main

May 8, 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()
```

[ ]:

id	target	comment_text	severe_toxicity	obscene	identity_attack	insult	threat	asian	\
59848	0.000000	This is so cool. It's like, 'would you want yo…	0.000000	0.0	0.000000	0.00000	0.0	NaN	
59849	0.000000	Thank you!! This would make my life a lot less…	0.000000	0.0	0.000000	0.00000	0.0	NaN	
59852	0.000000	This is such an urgent design problem; kudos t…	0.000000	0.0	0.000000	0.00000	0.0	NaN	
59855	0.000000	Is this something I'll be able to install on m…	0.000000	0.0	0.000000	0.00000	0.0	NaN	
59856	0.893617	haha you guys are a bunch of losers.							

```
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]
```

[ ]:
```
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 …
```

[ ]:
```
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
```

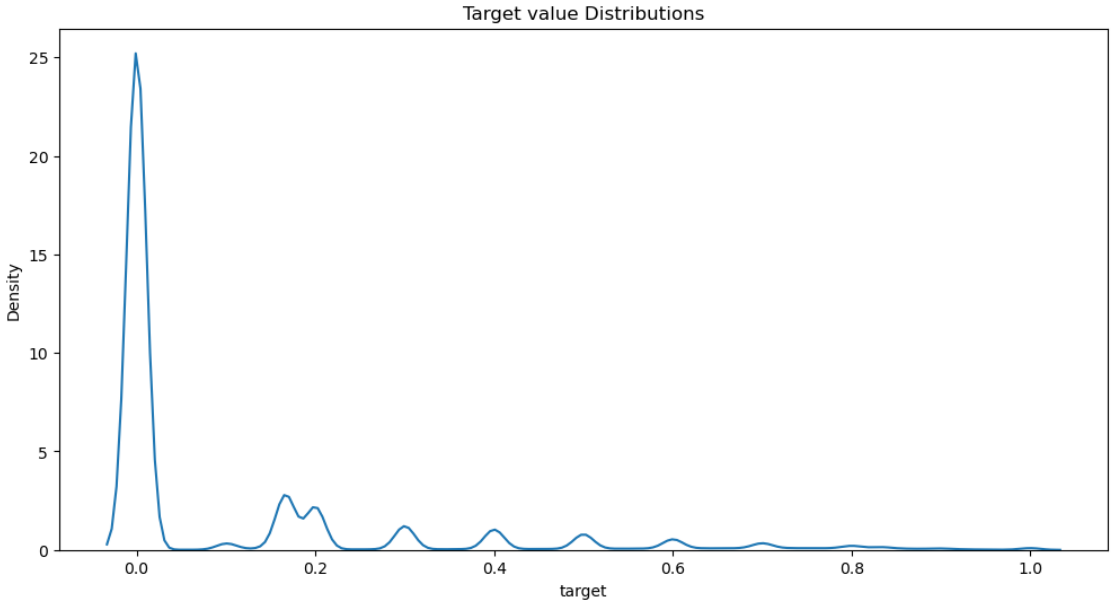
[ ]:
```
print("Train and test shape: {} {}".format(train_df.shape, test_df.shape))
```

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

1.2 Exploratory Data Analysis

1.3 1. Target Feature:

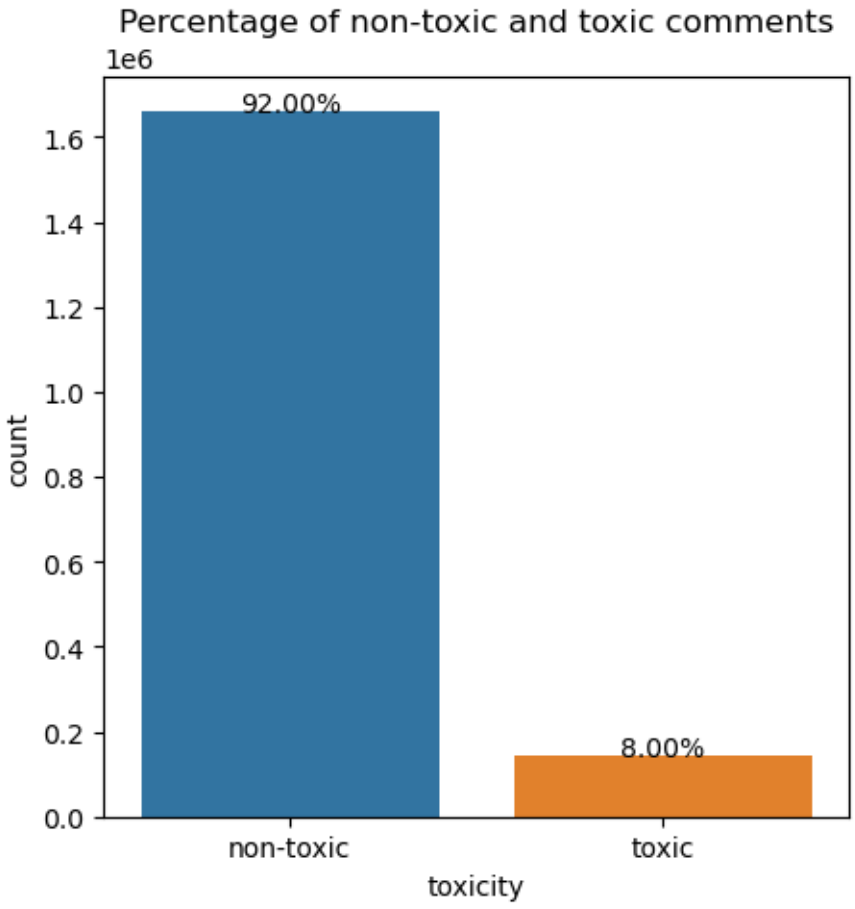
[ ]:
```
plt.figure(figsize=(12,6))
plt.title("Target value Distributions")
sns.distplot(train_df['target'], kde=True, hist=False, bins=240, label='target')
plt.show()
```



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)

[ ]:
```
# 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')
```

```
# 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

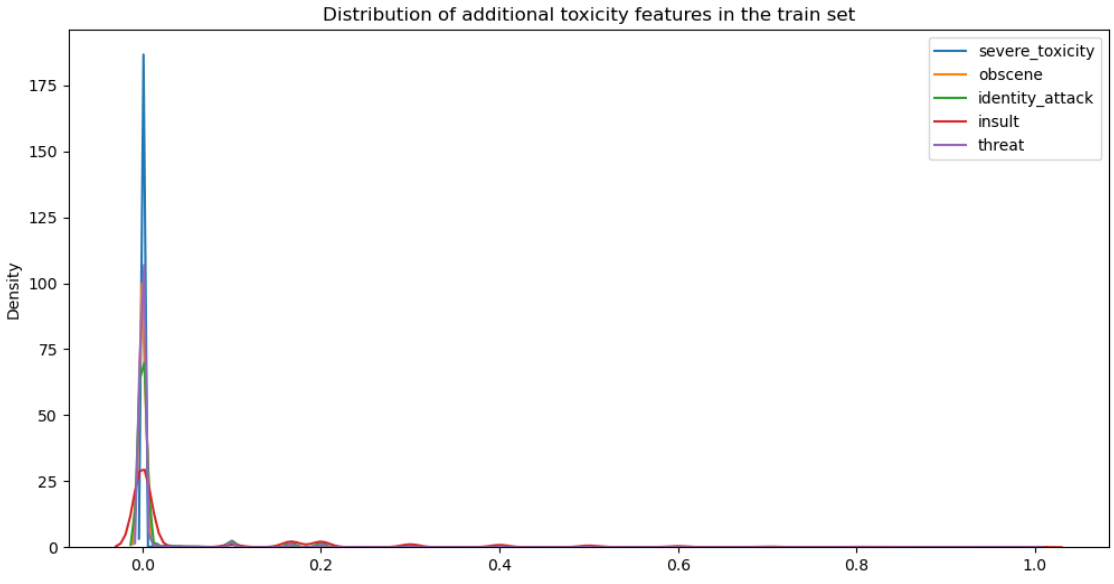
threat

insult

identity_attack

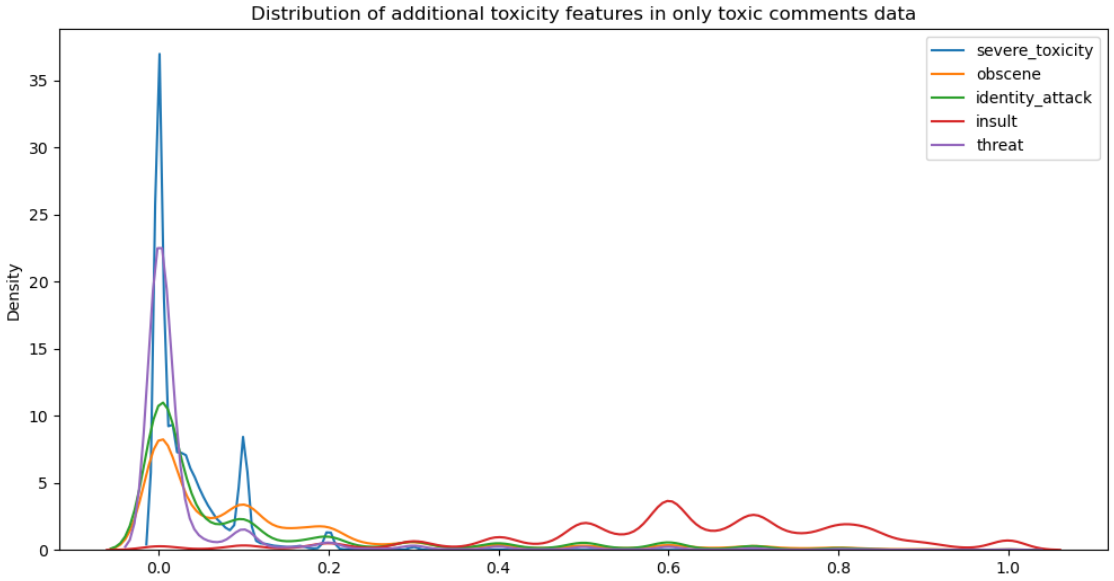
[ ]:
```
def plot_features_distribution(features, title, data):
    plt.figure(figsize=(12,6))
    plt.title(title)
    for feature in features:
        sns.distplot(data[feature],kde=True,hist=False, bins=240, label=feature)
    plt.xlabel('')
    plt.legend()
    plt.show()
```

[ ]:
```
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'
```

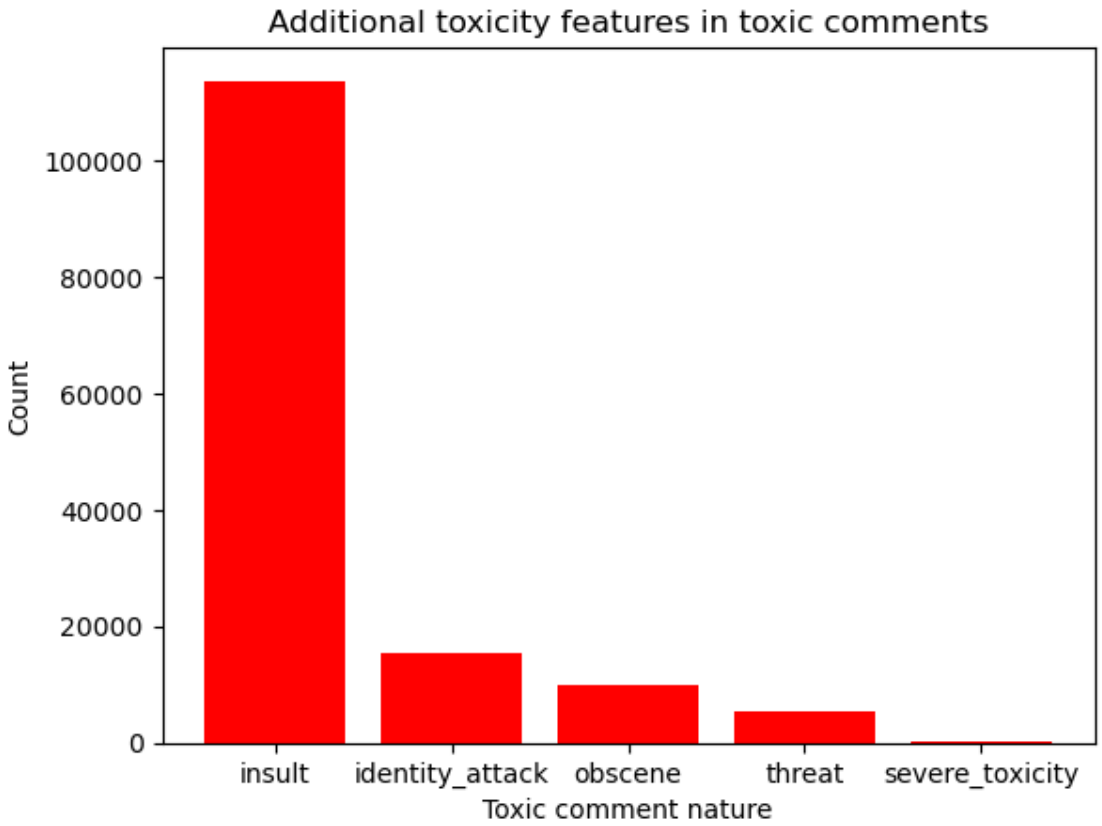
[ ]:
```
# If toxicity rating < 0.5 then the comment is non-toxic else it is toxic.
# Get toxic and non-toxic comments.
temp = train_df['target'].apply(lambda x: "non-toxic" if x < 0.5 else "toxic")
print(temp)
```

```
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.

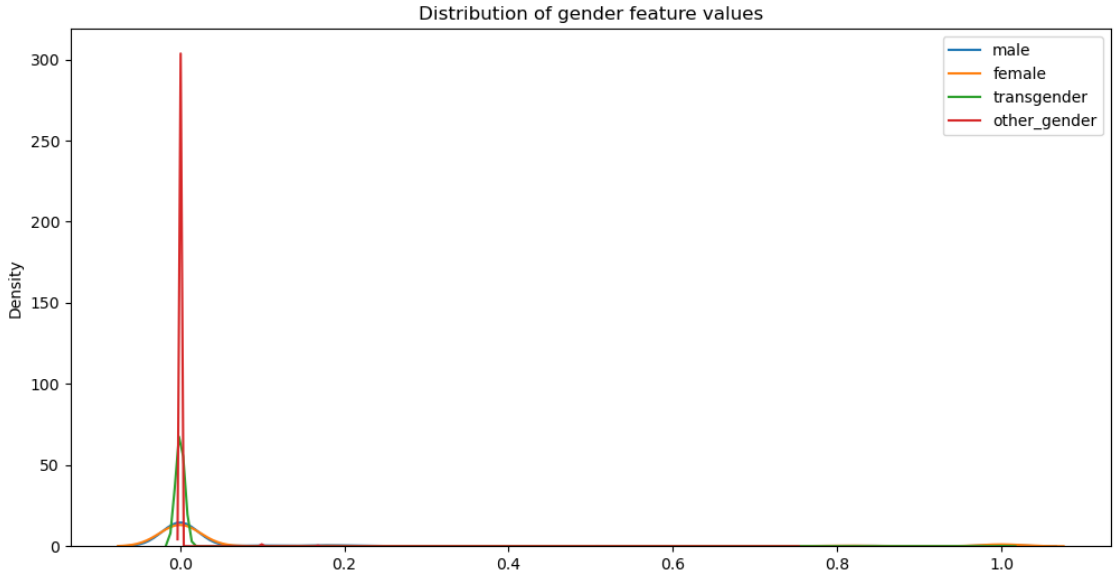
1.5 3. Identity Attributes:

Sensitive topics:

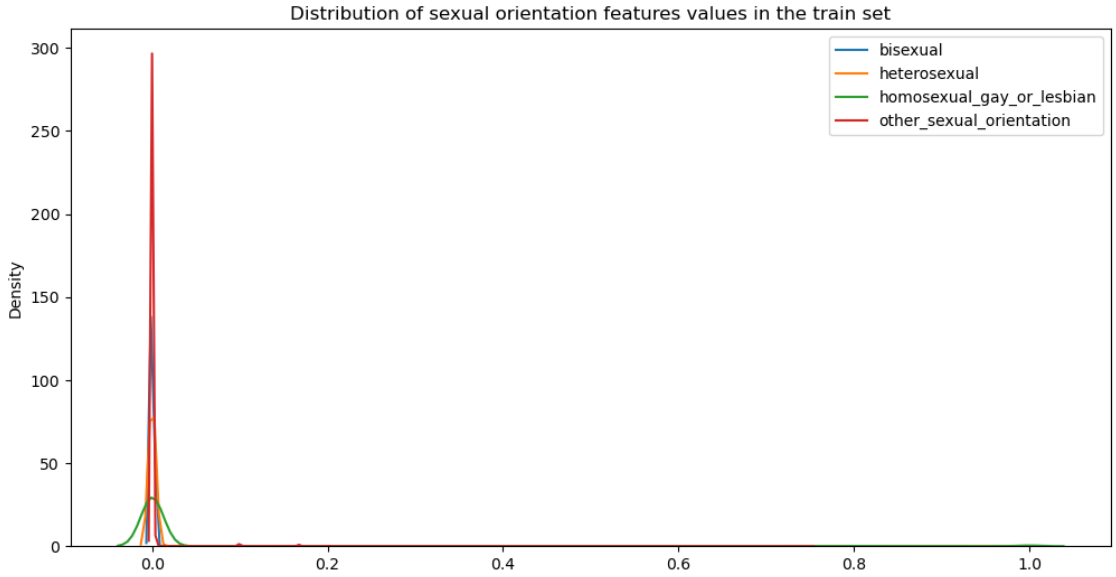
- male
- female
- homosexual_gay_or_lesbian
- bisexual
- heterosexual
- christian
- jewish
- muslim
- black
- white
- asian
- latino

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

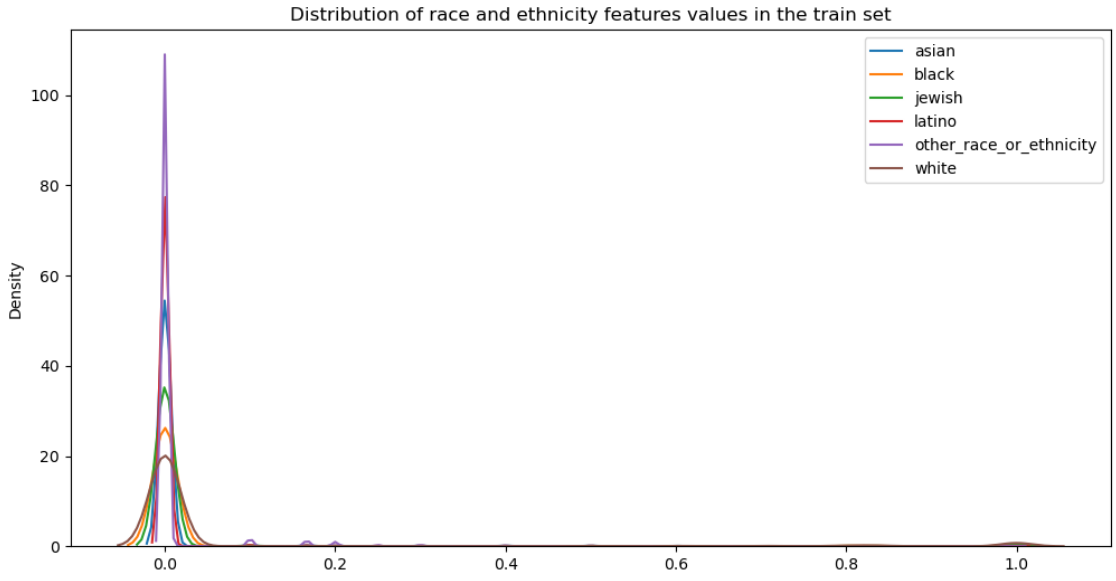
[ ]:
```
features = ['male', 'female', 'transgender', 'other_gender']
plot_features_distribution(features, "Distribution of gender feature values",␣
↪temp)
```



[ ]:
```
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)
```

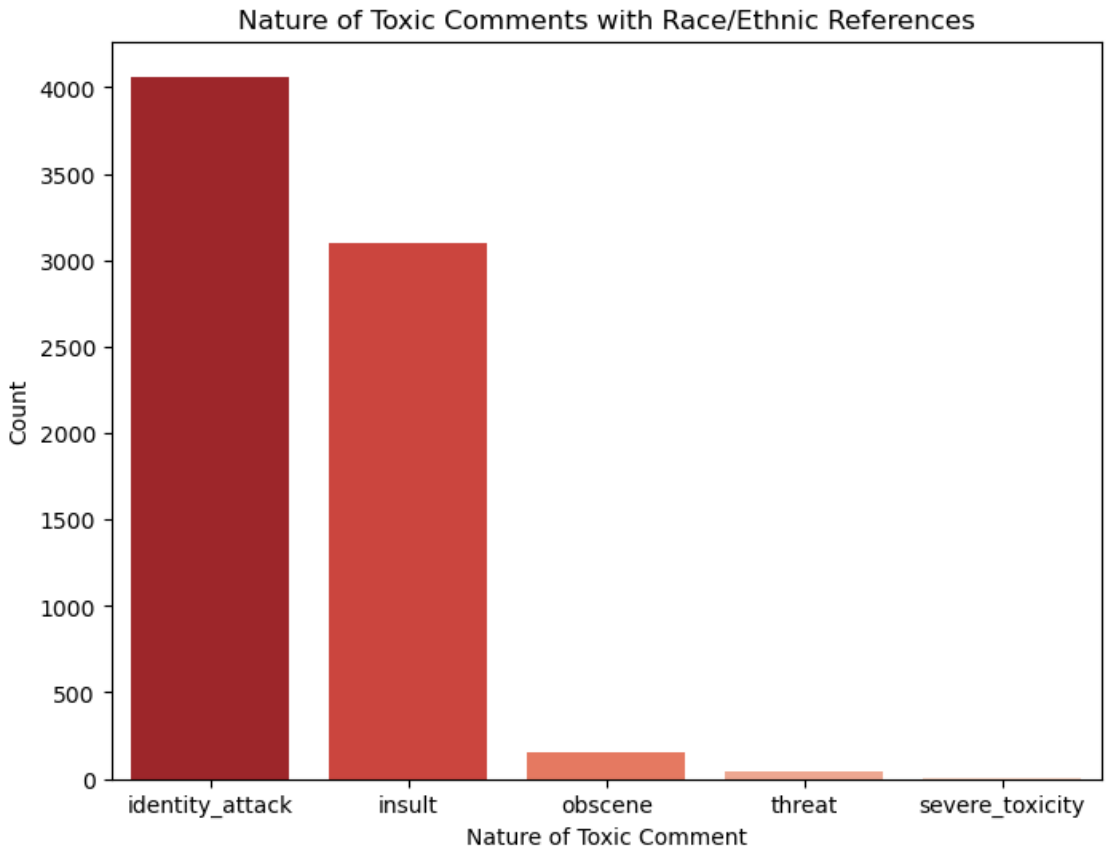


[ ]:
```
features = ['asian', 'black', 'jewish', 'latino', 'other_race_or_ethnicity',␣
 ↪'white']
plot_features_distribution(features, "Distribution of race and ethnicity␣
 ↪features values in the train set", temp)
```



[ ]:
```
# 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
```

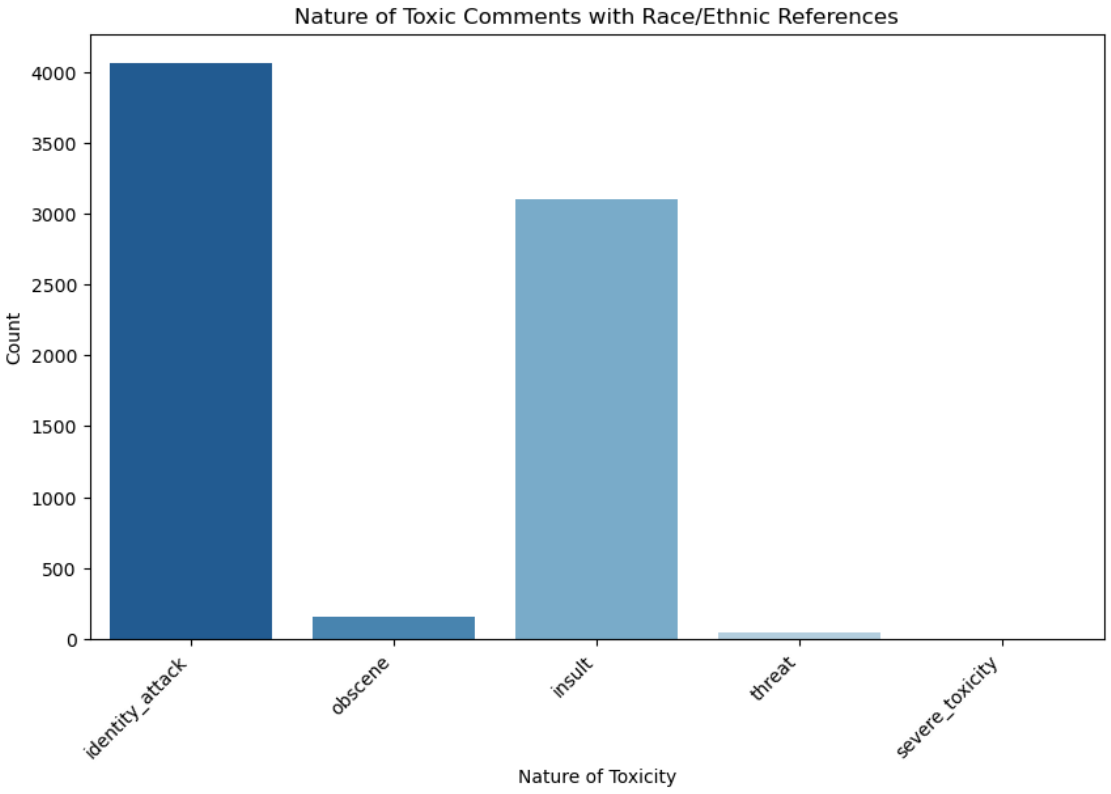
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# 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.

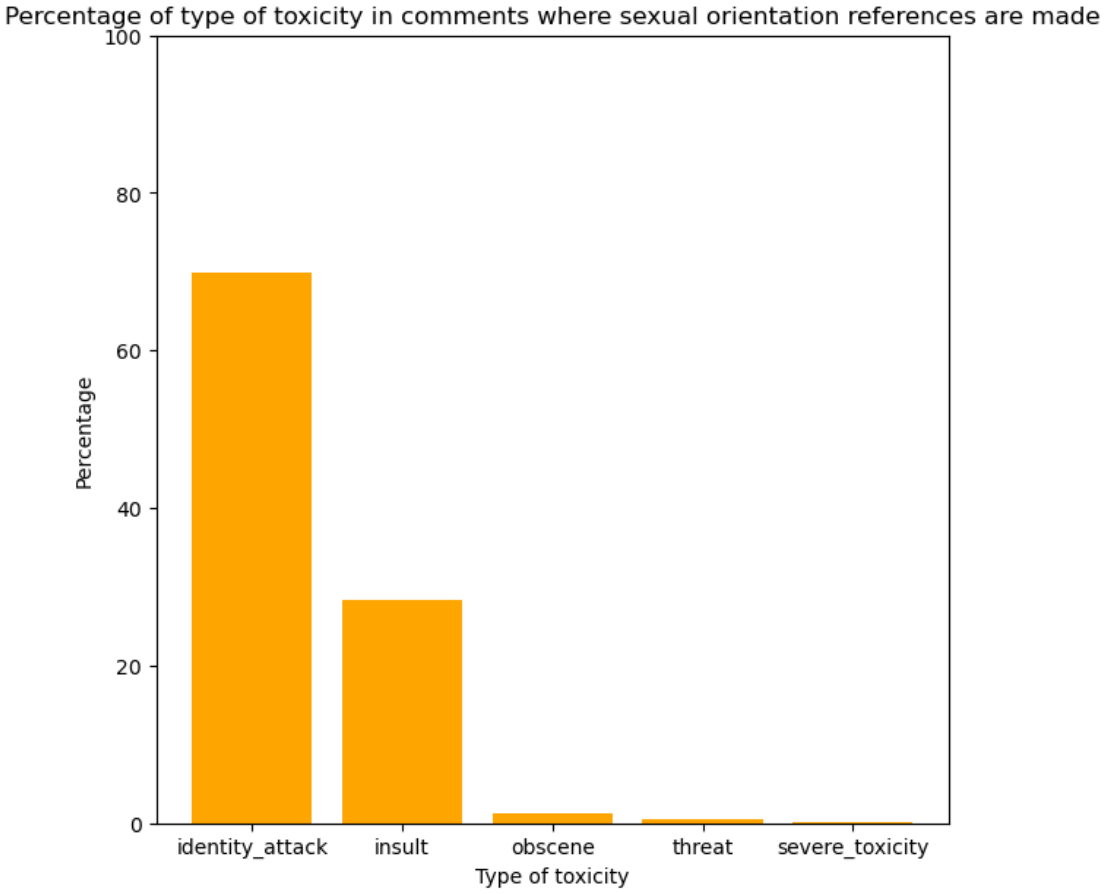
[ ]:
```
train_df2 = train_df
train_df2['insult'] = pd.to_numeric(train_df['insult'], errors='coerce')

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()
```



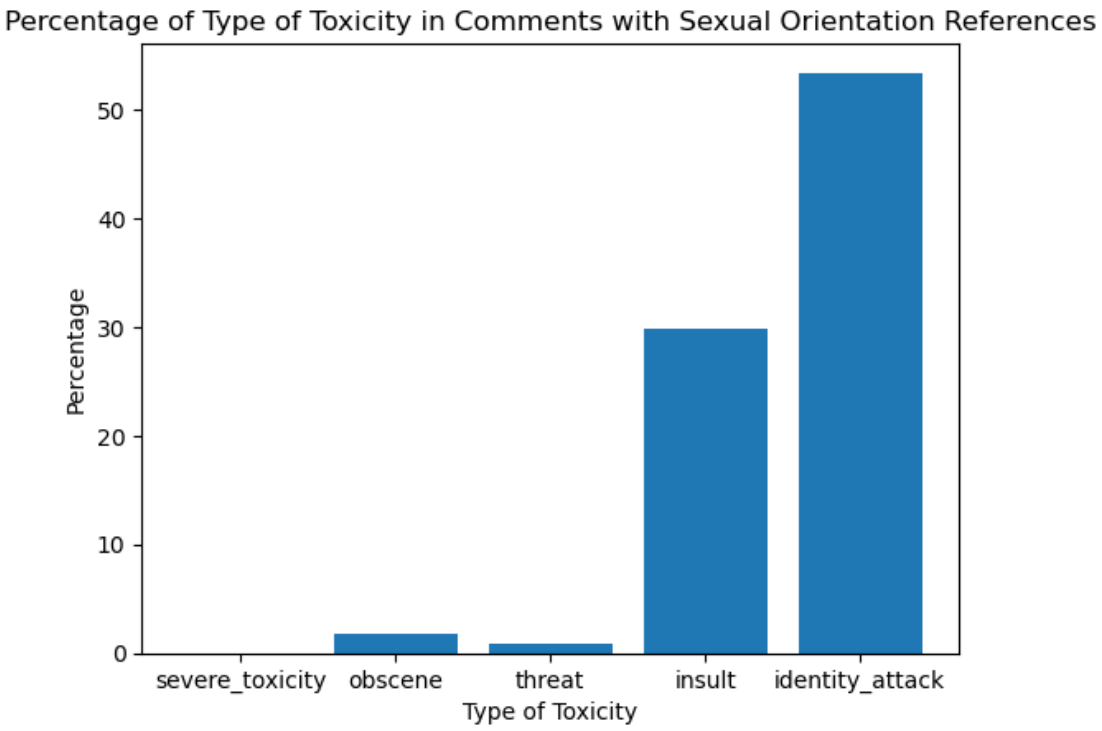
[ ]:
```
# Get data where sexual orientation references are made.
cond = (train_df['bisexual'] > 0.5) | (train_df['heterosexual'] > 0.5) |␣
 ↪(train_df['homosexual_gay_or_lesbian'] > 0.5) |␣
 ↪(train_df['other_sexual_orientation'] > 0.5)
temp = train_df[cond]
temp = temp[temp['target'] > 0.5]

# 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()
```



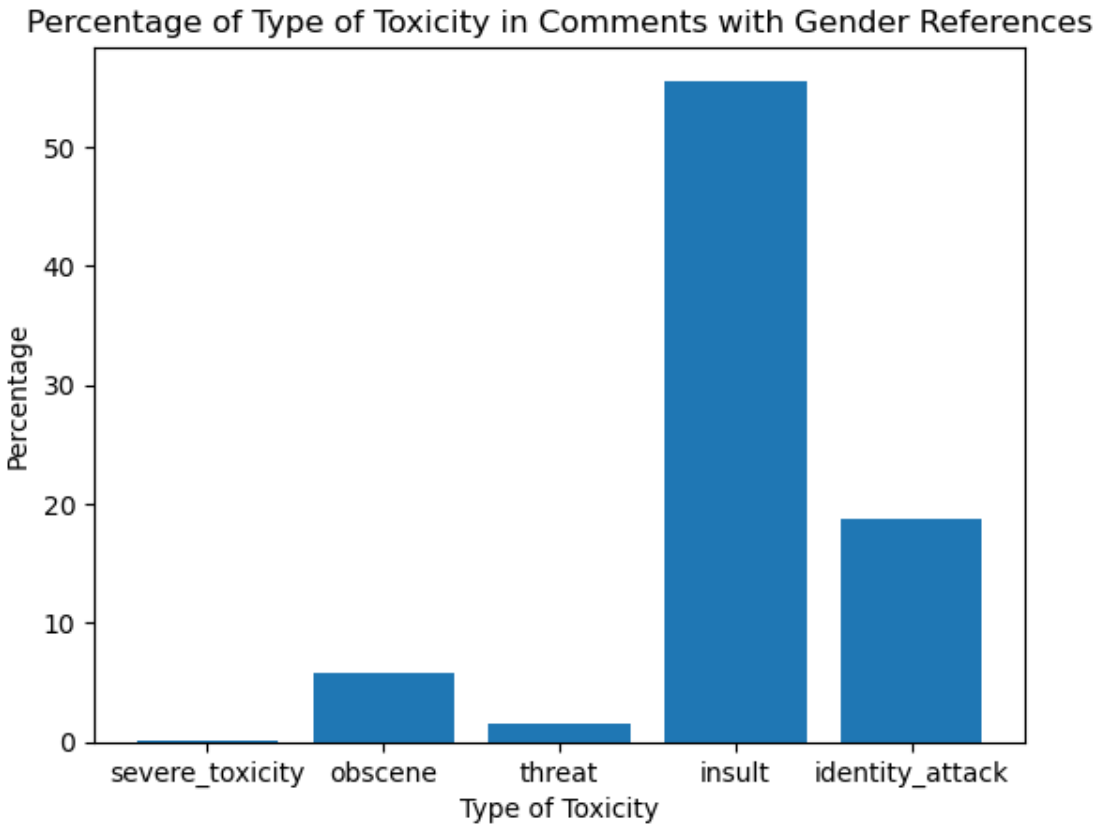
[ ]:
```
import matplotlib.pyplot as plt

# 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.

[ ]:
```
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

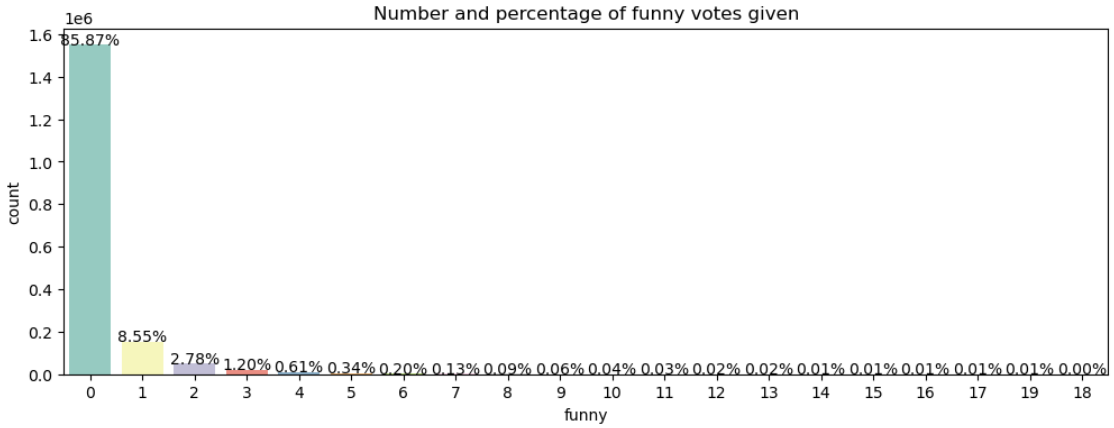
[ ]:
```
'''
This block of code will result in error for the following graphs

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:
```

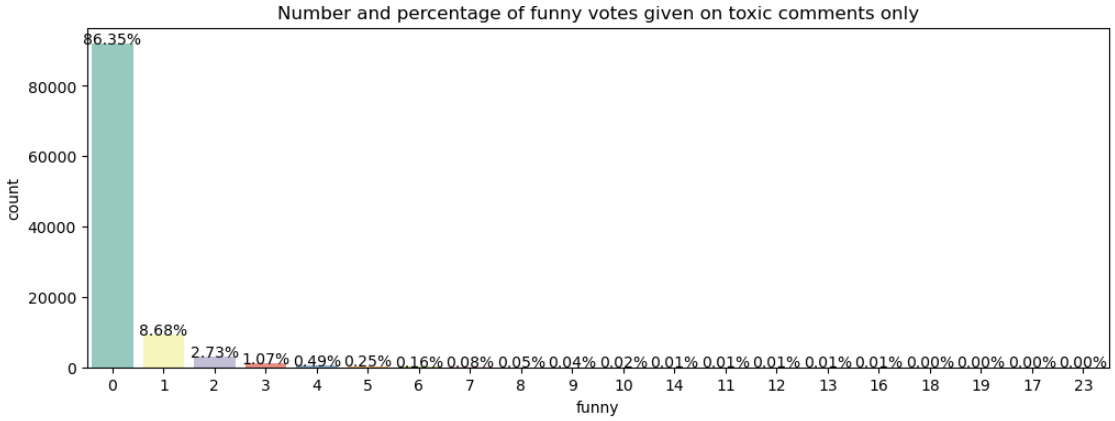
```
        height = p.get_height()
        ax.text(p.get_x()+p.get_width()/2.,
                height + 3,
                '{:1.2f}%'.format(100*height/total),
                ha="center")
    plt.show()   '''
```

[ ]:
```
' \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()   '
```

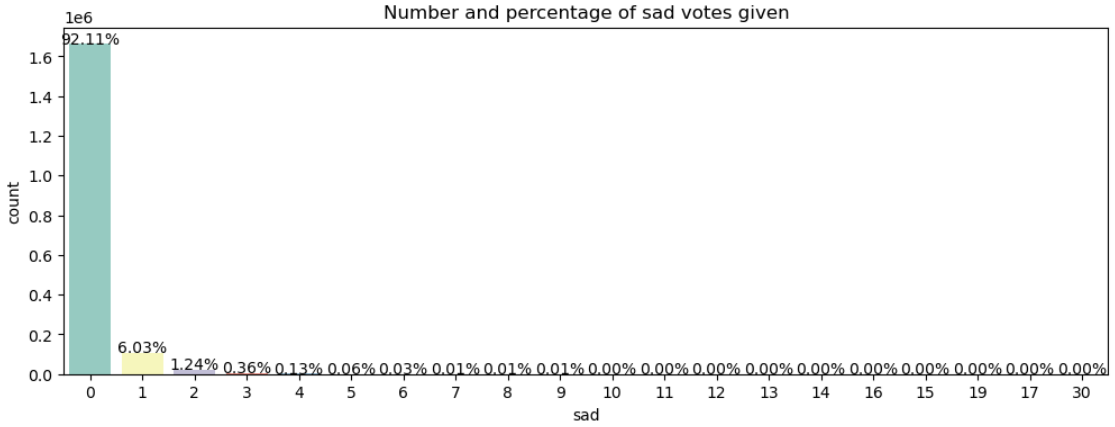
[ ]:
```
def plot_count(feature, title, data, size=1):
    f, ax = plt.subplots(1,1, figsize=(4*size,4))
    total = float(len(data))
    g = sns.countplot(x=feature, data=data, order=data[feature].value_counts().
 ↪index[:20], palette='Set3')
    g.set_title("Number and percentage of {}".format(title))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

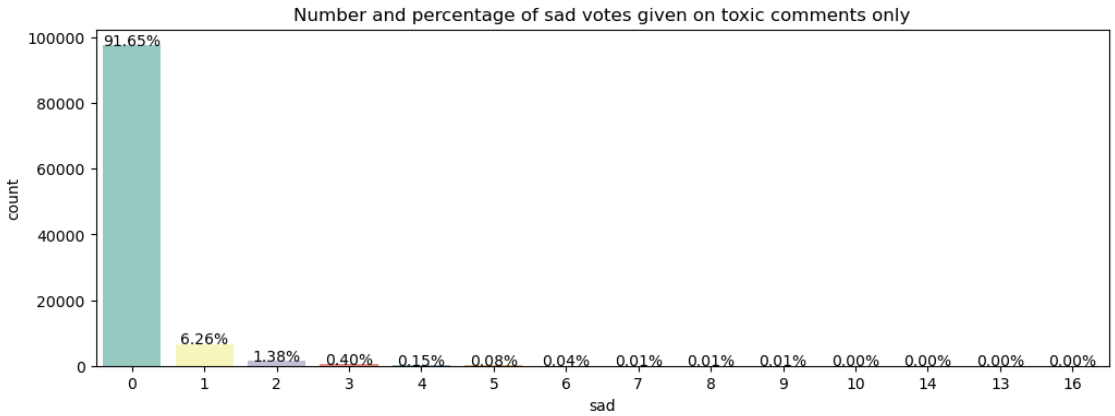
[ ]:
```
plot_count('funny','funny votes given', train_df, 3)
plot_count('funny', 'funny votes given on toxic comments only',␣
 ↪train_df[train_df['target'] > 0.5], 3)
```



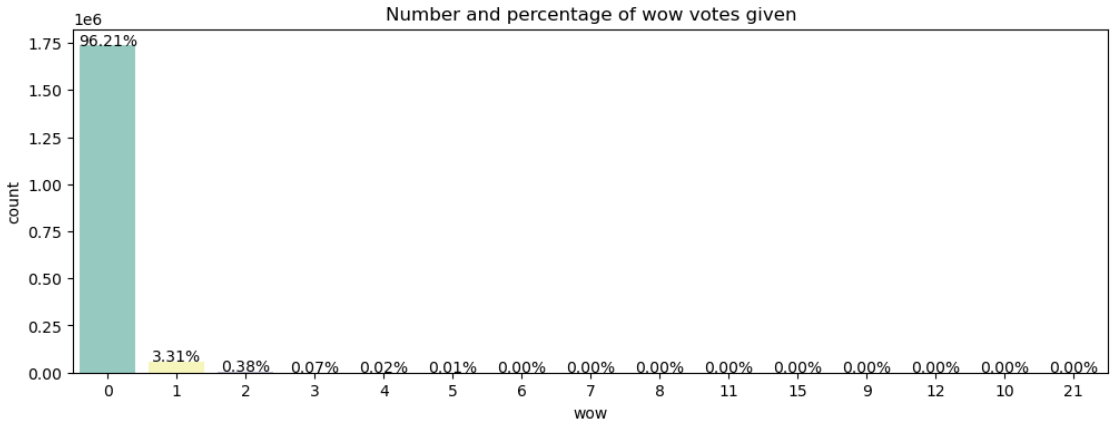


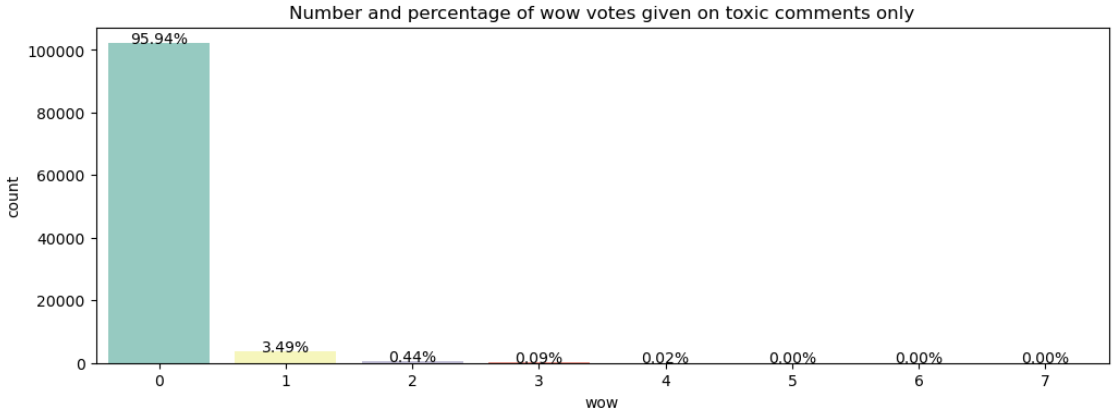
[ ]:
```
plot_count('sad','sad votes given', train_df, 3)
plot_count('sad', 'sad votes given on toxic comments only',␣
 ↪train_df[train_df['target'] > 0.5], 3)
```



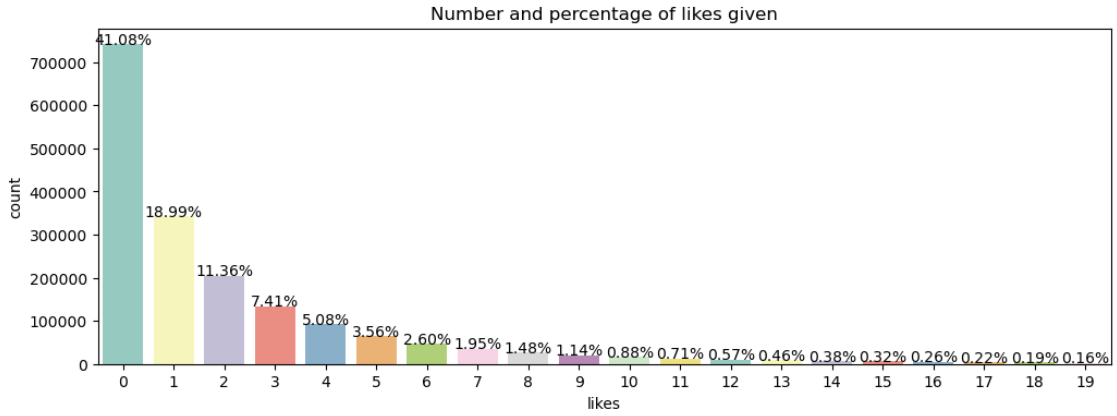


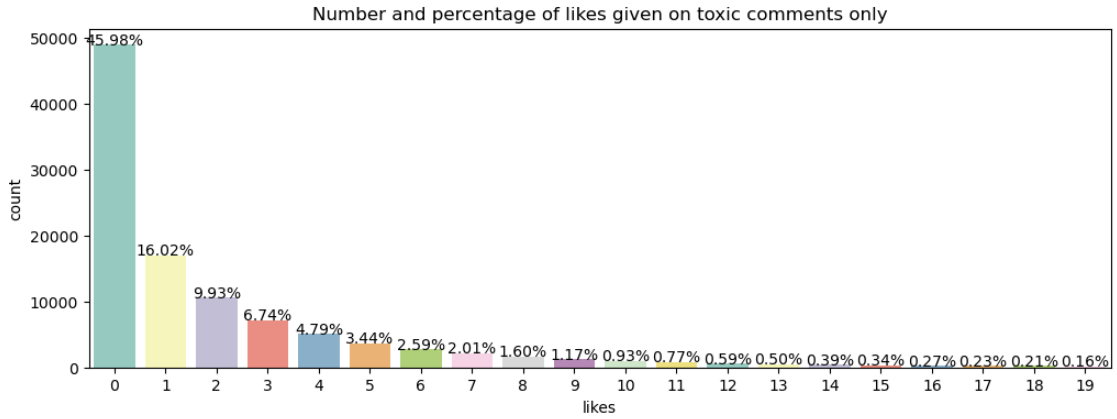
[ ]:
```
plot_count('wow','wow votes given', train_df, 3)
plot_count('wow', 'wow votes given on toxic comments only',␣
 ↪train_df[train_df['target'] > 0.5], 3)
```



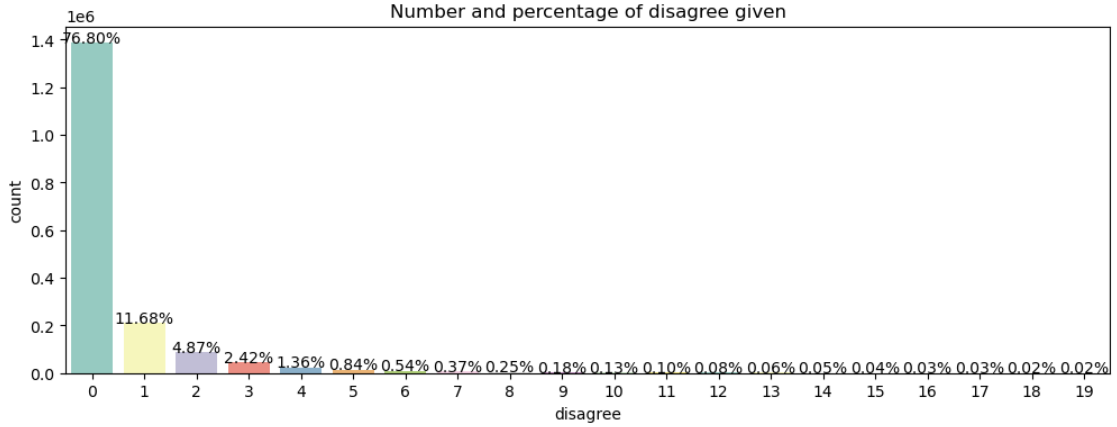


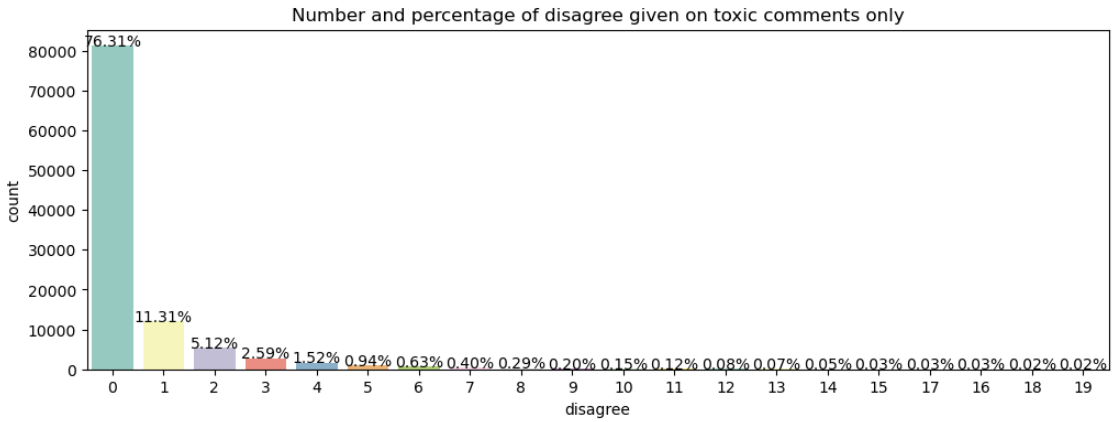
[ ]:
```
plot_count('likes','likes given', train_df, 3)
plot_count('likes', 'likes given on toxic comments only',␣
 ↪train_df[train_df['target'] > 0.5], 3)
```





[ ]:
```
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()
```

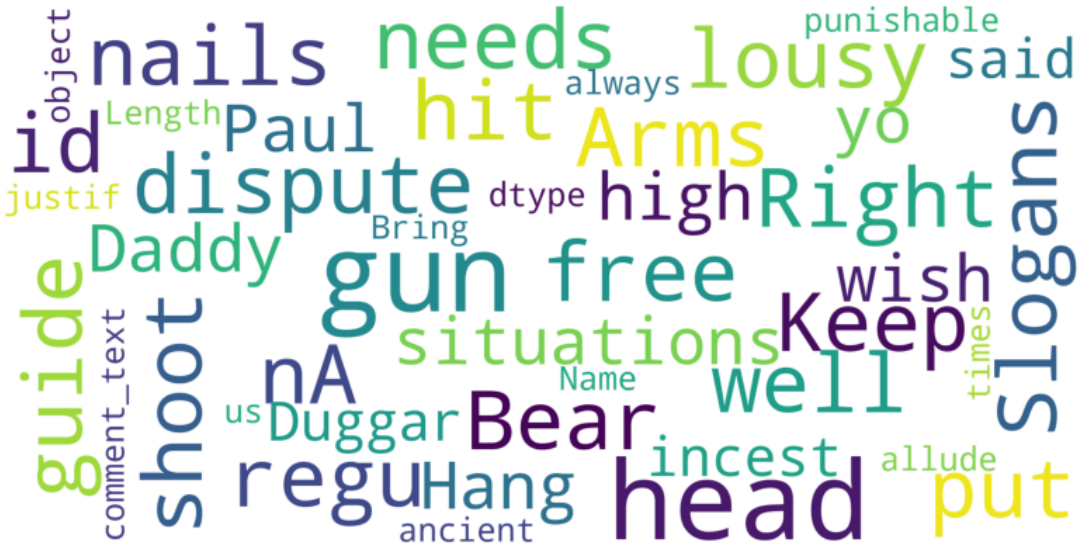
[ ]:
```
show_wordcloud(train_df['comment_text'].sample(20000), title = 'Prevalent words␣
↪in comments - train data')
```

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')
```

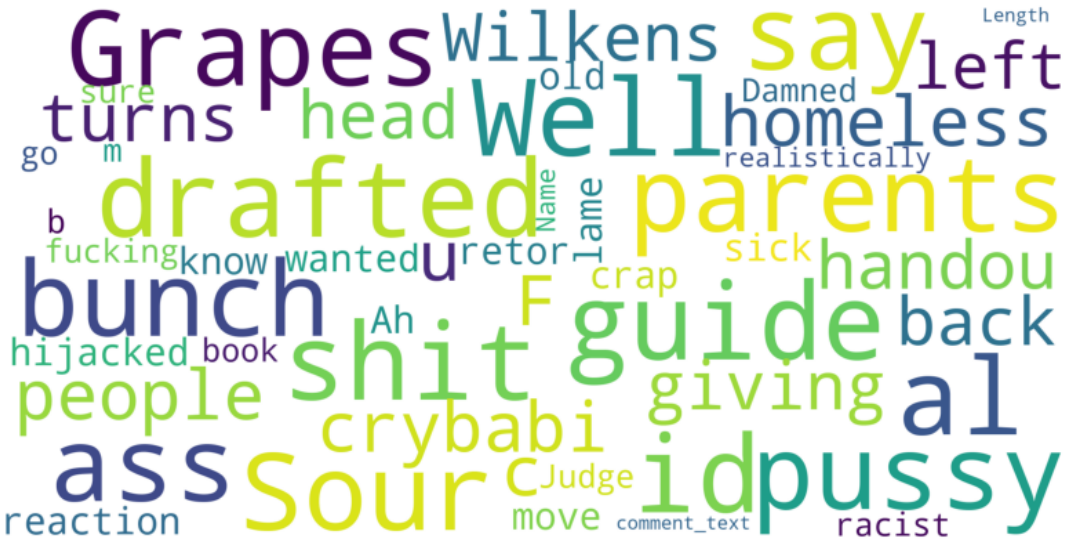
Prevalent comments with insult score > 0.75

[ ]:
```
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

[ ]:
```
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

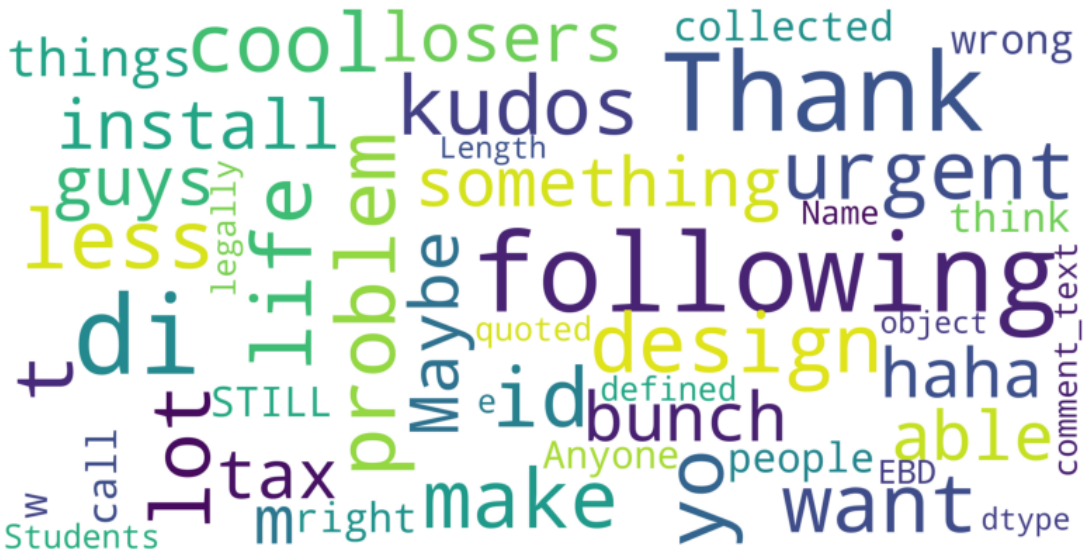
```
[ ]: show_wordcloud(train_df.loc[train_df['target'] < 0.25]['comment_text'],
                    title = 'Prevalent words in comments with target score < 0.25')
```

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')
```

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)
```

[ ]:
```
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)
```

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

[ ]:

id	preprocessed_text
522489	probabl bar set low hillari sad sheepl smart e…
5107660	thank sen dunleavi vote principl truth matter …
731719	come play dumb
5665796	get rid led light non sequitur ever one howev …
5043885	exact great incom 30 year save basic noth spen…

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

[ ]:

id	preprocessed_text
5436250	poster respond seem understand scientif proces…
6255599	care okay build monster home govern butt local…
5445055	seem like insight echo overal industri opinion…
6065875	mental incompet potus realli sake world let al…
313926	35 000 commiss transact quit lot money buy lot…

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

[ ]:

id	preprocessed_text
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…

[ ]:
```
# 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:

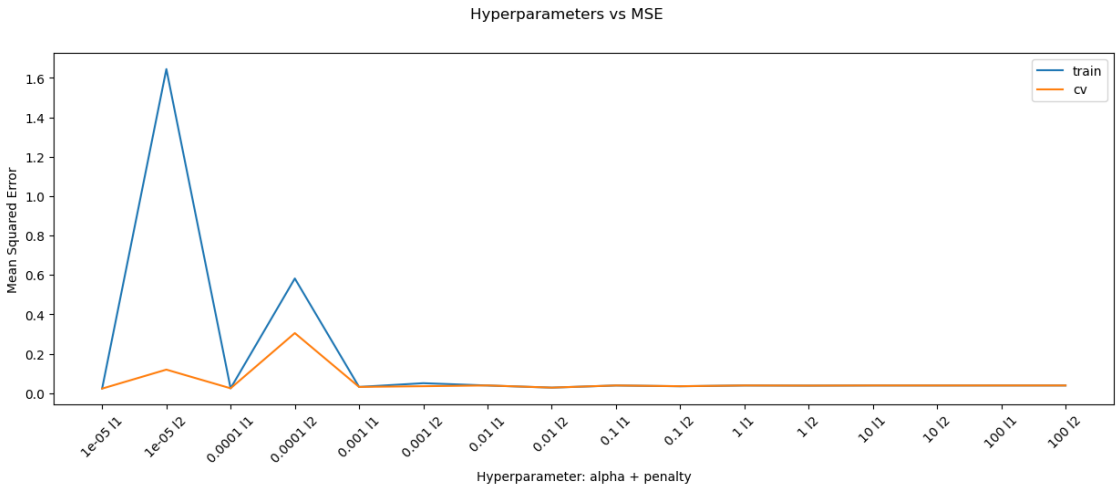
[ ]:
```
# Performing hyperparameter tuning:
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
penalty = ['l1', 'l2']
xticks = []
tr_errors = []
cv_errors = []
best_model = None
best_error = 100
for a in alpha:
    for p in penalty:
        xticks.append(str(a) + ' ' + p)
        print(str(a) + ' ' + 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:  0.023836357836215686
Mean Squared Error on cv set:  0.022995330405120723
**************************************************
1e-05 l2 :
Mean Squared Error on train set:  1.6449003068095598
Mean Squared Error on cv set:  0.11933811144844796
**************************************************
0.0001 l1 :
Mean Squared Error on train set:  0.02447173787310386
Mean Squared Error on cv set:  0.024527201860137578
**************************************************
0.0001 l2 :
Mean Squared Error on train set:  0.5818971377161595
Mean Squared Error on cv set:  0.30472412447214586
**************************************************
0.001 l1 :
Mean Squared Error on train set:  0.031436430325507025
Mean Squared Error on cv set:  0.03153559646687563
**************************************************
0.001 l2 :
Mean Squared Error on train set:  0.0504213342077689
Mean Squared Error on cv set:  0.03511870695431834
**************************************************
0.01 l1 :
Mean Squared Error on train set:  0.03878989453804379
Mean Squared Error on cv set:  0.038985636568792344
**************************************************
```

```
0.01 l2 :
Mean Squared Error on train set:  0.027927678895015073
Mean Squared Error on cv set:  0.02804781049760689
**************************************************
0.1 l1 :
Mean Squared Error on train set:  0.038789894412469036
Mean Squared Error on cv set:  0.03898563377427119
**************************************************
0.1 l2 :
Mean Squared Error on train set:  0.03491475112100914
Mean Squared Error on cv set:  0.03508484091129964
**************************************************
1 l1 :
Mean Squared Error on train set:  0.03878990259156812
Mean Squared Error on cv set:  0.038985613767767074
**************************************************
1 l2 :
Mean Squared Error on train set:  0.03804400577320498
Mean Squared Error on cv set:  0.03823584810608087
**************************************************
10 l1 :
Mean Squared Error on train set:  0.038789917473624244
Mean Squared Error on cv set:  0.03898561007646659
**************************************************
10 l2 :
Mean Squared Error on train set:  0.0386922418338348
Mean Squared Error on cv set:  0.03888693026849541
**************************************************
100 l1 :
Mean Squared Error on train set:  0.03878990036488324
Mean Squared Error on cv set:  0.03898566225976059
**************************************************
100 l2 :
Mean Squared Error on train set:  0.0387808358578927
Mean Squared Error on cv set:  0.03897650231134728
**************************************************
```

[ ]:
```
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': 1e-05,
 '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': 'l1',
 '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,:]
```

```
# 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'
```

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

[ ]:
```
                   weights
stupid stupid     -0.130182
knee jerk         -0.075628
black white       -0.049673
fool peopl        -0.047908
ignor fact        -0.037218
black market      -0.032714
winner loser      -0.032227
dumb dumb         -0.031451
white hous        -0.029207
mass shoot        -0.026813
can               -0.022704
get sick          -0.022240
magaph            -0.021601
racist misogynist -0.020718
white black       -0.020343
gerald butt       -0.019668
mental health     -0.019394
knee              -0.018735
winner            -0.017685
men women         -0.017086
```

1.2 Decision Trees:

1.2.1 Hyperparameter Tuning:

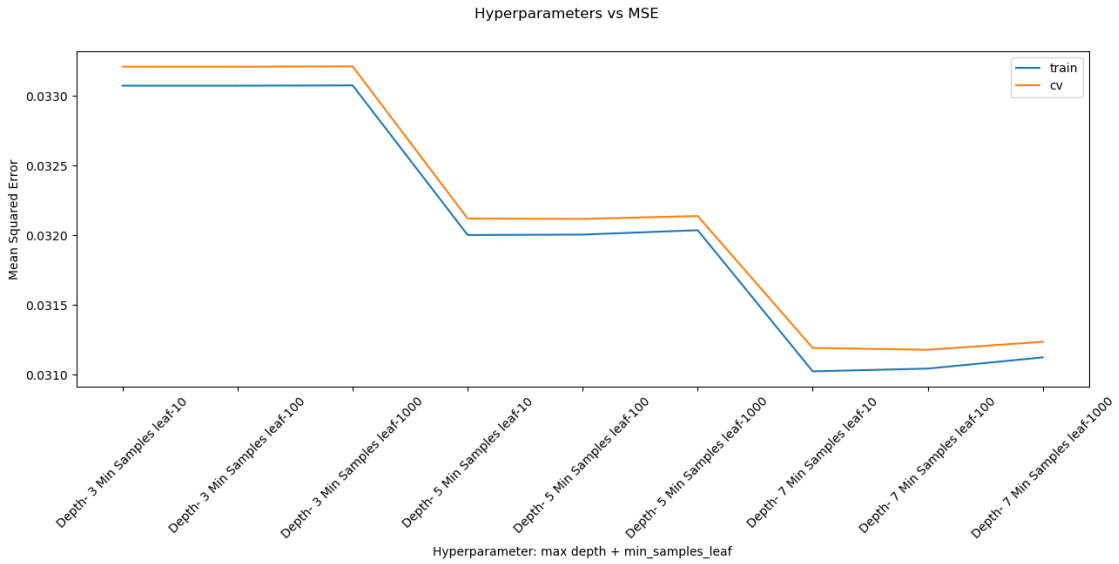
[ ]:
```
# Performing hyperparameter tuning:
max_depth = [3, 5, 7]
min_samples = [10, 100, 1000]
xticks = []
tr_errors = []
```

```
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:  0.03307213127121194
Mean Squared Error on cv set:  0.03320847754219259
**************************************************
Depth- 3 Min Samples leaf-100 :
Mean Squared Error on train set:  0.033072131271211926
Mean Squared Error on cv set:  0.033208477542192597
**************************************************
Depth- 3 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.03307428191859929
Mean Squared Error on cv set:  0.03321025042838812
**************************************************
Depth- 5 Min Samples leaf-10 :
Mean Squared Error on train set:  0.03200072513211591
Mean Squared Error on cv set:  0.03211864319991914
**************************************************
Depth- 5 Min Samples leaf-100 :
Mean Squared Error on train set:  0.03200380856300675
```

```
Mean Squared Error on cv set:  0.0321162197037969
**************************************************
Depth- 5 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.032035374707821314
Mean Squared Error on cv set:  0.0321368971630689
**************************************************
Depth- 7 Min Samples leaf-10 :
Mean Squared Error on train set:  0.031022678752460318
Mean Squared Error on cv set:  0.031190515454756786
**************************************************
Depth- 7 Min Samples leaf-100 :
Mean Squared Error on train set:  0.031042383508694815
Mean Squared Error on cv set:  0.031177265480264175
**************************************************
Depth- 7 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.03112272864928463
Mean Squared Error on cv set:  0.031234676345453797
**************************************************
```

[ ]:
```
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	0.399523
idiot	0.264382
pathet	0.068405
fool	0.067381
moron	0.063471
racist	0.059112
hypocrit	0.056195
would	0.004134
one	0.003823
year	0.003134
peopl	0.002072
white	0.001639
time	0.001452
even	0.001396
get	0.000742
also	0.000581
work	0.000461
fool peopl	0.000374
see	0.000362
want	0.000285

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:

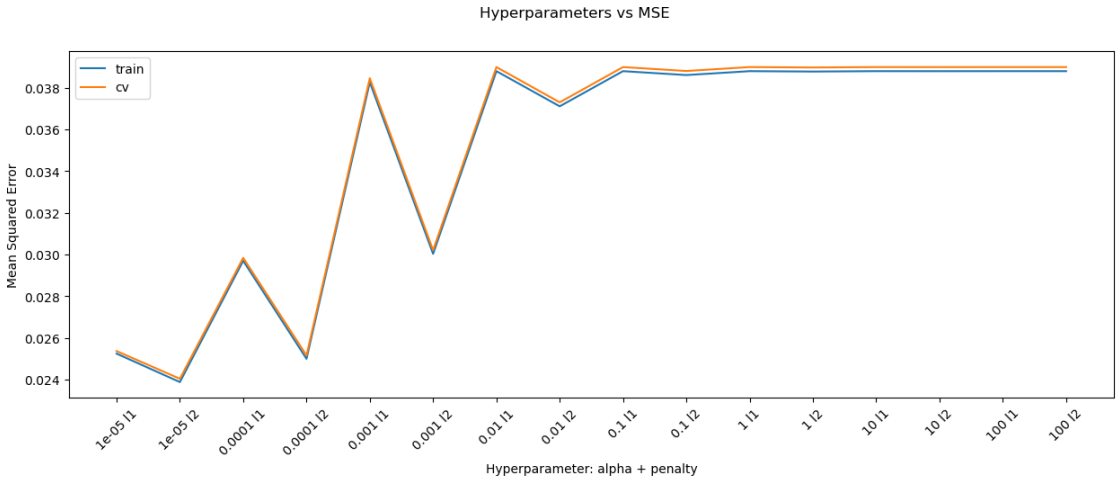
[ ]:
```
# Performing hyperparameter tuning:
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
penalty = ['l1', 'l2']
xticks = []
tr_errors = []
cv_errors = []
best_model = None
best_error = 100
for a in alpha:
    for p in penalty:
        xticks.append(str(a) + ' ' + p)
        print(str(a) + ' ' + 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:  0.025250054263804354
Mean Squared Error on cv set:  0.025372586833219943
**************************************************
1e-05 l2 :
Mean Squared Error on train set:  0.023887022342498364
Mean Squared Error on cv set:  0.02404554702921834
**************************************************
0.0001 l1 :
Mean Squared Error on train set:  0.029701640819809486
Mean Squared Error on cv set:  0.029841383472502682
**************************************************
0.0001 l2 :
Mean Squared Error on train set:  0.025002349064471168
Mean Squared Error on cv set:  0.025163092811001757
**************************************************
0.001 l1 :
Mean Squared Error on train set:  0.03825771281182615
Mean Squared Error on cv set:  0.03845286672344152
**************************************************
0.001 l2 :
Mean Squared Error on train set:  0.030032286597954913
Mean Squared Error on cv set:  0.030206019273114133
**************************************************
0.01 l1 :
Mean Squared Error on train set:  0.038789894701071075
Mean Squared Error on cv set:  0.03898562804173839
**************************************************
0.01 l2 :
Mean Squared Error on train set:  0.037104280584366024
Mean Squared Error on cv set:  0.037294983821628575
**************************************************
0.1 l1 :
Mean Squared Error on train set:  0.038789895291882905
Mean Squared Error on cv set:  0.03898564284292089
**************************************************
0.1 l2 :
Mean Squared Error on train set:  0.038602549254522604
Mean Squared Error on cv set:  0.038797697504416966
**************************************************
1 l1 :
Mean Squared Error on train set:  0.03878990127646504
Mean Squared Error on cv set:  0.0389856648936713
```

```
**************************************************
1 l2 :
Mean Squared Error on train set:  0.03877061532361369
Mean Squared Error on cv set:  0.03896622709617408
**************************************************
10 l1 :
Mean Squared Error on train set:  0.038789897547565975
Mean Squared Error on cv set:  0.03898565304746246
**************************************************
10 l2 :
Mean Squared Error on train set:  0.03878815449907428
Mean Squared Error on cv set:  0.038983897342612414
**************************************************
100 l1 :
Mean Squared Error on train set:  0.03878990192959554
Mean Squared Error on cv set:  0.038985666711387486
**************************************************
100 l2 :
Mean Squared Error on train set:  0.03878965232265972
Mean Squared Error on cv set:  0.03898542298249892
**************************************************
```

[ ]:
```
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	1.569629
idiot	1.266662
fool	0.657220
ignor	0.605213
dumb	0.592775
pathet	0.583823
moron	0.575642
ridicul	0.561020
loser	0.559009
liar	0.519340
hypocrit	0.517374
crap	0.512626
racist	0.497301
white	0.472975
troll	0.454203
damn	0.438325
kill	0.435202
clown	0.432796
black	0.431933
silli	0.429068

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

[ ]:

	weights
thank	-0.092360
interest	-0.087282
agre	-0.077450
stori	-0.077260
great	-0.071112
good	-0.069928
may	-0.068962
new	-0.068652
differ	-0.067376
issu	-0.067184
chang	-0.066715
com	-0.066144
point	-0.065153

```
work     -0.065107
articl   -0.064814
year     -0.064604
number   -0.064122
http     -0.061467
investig -0.060985
happen   -0.059328
```

2.2 Decision Trees:

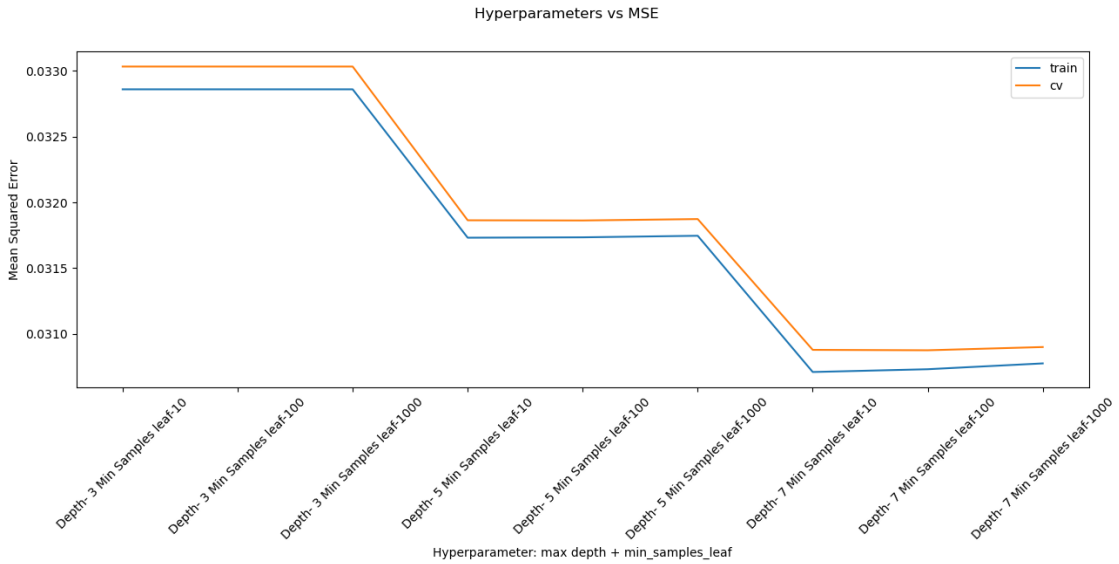
2.2.1 Hyperparameter Tuning:

[ ]:
```
# 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:  0.03285900910094292
```

```
Mean Squared Error on cv set:  0.03303194705923938
**************************************************
Depth- 3 Min Samples leaf-100 :
Mean Squared Error on train set:  0.03285900910094292
Mean Squared Error on cv set:  0.03303194705923939
**************************************************
Depth- 3 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.03285900910094292
Mean Squared Error on cv set:  0.03303194705923938
**************************************************
Depth- 5 Min Samples leaf-10 :
Mean Squared Error on train set:  0.0317314615621315
Mean Squared Error on cv set:  0.03186385013087883
**************************************************
Depth- 5 Min Samples leaf-100 :
Mean Squared Error on train set:  0.031734293901719064
Mean Squared Error on cv set:  0.03186235866043612
**************************************************
Depth- 5 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.03174615445189408
Mean Squared Error on cv set:  0.031873078228663955
**************************************************
Depth- 7 Min Samples leaf-10 :
Mean Squared Error on train set:  0.030710958871119038
Mean Squared Error on cv set:  0.030879295161989388
**************************************************
Depth- 7 Min Samples leaf-100 :
Mean Squared Error on train set:  0.030732047615251796
Mean Squared Error on cv set:  0.030876105245497155
**************************************************
Depth- 7 Min Samples leaf-1000 :
Mean Squared Error on train set:  0.030776332862837024
Mean Squared Error on cv set:  0.03090059010354151
**************************************************
```

[ ]:
```
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	0.407776
idiot	0.267245
fool	0.071524
pathet	0.069263
moron	0.064115
white	0.057197
hypocrit	0.054854
racist	0.005537
trump	0.000752
peopl	0.000504
ignor	0.000319
presid	0.000255
show	0.000125
like	0.000122
thing	0.000059
keep	0.000052
see	0.000043
left	0.000040
money	0.000040
work	0.000038

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)          20126100

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

 cu_dnnlstm_1 (CuDNNLSTM)    (None, 128)               132096

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 20,376,085
Trainable params: 20,376,085
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]
```

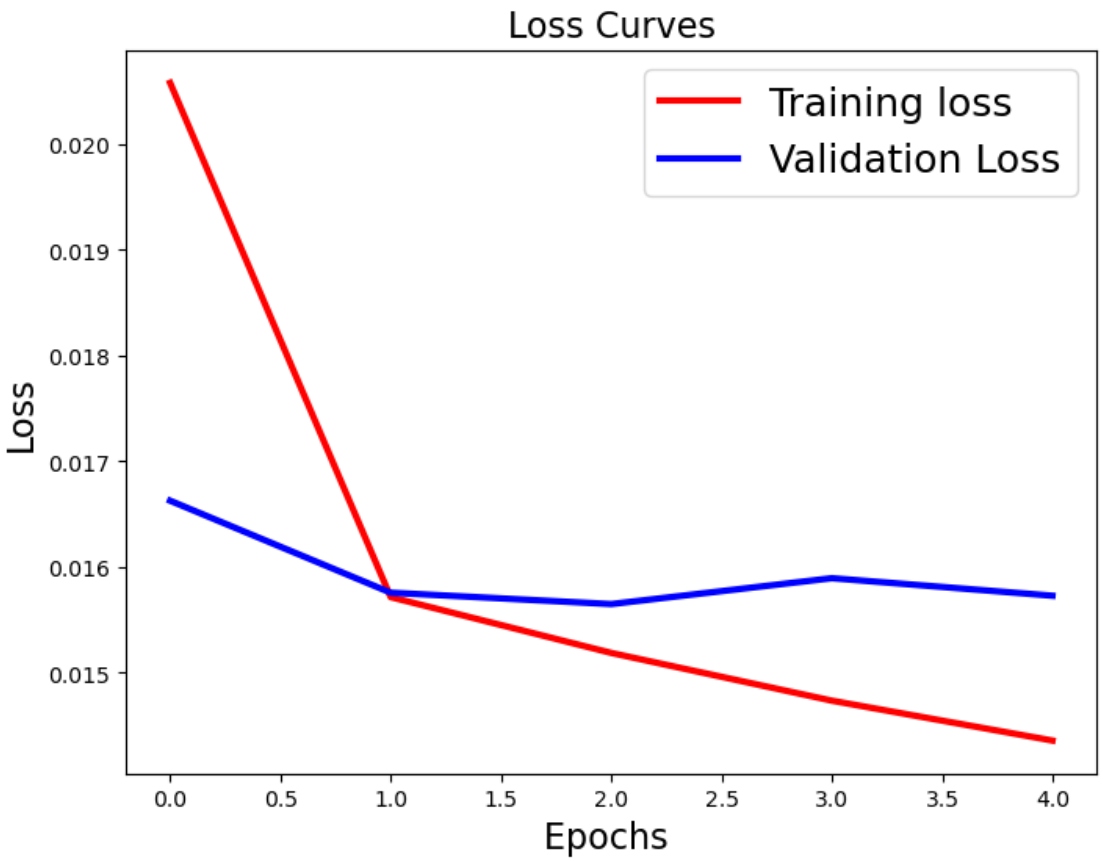
[ ]:
```
history = model.fit(lstm_train, y_train, epochs=5, batch_size=2048,␣
 ↪validation_data=(lstm_cv, y_cv), verbose = 1, callbacks=callbacks_list)
```

```
Epoch 1/5

2023-05-07 14:26:25.058425: W
tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz

661/661 [==============================] - ETA: 0s - loss: 0.0206 - mse: 0.0206
Epoch 1: val_loss improved from -inf to 0.01663, saving model to weights-
improvement-01-0.02.hdf5
661/661 [==============================] - 3142s 5s/step - loss: 0.0206 - mse:
0.0206 - val_loss: 0.0166 - val_mse: 0.0166
Epoch 2/5
661/661 [==============================] - ETA: 0s - loss: 0.0157 - mse: 0.0157
Epoch 2: val_loss did not improve from 0.01663
661/661 [==============================] - 3476s 5s/step - loss: 0.0157 - mse:
0.0157 - val_loss: 0.0158 - val_mse: 0.0158
Epoch 3/5
661/661 [==============================] - ETA: 0s - loss: 0.0152 - mse: 0.0152
Epoch 3: val_loss did not improve from 0.01663
661/661 [==============================] - 4343s 7s/step - loss: 0.0152 - mse:
0.0152 - val_loss: 0.0156 - val_mse: 0.0156
Epoch 4/5
661/661 [==============================] - ETA: 0s - loss: 0.0147 - mse: 0.0147
Epoch 4: val_loss did not improve from 0.01663
661/661 [==============================] - 5197s 8s/step - loss: 0.0147 - mse:
0.0147 - val_loss: 0.0159 - val_mse: 0.0159
Epoch 5/5
661/661 [==============================] - ETA: 0s - loss: 0.0144 - mse: 0.0144
Epoch 5: val_loss did not improve from 0.01663
661/661 [==============================] - 5043s 8s/step - loss: 0.0144 - mse:
0.0144 - val_loss: 0.0157 - val_mse: 0.0157
```

[ ]:
```
# 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')



End of Assessment task 2 Part B

by

Hemang Sharma (24695785)

Nusrat Zahan (14367472)

Rajveer Singh Saini (14368005)