Case Study Part 2: Insights

This notebook is running on a Google Colab L4 CPU environment. In this section, the metrics inferred from the data in part 1 of the case study are clustered, analyzed and visualized.

```
%%capture
#!pip install numpy --upgrade #(uncomment install lines if running first time in google colab)
#!pip install scipy --upgrade
#!pip install gensim #(refresh kernel after installing to load new library and recomment installs)
```

```
%%capture
import pandas as pd
import gensim #gensim is used for a latent text embedding model for clustering text by meaning
from gensim.models import LdaMulticore
from gensim.corpora import Dictionary
from gensim.models.coherencemodel import CoherenceModel
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
!pip install wordcloud
from wordcloud import WordCloud
```

```
from gensim.models.phrases import Phrases
from sklearn.feature_extraction.text import TfidfVectorizer
#from sklearn.cluster import KMeans
#from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score, silhouette_score
import numpy as np
```

```
%%capture
!pip install datasets
from datasets import load_dataset
from google.colab import userdata
```

```
data = load_dataset("skeskinen/TinyStories-GPT4")
```

```
data_df = data['train'].to_pandas()
```

```
word_occurrences = pd.read_csv('word_occurrences.csv')
```

```
feature_occurrences = pd.read_csv('feature_occurrences.csv')
```

Statistical Analysis and Visualization

In this section the statistical metrics calculated from the training data is analyzed and visualized for making insights with regards to the stated objectives. Furthermore, given the complexity in understanding the level of literacy from the data - as no 'literacy level' column is within the data - a language model is used to classify the responses.

Mental Health

As for mental-health, goals may be defined in objective terms by inferring the emotions, subjects, topics, genres, or any commonly used keywords related to mental-health within the data. For this purpose, we can use the list of words and narrative features provided within the prompts provided to the AI in the training data for each row in the dataset. By counting the occurrences of the 'words' and 'features' columns used within the prompt, we can make some insights of the mental-health objectives.

Objective: Highlight narrative features that relate to mental health.

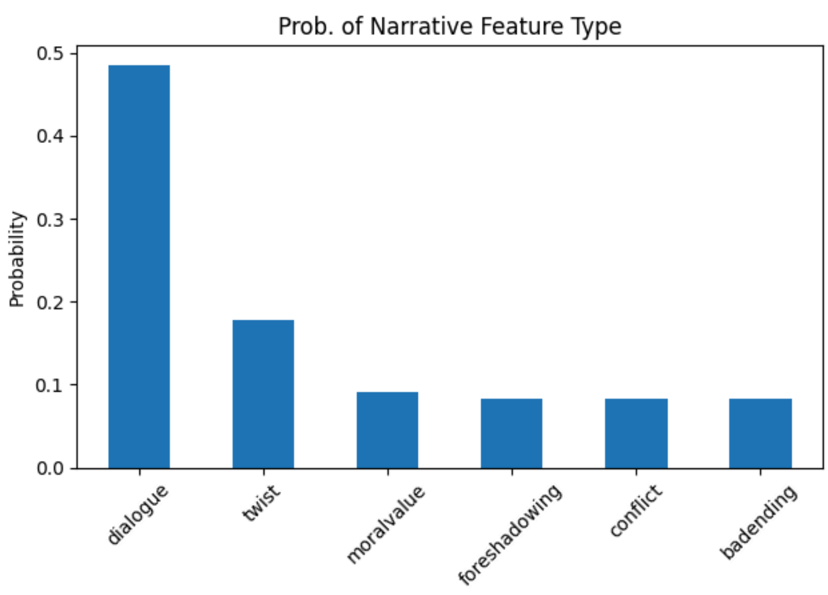
Narrative Features

Some of the 'feature narrative' words provided to the model within the prompt template's in the training dataset give insight into whether the story may contain any emotional subjects or topics related to mental health. A bar plot is shown that highlights the amount of each feature narrative.

```
# converting the counts of feature occurrences to probability values
feature_occurence_probabilities = feature_occurrences['feature_occurrences'] / feature_occurrences['feature_occurrences'].su
```

```
feature_occurence_probabilities.index = feature_occurrences['feature']
```

```
ax = feature_occurence_probabilities.sort_values(ascending=False).plot(
    kind='bar',
    title='Prob. of Narrative Feature Type'
)

# Correctly set labels using set methods
plt.ylabel('Probability')
plt.xlabel('Narrative Feature')
plt.xticks(rotation=45)

# Optional: Adjust layout to prevent label cutoff
plt.tight_layout()

plt.show()
```




```
feature_occurrences.describe()
```

	feature_occurrences
count	6.000000e+00
mean	5.059540e+05
std	4.859297e+05
min	2.503000e+05
25%	2.507192e+05
50%	2.624705e+05
75%	4.730752e+05

Mental Health Insight

We can see from the bar chart that the majority of narratives have dialogue (i.e speech between characters in a story). Similarly, 'twists' and 'moralvalue' are 2nd and 3rd respectively, highlighting how the majority of prompts seek to generate interesting stories where characters interact and exemplify moral values. On the other end of the spectrum, the story prompts in the training data generate stories with narratives like foreshadowing, conflict, and bad endings. As our target demographic is youth, it might be a good idea to do further investigation on story prompts that contain the 'conflict' and 'badending' narrative tags - these combined tags are contained in roughly **20%** of the prompts.

Statistical Insight

The description of the feature occurrences show that the average count of prompts with each narrative type is typically ~500,000 for each. However, the 50% quantile is ~260,000 showing that the majority of narrative types are roughly half as large as the mean. This implies that there are outlier narratives skewing the distribution up. The 'dialogue' and 'twist' narrative types are outliers, and are contained within ~50% and ~20% of the story prompts respectively, and combined contribute to ~70% of the story prompts. Similarly, the standard deviation value of ~480,000 shows how the variation in prompt counts is really high, further showing how the outliers ('dialogue', 'twist') are overshadowing the remaining narrative types ('moralvalue', 'foreshadowing', 'conflict', 'badending').

Note: Most of the prompts range in number of narrative tags, and may contain either none or all of the tags described above. As such, it is worth investigating how these tags correlate. For example, a combined 'badending' and 'conflict' narrative lacking a 'moralvalue' tag might lead to risky or insensitive storytelling.

Key Words

The key words column 'words' in the data are specific key vocabulary words provided to the AI in the story generation prompt. These words differ from the narrative feature words as the key words must be included in the generated story - whereas the narrative features are abstract storytelling features.

```
# converting the counts of feature occurrences to probability values
word_occurrence_probabilities = word_occurrences['word_occurrences'] / word_occurrences['word_occurrences'].sum()
```

```
word_occurrence_probabilities.index = word_occurrences['word']
```

```
word_occurrence_probabilities.describe()
```

	word_occurrences
count	1603.000000
mean	0.000624
std	0.000422
min	0.000294
25%	0.000311
50%	0.000319
75%	0.000854
max	0.002497

As the collection of vocabulary words within the dataset is large (~1604), the data must be clustered along the words' sentiment/meaning using a text embedding model. This way, we can reduce the dimensionality of the data while somewhat preserving the intended meaning of the collection. Given the individual probabilities for each word are very small (on the scale of 1/1000 likeliness), clustering to reduce the dimensionality will help understand the salient groups of words that are commonly paired within each vocab key collection.

```
# step 1: prepare the n-grams
data_df['words'].head() # every collection of words is already of length 3 (trigram format)
```

	words
0	[receive, opera, red]
1	[use, sheet, blue]
2	[relax, bus, uncomfortable]
3	[sail, cricket, wide]
4	[pray, pigeon, creative]

Since the dataset is large, clustering can not be executed in one execution - even with Google Colabs biggest ram environment. Therefore, the data will be clustered in iterations via bootstrapping. In this way, the data will be clustered in batches and then aggregated.

```
# note: vectorizer must be initialized outside the loop
# in order to maintain the training data across batches

# Step 2: LDA Clustering the batched TF-IDF matrix with
def bootstrap_clustering(data, num_clusters, num_iterations, batch_size):
    """
    Performs bootstrapped clustering on the data, incorporating TF-IDF creation within the loop.

    Args:
        data: The data to cluster (the 'words' column of your DataFrame).
        num_topics: The number of clusters in the LDA clustering.
        num_iterations: The number of iterations.
        batch_size: The batch size.

    Returns:
        A list of cluster assignments for each data point.
    """

    all_topic_assignments = []
    batches = []
    topic_assignment_batches = []
    # Initialize lists to store cluster statistical metrics
    coherence_scores = []
    perplexity_scores = []

    data_size = len(data)

    for _ in range(num_iterations):
        # Randomly sample a batch of data
        batch_indices = np.random.choice(data_size, size=batch_size, replace=True)
        batch_data = data.iloc[batch_indices]['words']  # Select the 'words' column
        batches.append(batch_data)

        # Prepare data for LDA
        dictionary = Dictionary([word for word in batch_data.values])
        corpus = [dictionary.doc2bow(text) for text in batch_data.values]

        # Perform LDA on the batch
        lda_model = LdaMulticore(corpus, num_topics=num_topics, id2word=dictionary, passes=15, workers=2)
        batch_topic_assignments = [max(lda_model[doc], key=lambda item: item[1])[0] for doc in corpus]

        topic_assignment_batches.append(batch_topic_assignments)
        # Append the cluster assignments to the list
        all_topic_assignments.extend(batch_topic_assignments)
        # Calculate and store metrics for this iteration
        coherence_model = CoherenceModel(model=lda_model, texts=batch_data.values, dictionary=dictionary, coherence='c_v')
        coherence_score = coherence_model.get_coherence()
        coherence_scores.append(coherence_score)

        perplexity = lda_model.log_perplexity(corpus)
        perplexity_scores.append(perplexity)

    return all_topic_assignments, batches, topic_assignment_batches, coherence_scores, perplexity_scores
```

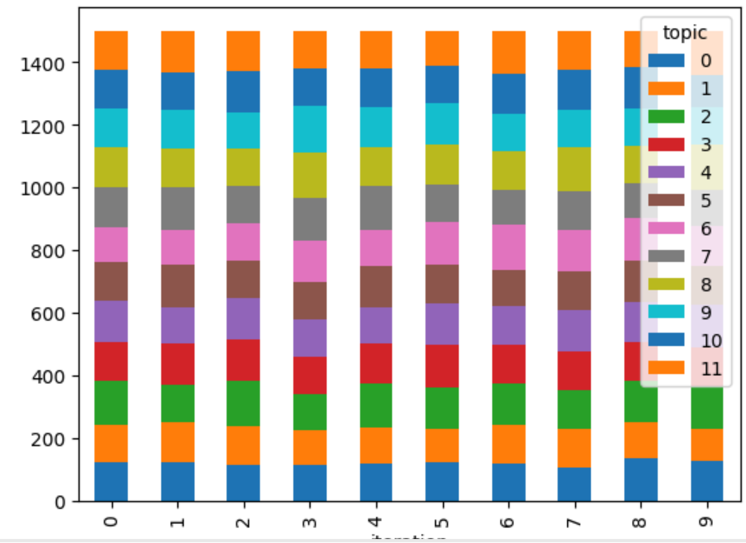
```
num_topics = 12 # 12 is chosen to be able to visualize wordcloud in grid format
num_iterations = 10  # Adjust as needed
batch_size = 1500  # This number is close to the maximum compute

# Perform bootstrapped clustering
topic_assignments, batches, topic_assignment_batches, coherence_scores, perplexity_scores = bootstrap_clustering(
    data_df, num_topics, num_iterations, batch_size
)

# Assign the cluster assignments to the original DataFrame
# word_trigram_df['cluster'] = cluster_assignments
```

```
cluster_assignments_df = pd.DataFrame(topic_assignments, columns=['topic'])
# adding in a cluster iteration index
cluster_assignments_df['iteration'] = cluster_assignments_df.index // batch_size
```

```
cluster_counts_by_iteration = cluster_assignments_df.groupby('iteration')['topic'].value_counts()
```

```
# let's visualize the distribution of cluster counts by iteration by showing a bar chart with a legend using the iteration
cluster_counts_by_iteration.unstack().plot(kind='bar', stacked=True, legend=True)
```

```
<Axes: xlabel='iteration'>
```



Since the topics discovered via LDA clustering are abstract, classification of these topics into a understandable theme may be done using a large language model with a prompt to classify into the existing objectives of 'mental health' or 'creativity'.

```
mental_health_topics = [
    "Dealing with Fear and Anxiety",
    "Coping with Sadness and Loss",
    "Managing Anger and Frustration",
    "Practicing Gratitude and Mindfulness",
    "Developing Healthy Relationships with Family",
    "Developing Healthy Relationships with Friends"
]

creativity_topics = [
    "Thinking Outside the Box",
    "Embracing Imagination and Curiosity",
    "Using Art and Expression",
    "Music",
    "Painting",
    "Sculpting"]
```

```
# let's create a function to call openAI's chatgpt-4o with a prompt to classify topics
import openai
import os

client = openai.OpenAI(api_key=userdata.get('openai'))
model = 'gpt-4o-mini'
def classify_topics(mental_health_topics, creativity_topics, topic_df_words_counts):
    """ this takes a preset list of mental health and creativity topics,
        and classifies the topic_ids by using the data in the topic_df_words
        note: there are 12 topic_ids, and 6 mental health topics, and 6 creativity topics
        thus only one topic_id can be classified into one of the 2 categories
        """
    prompt_template = (
        f"The following is a list of mental health topics: {mental_health_topics}",
        f"The following is a list of creative topics: {creativity_topics}",
        f"Use this list of words and their counts to classify the most likely topic: {topic_df_words_counts}",
        "Note: Each topic in either list can be assigned only once. Only respond with the topic chosen and nothing else. The
        "Topic: "
    )
    prompt = "\n".join(prompt_template)
    response = client.chat.completions.create(
        messages=[
            {"role": "system", "content": "You are comfortable classifying the language used in stories by simply inferring
            {"role": "user", "content": prompt}
```

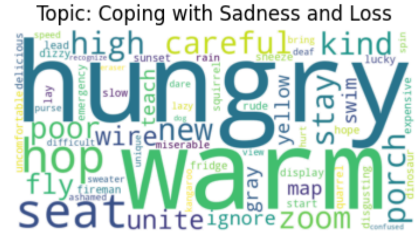
```
        ],
        model = model,
        temperature = 0.0
    )
    return response.choices[0].message.content

```

```
# Step 3: creating a word cloud for all topics within a single iteration
# add the cluster labels to the batch
sample_batch = batches[0]
batch_df = pd.DataFrame({'words': sample_batch, 'topic': topic_assignment_batches[0]})
# Determine the number of rows and columns for the grid
num_topics = len(batch_df['topic'].unique())
num_cols = 3  # Adjust as needed
num_rows = (num_topics + num_cols - 1) // num_cols

# Create the figure and axes for the grid
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))
i = 0
for topic_id in batch_df['topic'].unique():
    topic_df = batch_df[batch_df['topic'] == topic_id].copy()

    # Create a word cloud of the topics
    wordcloud = WordCloud(width=800, height=400, background_color='white', max_words=100)
    topic_df['words'] = topic_df['words'].apply(lambda x: ' '.join(x))
    # Concatenate all word strings in topic_df['words']
    wordcloud_text = ' '.join(topic_df['words'].values)
    # Filter counts of all words in the wordcloud_text and then select top 100 with count > 1
    wordcloud_text_counts = wordcloud.process_text(wordcloud_text)
    wordcloud_text_counts = {k: v for k, v in sorted(wordcloud_text_counts.items(), key=lambda item: item[1], reverse=True)}
    wordcloud_text_counts = dict(list(wordcloud_text_counts.items())[:100])
    wordcloud_text_counts = pd.Series(wordcloud_text_counts, index=wordcloud_text_counts.keys())
    # Filter to words with counts > 1
    wordcloud_text_counts = wordcloud_text_counts[wordcloud_text_counts > 1]
    wordcloud_text_filtered = ' '.join(wordcloud_text_counts.index)
    # Classifying the topic_df['words']
    topic = classify_topics(mental_health_topics, creativity_topics, wordcloud_text_counts.to_string())
    topic_df['topic'] = topic
    wordcloud.generate(wordcloud_text_filtered)
    # Plot the wordcloud on the corresponding subplot
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col]  # Get the current subplot
    ax.imshow(wordcloud, interpolation='bilinear')
    ax.axis('off')
    ax.set_title(f'Topic: {topic}')  # Add a title/label
    i += 1

```

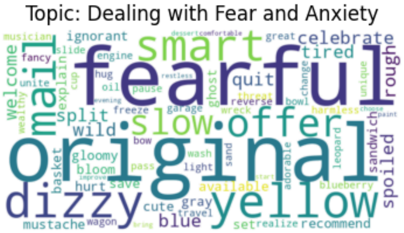
Topic: Creative

Topic: Creative

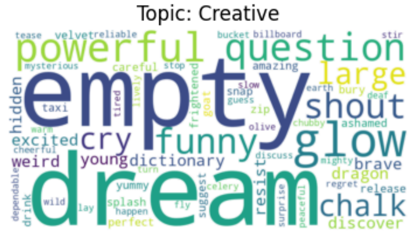
Topic: Coping with Sadness and Loss



Topic: Dealing with Fear and Anxiety

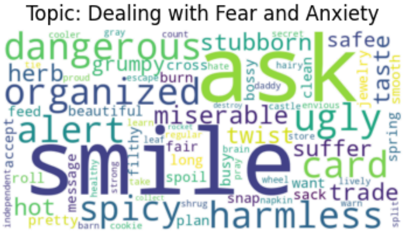


Topic: Creative



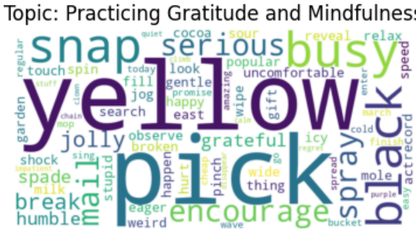
Topic: Creative

Topic: Dealing with Fear and Anxiety



Topic: Mental Health

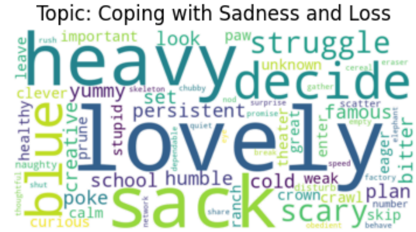
Topic: Practicing Gratitude and Mindfulness



Topic: Creative

Topic: Creative

Topic: Coping with Sadness and Loss

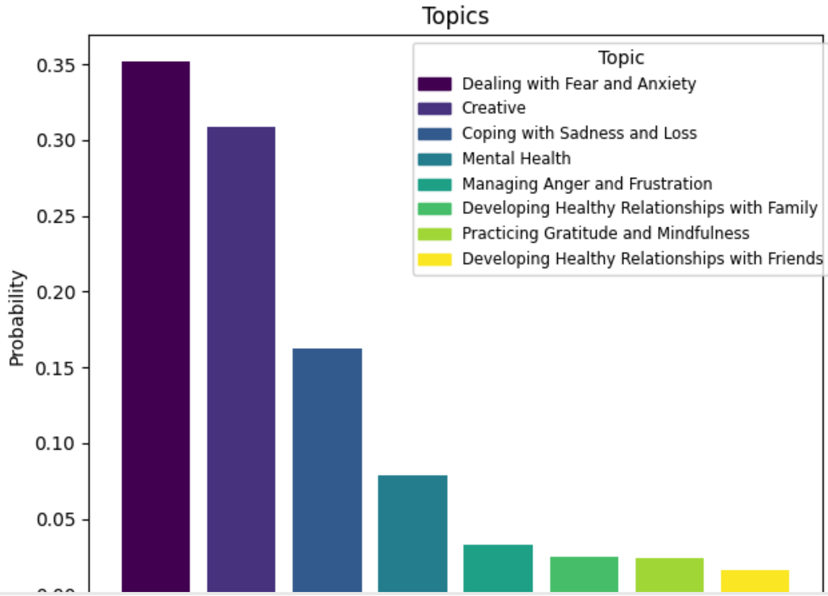


```
# Step 4: classifying all the topic_ids across all bootstrap cluster iterations
all_classified_topic_batches = []
for i in range(len(topic_assignment_batches)):
    batch_df = pd.DataFrame({'words': batches[i], 'topic': topic_assignment_batches[i]})
    for topic_id in batch_df['topic'].unique():
        topic_df = batch_df[batch_df['topic'] == topic_id].copy()
        topic_df['words'] = topic_df['words'].apply(lambda x: ' '.join(x))
        # Concatenate all word strings in topic_df['words']
        wordcloud_text = ' '.join(topic_df['words'].values)
        # Filter counts of all words in the wordcloud_text and then select top 100 with count > 1
        wordcloud_text_counts = wordcloud.process_text(wordcloud_text)
        wordcloud_text_counts = {k: v for k, v in sorted(wordcloud_text_counts.items(), key=lambda item: item[1], reverse=Tr
        wordcloud_text_counts = dict(list(wordcloud_text_counts.items())[:100])
        wordcloud_text_counts = pd.Series(wordcloud_text_counts, index=wordcloud_text_counts.keys())
        # Filter to words with counts > 1
        wordcloud_text_counts = wordcloud_text_counts[wordcloud_text_counts > 1]
        topic = classify_topics(mental_health_topics, creativity_topics, wordcloud_text_counts.to_string())
        topic_df['topic'] = topic
        all_classified_topic_batches.append(topic_df)
```

```
# concatenating all tables for summary
all_classified_topic_batches_df = pd.concat(all_classified_topic_batches)
```

```
def plot_topic_counts(df, title="Topics", xlabel="Topic", ylabel="Probability"):
    # Getting topic counts and labels
    topic_counts = df['topic'].value_counts(normalize=True)
    labels = topic_counts.index.tolist()
    counts = topic_counts.values.tolist()

    # Plotting the bars with distinct colors
    # Generating distinct colors using a colormap
    num_bars = len(labels)
    cmap = plt.get_cmap('viridis', num_bars)  # Choose a colormap

    bars = plt.bar(labels, counts, color=[cmap(i) for i in range(num_bars)])

    # Setting chart elements
    plt.title(title)
    # Hiding x-ticks and labels
    plt.xticks([])
    plt.ylabel(ylabel)
    #plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better visibility
    plt.tight_layout()  # Prevents labels from overlapping

    # Generating legend handles with x-axis labels as legend entries
    legend_handles = [mpatches.Patch(color=cmap(i), label=label) for i, label in enumerate(labels)]

    # Smaller legend with adjusted position
    plt.legend(handles=legend_handles, title=xlabel, loc='upper right', fontsize='small', bbox_to_anchor=(1.02, 1))

    plt.show()
```

```
plot_topic_counts(all_classified_topic_batches_df)
```



Mental Health Insight

The classification of the topic id's found via clustering of the key words using a Latent Dirichlet Allocation model for latent text embedding analysis is shown above. It is surprising that a majority of the classified topic labels fall under the 'mental health' category. However, despite the language model not classifying much of the sub-categories within the 'creativity' category, it still categorized much of the clusters as belonging to the general 'creative' label.

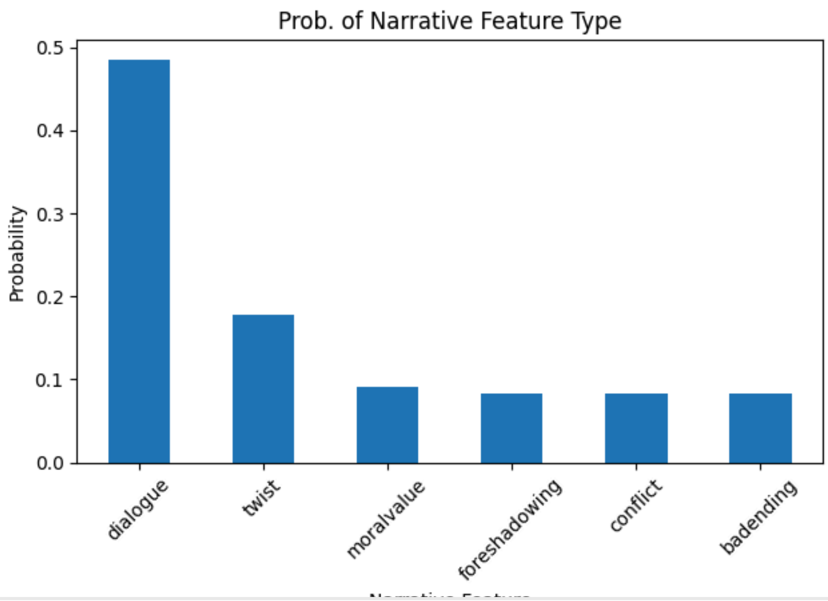
Statistical Insight

Creativity

Objectively speaking, creativity is a hard goal to define as it can be objectively defined in a multitude of way, as are the aforementioned topics of literacy and mental-health. However, many people might agree that creativity is somehow unique. Therefore, it may be possible to define the goal of creativity by understanding the level of variance in the models responses to similar prompts.

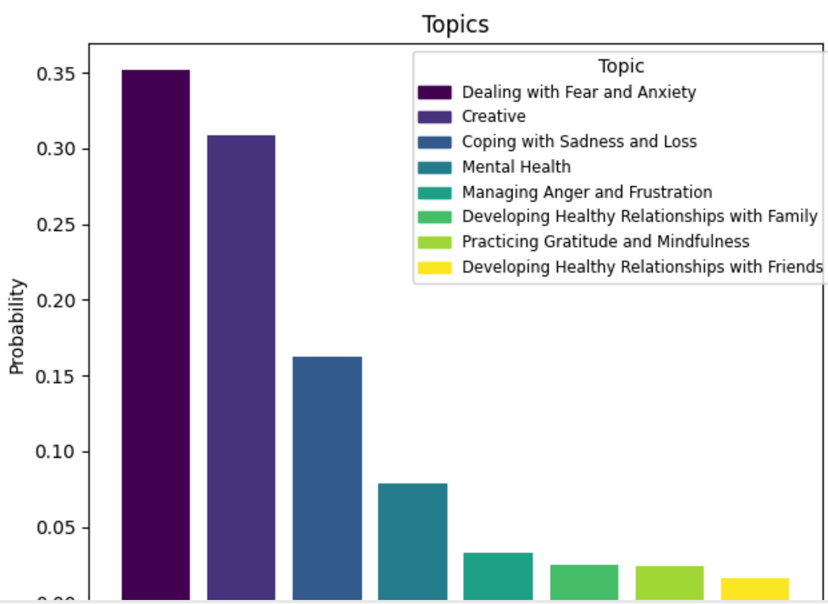
Narrative Features

```
ax = feature_occurence_probabilities.sort_values(ascending=False).plot(
    kind='bar',
    title='Prob. of Narrative Feature Type'
)

# Correctly set labels using set methods
plt.ylabel('Probability')
plt.xlabel('Narrative Feature')
plt.xticks(rotation=45)

# Optional: Adjust layout to prevent label cutoff
plt.tight_layout()

plt.show()
```



Key Words

```
1 plot_topic_counts(all_classified_topic_batches_df)
```



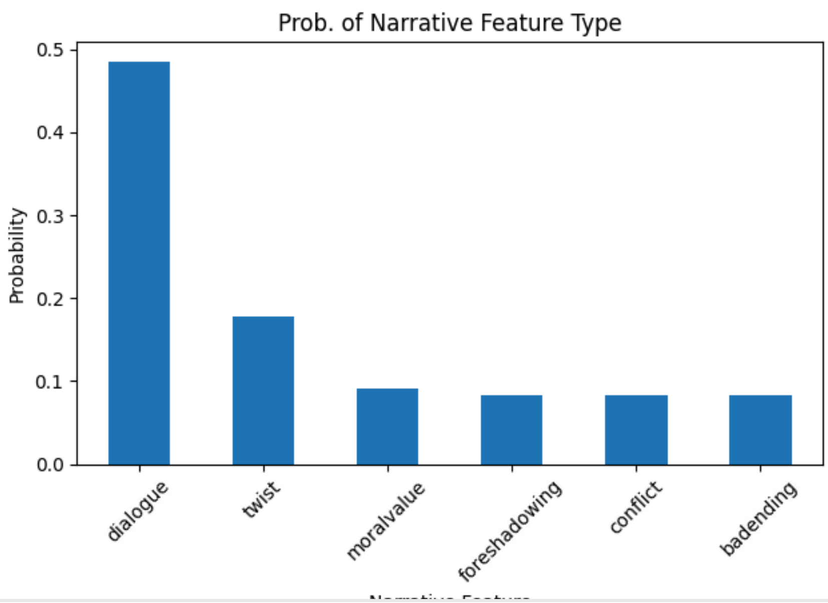
Literacy

WonderWords' literacy goals may be defined in objective terms by classifying whether the responses are at a lower or a higher reading level. As our application is targeting a youth demographic, utilizing the categorization system used in most libraries and school systems will help illustrate whether the training data is biased towards a specific set of reading levels.

Narrative Features

```
ax = feature_occurence_probabilities.sort_values(ascending=False).plot(
    kind='bar',
    title='Prob. of Narrative Feature Type'
)

# Correctly set labels using set methods
```

```
plt.ylabel('Probability')
plt.xlabel('Narrative Feature')
plt.xticks(rotation=45)

# Optional: Adjust layout to prevent label cutoff
plt.tight_layout()

plt.show()
```



Key Words

```
literacy_levels = [
    "Kindergarten (5–6)",
    "1st Grade (6–7)",
    "2nd Grade (7–8)",
    "3rd Grade (8–9)",
    "4th Grade (9–10)",
    "5th Grade (10–11)",
    "6th Grade (11–12)",
    "7th Grade (12–13)",
    "8th Grade (13–14)",
    "9th Grade (14–15)",
    "10th Grade (15–16)",
    "11th Grade (16–17)",
    "12th Grade (17–18)"
]
```

```
def classify_literacy(literacy_levels, topic_df_words_counts):
    """ this takes a preset list of literacy levels,
        and classifies the topic_ids by using the data in the topic_df_words
        note: there are 12 topic_ids and 13 corresponding literacy levels.
        """
    prompt_template = (
        f"The following is a list of education levels and corresponding ages: {literacy_levels}",
        f"Use this list of words and their probabilities to classify the most likely level for this set of words: {topic_df_
        "Note: Each topic in either list can be assigned only once. Only respond with the topic chosen and nothing else.",
        "Nearest literacy level: "
    )
    prompt = "\n".join(prompt_template)
    response = client.chat.completions.create(
        messages=[
            {"role": "system", "content": "You are comfortable classifying the language used in stories by simply inferring
            {"role": "user", "content": prompt}
        ],
        model = model,
        temperature = 0.0
    )
    return response.choices[0].message.content
```

```python
# Step 3: creating a word cloud for all topics within a single iteration
# add the cluster labels to the batch
sample_batch = batches[0]
batch_df = pd.DataFrame({'words': sample_batch, 'topic': topic_assignment_batches[0]})
# Determine the number of rows and columns for the grid
num_topics = len(batch_df['topic'].unique())
num_cols = 3  # Adjust as needed
num_rows = (num_topics + num_cols - 1) // num_cols

# Create the figure and axes for the grid
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))
i = 0
for topic_id in batch_df['topic'].unique():
    topic_df = batch_df[batch_df['topic'] == topic_id].copy()

    # Create a word cloud of the topics
    wordcloud = WordCloud(width=800, height=400, background_color='white', max_words=100)
    topic_df['words'] = topic_df['words'].apply(lambda x: ' '.join(x))
    # Concatenate all word strings in topic_df['words']
    wordcloud_text = ' '.join(topic_df['words'].values)
    # Filter counts of all words in the wordcloud_text and then select top 100 with count > 1
    wordcloud_text_counts = wordcloud.process_text(wordcloud_text)
    wordcloud_text_counts = {k: v for k, v in sorted(wordcloud_text_counts.items(), key=lambda item: item[1], reverse=True)}
    wordcloud_text_counts = dict(list(wordcloud_text_counts.items())[:100])
    wordcloud_text_counts = pd.Series(wordcloud_text_counts, index=wordcloud_text_counts.keys())
    # Filter to words with counts > 1
    wordcloud_text_counts = wordcloud_text_counts[wordcloud_text_counts > 1]
    wordcloud_text_filtered = ' '.join(wordcloud_text_counts.index)
    # Classifying the topic_df['words']
    topic = classify_literacy(literacy_levels, wordcloud_text_counts.to_string())
    topic_df['topic'] = topic
    wordcloud.generate(wordcloud_text_filtered)
    # Plot the wordcloud on the corresponding subplot
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col]  # Get the current subplot
    ax.imshow(wordcloud, interpolation='bilinear')
    ax.axis('off')
    ax.set_title(f'Topic: {topic}')  # Add a title/label
    i += 1
```


```python
# Step 4: classifying all the topic_ids across all bootstrap cluster iterations
all_classified_literacy_batches = []
for i in range(len(topic_assignment_batches)):
    batch_df = pd.DataFrame({'words': batches[i], 'topic': topic_assignment_batches[i]})
    for topic_id in batch_df['topic'].unique():
        topic_df = batch_df[batch_df['topic'] == topic_id].copy()
        topic_df['words'] = topic_df['words'].apply(lambda x: ' '.join(x))
        # Concatenate all word strings in topic_df['words']
        wordcloud_text = ' '.join(topic_df['words'].values)
        # Filter counts of all words in the wordcloud_text and then select top 100 with count > 1
        wordcloud_text_counts = wordcloud.process_text(wordcloud_text)
        wordcloud_text_counts = {k: v for k, v in sorted(wordcloud_text_counts.items(), key=lambda item: item[1], reverse=Tr
        wordcloud_text_counts = dict(list(wordcloud_text_counts.items())[:100])
        wordcloud_text_counts = pd.Series(wordcloud_text_counts, index=wordcloud_text_counts.keys())
        # Filter to words with counts > 1
        wordcloud_text_counts = wordcloud_text_counts[wordcloud_text_counts > 1]
        topic = classify_literacy(literacy_levels, wordcloud_text_counts.to_string())
        topic_df['topic'] = topic
```