In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd
import tensorflow as tf# data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under th
e input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

```
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/test.csv
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/sample_submission.csv
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/test-processed-seqlen128.csv
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/validation-processed-seqlen128.csv
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/jigsaw-toxic-comment-train-
processed-seqlen128.csv
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/validation.csv
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/jigsaw-unintended-bias-train.csv
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/jigsaw-toxic-comment-train.csv
/kaggle/input/jigsaw-multilingual-toxic-comment-classification/jigsaw-unintended-bias-train-
processed-seqlen128.csv
/kaggle/input/images/grenade.png
/kaggle/input/images/swords.png
/kaggle/input/images/nuclear.png
/kaggle/input/images/safe-zone.png
/kaggle/input/images/boxing.png
```

In [2]:

```
!pip install -q pyicu
!pip install -q pycld2
!pip install -q polyglot
!pip install -q textstat
!pip install -q googletrans
```

In [3]:

```
import warnings
warnings.filterwarnings("ignore")
import os
import gc
import re
import folium
import textstat
from scipy import stats
from colorama import Fore, Back, Style, init

import math
import numpy as np
import scipy as sp
import pandas as pd

import random
import networkx as nx
from pandas import Timestamp

from PIL import Image
from IPython.display import SVG
from keras.utils import model_to_dot

import requests
```

```
import requests
from IPython.display import HTML

import seaborn as sns
from tqdm import tqdm
import matplotlib.cm as cm
import matplotlib.pyplot as plt

tqdm.pandas()

import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots

import transformers
import tensorflow as tf

from tensorflow.keras.callbacks import Callback
from sklearn.metrics import accuracy_score, roc_auc_score
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, CSVLogger

from tensorflow.keras.models import Model
from kaggle_datasets import KaggleDatasets
from tensorflow.keras.optimizers import Adam
from tokenizers import BertWordPieceTokenizer
from tensorflow.keras.layers import Dense, Input, Dropout, Embedding
from tensorflow.keras.layers import LSTM, GRU, Conv1D, SpatialDropout1D

from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import activations
from tensorflow.keras import constraints
from tensorflow.keras import initializers
from tensorflow.keras import regularizers

import tensorflow.keras.backend as K
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.activations import *
from tensorflow.keras.constraints import *
from tensorflow.keras.initializers import *
from tensorflow.keras.regularizers import *

from sklearn import metrics
from sklearn.utils import shuffle
from gensim.models import Word2Vec
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.feature_extraction.text import TfidfVectorizer,\
                                            CountVectorizer,\
                                            HashingVectorizer

from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.tokenize import TweetTokenizer

import nltk
from textblob import TextBlob

from nltk.corpus import wordnet
from nltk.corpus import stopwords
from googletrans import Translator
from nltk import WordNetLemmatizer
from polyglot.detect import Detector
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud, STOPWORDS
from nltk.sentiment.vader import SentimentIntensityAnalyzer

stopword=set(STOPWORDS)

lem = WordNetLemmatizer()
tokenizer=TweetTokenizer()

np.random.seed(0)
```

```
Using TensorFlow backend.
```

Importing twython this is an important Twitter package for comment analysis

In [4]:

```
#importing important packages
! pip install -q twython

from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

In [5]:

```
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
import transformers
from transformers import TFAutoModel, AutoTokenizer
from tokenizers import Tokenizer, models, pre_tokenizers, decoders, processors
```

In [6]:

```
train1 = pd.read_csv("/kaggle/input/jigsaw-multilingual-toxic-comment-classification/jigsaw-toxic-comment-train.csv")
train2 = pd.read_csv("/kaggle/input/jigsaw-multilingual-toxic-comment-classification/jigsaw-unintended-bias-train.csv")
train2.toxic = train2.toxic.round().astype(int)

valid = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-comment-classification/validation.csv')
test = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-comment-classification/test.csv')
sub = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-comment-classification/sample_submission.csv')
```

In [7]:

```
train1.tail()
```

Out[7]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
223544	fff8f64043129fa2	:Jerome, I see you never got around to this…! ...	0	0	0	0	0	0
223545	fff9d70fe0722906	==Lucky bastard== \n http://wikimediafoundatio...	0	0	0	0	0	0
223546	fffa8a11c4378854	==shame on you all!!!== \n\n You want to speak...	0	0	0	0	0	0
223547	fffac2a094c8e0e2	MEL GIBSON IS A NAZI BITCH WHO MAKES SHITTY MO...	1	0	1	0	1	0
223548	fffb5451268fb5ba	" \n\n == Unicorn lair discovery == \n\n Suppo...	0	0	0	0	0	0

In [8]:

```
test.head()
```

Out[8]:

	id	content	lang
0	0	Doctor Who adlı viki başlığına 12. doctor olar...	tr
1	1	Вполне возможно, но я пока не вижу необходимо...	ru
2	2	Quindi tu sei uno di quelli conservativi , ...	it
3	3	Malesef gerçekleştirilmedi ancak şöyle bir şey...	tr
4	4	:Resim:Seldabagcan.jpg resminde kaynak sorunu ...	tr

LET'S LOOK AT THE DIVISION

In [9]:

```
print('TRAIN::',train1.shape[0],'TEST::',test.shape[0])
```

```
TRAIN:: 223549 TEST:: 63812
```

FINDING THE MOST COMMON WORDS

In [10]:

```
from wordcloud import WordCloud
import plotly.express as px
import matplotlib.pyplot as plt

def remove_nan(word):
    if type(word)==str:
        return word.replace("\n", "")
    else:
        return ""

#replacing nan with ''
text = ' '.join([remove_nan(x) for x in train1['comment_text']])

wordcloud = WordCloud(max_font_size = None,background_color = 'red',collocations=False,width =1500,
height = 1200).generate(text)
fig = px.imshow(wordcloud)
fig.update_layout(title_text = ' Most Common words in comments')
```

MOST COMMON WORDS WE SEE HERE ARE- article, page,talk, will,one , will, edit ,obscence words are less often

Dealing With Null values

In [11]:

```
print('Train Data Null values:')
check_null = train1.isnull().sum()
print(check_null)
print('Test Data Null values:')
```

```
check_null = test.isnull().sum()
print(check_null)
```

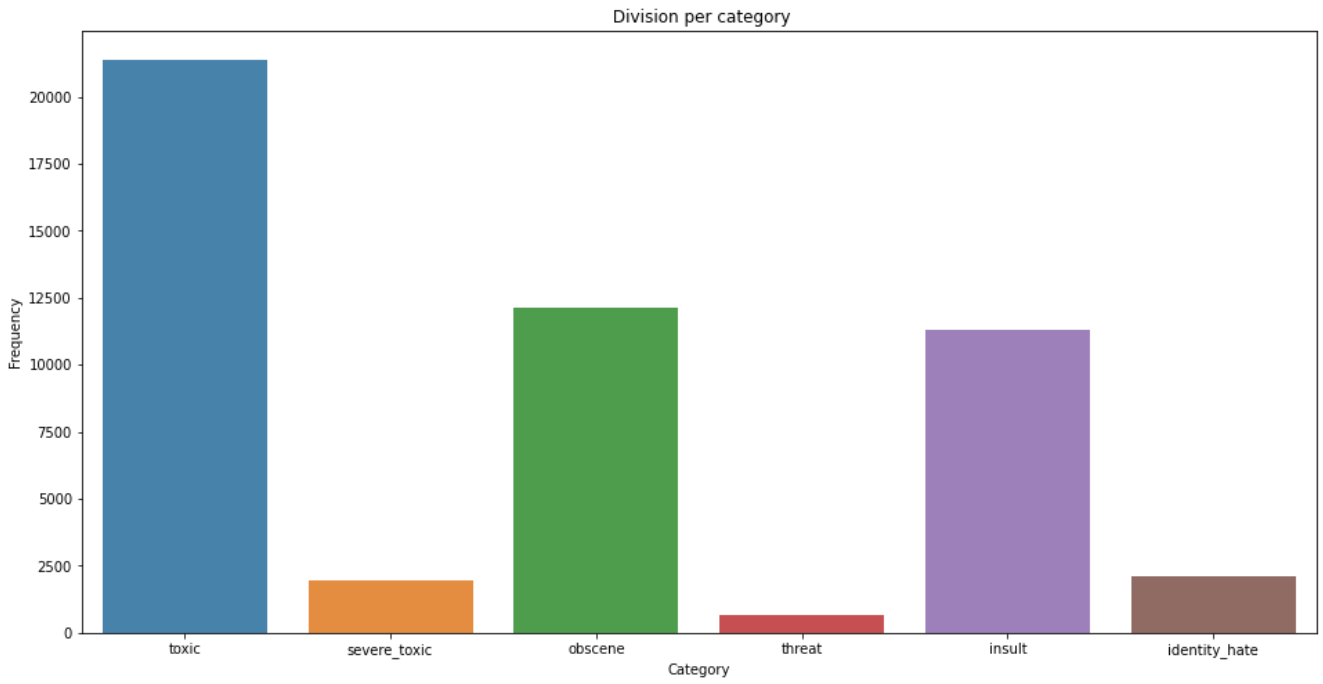
```
Train Data Null values:
id               0
comment_text     0
toxic            0
severe_toxic     0
obscene          0
threat           0
insult           0
identity_hate    0
dtype: int64
Test Data Null values:
id         0
content    0
lang       0
dtype: int64
```

There are no null values the data is clean so far :)

Crrating a bar plot for different negative comments category

In [12]:

```
import seaborn as sns
target_columns = train1.iloc[:,2:].sum()
plt.figure(figsize=(16,8))
ax = sns.barplot(target_columns.index, target_columns.values,alpha=0.9 )
plt.title(" Division per category")
plt.xlabel("Category")
plt.ylabel("Frequency")


plt.show()
```



In [13]:

```
#Checking  the class imbalance
x=train1.iloc[:,2:].sum()
rowsums = train1.iloc[:,2:].sum(axis=1)
train1['clean']= (rowsums==0)
#Checking how many comments are totally clean without any negative tags
train1['clean'].sum()
print('Total comments are  =  ',len(train1))
print('Total clean comments = ',train1['clean'].sum())
```

```
print('Total number of tags',x.sum())
```

```
Total comments are  =   223549
Total clean comments =  201081
Total number of tags 49596
```

Checking for multiple tags

In [14]:

```
train_data=train1
x_data=train_data.iloc[:,2:].sum(axis=1).value_counts()
#plot
import plotly.express as px
fig = px.bar(x_data, x=x_data.index, y=x_data.values)
fig.update_layout(title_text="Multiple tags", template="plotly_white")
fig.show()
```

LET'S GENERATE SOME MORE NASTY COMMENTS FOR FUN !!

In [15]:

```
train_data=train1
import markovify as mk
doc = train_data.loc[train_data.clean==0,'comment_text'].tolist()
text_model = mk.Text(doc)
for i in range(5):
    print(text_model.make_sentence())
```

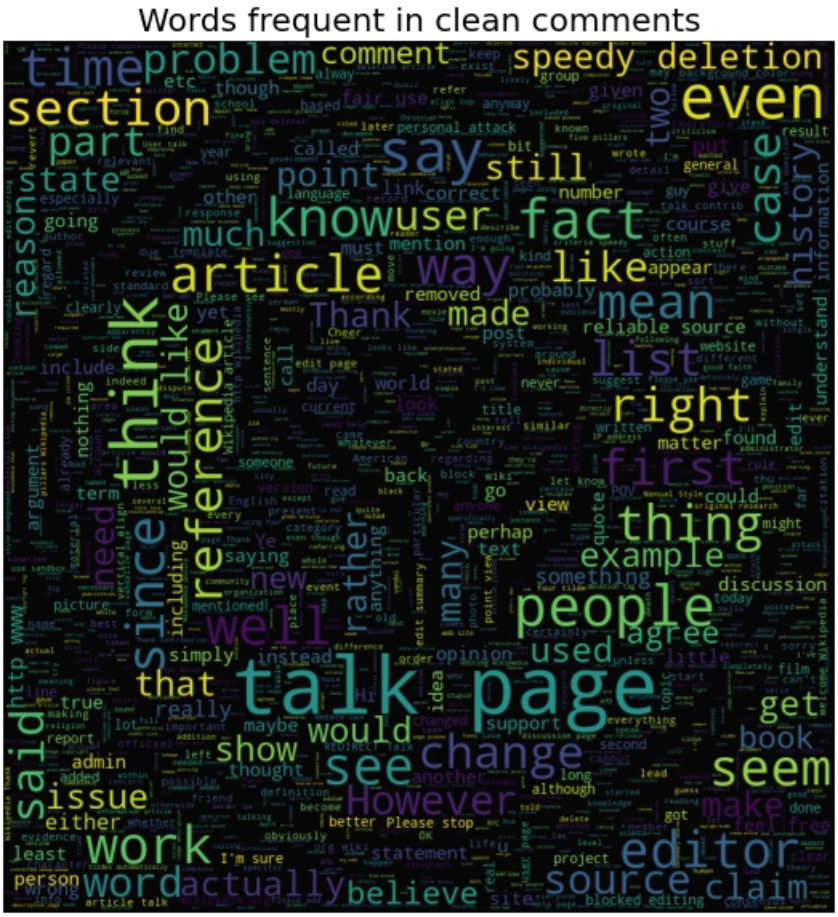
```
Funny how I feel strongly that it is - is the one who's got his head off.
While you'll never make, unless they were just as evil and he is a homo like mitt romney is.
Of course I understand that Fuck use to had so many f.ucking friendship offers to you, then we can
get cancer and die bitch ass nigga eat shit get rid of the Met web-site.
go suck your pussy Fucking in your mother every day, and every black community racist.
I'm honestly surprised you did.
```

Now let's create word clouds of different target tag categories i.e Obscene, Toxic, Identity Threat etc

In [16]:

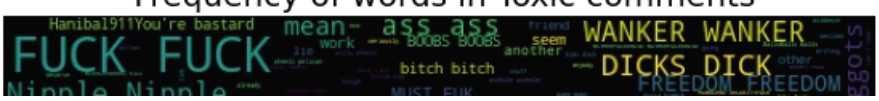
In [16]:

```
from PIL import Image
from nltk.corpus import stopwords
stop_words=set(stopwords.words())
clean_mask = np.array(Image.open("../input/images/safe-zone.png"))
clean_mask = clean_mask[:,:,1]
#wordcloud for clean comments
subset = train_data[train_data.clean==1]
text = subset.comment_text.values
wc = WordCloud(background_color='black',max_words=2000,mask=clean_mask,stopwords=stop_words)
wc.generate(" ".join(text))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title('Words frequent in clean comments',fontsize=20)
plt.imshow(wc.recolor(colormap = 'viridis',random_state=17), alpha=0.98)
plt.show()
```



In [17]:

```
from PIL import Image
from nltk.corpus import stopwords
stop_words=set(stopwords.words())
clean_mask = np.array(Image.open("../input/images/nuclear.png"))
clean_mask = clean_mask[:,:,1]
#wordcloud for clean comments
subset = train_data[train_data.toxic==1]
text = subset.comment_text.values
wc = WordCloud(background_color='black',max_words=2000,mask=clean_mask,stopwords=stop_words)
wc.generate(" ".join(text))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title(' Frequency of words in Toxic comments',fontsize=20)
plt.imshow(wc.recolor(colormap = 'viridis',random_state=17), alpha=0.98)
plt.show()
```

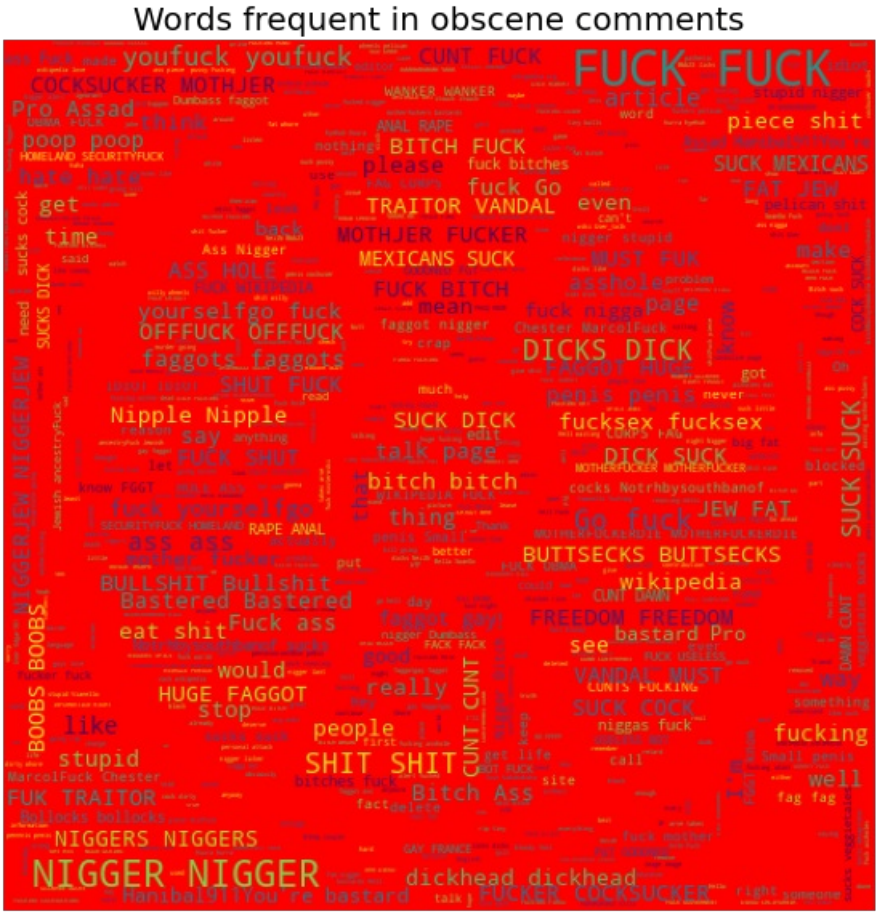


In [18]:

```
from PIL import Image
from nltk.corpus import stopwords
stop_words=set(stopwords.words())
clean_mask = np.array(Image.open("../input/images/nuclear.png"))
clean_mask = clean_mask[:,:,1]
#wordcloud for clean comments
subset = train_data[train_data.severe_toxic==1]
text = subset.comment_text.values
wc = WordCloud(background_color='white',max_words=2000,mask=clean_mask,stopwords=stop_words)
wc.generate(" ".join(text))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title('Words frequent in Severe comments',fontsize=20)
plt.imshow(wc.recolor(colormap = 'viridis',random_state=17), alpha=0.98)
plt.show()
```

In [19]:

```
from PIL import Image
from nltk.corpus import stopwords
stop_words=set(stopwords.words())
clean_mask = np.array(Image.open("../input/images/nuclear.png"))
clean_mask = clean_mask[:,:,1]
#wordcloud for clean comments
subset = train_data[train_data.obscene==1]
text = subset.comment_text.values
wc = WordCloud(background_color='red',max_words=2000,mask=clean_mask,stopwords=stop_words)
wc.generate(" ".join(text))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title('Words frequent in obscene comments',fontsize=20)
plt.imshow(wc.recolor(colormap = 'viridis',random_state=17), alpha=1)
plt.show()
```



In [20]:

```
from PIL import Image
from nltk.corpus import stopwords
stop_words=set(stopwords.words())
clean_mask = np.array(Image.open("../input/images/nuclear.png"))
clean_mask = clean_mask[:,:,1]
#wordcloud for clean comments
subset = train_data[train_data.insult==1]
text = subset.comment_text.values
wc = WordCloud(background_color='black',max_words=2000,mask=clean_mask,stopwords=stop_words)
wc.generate(" ".join(text))
plt.figure(figsize=(20,10))
```

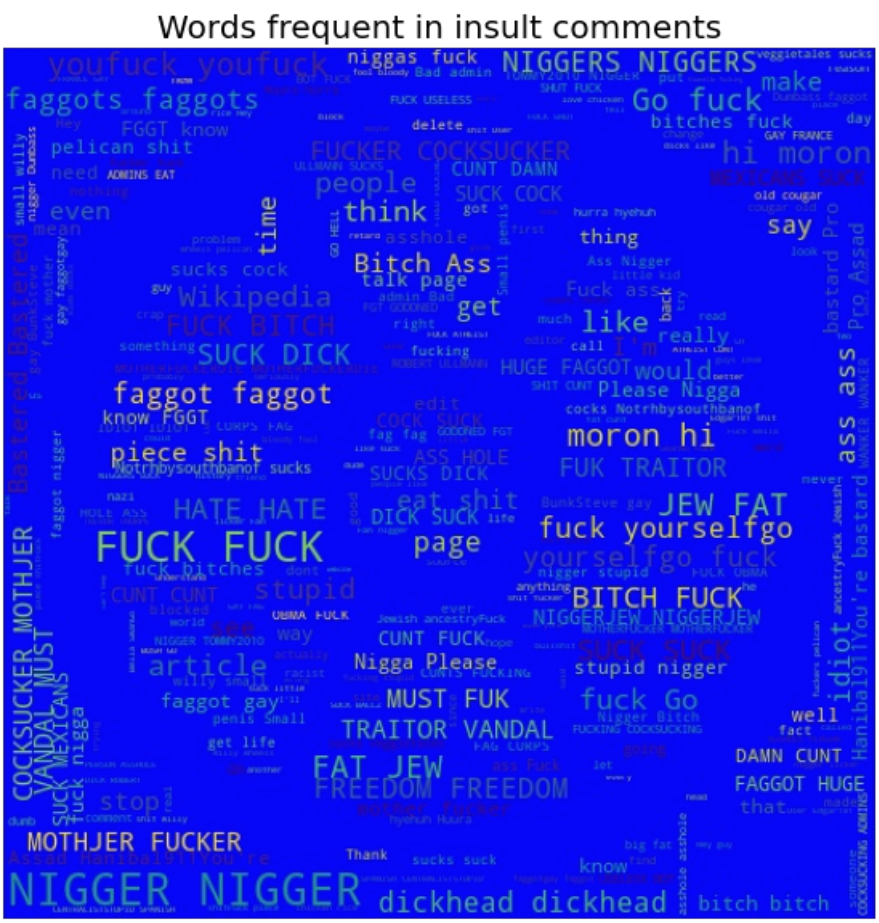
```
plt.axis('off')
plt.title('Words frequent in insult comments',fontsize=20)
plt.imshow(wc.recolor(colormap = 'viridis',random_state=17), alpha=1)
plt.show()
```

In [21]:

```
from PIL import Image
from nltk.corpus import stopwords
stop_words=set(stopwords.words())
clean_mask = np.array(Image.open("../input/images/nuclear.png"))
clean_mask = clean_mask[:,:,1]
#wordcloud for clean comments
subset = train_data[train_data.threat==1]
text = subset.comment_text.values
wc = WordCloud(background_color='pink',max_words=2000,mask=clean_mask,stopwords=stop_words)
wc.generate(" ".join(text))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title('Words frequent in threat comments',fontsize=20)
plt.imshow(wc.recolor(colormap = 'viridis',random_state=17), alpha=1)
plt.show()
```

In [22]:

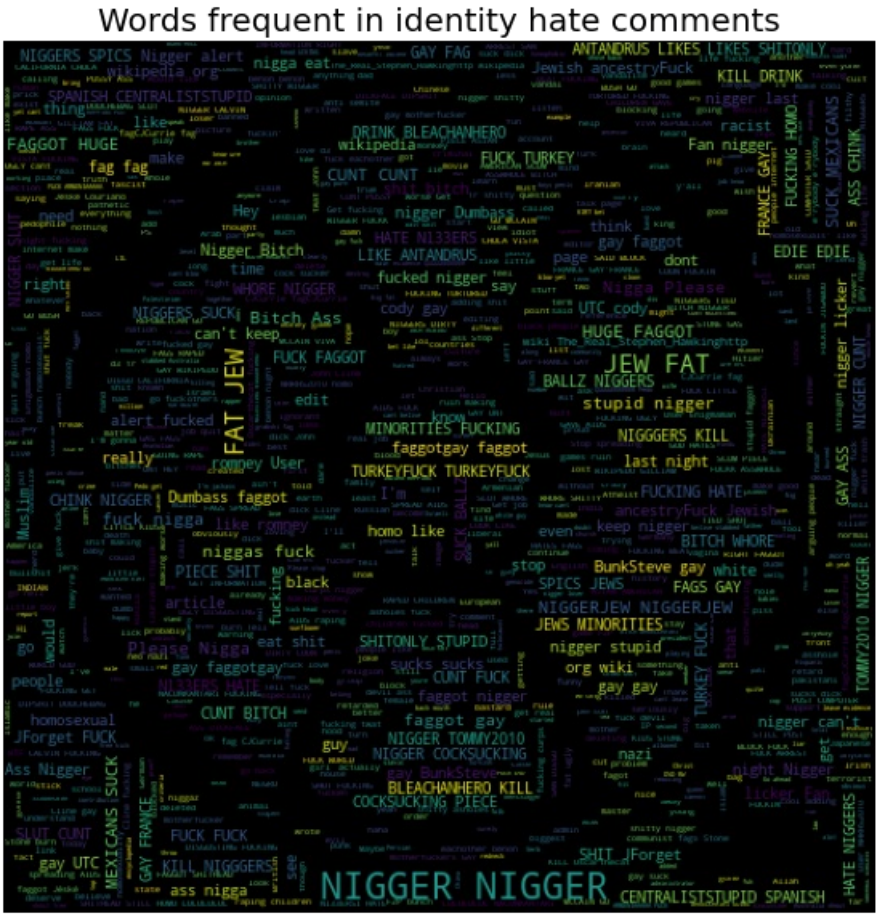
```
from PIL import Image
from nltk.corpus import stopwords
stop_words=set(stopwords.words())
clean_mask = np.array(Image.open("../input/images/nuclear.png"))
clean_mask = clean_mask[:,:,1]
#wordcloud for clean comments
subset = train_data[train_data.insult==1]
text = subset.comment_text.values
wc = WordCloud(background_color='blue',max_words=2000,mask=clean_mask,stopwords=stop_words)
wc.generate(" ".join(text))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title('Words frequent in insult comments',fontsize=20)
plt.imshow(wc.recolor(colormap = 'viridis',random_state=17), alpha=0.98)
plt.show()
```



In [23]:

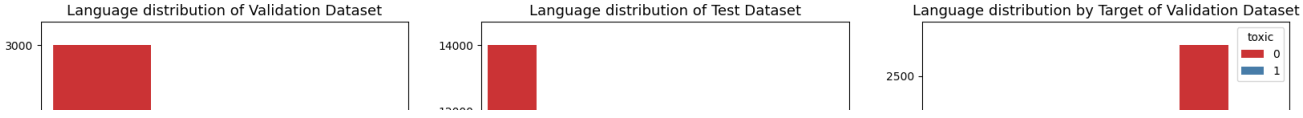
```
from PIL import Image
```

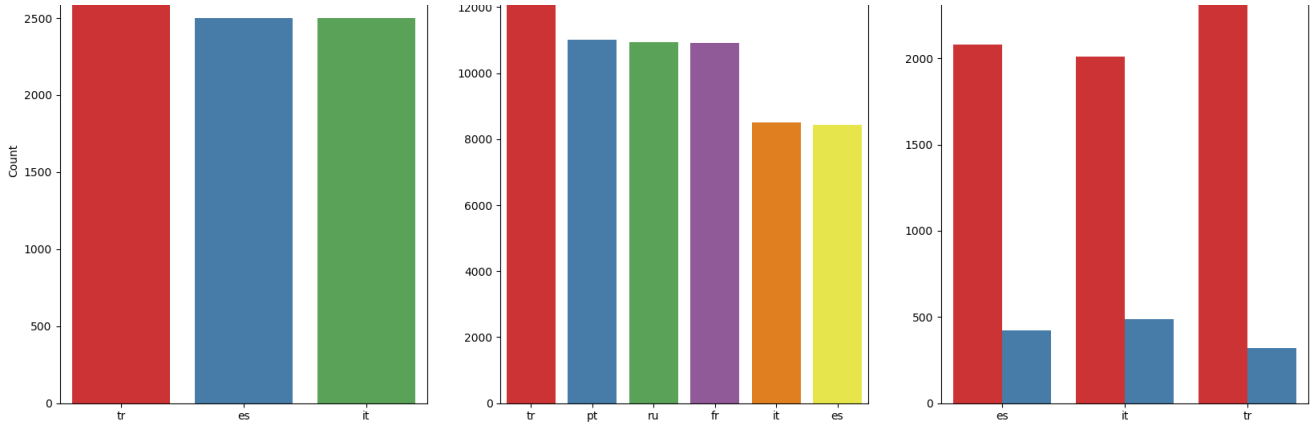
```
from nltk.corpus import stopwords
stop_words=set(stopwords.words())
clean_mask = np.array(Image.open("../input/images/nuclear.png"))
clean_mask = clean_mask[:,:,1]
#wordcloud for clean comments
subset = train_data[train_data.identity_hate==1]
text = subset.comment_text.values
wc = WordCloud(background_color='black',max_words=2000,mask=clean_mask,stopwords=stop_words)
wc.generate(" ".join(text))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title('Words frequent in identity hate comments',fontsize=20)
plt.imshow(wc.recolor(colormap = 'viridis',random_state=17), alpha=1)
plt.show()
```



In [24]:

```
test_data=test
val_data=valid
fig,axes = plt.subplots(ncols=3,figsize=(17,7),dpi=100)
temp = val_data['lang'].value_counts()
sns.barplot(temp.index,temp,ax=axes[0],palette='Set1')

temp = test_data['lang'].value_counts()
sns.barplot(temp.index,temp,ax=axes[1],palette='Set1')
sns.countplot(data=val_data , x='lang' , hue='toxic',ax=axes[2],palette='Set1')
axes[0].set_ylabel('Count')
axes[1].set_ylabel(' ')
axes[2].set_ylabel(' ')
axes[2].set_xlabel(' ')
axes[0].set_title('Language distribution of Validation Dataset',fontsize=13)
axes[1].set_title('Language distribution of Test Dataset', fontsize=13)
axes[2].set_title('Language distribution by Target of Validation Dataset',fontsize=13)
plt.tight_layout()
plt.show()
```





Feature Engineering

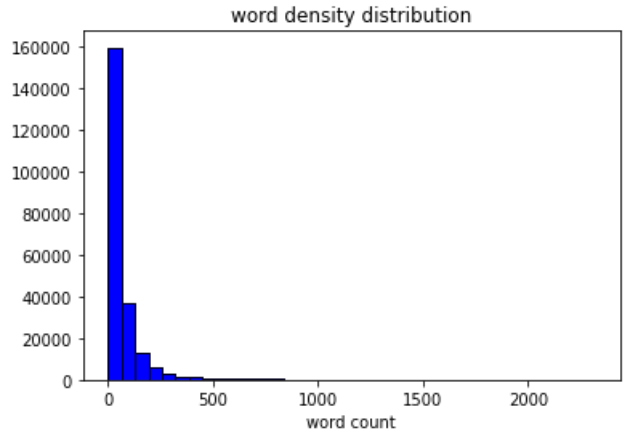
Plot of word density per comment.

In [25]:

```
def word_counter(x):
    if type(x) is str:
        return len(x.split())
    else:
        return 0

train_data['comment_word'] = train_data['comment_text'].apply(word_counter)


import matplotlib.pyplot as plt
import seaborn as sns

# matplotlib histogram
plt.hist(train_data['comment_word'], color = 'blue', edgecolor = 'black',
         bins = int(180/5))

# seaborn histogram
sns.distplot(train_data['comment_word'], hist=True, kde=False,
             bins=int(180/5), color = 'blue',
             hist_kws={'edgecolor':'black'})
# Add labels
plt.title('word density distribution')
plt.xlabel('word count')
plt.ylabel('')
```

Out[25]:

```
Text(0, 0.5, '')
```



Sentiment Analysis

Sentiment Analysis

Generally comments with negative sentiments tend to be more toxic

In [26]:

```
from tqdm import tqdm, tqdm_notebook
tqdm_notebook().pandas()
def polarity(x):
    if type(x) == str:
        return SIA.polarity_scores(x)
    else:
        return 1000

SIA = SentimentIntensityAnalyzer()
train_data["polarity"] = train_data["comment_text"].progress_apply(polarity)
```

In [27]:

```
import plotly.graph_objects as go

fig = go.FigureWidget(go.Histogram(x=[pols["neg"] for pols in train_data["polarity"] if pols["neg"]
!= 0], marker=dict(
            color='skyblue')
    ))

fig.update_layout(xaxis_title="Negativity level", title_text="Negativity sentiment", template="simp
le_white")
fig
```

In [28]:

```
val = val_data
train = train_data

def clean(text):
    text = text.fillna("fillna").str.lower()
    text = text.map(lambda x: re.sub('\\n',' ',str(x)))
    text = text.map(lambda x: re.sub("\[\[User.*",'',str(x)))
    text = text.map(lambda x: re.sub("\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}",'',str(x)))
    text = text.map(lambda x: re.sub("\(http://.*?\s\(http://.*\)",'',str(x)))
    return text

val["comment_text"] = clean(val["comment_text"])
test_data["content"] = clean(test_data["content"])
train["comment_text"] = clean(train["comment_text"])
```

In [29]:

```
# Modelling
```

In [30]:

```
def fast_encode(texts, tokenizer, chunk_size=256, maxlen=512):
    tokenizer.enable_truncation(max_length=maxlen)
    tokenizer.enable_padding(max_length=maxlen)
    all_ids=[]

    for i in range(0, len(texts), chunk_size):
        text_chunk = texts[i:i+chunk_size].tolist()
        encs = tokenizer.encode_batch(text_chunk)
        all_ids.extend([enc.ids for enc in encs])

    return np.array(all_ids)
```

In [31]:

```
def regular_encode(texts,tokenizer,maxlen=512):
    enc_di = tokenizer.batch_encode_plus(
    texts,
    return_attention_masks = False,
    return_token_type_ids = False,
    pad_to_max_length = True,
    max_length = maxlen
    )

    return np.array(enc_di['input_ids'])
```

In [32]:

```
def build_roberta_model(transformer, max_len=512):
    input_word_ids = Input(shape=(max_len,), dtype=tf.int32, name="input_word_ids")
    sequence_output = transformer(input_word_ids)[0]
    cls_token = sequence_output[:, 0, :]
    #cls_token = Dense(500, activation="relu")(cls_token)
    #cls_token = Dropout(0.1)(cls_token)
    out = Dense(1, activation='sigmoid')(cls_token)

    model = Model(inputs=input_word_ids, outputs=out)
    model.compile(Adam(lr=1e-5),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

In [33]:

```
#TPU
try:

    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU', tpu.master())
except ValueError:
    tpu=None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    strategy = tf.distribute.get_strategy()

print("REPLICAS:", strategy.num_replicas_in_sync)
```

```
Running on TPU grpc://10.0.0.2:8470
REPLICAS: 8
```

TPU Configuration

In [34]:

```
AUTO = tf.data.experimental.AUTOTUNE

GCS_DS_PATH = KaggleDatasets().get_gcs_path('jigsaw-multilingual-toxic-comment-classification')

EPOCHS = 2
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
MAX_LEN = 192
MODEL = 'jplu/tf-xlm-roberta-large'
```

In [35]:

```
tokenizer = AutoTokenizer.from_pretrained(MODEL)
```

In [36]:

```
train = pd.concat([
        train1[['comment_text', 'toxic']],
        train2[['comment_text', 'toxic']].query('toxic == 1'),
        train2[['comment_text', 'toxic']].query('toxic == 0').sample(n=100000, random_state=0)

])
```

In [37]:

```
%%time

x_train = regular_encode(train.comment_text.values, tokenizer, maxlen=MAX_LEN)

x_valid = regular_encode(valid.comment_text.values, tokenizer, maxlen = MAX_LEN)

x_test  = regular_encode(test.content.values, tokenizer, maxlen=MAX_LEN)

y_train = train.toxic.values
y_valid = valid.toxic.values
```

```
CPU times: user 7min 57s, sys: 2.48 s, total: 8min
Wall time: 7min 59s
```

Encoding comments for compatibility and getting targets

In [38]:

```
train_dataset = (
        tf.data.Dataset.from_tensor_slices((x_train,y_train)).repeat().shuffle(2048).batch(BATCH_SI
ZE).prefetch(AUTO) )


valid_dataset = (
        tf.data.Dataset.from_tensor_slices((x_valid,y_valid)).batch(BATCH_SIZE).cache().prefetch(AU
TO)
)

test_dataset = (
        tf.data.Dataset.from_tensor_slices(x_test).batch(BATCH_SIZE)
)
```

In [39]:

```
%%time
with strategy.scope():
    transformer_layer = TFAutoModel.from_pretrained(MODEL)
    model = build_roberta_model(transformer_layer, max_len=MAX_LEN)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_word_ids (InputLayer)  [(None, 192)]             0
_________________________________________________________________
tf_roberta_model (TFRobertaM ((None, 192, 1024), (None 559890432
_________________________________________________________________
tf_op_layer_strided_slice (T [(None, 1024)]            0
_________________________________________________________________
dense (Dense)                (None, 1)                 1025
=================================================================
Total params: 559,891,457
Trainable params: 559,891,457
Non-trainable params: 0
_________________________________________________________________
CPU times: user 1min 56s, sys: 42.2 s, total: 2min 38s
Wall time: 2min 37s
```

Training

In [40]:

```
N_STEPS = x_train.shape[0] // BATCH_SIZE


train_history = model.fit(
                train_dataset, steps_per_epoch=N_STEPS,validation_data=valid_dataset,epochs=EPOCHS
)
```

```
Train for 3404 steps, validate for 63 steps
Epoch 1/2
3404/3404 [==============================] - 1846s 542ms/step - loss: 0.0716 - accuracy: 0.9722 -
val_loss: 0.3179 - val_accuracy: 0.8731
Epoch 2/2
3404/3404 [==============================] - 1631s 479ms/step - loss: 0.0561 - accuracy: 0.9781 -
val_loss: 0.3070 - val_accuracy: 0.8779
```

Output

In [41]:

```
sub['toxic'] = model.predict(test_dataset, verbose=1)
sub.to_csv('submission.csv', index=False)
```

```
499/499 [==============================] - 114s 228ms/step
```

In [ ]:

```

```