In [2]:

```python
import pandas as pd

# List of new file names
file_names = [
   'AmericanPolitics_ner.csv',
    'Liberal_ner.csv',
    'obama_ner.csv',
    'Presidentialpoll_ner.csv',
    'PresidentialRaceMemes_ner.csv',
    'uspolitics_ner.csv'
]

# Read and append all CSV files into one DataFrame
combined_df = pd.concat([pd.read_csv(file) for file in file_names])

# Save the combined DataFrame to a new CSV file if needed
combined_df.to_csv('combined_US_Presidents.csv', index=False)

# Display the first few rows of the combined DataFrame to verify
combined_df.head()
```

Out[2]:

	Unnamed: 0	created_utc_x	title	selftext	created_utc_y	subreddit	link_id	
0	92	2011-02-14 20:35:35	Should Donald Trump run for president?	NaN	2011-02-14 22:41:26	AmericanPolitics	t3_flcgc	Shou billic that millior
1	255	2011-04-20 01:18:06	Donald Trump is such an embarrassment to Ameri...	NaN	2011-04-20 12:55:47	AmericanPolitics	t3_gu2an	d trump controll an
2	256	2011-04-20 01:18:06	Donald Trump is such an embarrassment to Ameri...	NaN	2011-04-20 20:22:13	AmericanPolitics	t3_gu2an	Everyo matter your should
3	440	2011-06-28 08:54:25	Borrowing and spending the GOP way -- The big ...	NaN	2011-06-28 12:13:43	AmericanPolitics	t3_ib5c3	&gt;Cor th sigr GOP pc
4	453	2011-06-30 17:13:34	MSNBC suspends journalist over Barack Obama in...	NaN	2011-06-30 17:40:27	AmericanPolitics	t3_iderp	So is fact th said ' inst

In [3]:

```python
combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 55385 entries, 0 to 13583
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Unnamed: 0       55385 non-null  int64 
 1   created_utc_x    55385 non-null  object
 2   title            55385 non-null  object
 3   selftext         11321 non-null  object
 4   created_utc_y    55385 non-null  object
 5   subreddit        55385 non-null  object
 6   link_id          55385 non-null  object
 7   body             55382 non-null  object
 8   score            55385 non-null  int64 
 9   persons_title    55385 non-null  object
 10  persons_selftext 55385 non-null  object
 11  persons_body     55385 non-null  object
dtypes: int64(2), object(10)
memory usage: 5.5+ MB
```

In [23]:

```python
import pandas as pd
import random
from detoxify import Detoxify
import matplotlib.pyplot as plt

# Initialize Detoxify model for toxicity analysis
model = Detoxify('original')

# Assuming combined_df is your combined DataFrame
combined_df = pd.concat([pd.read_csv(file) for file in file_names])

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

# Sample 20% of the DataFrame randomly
sampled_df = combined_df.sample(frac=0.20)

# Define the function to compute toxicity for each sentence
def compute_toxicity(text):
    if pd.isna(text):
        return 0
    # Split text into sentences
    sentences = text.split('.')
    # Compute toxicity for each sentence
    toxicity_scores = [model.predict(sentence)['toxicity'] for sentence in sentences i
    # Return the mean toxicity score for the text
    return sum(toxicity_scores) / len(toxicity_scores) if toxicity_scores else 0

# Compute toxicity scores for 'title', 'selftext', and 'body' columns in the sampled D
sampled_df['toxicity_title'] = sampled_df['title'].apply(compute_toxicity)
sampled_df['toxicity_selftext'] = sampled_df['selftext'].apply(compute_toxicity)
sampled_df['toxicity_body'] = sampled_df['body'].apply(compute_toxicity)
```

```
C:\Users\HP\AppData\Roaming\Python\Python311\site-packages\transformers\tokenization_
utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set. It wil
l be set to `True` by default. This behavior will be depracted in transformers v4.45,
and will be then set to `False` by default. For more details check this issue: http
s://github.com/huggingface/transformers/issues/31884
  warnings.warn(
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[23], line 30
     27     return sum(toxicity_scores) / len(toxicity_scores) if toxicity_scores els
e 0
     29 # Compute toxicity scores for 'title', 'selftext', and 'body' columns in the
sampled DataFrame
---> 30 sampled_df['toxicity_title'] = sampled_df['title'].apply(compute_toxicity)
     31 sampled_df['toxicity_selftext'] = sampled_df['selftext'].apply(compute_toxici
ty)
     32 sampled_df['toxicity_body'] = sampled_df['body'].apply(compute_toxicity)

File ~\AppData\Roaming\Python\Python311\site-packages\pandas\core\series.py:4924, in
Series.apply(self, func, convert_dtype, args, by_row, **kwargs)
   4789 def apply(
   4790     self,
   4791     func: AggFuncType,
   (...)
   4796     **kwargs,
   4797 ) -> DataFrame | Series:
   4798     """
   4799     Invoke function on values of Series.
   4800
   (...)
   4915     dtype: float64
   4916     """
   4917     return SeriesApply(
   4918         self,
   4919         func,
   4920         convert_dtype=convert_dtype,
   4921         by_row=by_row,
   4922         args=args,
   4923         kwargs=kwargs,
-> 4924     ).apply()

File ~\AppData\Roaming\Python\Python311\site-packages\pandas\core\apply.py:1427, in S
eriesApply.apply(self)
   1424     return self.apply_compat()
   1426 # self.func is Callable
-> 1427 return self.apply_standard()

File ~\AppData\Roaming\Python\Python311\site-packages\pandas\core\apply.py:1507, in S
eriesApply.apply_standard(self)
   1501 # row-wise access
   1502 # apply doesn't have a `na_action` keyword and for backward compat reasons
   1503 # we need to give `na_action="ignore"` for categorical data.
   1504 # TODO: remove the `na_action="ignore"` when that default has been changed in
   1505 #  Categorical (GH51645).
   1506 action = "ignore" if isinstance(obj.dtype, CategoricalDtype) else None
-> 1507 mapped = obj._map_values(
   1508     mapper=curried, na_action=action, convert=self.convert_dtype
   1509 )
   1511 if len(mapped) and isinstance(mapped[0], ABCSeries):
   1512     # GH#43986 Need to do list(mapped) in order to get treated as nested
   1513     #  See also GH#25959 regarding EA support
   1514     return obj._constructor_expanddim(list(mapped), index=obj.index)

File ~\AppData\Roaming\Python\Python311\site-packages\pandas\core\base.py:921, in Ind
exOpsMixin._map_values(self, mapper, na_action, convert)
    918 if isinstance(arr, ExtensionArray):
```

```
    919     return arr.map(mapper, na_action=na_action)
--> 921 return algorithms.map_array(arr, mapper, na_action=na_action, convert=conver
t)

File ~\AppData\Roaming\Python\Python311\site-packages\pandas\core\algorithms.py:1743,
in map_array(arr, mapper, na_action, convert)
   1741 values = arr.astype(object, copy=False)
   1742 if na_action is None:
-> 1743     return lib.map_infer(values, mapper, convert=convert)
   1744 else:
   1745     return lib.map_infer_mask(
   1746         values, mapper, mask=isna(values).view(np.uint8), convert=convert
   1747     )

File lib.pyx:2972, in pandas._libs.lib.map_infer()

Cell In[23], line 25, in compute_toxicity(text)
     23 sentences = text.split('.')
     24 # Compute toxicity for each sentence
---> 25 toxicity_scores = [model.predict(sentence)['toxicity'] for sentence in senten
ces if sentence.strip() != '']
     26 # Return the mean toxicity score for the text
     27 return sum(toxicity_scores) / len(toxicity_scores) if toxicity_scores else 0

Cell In[23], line 25, in <listcomp>(.0)
     23 sentences = text.split('.')
     24 # Compute toxicity for each sentence
---> 25 toxicity_scores = [model.predict(sentence)['toxicity'] for sentence in senten
ces if sentence.strip() != '']
     26 # Return the mean toxicity score for the text
     27 return sum(toxicity_scores) / len(toxicity_scores) if toxicity_scores else 0

File ~\AppData\Roaming\Python\Python311\site-packages\torch\utils\_contextlib.py:115,
in context_decorator.<locals>.decorate_context(*args, **kwargs)
    112 @functools.wraps(func)
    113 def decorate_context(*args, **kwargs):
    114     with ctx_factory():
--> 115         return func(*args, **kwargs)

File ~\AppData\Roaming\Python\Python311\site-packages\detoxify\detoxify.py:117, in De
toxify.predict(self, text)
    115 self.model.eval()
    116 inputs = self.tokenizer(text, return_tensors="pt", truncation=True, padding=T
rue).to(self.model.device)
--> 117 out = self.model(**inputs)[0]
    118 scores = torch.sigmoid(out).cpu().detach().numpy()
    119 results = {}

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:152
0, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hoo
ks or self._forward_pre_hooks
```

```
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File ~\AppData\Roaming\Python\Python311\site-packages\transformers\models\bert\modeli
ng_bert.py:1695, in BertForSequenceClassification.forward(self, input_ids, attention_
mask, token_type_ids, position_ids, head_mask, inputs_embeds, labels, output_attentio
ns, output_hidden_states, return_dict)
   1687 r"""
   1688 labels (`torch.LongTensor` of shape `(batch_size,)`, *optional*):
   1689     Labels for computing the sequence classification/regression loss. Indices
should be in `[0, ...,
   1690     config.num_labels - 1]`. If `config.num_labels == 1` a regression loss is
computed (Mean-Square loss), If
   1691     `config.num_labels > 1` a classification loss is computed (Cross-Entrop
y).
   1692 """
   1693 return_dict = return_dict if return_dict is not None else self.config.use_ret
urn_dict
-> 1695 outputs = self.bert(
   1696     input_ids,
   1697     attention_mask=attention_mask,
   1698     token_type_ids=token_type_ids,
   1699     position_ids=position_ids,
   1700     head_mask=head_mask,
   1701     inputs_embeds=inputs_embeds,
   1702     output_attentions=output_attentions,
   1703     output_hidden_states=output_hidden_states,
   1704     return_dict=return_dict,
   1705 )
   1707 pooled_output = outputs[1]
   1709 pooled_output = self.dropout(pooled_output)

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:152
0, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hoo
ks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File ~\AppData\Roaming\Python\Python311\site-packages\transformers\models\bert\modeli
ng_bert.py:1141, in BertModel.forward(self, input_ids, attention_mask, token_type_id
s, position_ids, head_mask, inputs_embeds, encoder_hidden_states, encoder_attention_m
ask, past_key_values, use_cache, output_attentions, output_hidden_states, return_dic
t)
   1134 # Prepare head mask if needed
   1135 # 1.0 in head_mask indicate we keep the head
   1136 # attention_probs has shape bsz x n_heads x N x N
   1137 # input head_mask has shape [num_heads] or [num_hidden_layers x num_heads]
   1138 # and head_mask is converted to shape [num_hidden_layers x batch x num_heads
x seq_length x seq_length]
   1139 head_mask = self.get_head_mask(head_mask, self.config.num_hidden_layers)
-> 1141 encoder_outputs = self.encoder(
   1142     embedding_output,
   1143     attention_mask=extended_attention_mask,
   1144     head_mask=head_mask,
   1145     encoder_hidden_states=encoder_hidden_states,
   1146     encoder_attention_mask=encoder_extended_attention_mask,
   1147     past_key_values=past_key_values,
   1148     use_cache=use_cache,
   1149     output_attentions=output_attentions,
   1150     output_hidden_states=output_hidden_states,
   1151     return_dict=return_dict,
   1152 )
   1153 sequence_output = encoder_outputs[0]
   1154 pooled_output = self.pooler(sequence_output) if self.pooler is not None else
None

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:152
0, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hoo
ks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File ~\AppData\Roaming\Python\Python311\site-packages\transformers\models\bert\modeli
ng_bert.py:694, in BertEncoder.forward(self, hidden_states, attention_mask, head_mas
k, encoder_hidden_states, encoder_attention_mask, past_key_values, use_cache, output_
attentions, output_hidden_states, return_dict)
    683     layer_outputs = self._gradient_checkpointing_func(
    684         layer_module.__call__,
    685         hidden_states,
   (...)
    691         output_attentions,
    692     )
    693 else:
--> 694     layer_outputs = layer_module(
    695         hidden_states,
    696         attention_mask,
    697         layer_head_mask,
    698         encoder_hidden_states,
    699         encoder_attention_mask,
    700         past_key_value,
    701         output_attentions,
    702     )
    704 hidden_states = layer_outputs[0]
    705 if use_cache:

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:152
0, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hoo
ks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File ~\AppData\Roaming\Python\Python311\site-packages\transformers\models\bert\modeli
ng_bert.py:626, in BertLayer.forward(self, hidden_states, attention_mask, head_mask,
encoder_hidden_states, encoder_attention_mask, past_key_value, output_attentions)
    623     cross_attn_present_key_value = cross_attention_outputs[-1]
    624     present_key_value = present_key_value + cross_attn_present_key_value
--> 626 layer_output = apply_chunking_to_forward(
    627     self.feed_forward_chunk, self.chunk_size_feed_forward, self.seq_len_dim,
attention_output
    628 )
    629 outputs = (layer_output,) + outputs
    631 # if decoder, return the attn key/values as the last output

File ~\AppData\Roaming\Python\Python311\site-packages\transformers\pytorch_utils.py:2
39, in apply_chunking_to_forward(forward_fn, chunk_size, chunk_dim, *input_tensors)
    236     # concatenate output at same dimension
    237     return torch.cat(output_chunks, dim=chunk_dim)
--> 239 return forward_fn(*input_tensors)

File ~\AppData\Roaming\Python\Python311\site-packages\transformers\models\bert\modeli
ng_bert.py:638, in BertLayer.feed_forward_chunk(self, attention_output)
    637 def feed_forward_chunk(self, attention_output):
--> 638     intermediate_output = self.intermediate(attention_output)
    639     layer_output = self.output(intermediate_output, attention_output)
    640     return layer_output

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:152
0, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hoo
ks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
```

```
   1522 try:
   1523     result = None

File ~\AppData\Roaming\Python\Python311\site-packages\transformers\models\bert\modeli
ng_bert.py:538, in BertIntermediate.forward(self, hidden_states)
    537 def forward(self, hidden_states: torch.Tensor) -> torch.Tensor:
--> 538     hidden_states = self.dense(hidden_states)
    539     hidden_states = self.intermediate_act_fn(hidden_states)
    540     return hidden_states

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\module.py:152
0, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hoo
ks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File ~\AppData\Roaming\Python\Python311\site-packages\torch\nn\modules\linear.py:116,
in Linear.forward(self, input)
    115 def forward(self, input: Tensor) -> Tensor:
--> 116     return F.linear(input, self.weight, self.bias)

KeyboardInterrupt: 
```

In [ ]:

```python
sampled_df.head()
```

In [ ]:

```python
sampled_df.info()
```

In [6]:

```python
# Save the sampled DataFrame to the specified file path
output_file_path = 'C:\\Users\\HP\\Desktop\\Middlesex Course Content\\Giovanni Propose

sampled_df.to_csv(output_file_path, index=False)
```

In [6]:

```python
sampled_df = pd.read_csv('US_Selected_President_Toxicity.csv')

# Display the first few rows to ensure it's loaded correctly
sampled_df.head()
```

Out[6]:

	Unnamed: 0	created_utc_x	title	selftext	created_utc_y	subreddit	link_id	
0	123606	2022-05-02 23:13:13	Opinion | New round of text messages exposes F...	NaN	2022-05-02 23:14:21	uspolitics	t3_uh2end	& c
1	76491	2020-06-08 05:32:48	To go up against Trump, the DNC picks... Joe B...	NaN	2020-06-08 21:47:44	PresidentialRaceMemes	t3_gysvlg	wh nc it's
2	160377	2021-05-29 16:14:47	How Mitch McConnell killed the US Capitol atta...	NaN	2021-05-29 22:40:36	Liberal	t3_nnqps4	I i Rep w
3	163600	2022-12-19 19:31:38	January 6 committee says Donald Trump violated...	NaN	2022-12-19 23:08:51	uspolitics	t3_zq13wb	[
4	38225	2020-05-22 18:08:17	So I wrote back...	NaN	2020-05-22 21:49:13	PresidentialRaceMemes	t3_gooiv0	Tru s

For each subreddit, compute the volume of posts and the average toxicity (even computed on a sample of posts/comments)

In [29]:

```
# Calculate the volume of posts per subreddit
volume_per_subreddit = sampled_df['subreddit'].value_counts()

# Calculate the average toxicity per subreddit
toxicity_columns = ['toxicity_title', 'toxicity_selftext', 'toxicity_body']
average_toxicity_per_subreddit = sampled_df.groupby('subreddit')[toxicity_columns].mea

# Combine the volume and average toxicity into a single DataFrame
result_df = volume_per_subreddit.to_frame(name='volume').join(average_toxicity_per_sub

result_df
```

Out[29]:

subreddit	volume	toxicity_title	toxicity_selftext	toxicity_body
Liberal	2758	0.055781	0.010714	0.090858
PresidentialRaceMemes	2747	0.089217	0.000326	0.106512
uspolitics	2742	0.039748	0.002658	0.096191
obama	1034	0.016698	0.002645	0.055371
Presidentialpoll	923	0.013800	0.006540	0.057242
AmericanPolitics	873	0.056920	0.000031	0.108735

Sorted by Avg Toxicity

In [30]:

```python
# Combine the volume and average toxicity into a single DataFrame
result_df = volume_per_subreddit.to_frame(name='volume').join(average_toxicity_per_sub

# Calculate the overall average toxicity for each subreddit
result_df['avg_toxicity'] = result_df[toxicity_columns].mean(axis=1)

# Sort the DataFrame by overall average toxicity in descending order
sorted_result_df = result_df.sort_values(by='avg_toxicity', ascending=False)

# Display the sorted results
sorted_result_df
```

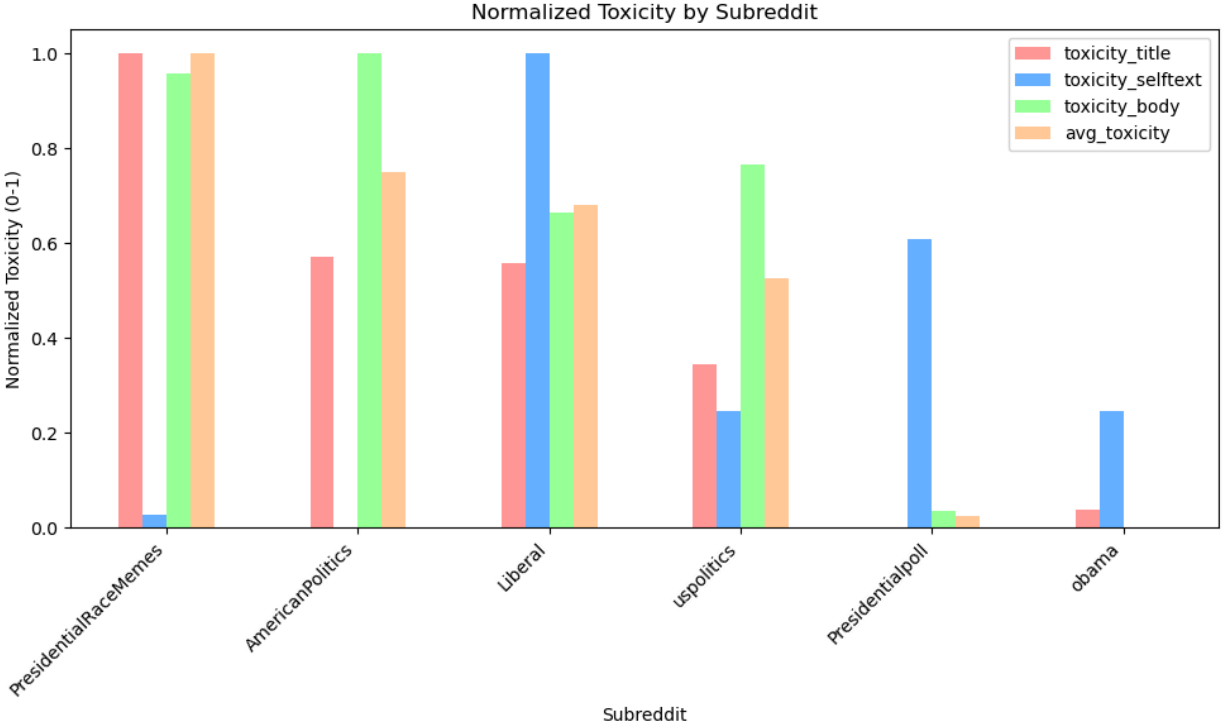
Out[30]:

subreddit	volume	toxicity_title	toxicity_selftext	toxicity_body	avg_toxicity
PresidentialRaceMemes	2747	0.089217	0.000326	0.106512	0.065351
AmericanPolitics	873	0.056920	0.000031	0.108735	0.055229
Liberal	2758	0.055781	0.010714	0.090858	0.052451
uspolitics	2742	0.039748	0.002658	0.096191	0.046199
Presidentialpoll	923	0.013800	0.006540	0.057242	0.025861
obama	1034	0.016698	0.002645	0.055371	0.024905

In [31]:

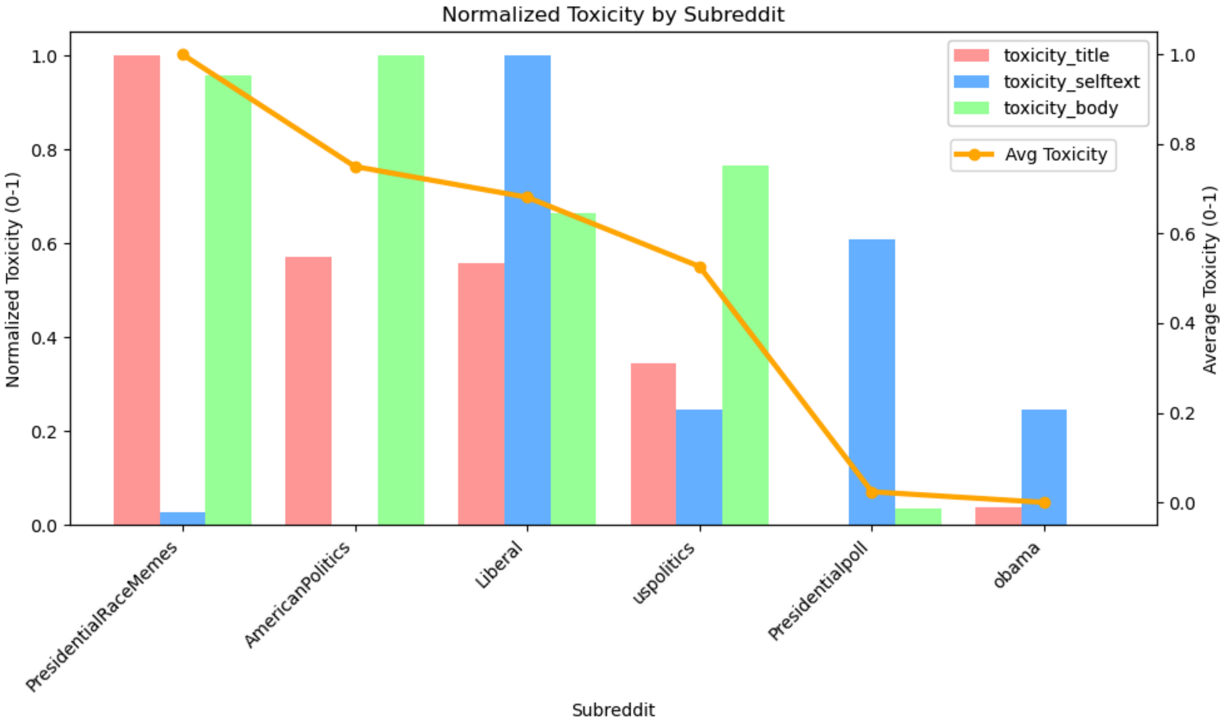
```python
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt

# Provided data
data = {
    'subreddit': ['PresidentialRaceMemes', 'AmericanPolitics', 'Liberal', 'uspolitics'
    'volume': [2747, 873, 2758, 2742, 923, 1034],
    'toxicity_title': [0.089217, 0.056920, 0.055781, 0.039748, 0.013800, 0.016698],
    'toxicity_selftext': [0.000326, 0.000031, 0.010714, 0.002658, 0.006540, 0.002645],
    'toxicity_body': [0.106512, 0.108735, 0.090858, 0.096191, 0.057242, 0.055371],
    'avg_toxicity': [0.065351, 0.055229, 0.052451, 0.046199, 0.025861, 0.024905]
}
```

```
# Creating a DataFrame
df = pd.DataFrame(data)

# Normalize the toxicity columns using Min-Max Scaling
columns_to_normalize = ['toxicity_title', 'toxicity_selftext', 'toxicity_body', 'avg_t

# Apply Min-Max normalization
df_normalized = df.copy()  # Create a copy of the original data
for col in columns_to_normalize:
    df_normalized[col] = (df[col] - df[col].min()) / (df[col].max() - df[col].min())

# Plotting the normalized values for graphical representation
df_normalized.set_index('subreddit', inplace=True)

# Plot normalized toxicity data
df_normalized[columns_to_normalize].plot(kind='bar', figsize=(10, 6), color=['#FF9999'
plt.title('Normalized Toxicity by Subreddit')
plt.ylabel('Normalized Toxicity (0-1)')
plt.xlabel('Subreddit')
plt.xticks(rotation=45, ha='right')
plt.legend(loc='upper right')
plt.tight_layout()

# Show the plot
plt.show()
```



In [32]:

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt

# Provided data
data = {
    'subreddit': ['PresidentialRaceMemes', 'AmericanPolitics', 'Liberal', 'uspolitics'
    'volume': [2747, 873, 2758, 2742, 923, 1034],
    'toxicity_title': [0.089217, 0.056920, 0.055781, 0.039748, 0.013800, 0.016698],
    'toxicity_selftext': [0.000326, 0.000031, 0.010714, 0.002658, 0.006540, 0.002645],
    'toxicity_body': [0.106512, 0.108735, 0.090858, 0.096191, 0.057242, 0.055371],
    'avg_toxicity': [0.065351, 0.055229, 0.052451, 0.046199, 0.025861, 0.024905]
}

# Creating a DataFrame
df = pd.DataFrame(data)

# Normalize the toxicity columns using Min-Max Scaling
columns_to_normalize = ['toxicity_title', 'toxicity_selftext', 'toxicity_body', 'avg_t

# Apply Min-Max normalization
df_normalized = df.copy()  # Create a copy of the original data
for col in columns_to_normalize:
    df_normalized[col] = (df[col] - df[col].min()) / (df[col].max() - df[col].min())

# Set 'subreddit' as index for labeling x-axis
df_normalized.set_index('subreddit', inplace=True)

# Plotting the data
fig, ax1 = plt.subplots(figsize=(10, 6))

# Plot bar graph for title, selftext, body toxicities
bar_colors = ['#FF9999', '#66B2FF', '#99FF99']
df_normalized[['toxicity_title', 'toxicity_selftext', 'toxicity_body']].plot(kind='bar
ax1.set_ylabel('Normalized Toxicity (0-1)')
ax1.set_xlabel('Subreddit')
ax1.set_title('Normalized Toxicity by Subreddit')

# Ensure the x-axis labels are the subreddit names
ax1.set_xticks(range(len(df_normalized.index)))
ax1.set_xticklabels(df_normalized.index, rotation=45, ha='right')  # Rotate x-axis lab

# Create a secondary axis for average toxicity as a line graph
ax2 = ax1.twinx()
ax2.plot(df_normalized.index, df_normalized['avg_toxicity'], color='orange', marker='o
ax2.set_ylabel('Average Toxicity (0-1)')

# Adding the legends for both bar and line plots at the upper right
bars_legend = ax1.legend(df_normalized[['toxicity_title', 'toxicity_selftext', 'toxici
line_legend = ax2.legend(['Avg Toxicity'], loc='upper right', bbox_to_anchor=(0.97, 0.

# Ensure both legends are visible without overlapping
ax1.add_artist(bars_legend)

# Adjust layout to make sure everything fits
plt.tight_layout()

# Show plot
plt.show()
```



In [33]:
```
import re
import pandas as pd

# Define the dictionary mapping variations to the standard form
name_variations = {
    r'\b(?:donald\s+trump|trump|donald(?:\s+trump)?)\b': 'Donald Trump',
    r'\b(?:joe\s+biden|biden|joe(?:\s+biden)?)\b': 'Joe Biden',
    r'\b(?:kamala\s+harris|harris|kamala(?:\s+harris)?)\b': 'Kamala Harris',
    r'\b(?:george\s+w\.\s+bush|bush|george(?:\s+w\.\s+bush)?)\b': 'George W. Bush',
    r'\b(?:barack\s+obama|obama|barack(?:\s+obama)?)\b': 'Barack Obama',
}

# Function to replace variations with standard names
def standardize_names(text):
    for pattern, standard_name in name_variations.items():
        text = re.sub(pattern, standard_name, text, flags=re.IGNORECASE)
    return text
```

For every US president, compute the toxicity and number of mentions in each subreddit

In [34]:
```
import re
import pandas as pd

# Define the dictionary mapping variations to the standard form
name_variations = {
  r'\b(?:george\s+w\.\s+bush|bush|george(?:\s+w\.\s+bush)?)\b': 'George W. Bush',
  r'\b(?:barack\s+obama|obama|barack(?:\s+obama)?)\b': 'Barack Obama',
  r'\b(?:bill\s+clinton|clinton|bill(?:\s+clinton)?)\b': 'Bill Clinton',
  r'\b(?:donald\s+trump|trump|donald(?:\s+trump)?)\b': 'Donald Trump',
  r'\b(?:joe\s+biden|biden|joe(?:\s+biden)?)\b': 'Joe Biden',
  r'\b(?:kamala\s+harris|harris|kamala(?:\s+harris)?)\b': 'Kamala Harris'
}

# Function to standardize names in a text
def standardize_names(text):
    for pattern, standard_name in name_variations.items():
        text = re.sub(pattern, standard_name, text, flags=re.IGNORECASE)
    return text

# Function to extract persons from a given text
def extract_persons(text):
    persons_found = []
    for pattern, standard_name in name_variations.items():
        if re.search(pattern, text, flags=re.IGNORECASE):
            persons_found.append(standard_name)
    return persons_found
```

In [35]:

```
# Convert all relevant columns to strings and fill NaNs with empty strings
sampled_df['title'] = sampled_df['title'].fillna('').astype(str)
sampled_df['selftext'] = sampled_df['selftext'].fillna('').astype(str)
sampled_df['body'] = sampled_df['body'].fillna('').astype(str)

# Standardize names in relevant columns
sampled_df['title'] = sampled_df['title'].apply(standardize_names)
sampled_df['selftext'] = sampled_df['selftext'].apply(standardize_names)
sampled_df['body'] = sampled_df['body'].apply(standardize_names)

# Extract persons from relevant columns
sampled_df['persons_title'] = sampled_df['title'].apply(extract_persons)
sampled_df['persons_selftext'] = sampled_df['selftext'].apply(extract_persons)
sampled_df['persons_body'] = sampled_df['body'].apply(extract_persons)

# Initialize an empty list to store results
results = []

# Iterate over each president
for president in set(name_variations.values()):
    # Filter rows where the president is mentioned in any of the columns
    mentions_df = sampled_df[
        sampled_df['persons_title'].apply(lambda x: president in x) |
        sampled_df['persons_selftext'].apply(lambda x: president in x) |
        sampled_df['persons_body'].apply(lambda x: president in x)
    ]

    # Group by subreddit and calculate the number of mentions and average toxicity
    grouped = mentions_df.groupby('subreddit').agg(
        mentions=('title', 'size'),
        avg_toxicity_title=('toxicity_title', 'mean'),
        avg_toxicity_selftext=('toxicity_selftext', 'mean'),
        avg_toxicity_body=('toxicity_body', 'mean')
    ).reset_index()

    # Add a column for the president's name
    grouped['president'] = president

    # Append the result to the list
    results.append(grouped)

# Concatenate all results into a single DataFrame
results_df = pd.concat(results, ignore_index=True)
```

```
# Display the results
results_df
```

Out[35]:

	subreddit	mentions	avg_toxicity_title	avg_toxicity_selftext	avg_toxicity_body	preside
0	AmericanPolitics	614	0.056452	0.000041	0.104964	Dona Trur
1	Liberal	2134	0.060496	0.012741	0.091588	Dona Trur
2	PresidentialRaceMemes	875	0.082616	0.000441	0.101608	Dona Trur
3	Presidentialpoll	460	0.010859	0.007183	0.049262	Dona Trur
4	obama	83	0.055568	0.000037	0.043310	Dona Trur
5	uspolitics	2322	0.035533	0.002654	0.095953	Dona Trur
6	AmericanPolitics	76	0.113220	0.000000	0.086676	Bara Obar
7	Liberal	253	0.061366	0.011242	0.090059	Bara Obar
8	PresidentialRaceMemes	212	0.067503	0.000514	0.092119	Bara Obar
9	Presidentialpoll	223	0.017083	0.008239	0.060269	Bara Obar
10	obama	1016	0.015971	0.002691	0.055527	Bara Obar
11	uspolitics	143	0.034819	0.007491	0.066935	Bara Obar
12	AmericanPolitics	317	0.032044	0.000015	0.140476	Joe Bid
13	Liberal	753	0.027271	0.012330	0.077253	Joe Bid
14	PresidentialRaceMemes	2432	0.090722	0.000357	0.109750	Joe Bid
15	Presidentialpoll	432	0.010167	0.006394	0.063641	Joe Bid
16	obama	53	0.005595	0.000026	0.028930	Joe Bid
17	uspolitics	754	0.055075	0.006800	0.085807	Joe Bid
18	AmericanPolitics	95	0.036438	0.000000	0.047518	Clint
19	Liberal	390	0.154513	0.040444	0.069826	Clint
20	PresidentialRaceMemes	236	0.070411	0.000484	0.090562	Clint
21	Presidentialpoll	408	0.015777	0.007765	0.055501	Clint
22	obama	85	0.009838	0.003669	0.063968	Clint

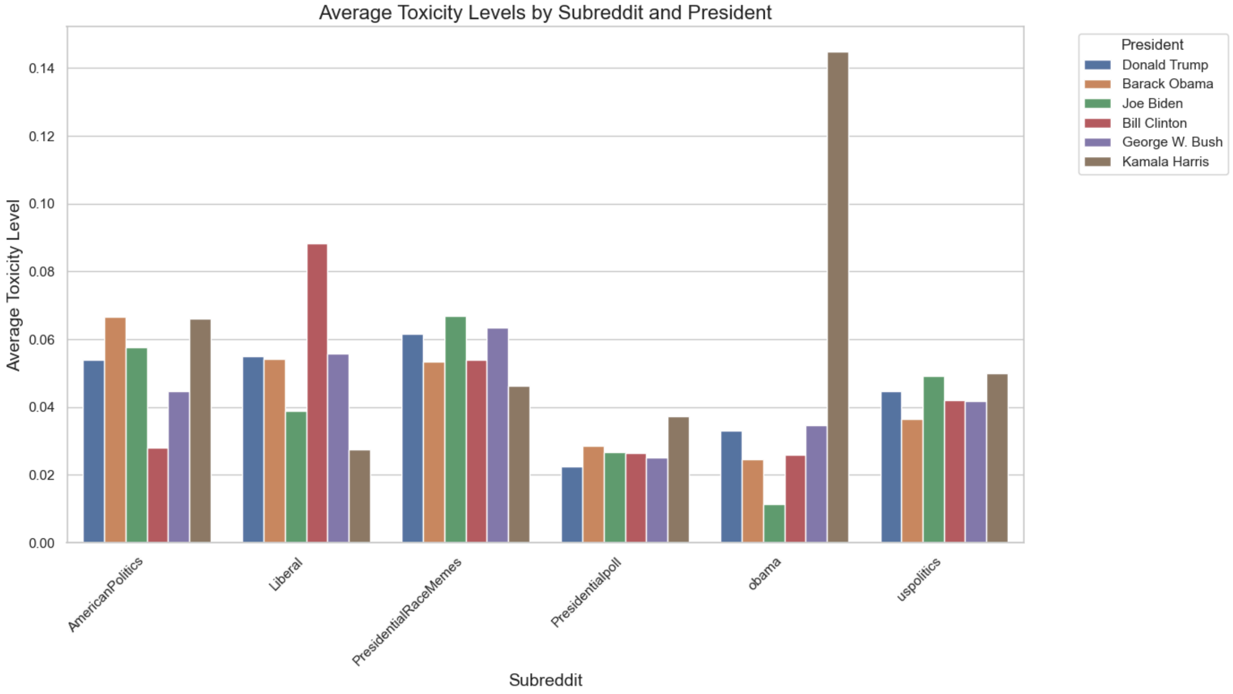
	subreddit	mentions	avg_toxicity_title	avg_toxicity_selftext	avg_toxicity_body	preside
23	uspolitics	225	0.041159	0.007781	0.077315	Clint
24	AmericanPolitics	52	0.085455	0.000013	0.048338	Geor W. Bu
25	Liberal	126	0.064295	0.003834	0.099221	Geor W. Bu
26	PresidentialRaceMemes	80	0.105543	0.000143	0.084751	Geor W. Bu
27	Presidentialpoll	363	0.013476	0.008572	0.053223	Geor W. Bu
28	obama	123	0.032773	0.003366	0.067635	Geor W. Bu
29	uspolitics	102	0.047255	0.011424	0.066331	Geor W. Bu
30	AmericanPolitics	14	0.080407	0.000000	0.117946	Kam Har
31	Liberal	85	0.008162	0.029153	0.044934	Kam Har
32	PresidentialRaceMemes	81	0.015716	0.000622	0.122366	Kam Har
33	Presidentialpoll	43	0.001292	0.007387	0.102910	Kam Har
34	obama	2	0.012199	0.000000	0.422453	Kam Har
35	uspolitics	37	0.016691	0.008951	0.124550	Kam

In [38]:

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

# Assuming you have the results_df DataFrame ready from your previous code

# Set the style of the visualization
sns.set(style="whitegrid")

# Create a bar plot for average toxicity by subreddit and president
plt.figure(figsize=(14, 8))
sns.barplot(data=results_df.melt(id_vars=['subreddit', 'president'],
                                   value_vars=['avg_toxicity_title', 'avg_toxicity_sel
             x='subreddit', y='value', hue='president', ci=None)

# Add titles and labels
plt.title('Average Toxicity Levels by Subreddit and President', fontsize=16)
plt.ylabel('Average Toxicity Level', fontsize=14)
plt.xlabel('Subreddit', fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.legend(title='President', bbox_to_anchor=(1.05, 1), loc='upper left')
```

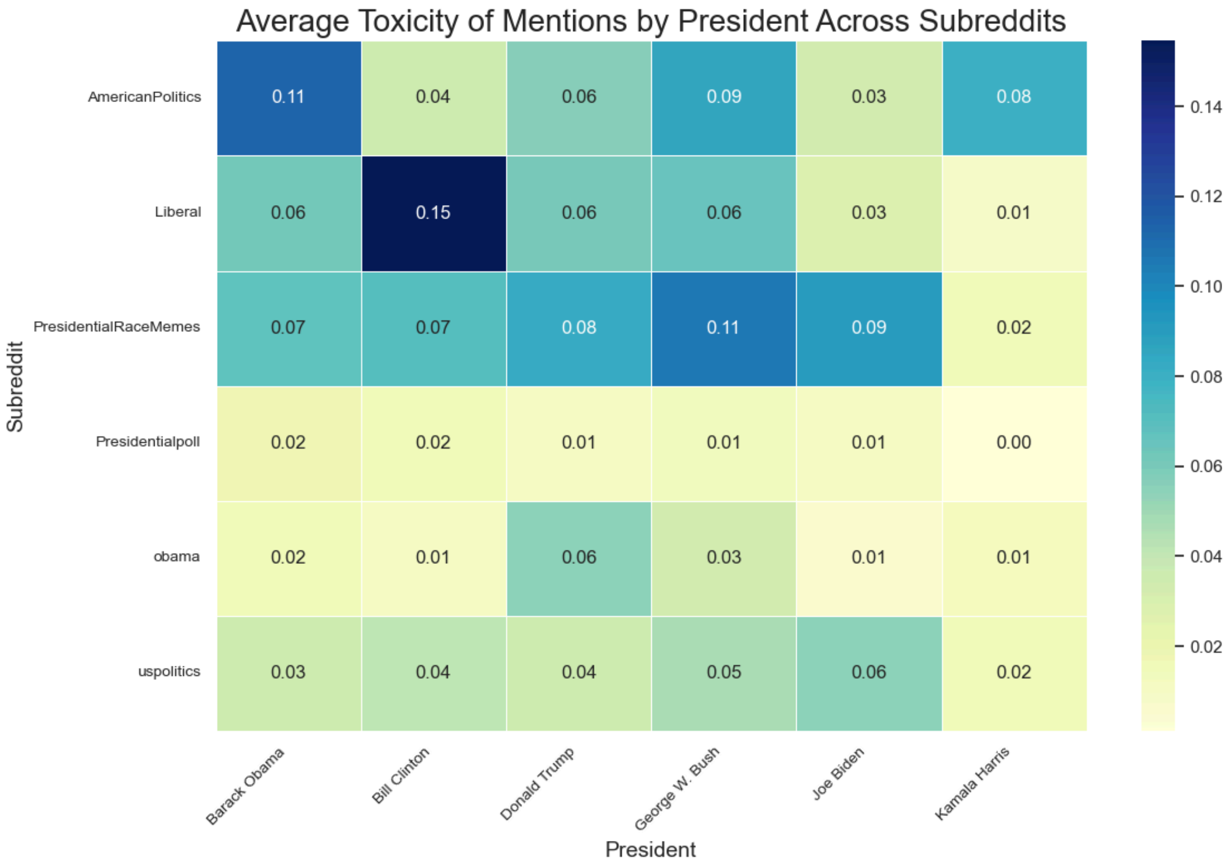
```
# Show the plot
plt.tight_layout()
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_41148\2175789254.py:12: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(data=results_df.melt(id_vars=['subreddit', 'president'],
```



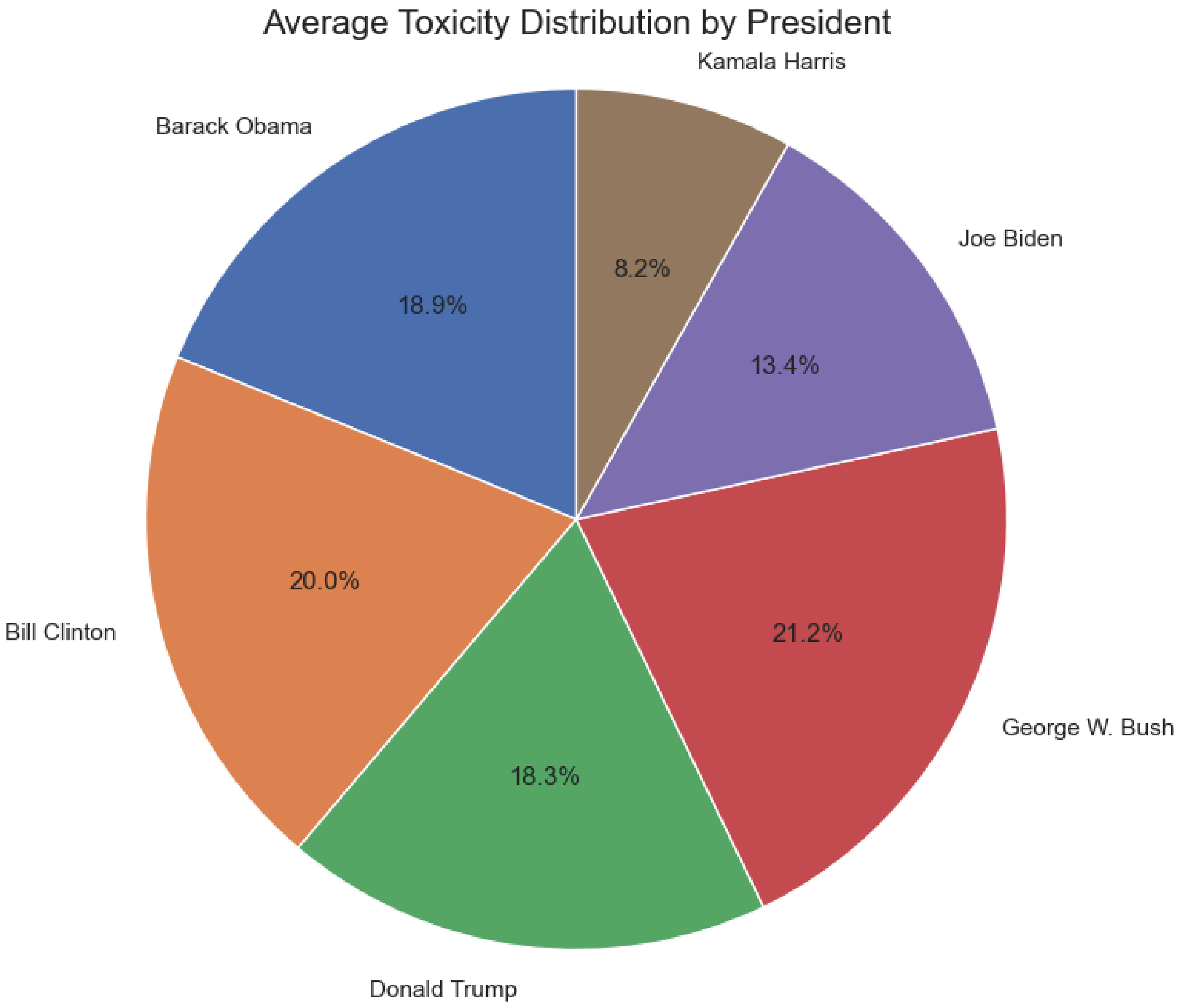
In [46]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming results_df is created from the previous code snippet

# Pivot the DataFrame for heatmap
heatmap_data = results_df.pivot_table(
    index='subreddit',
    columns='president',
    values='avg_toxicity_title',  # You can change this to other toxicity measures if
    fill_value=0  # Fill NaN with 0
)

# Create the heatmap
plt.figure(figsize=(12, 8))  # Adjust the figure size for better readability
sns.heatmap(heatmap_data, cmap='YlGnBu', annot=True, fmt=".2f", linewidths=.5)

# Rotate y-axis labels for better readability
plt.yticks(rotation=0, fontsize=10)
plt.xticks(rotation=45, ha='right', fontsize=10)

# Add titles and labels
plt.title('Average Toxicity of Mentions by President Across Subreddits', fontsize=20)
plt.xlabel('President', fontsize=14)
plt.ylabel('Subreddit', fontsize=14)
```

```
# Show the plot
plt.tight_layout()
plt.show()
```



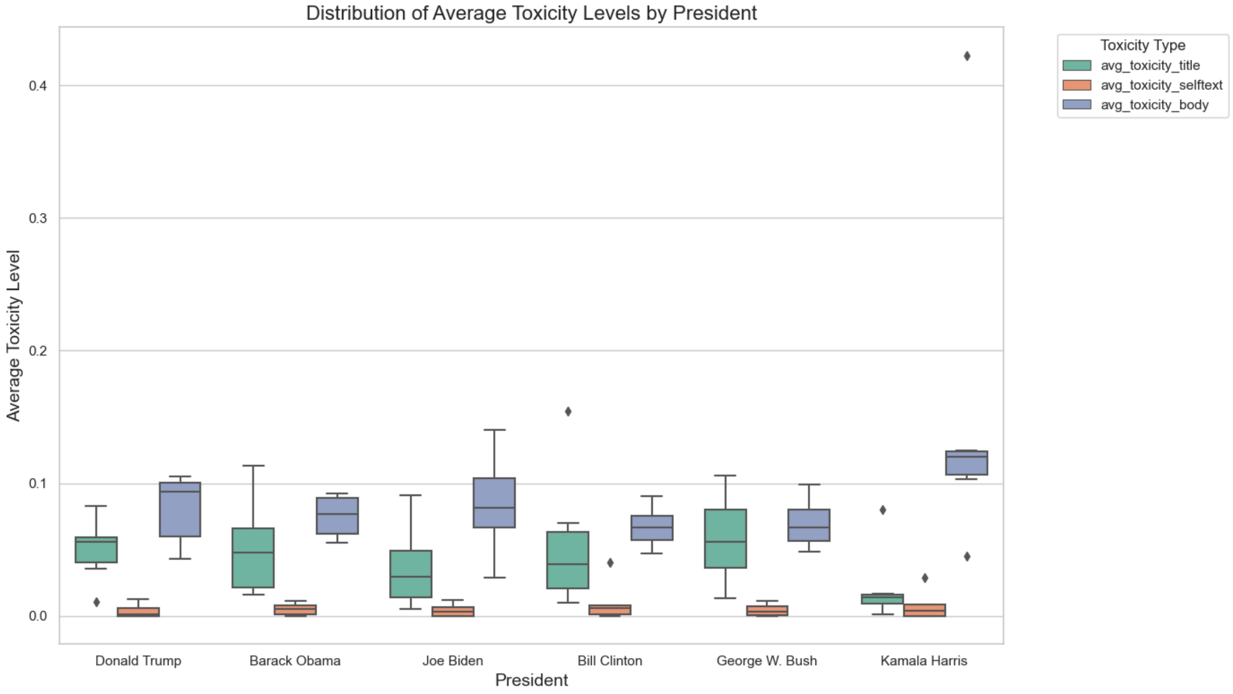
In [39]:

```
# Calculate overall average toxicity for each president
avg_toxicity_president = results_df.groupby('president').agg(
    avg_toxicity=('avg_toxicity_title', 'mean')
).reset_index()

# Create a pie chart for average toxicity by president
plt.figure(figsize=(10, 8))
plt.pie(avg_toxicity_president['avg_toxicity'], labels=avg_toxicity_president['preside
plt.title('Average Toxicity Distribution by President', fontsize=16)
plt.axis('equal')  # Equal aspect ratio ensures that pie chart is circular
plt.show()
```



In [40]:

```
# Create a box plot for average toxicity levels by president
plt.figure(figsize=(14, 8))
sns.boxplot(data=results_df.melt(id_vars=['president'],
                                   value_vars=['avg_toxicity_title', 'avg_toxicity_sel
             x='president', y='value', hue='variable', palette='Set2')

# Add titles and labels
plt.title('Distribution of Average Toxicity Levels by President', fontsize=16)
plt.ylabel('Average Toxicity Level', fontsize=14)
plt.xlabel('President', fontsize=14)
plt.legend(title='Toxicity Type', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show the plot
plt.tight_layout()
plt.show()
```



In [41]:

```
# Calculate the overall average toxicity for sorting
results_df['avg_toxicity'] = results_df[['avg_toxicity_title', 'avg_toxicity_selftext'

# Sort by the overall average toxicity in descending order
sorted_results_df = results_df.sort_values(by='avg_toxicity', ascending=False)

# Display the sorted results
sorted_results_df
```

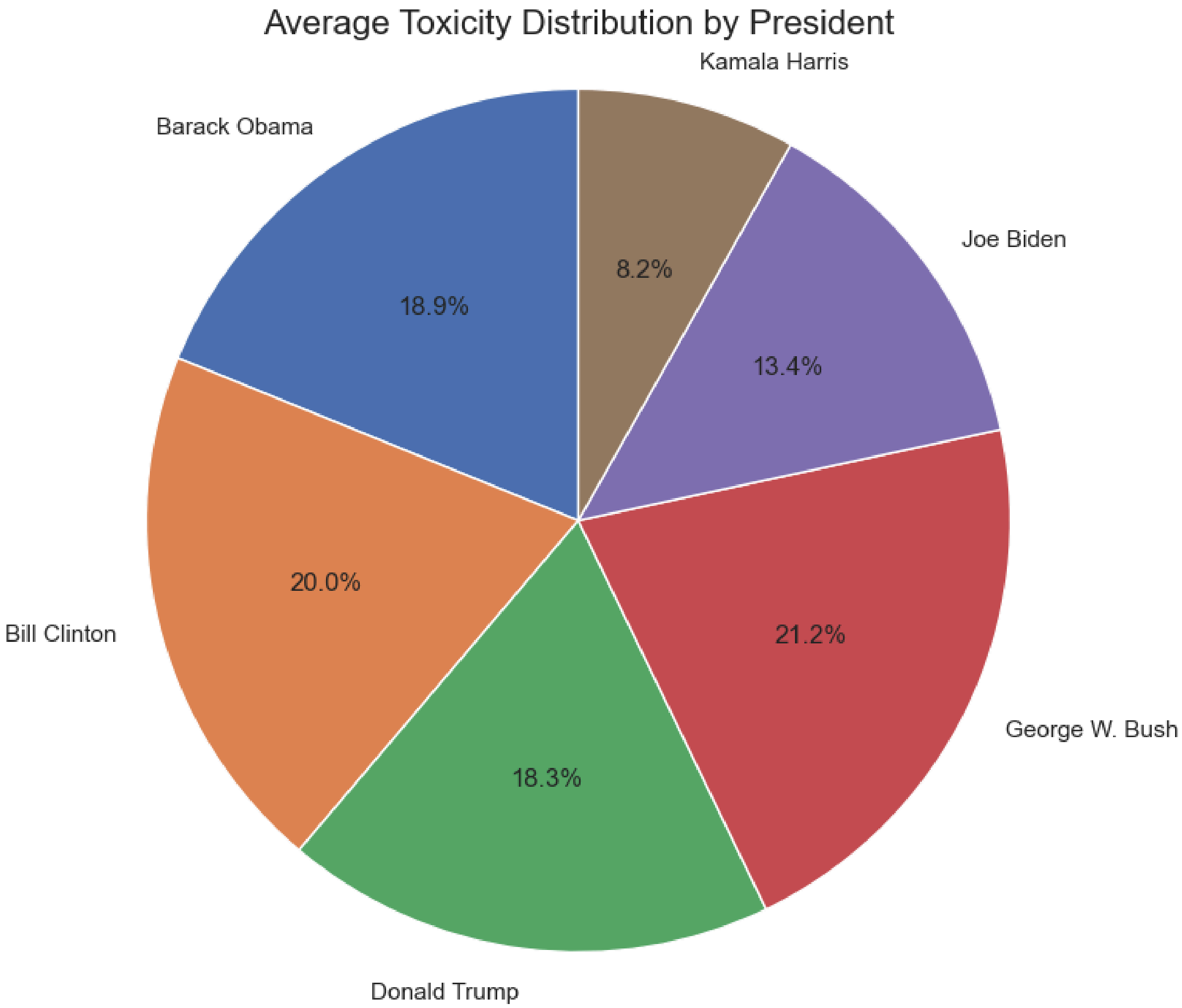
Out[41]:

	subreddit	mentions	avg_toxicity_title	avg_toxicity_selftext	avg_toxicity_body	preside
34	obama	2	0.012199	0.000000	0.422453	Kam Ha
19	Liberal	390	0.154513	0.040444	0.069826	Clint
14	PresidentialRaceMemes	2432	0.090722	0.000357	0.109750	Joe Bid
6	AmericanPolitics	76	0.113220	0.000000	0.086676	Bara Oba
30	AmericanPolitics	14	0.080407	0.000000	0.117946	Kam Ha
26	PresidentialRaceMemes	80	0.105543	0.000143	0.084751	Geor W. Bu
2	PresidentialRaceMemes	875	0.082616	0.000441	0.101608	Don Tru
12	AmericanPolitics	317	0.032044	0.000015	0.140476	Joe Bid
25	Liberal	126	0.064295	0.003834	0.099221	Geor W. Bu
1	Liberal	2134	0.060496	0.012741	0.091588	Don Tru
7	Liberal	253	0.061366	0.011242	0.090059	Bara Oba
20	PresidentialRaceMemes	236	0.070411	0.000484	0.090562	Clint
0	AmericanPolitics	614	0.056452	0.000041	0.104964	Don Tru
8	PresidentialRaceMemes	212	0.067503	0.000514	0.092119	Bara Oba
35	uspolitics	37	0.016691	0.008951	0.124550	Kam Ha
17	uspolitics	754	0.055075	0.006800	0.085807	Joe Bid
32	PresidentialRaceMemes	81	0.015716	0.000622	0.122366	Kam Ha
5	uspolitics	2322	0.035533	0.002654	0.095953	Don Tru
24	AmericanPolitics	52	0.085455	0.000013	0.048338	Geor W. Bu
23	uspolitics	225	0.041159	0.007781	0.077315	Clint
29	uspolitics	102	0.047255	0.011424	0.066331	Geor W. Bu
13	Liberal	753	0.027271	0.012330	0.077253	Joe Bid
33	Presidentialpoll	43	0.001292	0.007387	0.102910	Kam Ha

	subreddit	mentions	avg_toxicity_title	avg_toxicity_selftext	avg_toxicity_body	preside
11	uspolitics	143	0.034819	0.007491	0.066935	Bara Obai
28	obama	123	0.032773	0.003366	0.067635	Geor W. Bu
4	obama	83	0.055568	0.000037	0.043310	Dona Trur
9	Presidentialpoll	223	0.017083	0.008239	0.060269	Bara Obai
18	AmericanPolitics	95	0.036438	0.000000	0.047518	Clint
31	Liberal	85	0.008162	0.029153	0.044934	Kam Hai
15	Presidentialpoll	432	0.010167	0.006394	0.063641	Joe Bid
21	Presidentialpoll	408	0.015777	0.007765	0.055501	Clint
22	obama	85	0.009838	0.003669	0.063968	Clint
27	Presidentialpoll	363	0.013476	0.008572	0.053223	Geor W. Bu
10	obama	1016	0.015971	0.002691	0.055527	Bara Obai
3	Presidentialpoll	460	0.010859	0.007183	0.049262	Dona Trur
16	obama	53	0.005595	0.000026	0.028930	Joe Bid

In [53]:

```python
# Calculate overall average toxicity for each president
avg_toxicity_president = results_df.groupby('president').agg(
    avg_toxicity=('avg_toxicity_title', 'mean')
).reset_index()

# Create a pie chart for average toxicity by president
plt.figure(figsize=(10, 8))
plt.pie(avg_toxicity_president['avg_toxicity'], labels=avg_toxicity_president['preside
plt.title('Average Toxicity Distribution by President', fontsize=16)
plt.axis('equal')  # Equal aspect ratio ensures that pie chart is circular
plt.show()
```

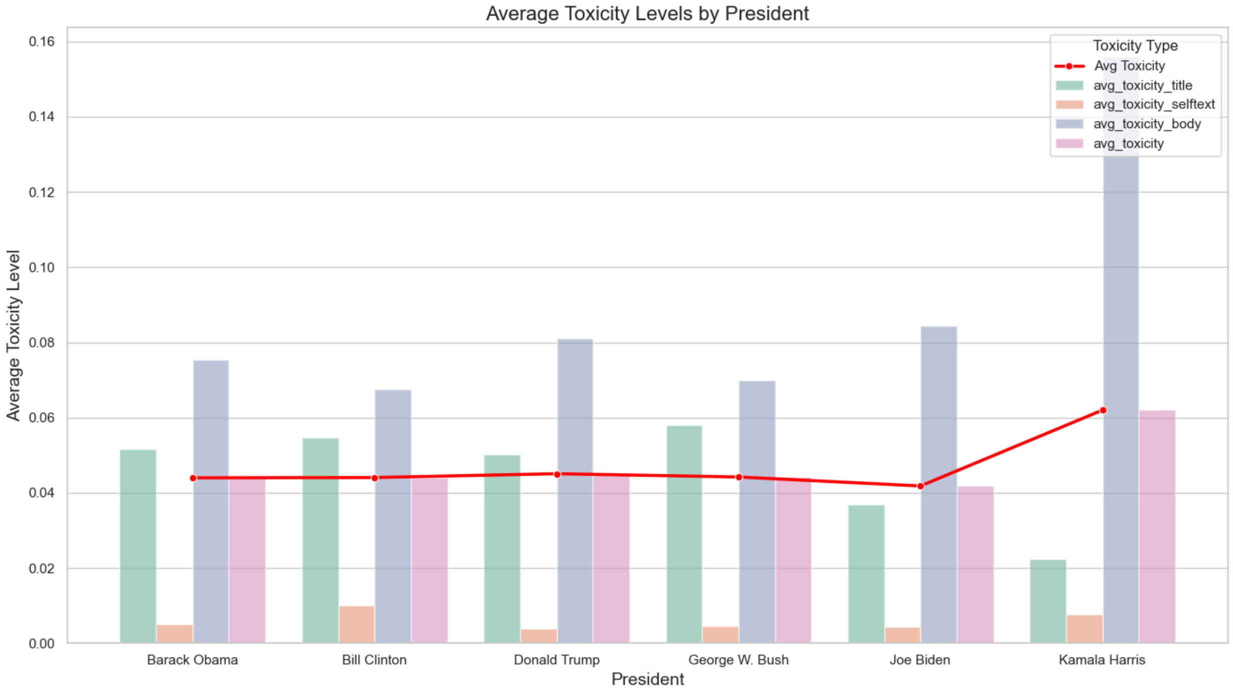


In [54]:

```
# Group by president and calculate mean toxicity
agg_toxicity = results_df.groupby('president').agg({
    'avg_toxicity_title': 'mean',
    'avg_toxicity_selftext': 'mean',
    'avg_toxicity_body': 'mean',
    'avg_toxicity': 'mean'
}).reset_index()

# Melt the DataFrame to long format for seaborn
agg_toxicity_melted = agg_toxicity.melt(id_vars='president', var_name='toxicity_type',

# Set the aesthetics for the plot
plt.figure(figsize=(14, 8))

# Create bar plot for toxicity types
bar_plot = sns.barplot(data=agg_toxicity_melted, x='president', y='toxicity_value', hu

# Overlay line plot for average toxicity
line_plot = sns.lineplot(data=agg_toxicity, x='president', y='avg_toxicity', marker='c

# Add titles and labels
plt.title('Average Toxicity Levels by President', fontsize=16)
plt.ylabel('Average Toxicity Level', fontsize=14)
plt.xlabel('President', fontsize=14)
plt.legend(title='Toxicity Type', loc='upper right')

# Show the plot
```

```
plt.tight_layout()
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



SENTIMENT ANALYSIS

In [58]:

```
from textblob import TextBlob

# Function to calculate sentiment
def get_sentiment(text):
    return TextBlob(text).sentiment.polarity  # Returns a value between -1 (negative)

# Apply sentiment analysis to relevant columns
sampled_df['sentiment_title'] = sampled_df['title'].apply(get_sentiment)
sampled_df['sentiment_selftext'] = sampled_df['selftext'].apply(get_sentiment)
sampled_df['sentiment_body'] = sampled_df['body'].apply(get_sentiment)

# Optionally, calculate average sentiment across the columns
sampled_df['avg_sentiment'] = sampled_df[['sentiment_title', 'sentiment_selftext', 'se

# Display the updated DataFrame
sampled_df[['title', 'sentiment_title', 'selftext', 'sentiment_selftext', 'body', 'ser
```

Out[58]:

	title	sentiment_title	selftext	sentiment_selftext	body
0	Opinion | New round of text messages exposes F...	-0.031818		0.0	&gt;It’s a common point of curiosity among san...
1	To go up against Donald Trump, the DNC picks.....	0.000000		0.0	where's he now lmao, it's time to face the rea...
2	How Mitch McConnell killed the US Capitol atta...	-0.200000		0.0	I have no idea why the Republicans would not w...
3	January 6 committee says Donald Trump violated...	-0.200000		0.0	[deleted]
4	So I wrote back...	0.000000		0.0	\nHey Donald Trump, the science is on my side,...
...	...	...	...	...	...
11072	Joe Biden when he decided to remove student lo...	0.000000		0.0	2005\n\nhttps://www.theguardian.com/us-news/20...
11073	The Joe Biden admin’s press freedom record sho...	0.000000		0.0	I see the useful idiots are at it again. Barac...
11074	In his own words - Stippling Barack Obama. A p...	0.275000		0.0	\nYou have posted a link to a video website. F...
11075	Joe Biden concluding his response.	0.000000		0.0	Welcome to /r/PresidentialRaceMemes! Make sure...

	title	sentiment_title	selftext	sentiment_selftext	body
11076	Barack Obama says health care law has led to 5...	0.350000		0.0	Uh, thanks Barack Obama?
11077	7...				

In [63]:

```python
import re
import pandas as pd
from textblob import TextBlob  # Ensure this is imported for sentiment analysis

# Define the dictionary mapping variations to the standard form
name_variations = {
    r'\b(?:george\s+w\.\s+bush|bush|george(?:\s+w\.\s+bush)?)\b': 'George W. Bush',
    r'\b(?:barack\s+obama|obama|barack(?:\s+obama)?)\b': 'Barack Obama',
    r'\b(?:bill\s+clinton|clinton|bill(?:\s+clinton)?)\b': 'Bill Clinton',
    r'\b(?:donald\s+trump|trump|donald(?:\s+trump)?)\b': 'Donald Trump',
    r'\b(?:joe\s+biden|biden|joe(?:\s+biden)?)\b': 'Joe Biden',
    r'\b(?:kamala\s+harris|harris|kamala(?:\s+harris)?)\b': 'Kamala Harris'
}

# Function to standardize names in a text
def standardize_names(text):
    for pattern, standard_name in name_variations.items():
        text = re.sub(pattern, standard_name, text, flags=re.IGNORECASE)
    return text

# Function to extract persons from a given text
def extract_persons(text):
    persons_found = []
    for pattern, standard_name in name_variations.items():
        if re.search(pattern, text, flags=re.IGNORECASE):
            persons_found.append(standard_name)
    return persons_found

# Function to calculate sentiment using TextBlob
def get_sentiment(text):
    return TextBlob(text).sentiment.polarity  # Returns a sentiment score between -1 a

# Assuming sampled_df is your DataFrame
# Convert all relevant columns to strings and fill NaNs with empty strings
sampled_df['title'] = sampled_df['title'].fillna('').astype(str)
sampled_df['selftext'] = sampled_df['selftext'].fillna('').astype(str)
sampled_df['body'] = sampled_df['body'].fillna('').astype(str)

# Standardize names in relevant columns
sampled_df['title'] = sampled_df['title'].apply(standardize_names)
sampled_df['selftext'] = sampled_df['selftext'].apply(standardize_names)
sampled_df['body'] = sampled_df['body'].apply(standardize_names)

# Extract persons from relevant columns
sampled_df['persons_title'] = sampled_df['title'].apply(extract_persons)
sampled_df['persons_selftext'] = sampled_df['selftext'].apply(extract_persons)
sampled_df['persons_body'] = sampled_df['body'].apply(extract_persons)
```

```
# Initialize an empty list to store results
results = []

# Initialize an empty list to store results
results = []

# Iterate over each president
for president in set(name_variations.values()):
    # Filter rows where the president is mentioned in any of the columns
    mentions_df = sampled_df[
        sampled_df['persons_title'].apply(lambda x: president in x) |
        sampled_df['persons_selftext'].apply(lambda x: president in x) |
        sampled_df['persons_body'].apply(lambda x: president in x)
    ]

    # Group by subreddit and calculate the number of mentions, average toxicity, and a
    grouped = mentions_df.groupby('subreddit').agg(
        mentions=('title', 'size'),
        avg_sentiment_title=('sentiment_title', 'mean'),         # Calculate average s
        avg_sentiment_selftext=('sentiment_selftext', 'mean'),   # Calculate average s
        avg_sentiment_body=('sentiment_body', 'mean')            # Calculate average s
    ).reset_index()

    # Calculate overall average sentiment across all sentiment columns
    grouped['avg_sentiment'] = grouped[['avg_sentiment_title', 'avg_sentiment_selftext

    # Add a column for the president's name
    grouped['president'] = president

    # Append the result to the list
    results.append(grouped)

# Concatenate all results into a single DataFrame
results_df = pd.concat(results, ignore_index=True)

# Display the results
results_df
```

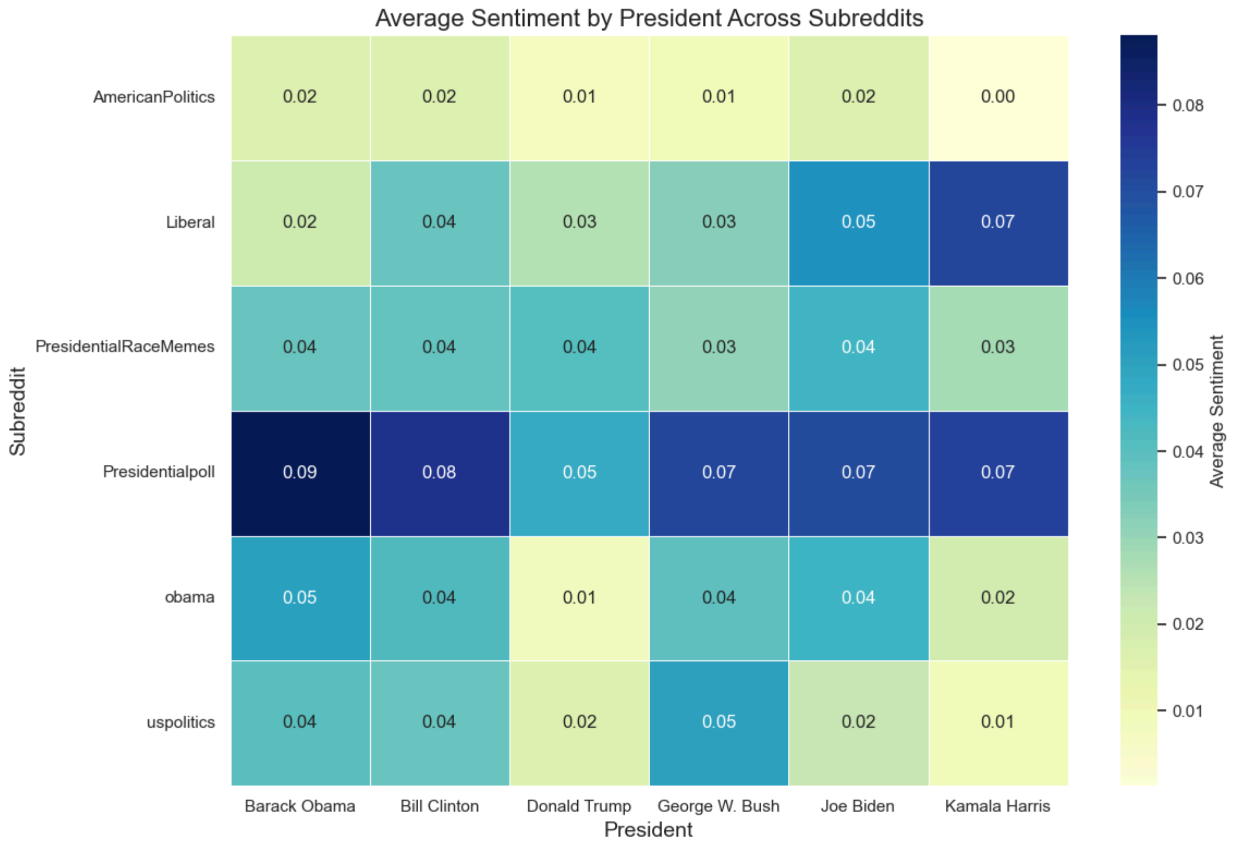
Out[63]:

	subreddit	mentions	avg_sentiment_title	avg_sentiment_selftext	avg_sentiment_body
0	AmericanPolitics	614	0.001259	0.000203	0.021676
1	Liberal	2134	0.017891	0.013190	0.046023
2	PresidentialRaceMemes	875	0.028056	0.014209	0.079695
3	Presidentialpoll	460	0.000013	0.068514	0.073407
4	obama	83	-0.024290	0.003313	0.044980
5	uspolitics	2322	0.011332	0.003526	0.035551
6	AmericanPolitics	76	-0.002055	0.000000	0.051514
7	Liberal	253	-0.001128	0.010852	0.051663
8	PresidentialRaceMemes	212	0.005195	0.048294	0.057677
9	Presidentialpoll	223	0.069318	0.066486	0.128383
10	obama	1016	0.057489	0.008205	0.086883
11	uspolitics	143	0.061273	0.006361	0.051674
12	AmericanPolitics	317	0.032983	0.000000	0.018085
13	Liberal	753	0.059360	0.040547	0.063693
14	PresidentialRaceMemes	2432	0.053105	0.007606	0.072719
15	Presidentialpoll	432	0.073852	0.078999	0.061299
16	obama	53	0.043438	0.000000	0.090631
17	uspolitics	754	0.013717	0.004219	0.050403
18	AmericanPolitics	95	0.005426	0.000000	0.045400
19	Liberal	390	0.026849	0.008031	0.078312
20	PresidentialRaceMemes	236	0.032647	0.043586	0.039486
21	Presidentialpoll	408	0.070385	0.072649	0.091650
22	obama	85	0.004076	0.022325	0.098119

	subreddit	mentions	avg_sentiment_title	avg_sentiment_selftext	avg_sentiment_body
23	uspolitics	225	0.054350	0.002467	0.056538
24	AmericanPolitics	52	-0.014128	0.000000	0.041077
25	Liberal	126	0.033673	0.008886	0.055256
26	PresidentialRaceMemes	80	0.007845	0.023817	0.060577
27	Presidentialpoll	363	0.068408	0.056822	0.089266
28	obama	123	-0.009104	0.021798	0.105254
29	uspolitics	102	0.070975	0.041334	0.038074
30	AmericanPolitics	14	0.014286	0.000000	-0.010359
31	Liberal	85	0.039531	0.100116	0.075362
32	PresidentialRaceMemes	81	0.030960	0.033335	0.020933
33	Presidentialpoll	43	0.104651	0.084857	0.027972
34	obama	2	0.000000	0.000000	0.056061
35	uspolitics	37	-0.008258	0.029160	0.006522

In [64]:

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming results_df is already defined and contains avg_sentiment and subreddit colu

# Pivot the DataFrame for heatmap
heatmap_data = results_df.pivot_table(
    index='subreddit',
    columns='president',
    values='avg_sentiment',
    fill_value=0  # Fill NaN values with 0
)

# Set the size of the heatmap
plt.figure(figsize=(12, 8))

# Create the heatmap
sns.heatmap(
    heatmap_data,
    cmap='YlGnBu',
    annot=True,  # Annotate cells with their values
    fmt=".2f",   # Format to 2 decimal places
    linewidths=.5,  # Lines between cells
    cbar_kws={'label': 'Average Sentiment'}  # Label for color bar
)

# Add titles and labels
plt.title('Average Sentiment by President Across Subreddits', fontsize=16)
plt.xlabel('President', fontsize=14)
plt.ylabel('Subreddit', fontsize=14)

# Show the plot
plt.tight_layout()
plt.show()
```

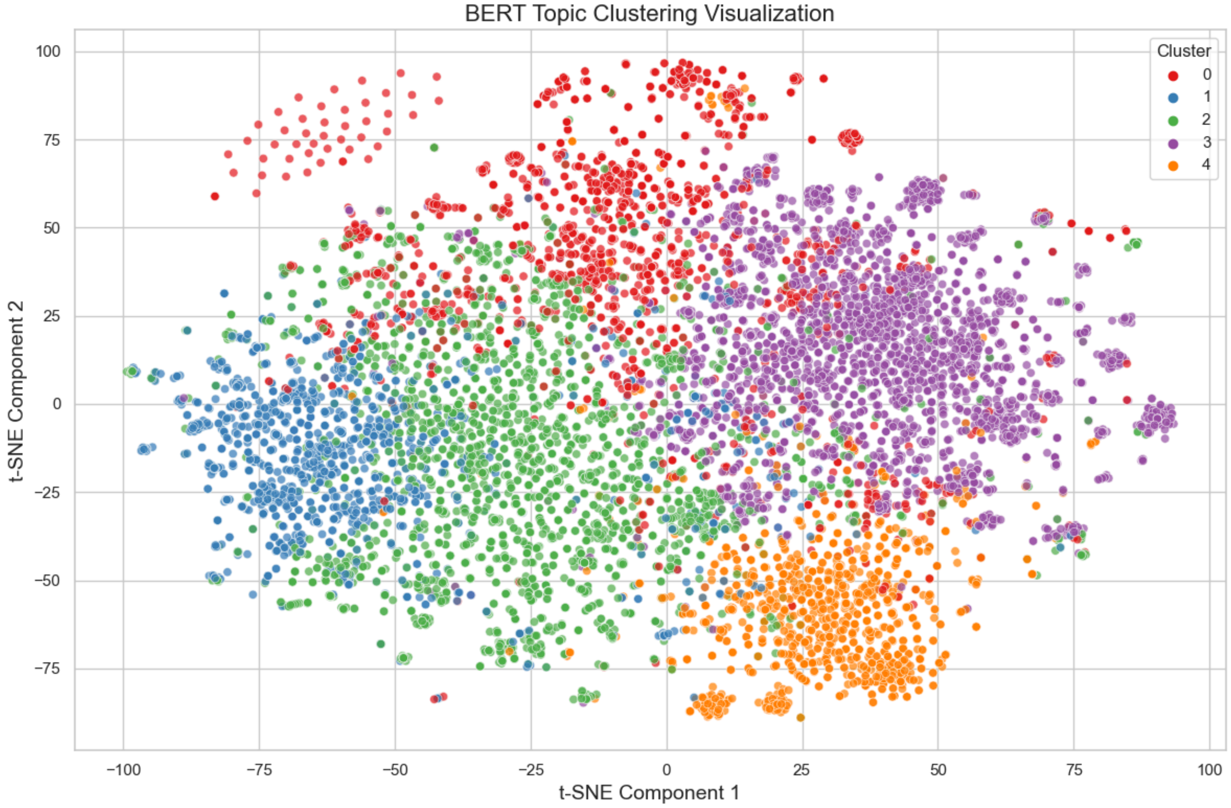


BERT TOPIC ANALYSER

In [65]:
```
import pandas as pd
from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns

# Load the BERT model
model = SentenceTransformer('all-MiniLM-L6-v2')  # You can choose a different model if

# Combine relevant text columns for analysis
sampled_df['combined_text'] = sampled_df['title'] + " " + sampled_df['selftext'] + " '
```

```python
# Generate BERT embeddings
embeddings = model.encode(sampled_df['combined_text'].tolist(), show_progress_bar=True

# Step 3: Cluster the embeddings
num_clusters = 5  # You can change this value based on your needs
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
sampled_df['cluster'] = kmeans.fit_predict(embeddings)

# Step 4: Visualize the clusters
# Reduce dimensions with t-SNE
tsne = TSNE(n_components=2, random_state=42)
tsne_results = tsne.fit_transform(embeddings)

# Add t-SNE results to DataFrame
sampled_df['x_tsne'] = tsne_results[:, 0]
sampled_df['y_tsne'] = tsne_results[:, 1]

# Step 5: Plot the clusters
plt.figure(figsize=(12, 8))
sns.scatterplot(data=sampled_df, x='x_tsne', y='y_tsne', hue='cluster', palette='Set1'

# Customize the plot
plt.title('BERT Topic Clustering Visualization', fontsize=16)
plt.xlabel('t-SNE Component 1', fontsize=14)
plt.ylabel('t-SNE Component 2', fontsize=14)
plt.legend(title='Cluster')
plt.tight_layout()

# Show the plot
plt.show()
```

```
C:\Users\HP\AppData\Roaming\Python\Python311\site-packages\transformers\tokenization_
utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set. It wil
l be set to `True` by default. This behavior will be depracted in transformers v4.45,
and will be then set to `False` by default. For more details check this issue: http
s://github.com/huggingface/transformers/issues/31884
  warnings.warn(
Batches:   0%|          | 0/347 [00:00<?, ?it/s]
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



In [67]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

# Assuming sampled_df contains the text data in a column called 'text'
documents = sampled_df['title'] + " " + sampled_df['selftext'] + " " + sampled_df['boc
```

In [68]:
```
# Create a CountVectorizer to create a document-term matrix
vectorizer = CountVectorizer(stop_words='english', max_features=1000)  # You can adjus
dtm = vectorizer.fit_transform(documents)
```

In [69]:
```
from sklearn.decomposition import LatentDirichletAllocation

# Set the number of topics
num_topics = 5  # Adjust as needed

# Apply LDA
lda = LatentDirichletAllocation(n_components=num_topics, random_state=42)
lda.fit(dtm)
```

Out[69]:
```
▾ LatentDirichletAllocation
LatentDirichletAllocation(n_components=5, random_state=42)
```

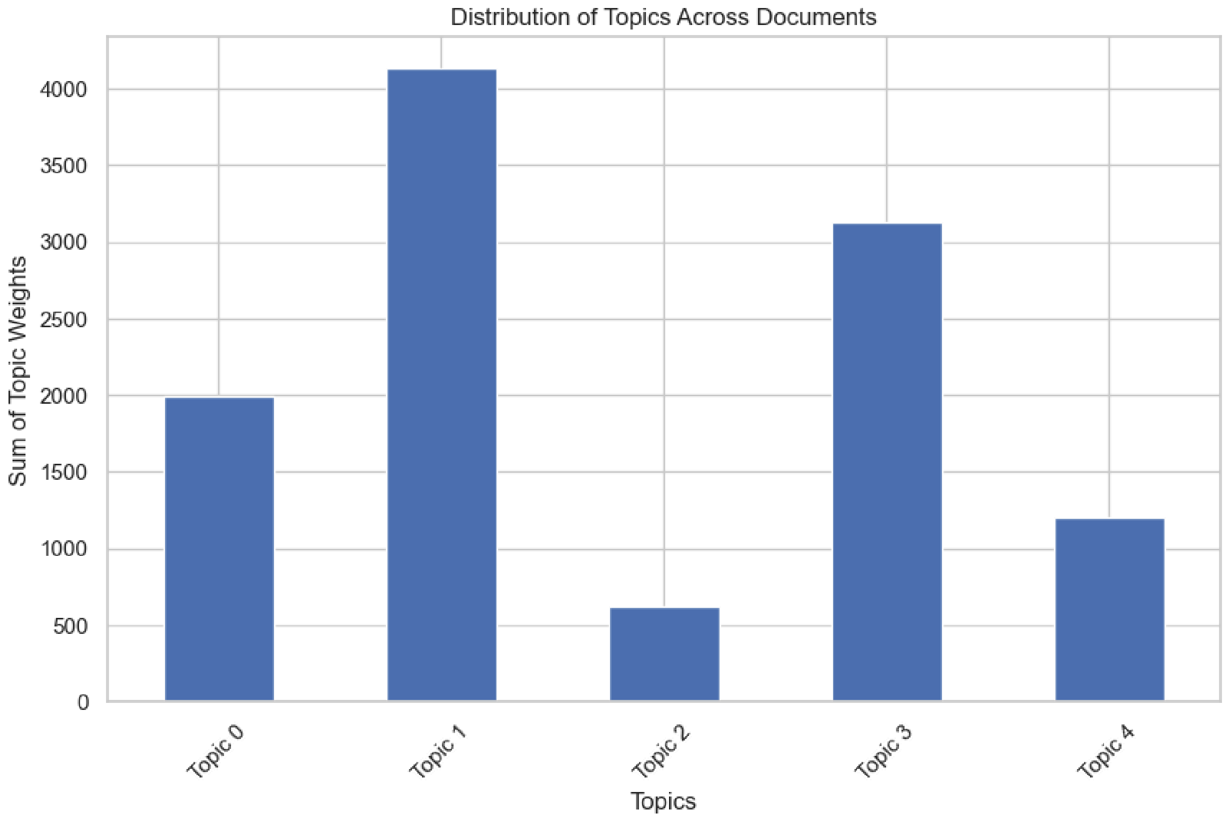
In [70]:
```
def get_topic_names(lda_model, vectorizer, n_words=5):
    topic_names = []
    for topic in lda_model.components_:
        words = vectorizer.get_feature_names_out()
        top_words_indices = topic.argsort()[-n_words:][::-1]
        topic_names.append([words[i] for i in top_words_indices])
    return topic_names

# Get topic names
```

```
topic_names = get_topic_names(lda, vectorizer)
print("Topics and their top words:")
for i, topic in enumerate(topic_names):
    print(f"Topic {i}: {', '.join(topic)}")
```

```
Topics and their top words:
Topic 0: barack, obama, joe, biden, com
Topic 1: joe, biden, people, just, like
Topic 2: election, vs, trump, donald, joe
Topic 3: trump, donald, https, com, www
Topic 4: amp, george, bush, president, https
```

In [71]:

```
import matplotlib.pyplot as plt

# Create a DataFrame to visualize the topic distribution
topic_distribution = lda.transform(dtm)
topic_distribution_df = pd.DataFrame(topic_distribution, columns=[f'Topic {i}' for i i

# Sum the topic distribution across all documents
topic_sum = topic_distribution_df.sum()

# Plot the distribution of topics
plt.figure(figsize=(10, 6))
topic_sum.plot(kind='bar')
plt.title('Distribution of Topics Across Documents')
plt.xlabel('Topics')
plt.ylabel('Sum of Topic Weights')
plt.xticks(rotation=45)
plt.show()
```



In [ ]:

```
import pandas as pd
from bertopic import BERTopic

# Assuming your DataFrame is called sampled_df and has a column 'created_utc' and 'sel
# Convert 'created_utc' to datetime if it isn't already
sampled_df['created_utc'] = pd.to_datetime(sampled_df['created_utc'])

# Extract the text for topic modeling
texts = sampled_df['selftext'].tolist()  # You can change this to any other column

# Initialize BERTopic
topic_model = BERTopic()

# Fit the model to your data
topics, _ = topic_model.fit_transform(texts)

# Add topics to your DataFrame
sampled_df['Topic'] = topics

# Group by date to count the number of topics over time
sampled_df['Date'] = sampled_df['created_utc'].dt.date
topic_counts = sampled_df.groupby(['Date', 'Topic']).size().reset_index(name='Counts')

# Visualize topics over time
topic_model.visualize_topics_over_time(topic_counts, top_n_topics=5)  # Adjust top_n_t
```

```
C:\Users\HP\AppData\Roaming\Python\Python311\site-packages\pandas\core\arrays\masked.
py:60: UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version
'1.3.5' currently installed).
  from pandas.core import (
```

In [5]:

```
import re
import pandas as pd

# Assuming combined_df is your DataFrame
# Combine relevant text fields into one column for topic modeling
combined_df['text'] = combined_df['title'].fillna('') + ' ' + combined_df['selftext'].

# Clean the text data
combined_df['text'] = combined_df['text'].apply(lambda x: re.sub(r"http\S+", "", x).lc
combined_df['text'] = combined_df['text'].apply(lambda x: " ".join(filter(lambda word:
combined_df['text'] = combined_df['text'].apply(lambda x: " ".join(re.sub("[^a-zA-Z]+'

# Remove rows with empty text
combined_df = combined_df[combined_df['text'] != ""]

# Get timestamps (if you have a datetime column)
timestamps = combined_df['created_utc_x'].to_list()  # Assuming this is the relevant t
texts = combined_df['text'].to_list()  # This will be used for topic modeling
```

In [6]:

```
sampled_combined_df = combined_df.sample(n=1000)  # Adjust n as needed
texts = sampled_combined_df['text'].to_list()
```

Word Cloud for Most Discussed On Words(Frequency)

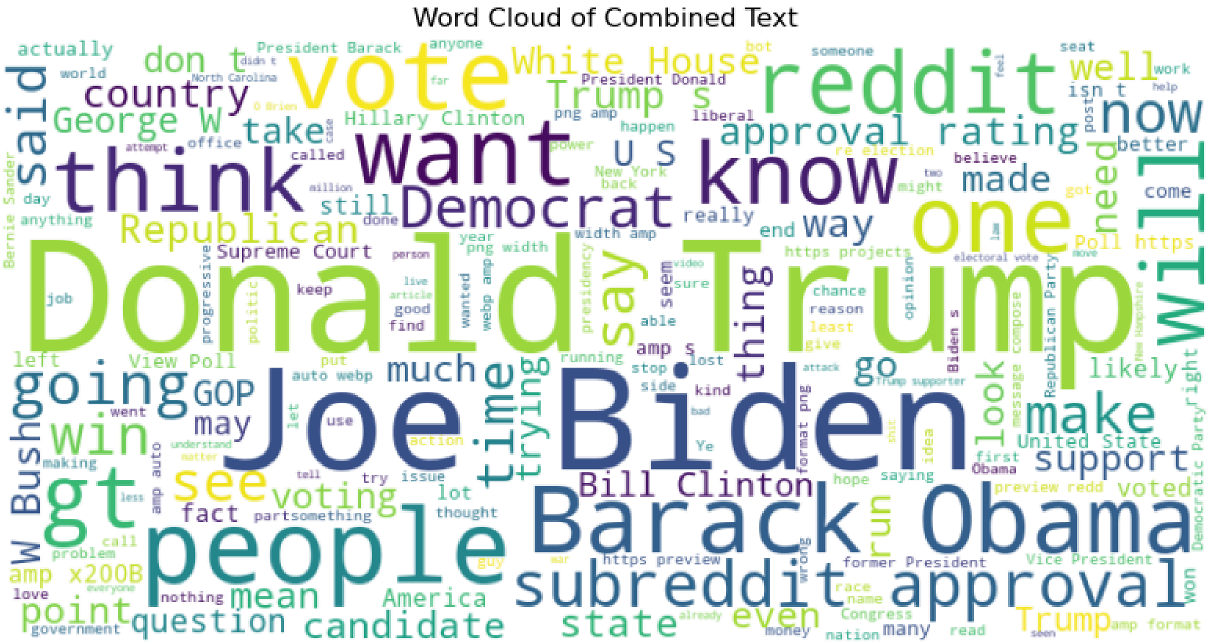
In [5]:

```
import pandas as pd

# Combine text from relevant columns into a single string
combined_df['combined_text'] = combined_df['title'] + ' ' + combined_df['selftext'].fi
```

```
# Convert to a single string
text_data = ' '.join(combined_df['combined_text'].tolist())
```

In [6]:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white', max_words=200).

# Plot the Word Cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Turn off axis numbers and ticks
plt.title('Word Cloud of Combined Text')
plt.show()
```



In [ ]:

In [ ]:

```
import pandas as pd
import re
from bertopic import BERTopic
from sklearn.feature_extraction.text import CountVectorizer

# Clean the text data
def clean_text(text):
    text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces with a single space
    text = re.sub(r'\W', ' ', text)   # Remove special characters
    return text.lower()

sampled_df['clean_body'] = sampled_df['body'].apply(lambda x: clean_text(x) if pd.notr

# Extract the text data
docs = sampled_df['clean_body'].tolist()
```

```
C:\Users\HP\AppData\Roaming\Python\Python311\site-packages\pandas\core\arrays\masked.
py:60: UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version
'1.3.5' currently installed).
  from pandas.core import (
```

In [1]:

```
protected_classes = [
    'race', 'color', 'national origin', 'sex', 'gender',
    'sexual orientation', 'age', 'disability', 'religion',
    'pregnancy', 'maternity'
]
```

In [ ]:

```
from bertopic import BERTopic
from sklearn.feature_extraction.text import CountVectorizer

# Define a custom CountVectorizer with the seeds
vectorizer_model = CountVectorizer(stop_words="english", vocabulary=protected_classes)

# Initialize BERTopic with the custom vectorizer
topic_model = BERTopic(vectorizer_model=vectorizer_model)

# Fit the model on the documents
topics, probabilities = topic_model.fit_transform(docs)
```

In [ ]: