True,
 'tol': 0.001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

1.1.2 Feature Importance:

[ ]:
```
# Printing the 20 most important features/words which contribute to a comment␣
↪being toxic.
feat_names = cnt_vec.get_feature_names_out()
weights = best_model.coef_
df = pd.DataFrame(data=weights, columns=['weights'], index=feat_names)
df.sort_values("weights", ascending=False).iloc[0:20,:]
```

```python
# Printing the 20 most important features/words which contribute to a comment␣
 ↪being toxic.
'''feat_names = cnt_vec.get_feature_names_out()
weights = best_model.feature_importances_
df = pd.DataFrame(data=weights, columns=['weights'], index=feat_names)
df.sort_values("weights", ascending=False).iloc[0:20,:]
'''
```

[ ]:
```
'feat_names = cnt_vec.get_feature_names_out()\nweights =
best_model.feature_importances_\ndf = pd.DataFrame(data=weights,
columns=[\'weights\'], index=feat_names)\ndf.sort_values("weights",
ascending=False).iloc[0:20,:]\n'
```

[ ]:
```python
# 20 most important features/words which contribute to comment being non-toxic.
df.sort_values("weights", ascending=True).iloc[0:20,:]
```

[ ]:

	weights
stupid stupid	-0.057107
left left	-0.033664
black white	-0.033161
fool peopl	-0.026811
ignor fact	-0.025922
knee jerk	-0.022985
great articl	-0.022486
winner loser	-0.021944
black market	-0.021253
thank	-0.021156
white hous	-0.020638
america great	-0.019288
mass shoot	-0.017990
make america	-0.017966
great job	-0.017429
awesom	-0.017378
peopl time	-0.016992
winner	-0.016474
men women	-0.016266
well said	-0.016020

1.2 Decision Trees:

1.2.1 Hyperparameter Tuning:

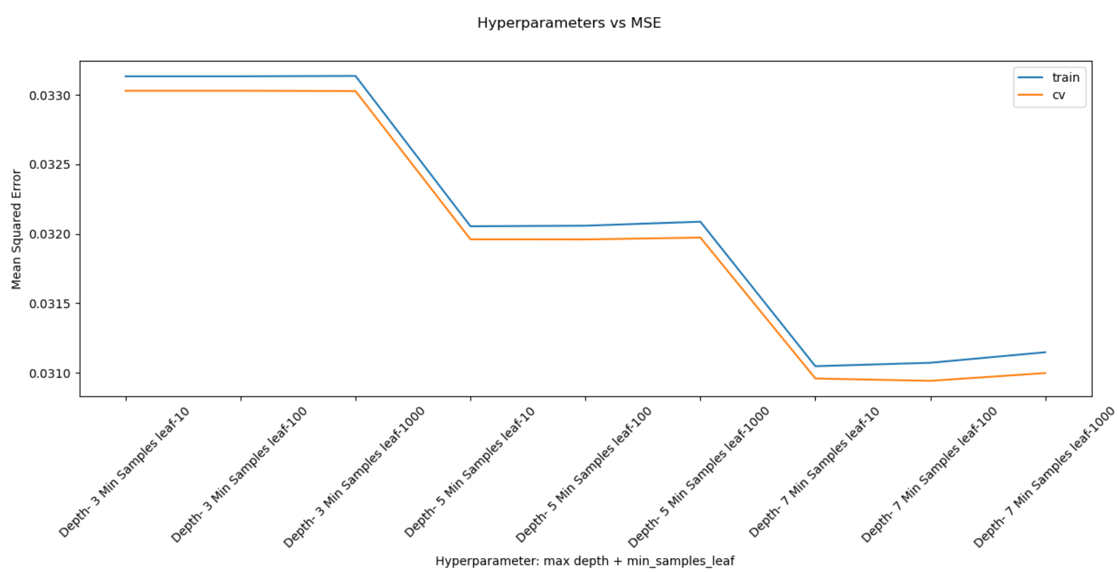
[ ]:
```python
# Performing hyperparameter tuning:
max_depth = [3, 5, 7]
min_samples = [10, 100, 1000]
xticks = []
tr_errors = []
```

```
cv_errors = []
best_model = None
best_error = 100
for d in max_depth:
    for samp in min_samples:
        xticks.append("Depth- " + str(d) + ' Min Samples leaf-' + str(samp))
        print("Depth- " + str(d) + ' Min Samples leaf-' + str(samp) + " :")

        model = DecisionTreeRegressor(max_depth=d, min_samples_leaf=samp)
        model.fit(bow_train, y_train) # Train

        preds = model.predict(bow_train) # Get predictions
        err = mean_squared_error(y_train['target'], preds) # Calculate error on␣
 ↪trainset
        tr_errors.append(err)
        print("Mean Squared Error on train set: ", err)

        preds = model.predict(bow_cv) # Get predictions on CV set
        err = mean_squared_error(y_cv['target'], preds) # Calculate error on cv␣
 ↪set
        cv_errors.append(err)
        print("Mean Squared Error on cv set: ", err)

        if err < best_error: # Get best model trained
            best_error = err
            best_model = model

        print("*"*50)
```

```
Depth- 3 Min Samples leaf-10 :
Mean Squared Error on train set:  0.03313298462875006
Mean Squared Error on cv set:  0.03302894924671027
**************************************************
Depth- 3 Min Samples leaf-100 :
Mean Squared Error on train set:  0.03313298462875007
Mean Squared Error on cv set:  0.03302894924671028
**************************************************
Depth- 3 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.03313568044338157
Mean Squared Error on cv set:  0.03302680074595134
**************************************************
Depth- 5 Min Samples leaf-10 :
Mean Squared Error on train set:  0.03205373667797802
Mean Squared Error on cv set:  0.03195976186233563
**************************************************
Depth- 5 Min Samples leaf-100 :
Mean Squared Error on train set:  0.03205768849162905
```

```
Mean Squared Error on cv set:  0.03195892592075731
**************************************************
Depth- 5 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.03208649172875205
Mean Squared Error on cv set:  0.031972115313431054
**************************************************
Depth- 7 Min Samples leaf-10 :
Mean Squared Error on train set:  0.03104633689989069
Mean Squared Error on cv set:  0.03095792988300593
**************************************************
Depth- 7 Min Samples leaf-100 :
Mean Squared Error on train set:  0.031071084588769577
Mean Squared Error on cv set:  0.030941043832055908
**************************************************
Depth- 7 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.031146787902501763
Mean Squared Error on cv set:  0.030997035555192114
**************************************************
```

[ ]:
```
plt.figure(figsize=(15,5))
plt.suptitle("Hyperparameters vs MSE")
plt.plot(range(len(max_depth) * len(min_samples)), tr_errors)
plt.plot(range(len(max_depth) * len(min_samples)), cv_errors)
plt.legend(['train', 'cv'])
plt.xticks(range(len(max_depth) * len(min_samples)), xticks, rotation=45)
plt.xlabel('Hyperparameter: max depth + min_samples_leaf')
plt.ylabel('Mean Squared Error')
plt.show()
```



```
[ ]: # Best models parameters:
best_model.get_params()
```

```
[ ]: {'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': 7,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 100,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

1.2.2 Feature Importance:

```
[ ]: weights = best_model.feature_importances_
df = pd.DataFrame(data=weights, columns=['weights'], index=feat_names)
df.sort_values("weights", ascending=False).iloc[0:20,:]
```

[ ]:

	weights
stupid	0.397572
idiot	0.262935
pathet	0.070121
fool	0.068156
moron	0.062549
white	0.058445
hypocrit	0.054983
racist	0.005800
one	0.004606
would	0.004120
year	0.003003
peopl	0.001907
even	0.001119
time	0.001109
also	0.000904
fool peopl	0.000459
state	0.000451
work	0.000417
get	0.000381
use	0.000237

1.9.2 2. Term Frequency - Inverse Document Frequency (TFIDF) :

[ ]:
```
tfidf_vec = TfidfVectorizer(ngram_range=(1,2), max_features=30000)
tfidf_train = tfidf_vec.fit_transform(X_train['preprocessed_text'])
tfidf_cv = tfidf_vec.transform(X_cv['preprocessed_text'])
tfidf_test = tfidf_vec.transform(X_test['preprocessed_text'])

print(tfidf_train.shape)
print(tfidf_cv.shape)
print(tfidf_test.shape)
```

```
(1353655, 30000)
(451219, 30000)
(97320, 30000)
```

2.1 SGDRegressor:

2.1.1 Hyperparameter Tuning:

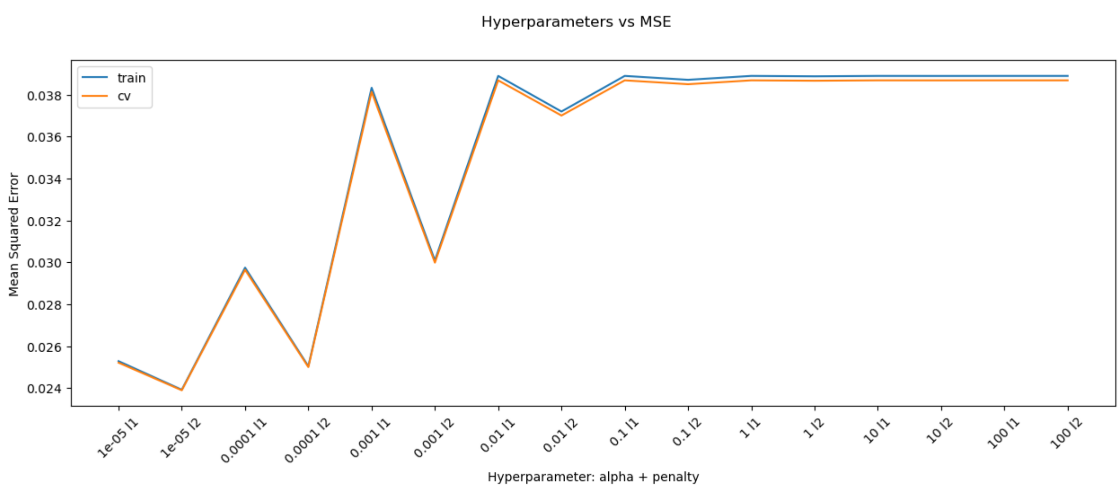
[ ]:
```
# Performing hyperparameter tuning:
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
penalty = ['l1', 'l2']
xticks = []
tr_errors = []
cv_errors = []
best_model = None
best_error = 100
for a in alpha:
    for p in penalty:
        xticks.append(str(a) + ' ' + p)
        print(str(a) + ' ' + p + " :")

        model = SGDRegressor(alpha=a, penalty=p)
        model.fit(tfidf_train, y_train) # Train

        preds = model.predict(tfidf_train) # Get predictions
        err = mean_squared_error(y_train['target'], preds) # Calculate error on␣
 ↪trainset
        tr_errors.append(err)
        print("Mean Squared Error on train set: ", err)

        preds = model.predict(tfidf_cv) # Get predictions on CV set
        err = mean_squared_error(y_cv['target'], preds) # Calculate error on cv␣
 ↪set
        cv_errors.append(err)
        print("Mean Squared Error on cv set: ", err)

        if err < best_error: # Get best model trained
```

```
            best_error = err
            best_model = model

    print("*"*50)
```

```
1e-05 l1 :
Mean Squared Error on train set:  0.025280435919366358
Mean Squared Error on cv set:  0.025206813046917097
**************************************************
1e-05 l2 :
Mean Squared Error on train set:  0.023915958092999283
Mean Squared Error on cv set:  0.023890186574119305
**************************************************
0.0001 l1 :
Mean Squared Error on train set:  0.029742153869101385
Mean Squared Error on cv set:  0.02963995922849148
**************************************************
0.0001 l2 :
Mean Squared Error on train set:  0.025036866281302077
Mean Squared Error on cv set:  0.024996408448019095
**************************************************
0.001 l1 :
Mean Squared Error on train set:  0.03832678793634069
Mean Squared Error on cv set:  0.038132931393494426
**************************************************
0.001 l2 :
Mean Squared Error on train set:  0.03008509394487517
Mean Squared Error on cv set:  0.029980521526456107
**************************************************
0.01 l1 :
Mean Squared Error on train set:  0.038891613950790785
Mean Squared Error on cv set:  0.03868055477546209
**************************************************
0.01 l2 :
Mean Squared Error on train set:  0.03719261296184899
Mean Squared Error on cv set:  0.037001063183267145
**************************************************
0.1 l1 :
Mean Squared Error on train set:  0.038891604050478694
Mean Squared Error on cv set:  0.03868050507886583
**************************************************
0.1 l2 :
Mean Squared Error on train set:  0.03870457202467621
Mean Squared Error on cv set:  0.038495512677272706
**************************************************
1 l1 :
Mean Squared Error on train set:  0.038891612015914435
Mean Squared Error on cv set:  0.0386804821944143
```

```
**************************************************
1 l2 :
Mean Squared Error on train set:  0.0388726749996241
Mean Squared Error on cv set:  0.03866171532518106
**************************************************
10 l1 :
Mean Squared Error on train set:  0.03889163390108171
Mean Squared Error on cv set:  0.03868060204064716
**************************************************
10 l2 :
Mean Squared Error on train set:  0.0388896989339024
Mean Squared Error on cv set:  0.03867864331598374
**************************************************
100 l1 :
Mean Squared Error on train set:  0.0388916098883576
Mean Squared Error on cv set:  0.038680542118703594
**************************************************
100 l2 :
Mean Squared Error on train set:  0.03889145202684925
Mean Squared Error on cv set:  0.03868034130501608
**************************************************
```

[ ]:
```
plt.figure(figsize=(15,5))
plt.suptitle("Hyperparameters vs MSE")
plt.plot(range(len(alpha) * len(penalty)), tr_errors)
plt.plot(range(len(alpha) * len(penalty)), cv_errors)
plt.legend(['train', 'cv'])
plt.xticks(range(len(alpha) * len(penalty)), xticks, rotation=45)
plt.xlabel('Hyperparameter: alpha + penalty')
plt.ylabel('Mean Squared Error')
plt.show()
```



2.1.2 Feature Importance:

[ ]:
```
# Printing the 20 most important features/words which contribute to a comment␣
↪being toxic.
feat_names = tfidf_vec.get_feature_names_out()
weights = best_model.coef_
df = pd.DataFrame(data=weights, columns=['weights'], index=feat_names)
df.sort_values("weights", ascending=False).iloc[0:20,:]
```

[ ]:

	weights
stupid	1.573779
idiot	1.265893
fool	0.664075
ignor	0.614256
dumb	0.595596
pathet	0.594740
moron	0.572136
ridicul	0.568214
loser	0.564755
liar	0.524785
crap	0.509863
hypocrit	0.503545
racist	0.491712
white	0.483304
troll	0.447571
kill	0.443806
black	0.436339
silli	0.436286
clown	0.431758
damn	0.431321

[ ]:
```
# 20 most important features/words which contribute to comment being non-toxic.
df.sort_values("weights", ascending=True).iloc[0:20,:]
```

[ ]:

	weights
thank	-0.094503
interest	-0.083377
agre	-0.080200
stori	-0.078429
great	-0.071006
good	-0.070586
may	-0.070051
new	-0.067940
point	-0.067277
work	-0.066807
number	-0.066480
com	-0.065822
differ	-0.065689

```
chang     -0.065612
year      -0.064588
issu      -0.063781
articl    -0.062882
happen    -0.062636
http      -0.062501
provid    -0.060242
```

2.2 Decision Trees:

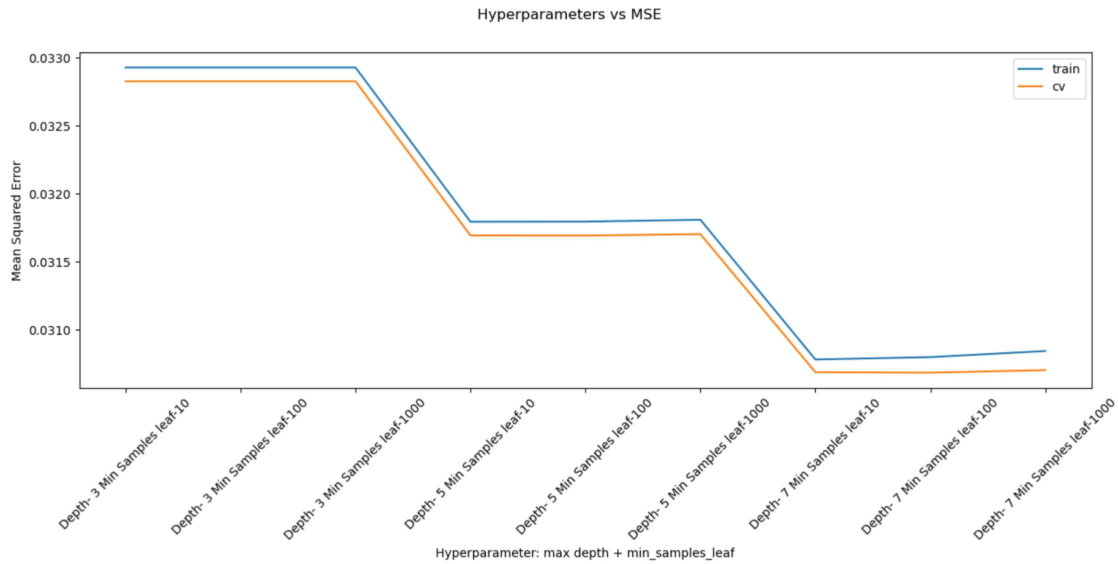
2.2.1 Hyperparameter Tuning:

[ ]:
```python
# Performing hyperparameter tuning:
max_depth = [3, 5, 7]
min_samples = [10, 100, 1000]
xticks = []
tr_errors = []
cv_errors = []
best_model = None
best_error = 100
for d in max_depth:
    for samp in min_samples:
        xticks.append("Depth- " + str(d) + ' Min Samples leaf-' + str(samp))
        print("Depth- " + str(d) + ' Min Samples leaf-' + str(samp) + " :")

        model = DecisionTreeRegressor(max_depth=d, min_samples_leaf=samp)
        model.fit(tfidf_train, y_train) # Train

        preds = model.predict(tfidf_train) # Get predictions
        err = mean_squared_error(y_train['target'], preds) # Calculate error on␣
 ↪trainset
        tr_errors.append(err)
        print("Mean Squared Error on train set: ", err)

        preds = model.predict(tfidf_cv) # Get predictions on CV set
        err = mean_squared_error(y_cv['target'], preds) # Calculate error on cv␣
 ↪set
        cv_errors.append(err)
        print("Mean Squared Error on cv set: ", err)

        if err < best_error: # Get best model trained
            best_error = err
            best_model = model

        print("*"*50)
```

```
Depth- 3 Min Samples leaf-10 :
Mean Squared Error on train set:  0.032927738478040314
```

```
Mean Squared Error on cv set:  0.032826573011590456
**************************************************
Depth- 3 Min Samples leaf-100 :
Mean Squared Error on train set:  0.03292773847804032
Mean Squared Error on cv set:  0.032826573011590456
**************************************************
Depth- 3 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.032927738478040314
Mean Squared Error on cv set:  0.032826573011590456
**************************************************
Depth- 5 Min Samples leaf-10 :
Mean Squared Error on train set:  0.03179432008144781
Mean Squared Error on cv set:  0.03169374494037711
**************************************************
Depth- 5 Min Samples leaf-100 :
Mean Squared Error on train set:  0.0317955063723966
Mean Squared Error on cv set:  0.03169278078097546
**************************************************
Depth- 5 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.031809025041135704
Mean Squared Error on cv set:  0.03170330674989337
**************************************************
Depth- 7 Min Samples leaf-10 :
Mean Squared Error on train set:  0.030782458379105846
Mean Squared Error on cv set:  0.030688973412743393
**************************************************
Depth- 7 Min Samples leaf-100 :
Mean Squared Error on train set:  0.030799726542875314
Mean Squared Error on cv set:  0.030685493052234895
**************************************************
Depth- 7 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.030844532409393416
Mean Squared Error on cv set:  0.03070411835721041
**************************************************
```

[ ]:
```
plt.figure(figsize=(15,5))
plt.suptitle("Hyperparameters vs MSE")
plt.plot(range(len(max_depth) * len(min_samples)), tr_errors)
plt.plot(range(len(max_depth) * len(min_samples)), cv_errors)
plt.legend(['train', 'cv'])
plt.xticks(range(len(max_depth) * len(min_samples)), xticks, rotation=45)
plt.xlabel('Hyperparameter: max depth + min_samples_leaf')
plt.ylabel('Mean Squared Error')
plt.show()
```



2.2.2 Feature Importance:

[ ]:
```
weights = best_model.feature_importances_
df = pd.DataFrame(data=weights, columns=['weights'], index=feat_names)
df.sort_values("weights", ascending=False).iloc[0:20,:]
```

[ ]:

	weights
stupid	0.407723
idiot	0.266989
pathet	0.071947
fool	0.071402
moron	0.063177
white	0.057472
hypocrit	0.053369
racist	0.005192
trump	0.000895
ignor	0.000618
peopl	0.000460
countri	0.000174
one	0.000115
thing	0.000111
like	0.000066
would	0.000066
know	0.000048
law	0.000044
state	0.000042
even	0.000035

1.9.3 3. Features for LSTM:

[ ]:
```
from tensorflow.keras.preprocessing import sequence

class LSTMFeaturization:

    def __init__(self):
        self.word_mapping = None
        self.total_words = None


    # Accepts a list of sentences and builds a vocabulary.
    def build_vocabulary(self, sentences):

        vocab = set()
        for x in sentences:
            for word in x.split():
                vocab.add(word)

        # Create a dictionary from vocabulary.
        vocab_dict = dict.fromkeys(vocab, 0)

        # Calculate count of each word..
        for x in sentences:
            for word in x.split():
                vocab_dict[word]+=1

        return vocab_dict



    # Accepts a dictionary (vocabulary) and gets the word number in dictionary␣
↪format
    def get_mapping(self, vocab_dict):

        # Get the number of each word into the corpus.
        k = []
        v = []
        for keys,val in vocab_dict.items():
            k.append(keys)
            v.append(val)

        kv = np.vstack((k,v)).T
        df = pd.DataFrame(columns=["Word","Count"], data=kv)
        df['Count'] = df['Count'].astype('int')

        # Sort the dataframe to get the largest count at first place
```

```
        df.sort_values(by=['Count'], ascending=False, inplace=True)

        # Give numbering to the most frequent word as 1 then next as 2 and so␣
↪on.
        df.reset_index(inplace=True)
        df['mapping'] = df.index + 1

        df.drop(columns=['index'], inplace=True)
        df.drop(columns=['Count'], inplace=True)

        # Convert to dictionary for easier processing.
        dictionary = dict(zip(df['Word'], df['mapping']))

        return dictionary


    # Accepts a list of sentences and generates vocabulary and word mappings.
    def fit(self, sentences):
        v = self.build_vocabulary(sentences)
        self.word_mapping = self.get_mapping(v)
        self.total_words = len(self.word_mapping)

    # Converts the sentences to number mappings.
    def transform(self, sentences, pad_length = 350):

        whole = list() # Stores mapping for all sentences
        for x in sentences: # for each sentence in list of sentences.

            part = list()
            for word in x.split(): # for each word
                if word in self.word_mapping:
                    part.append(self.word_mapping[word]) # Append mapped number.
            whole.append(part) # Append sentence.

        # Append additional values to make lengths equal.
        #whole = keras.preprocessing.sequence.pad_sequences(np.array(whole),␣
↪maxlen=pad_length)
        whole = sequence.pad_sequences(np.array(whole), maxlen=pad_length)

        return whole
```

[ ]:
```
lstmfeat = LSTMFeaturization()
lstmfeat.fit(X_train['preprocessed_text'])
```

[ ]:
```

```

[ ]:
```
lstm_train = lstmfeat.transform(X_train['preprocessed_text'])
lstm_test = lstmfeat.transform(X_test['preprocessed_text'])
lstm_cv = lstmfeat.transform(X_cv['preprocessed_text'])
```

[ ]:
```
print(lstm_train.shape)
print(lstm_cv.shape)
print(lstm_test.shape)
```

```
(1353655, 350)
(451219, 350)
(97320, 350)
```

[ ]:
```
np.save('lstm_train.npy', lstm_train)
np.save('lstm_cv.npy', lstm_cv)
np.save('lstm_test.npy', lstm_test)
```

[ ]:
```
# create the model
embedding_vecor_length = 100
total_words = lstmfeat.total_words
model = Sequential()
model.add(Embedding(total_words ,embedding_vecor_length, input_length=350))
model.add(CuDNNLSTM(128, return_sequences=True))
model.add(CuDNNLSTM(128))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mse'])
print(model.summary())
```

```
Metal device set to: Apple M1 Pro

systemMemory: 16.00 GB
maxCacheSize: 5.33 GB

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 350, 100)          20154400

 cu_dnnlstm (CuDNNLSTM)      (None, 350, 128)          117760

 cu_dnnlstm_1 (CuDNNLSTM)    (None, 128)               132096

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 20,404,385
Trainable params: 20,404,385
Non-trainable params: 0
```

```
-----------------------------------------------------------------
None
```

[ ]:
```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

[ ]:
```
filepath="weights-improvement-{epoch:02d}-{val_loss:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,␣
↪save_best_only=True, mode='max')
callbacks_list = [checkpoint]
```

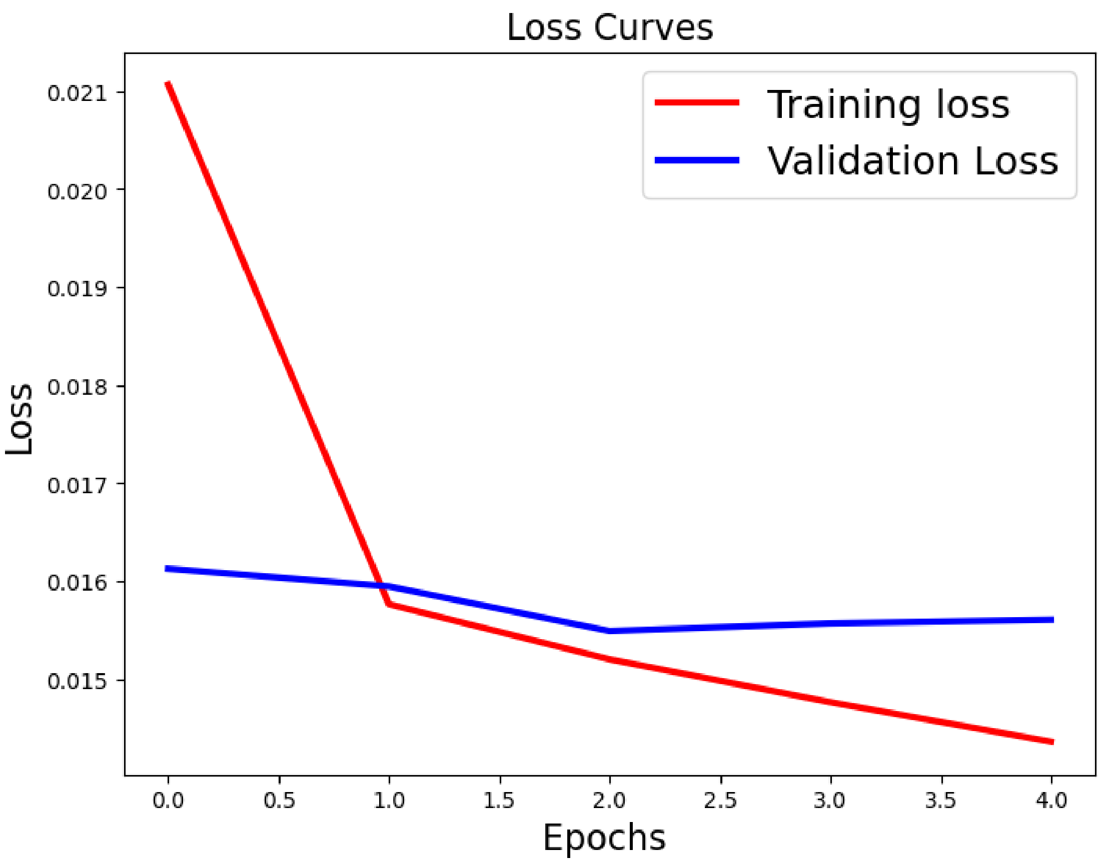
[ ]:
```
history = model.fit(lstm_train, y_train, epochs=5, batch_size=2048,␣
↪validation_data=(lstm_cv, y_cv), verbose = 1, callbacks=callbacks_list)
```

```
Epoch 1/5

2023-05-12 07:13:18.370818: W
tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz

661/661 [==============================] - ETA: 0s - loss: 0.0211 - mse: 0.0211
Epoch 1: val_loss improved from -inf to 0.01613, saving model to weights-
improvement-01-0.02.hdf5
661/661 [==============================] - 3131s 5s/step - loss: 0.0211 - mse:
0.0211 - val_loss: 0.0161 - val_mse: 0.0161
Epoch 2/5
661/661 [==============================] - ETA: 0s - loss: 0.0158 - mse: 0.0158
Epoch 2: val_loss did not improve from 0.01613
661/661 [==============================] - 3534s 5s/step - loss: 0.0158 - mse:
0.0158 - val_loss: 0.0159 - val_mse: 0.0159
Epoch 3/5
661/661 [==============================] - ETA: 0s - loss: 0.0152 - mse: 0.0152
Epoch 3: val_loss did not improve from 0.01613
661/661 [==============================] - 3667s 6s/step - loss: 0.0152 - mse:
0.0152 - val_loss: 0.0155 - val_mse: 0.0155
Epoch 4/5
661/661 [==============================] - ETA: 0s - loss: 0.0148 - mse: 0.0148
Epoch 4: val_loss did not improve from 0.01613
661/661 [==============================] - 3897s 6s/step - loss: 0.0148 - mse:
0.0148 - val_loss: 0.0156 - val_mse: 0.0156
Epoch 5/5
661/661 [==============================] - ETA: 0s - loss: 0.0144 - mse: 0.0144
Epoch 5: val_loss did not improve from 0.01613
661/661 [==============================] - 4422s 7s/step - loss: 0.0144 - mse:
0.0144 - val_loss: 0.0156 - val_mse: 0.0156
```

[ ]:
```
model.save('model.h5')
```

[ ]:
```
# Loss Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'b',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)
```

[ ]:
```
Text(0.5, 1.0, 'Loss Curves')
```



[ ]:
```
#inorder to load the model, just un-comment the following line
#model = keras.models.load_model('model.h5')
```

End of Assessment task 2 Part B

by

Hemang Sharma (24695785)

Nusrat Zahan (14367472)