Table of Python Scripts, Outputs, and Remarks

<table>
<tr><th colspan="3">Data Exploration</th></tr>
<tr><th>SCRIPT</th><th>OUTPUT</th><th>REMARKS</th></tr>
<tr>
<td>Python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import re
from google.colab import auth, drive
import gspread
from gspread_dataframe import get_as_dataframe
import langdetect

Authenticate and create the client
auth.authenticate_user()

Initialize gspread client
from google.auth import default
creds, _ = default()
gc = gspread.authorize(creds)

Open the Google Sheets file
spreadsheet = gc.open('Summative Activity 1 Prelim Data Mining')</td>
<td>Number of duplicate rows: 10
Total Number of SocMed comments: 511
Number of SocMed comments containing emojis: 0
Number of SocMed comments without emojis: 511</td>
<td>This block of code connects to the user's Google Drive to access the dataset used, cleaning it by removing NaN elements, checking for duplicate rows, replacing shortcuts/slang with proper words, and removing emojis. After which, it returns the number of SocMed comments as well as the number of duplicate rows.</td>
</tr>
</table>

```
# Function to load data from
Google Sheets
def load_data_from_sheet():
    worksheet =
spreadsheet.sheet1  # or you can
select by name or index
    # Load the data into a
Pandas DataFrame
    df =
get_as_dataframe(worksheet,
evaluate_formulas=True)
    # Drop any rows where all
elements are NaN
    df.dropna(how='all',
inplace=True)
    return df

# Fetch the data
data = load_data_from_sheet()

# New: Check for duplicates
duplicate_count =
data.duplicated().sum()
print(f"Number of duplicate
rows: {duplicate_count}")
data.drop_duplicates(inplace=Tru
e)

# Define a regex pattern to
match emojis
emoji_pattern = re.compile("["
```

```
u"\U0001F600-\U0001F64F"  # emoticons
u"\U0001F300-\U0001F5FF"  # symbols & pictographs
u"\U0001F680-\U0001F6FF"  # transport & map symbols
u"\U0001F700-\U0001F77F"  # alchemical symbols
u"\U0001F780-\U0001F7FF"  # Geometric Shapes Extended
u"\U0001F800-\U0001F8FF"  # Supplemental Arrows-C
u"\U0001F900-\U0001F9FF"  # Supplemental Symbols and Pictographs
u"\U0001FA00-\U0001FA6F"  # Chess Symbols
u"\U0001FA70-\U0001FAFF"  # Symbols and Pictographs Extended-A
u"\U00002702-\U000027B0"  # Dingbats
u"\U000024C2-\U0001F251"
```

```
                           "]+",
flags=re.UNICODE)

# Function to remove emojis
def remove_emojis(text):
    if isinstance(text, str):
        return
emoji_pattern.sub('', text)
    return text

# Function to replace
shortcuts/slang
def replace_shortcuts(text):
    shortcuts = {
        'u': 'you',
        'r': 'are',
        'y': 'why',
        'idk': "I don't know",
        'kuno': 'sabi daw',
        'd': 'hindi',
        'di': 'hindi',
        'yong': 'iyong',
        'yung': 'iyong',
        'hs': 'highschool',
        'ganun': 'ganon',
        'gnun': 'ganon',
        'smh': 'Shake my head',
        'wag': 'huwag',
        'confi': 'confidential',
        'ff': 'following',
        'socmed': 'social
media',
        'dun': 'doon',
        'gen': 'generation',
```

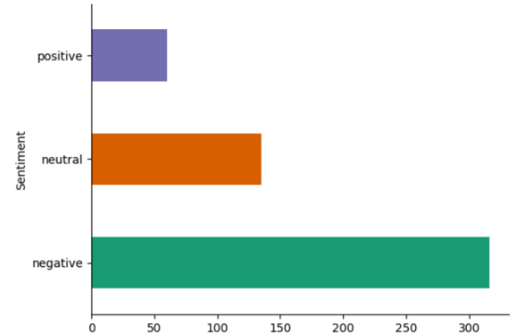
```
        'ung': 'iyong',
        'eto': 'ito',
        'ding': 'din',
        'gawing': 'gawin',
        'orayt': 'alright',
        'wdym': 'What do you
mean',
        'lng': 'lang',
        'tho': 'though',
        'nmp': 'National
Mathematics Program'
    }
    for short, full in
shortcuts.items():
        text = re.sub(r'\b' +
short + r'\b', full, text,
flags=re.IGNORECASE)
    return text

# Function to clean garbage text
def clean_garbage_text(text):
    # Remove HTML entities
    text =
re.sub(r'&amp;|&lt;|&gt;|&quot;|
&#x200B;', '', text)
    # Remove URLs
    text =
re.sub(r'http\S+|www.\S+', '',
text)
    return text

# Function to process text
def process_text(text):
    if isinstance(text, str):
```

```
        text =
remove_emojis(text)  # Remove
emojis first
        text = text.lower()  #
Convert to lowercase
        text =
replace_shortcuts(text)  #
Replace shortcuts/slang
        text =
clean_garbage_text(text)  #
Clean garbage text
        text =
re.sub(r'[^a-zA-Z\s]', '', text)
# Remove special characters,
punctuation, and numbers
    return text

# Makes objects in 'Content'
column be read as str
data['Content'] =
data['Content'].astype(str)

# Apply the text processing
function to the 'Content' column
data['Content'] =
data['Content'].apply(process_te
xt)

# Counts total SocMed comments
total =
data['Content'].value_counts().s
um()
```

<pre># Count SocMed comments containing emojis (this should be zero now after removing emojis) emoji_rows = data['Content'].apply(lambda x: bool(emoji_pattern.search(x))) emoji_count = emoji_rows.sum() # Count SocMed comments without emojis no_emoji_count = len(data) - emoji_rows.sum() print(f'Total Number of SocMed comments: {total}') print(f'Number of SocMed comments containing emojis: {emoji_count}') print(f'Number of SocMed comments without emojis: {no_emoji_count}')</pre>		

Python `data.head(10)`	| | Content | Sentiment | Platform | | 0 | yes to phase out matatag curriculum | negative | threads | | 1 | isa pa talagang problema ang sistema ng edukas... | negative | threads | | 2 | kasabay ng mga pagbabago sa kalendaryo at bago... | negative | threads | | 3 | describe matatag curriculum in one word hell | negative | threads | | 4 | no to matatag curriculum talagaaa tinambakan b... | negative | threads | | 5 | gunggung na matatag curriculum to | negative | threads | | 6 | hindi kami kasing tatag ng matatag curriculum ... | negative | threads | | 7 | grabe kailangan pala ng curriculum matatag fir... | negative | threads | | 8 | collaborative expertise on matatag curriculum ... | positive | threads | | 9 | lintik na matatag curriculum to dami tuloy gin... | negative | threads |	This block of code returns the first 10 rows of the dataset
Python `# For the shape and types of data` `print('Shape: ', data.shape)` `print('Data Types:')` `print(data.dtypes)`	`Shape:  (511, 3)` `Data Types:` `Content      object` `Sentiment    object` `Platform     object` `dtype: object`	This block of code returns the shape of the dataset and the types of data present
Python `# Visualization of content sentiment` `data.groupby('Sentiment').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))` `plt.gca().spines[['top', 'right',]].set_visible(False)`	`Sentiments on MATATAG Curriculum` `negative: 316` `neutral: 135` `positive: 60`	This block of code visualizes the differences of sentiments between the SocMed comments

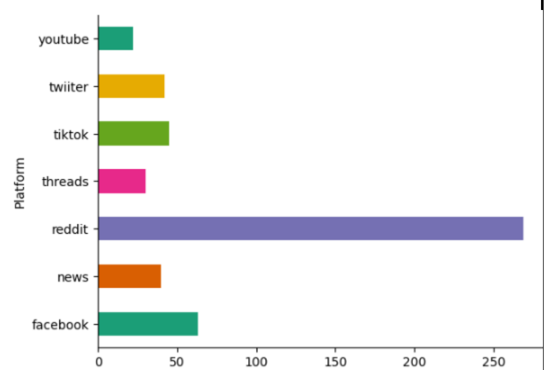
```python
# Count occurences of each
sentiment
print('Sentiments on MATATAG
Curriculum:')
counts =
data['Sentiment'].value_counts()
for sentiment, count in
counts.items():
  print(f"{sentiment}: {count}")
```



Python

```python
# Visualization of content per
platform
data.groupby('Platform').size().
plot(kind='barh',
color=sns.palettes.mpl_palette('
Dark2'))
plt.gca().spines[['top',
'right',]].set_visible(False)

# Count occurences of each
platform
print('Platforms used: ')
counts =
data['Platform'].value_counts()
for sentiment, count in
counts.items():
  print(f"{sentiment}: {count}")
```

```
Platforms used:
reddit: 269
facebook: 63
tiktok: 45
twiiter: 42
news: 40
threads: 30
youtube: 22
```



This block of code visualizes what platforms the SocMed comments were taken from

Topic Modeling		
SCRIPT	**OUTPUT**	**REMARKS**
Python `tagalog_stop_words_string =` `"""akin` `aking` `ako` `alin` `am` `amin` `aming` `ang` `ano` `anumang` `apat` `at` `atin` `ating` `ay` `ba` `bababa` `bago` `bakit` `bawat` `bilang` `dahil` `dalawa` `dapat` `dati` `din` `dito` `doon`		This block of code is for creating a string for tagalog stop words then converting them into a list

```
eh
gagawin
ganon
gayunman
ginagawa
ginawa
ginawang
gumawa
gusto
habang
hanggang
hindi
huwag
iba
ibaba
ibabaw
ibig
ikaw
ilagay
ilalim
ilan
inyong
isa
isang
itaas
ito
iyan
iyo
iyon
iyong
ka
kahit
kailangan
kailanman
```

```
kami
kanila
kanilang
kanino
kanya
kanyang
kapag
kapwa
karamihan
katiyakan
katulad
kaya
kaysa
ko
kong
kulang
kumuha
kung
laban
lahat
lamang
likod
lima
maaari
maaaring
maging
mahusay
makita
marami
marapat
masyado
may
mayroon
mga
```

```
minsan
mismo
mula
muli
na
nabanggit
naging
nagkaroon
nais
nakita
namin
napaka
narito
nasaan
ng
ngang
ngayon
ni
nila
nilang
nito
niya
niyan
niyang
noh
noon
o
pa
paano
pababa
paggawa
pagitan
pagkakaroon
pagkatapos
```

palabas pamamagitan panahon pangalawa para paraan pareho pataas pero pumunta pumupunta rin sa saan sabi sabihin sarili sila sino siya tatlo tayo tulad tungkol una yon yun walang nga si said one like mo		

ma nga kasi nga naman yung yan di lang lng also po talaga kz rn pls pang pag nya nang n mag smh nya""" # Convert the tagalog_stop_words_string to a list of stopwords tagalog_stop_words_list = tagalog_stop_words_string.split()		
		The following block of code fetches the essential NLTK packages for topic

<table>
<tr>
<td>

```python
# imports for nltk package for topic modelling
import nltk

# Download the 'punkt' tokenizer model
nltk.download('punkt')

# Downloading stopwords
nltk.download('stopwords')

# Downloading wordnet
nltk.download('wordnet')
```

</td>
<td>

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

</td>
<td>modeling. The punkt tokenizer is used for text segmentation and stopword identification, while WordNet is for lexical analysis.</td>
</tr>
<tr>
<td>

```python
# Loading of dataset and data pre processing
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

# Preprocessing steps
```

</td>
<td>

```
0                    yes phase matatag curriculum
1      talagang problema sistema edukasyon pilipinas ...
2      kasabay pagbabago kalendaryo bagong matatag cu...
3               describe matatag curriculum word hell
4      matatag curriculum talagaaa tinambakan gawain ...
                              ...
518                                   sana tanggalin k
519    salamat vp sara nagkamali boto inyo marunong k...
520    tanong dep ed sabado linggo binibigyan activit...
521    matatag tooth drill pagka graduate mangalakal ...
522    totoong matatag sana elementary senior high sc...
Name: Processed_Content, Length: 511, dtype: object
```

</td>
<td>This block of code preprocesses the dataset, ensuring that the content column contains strings by filling missing values and converting non-string entries. Afterwards, punctuation, numbers, and special characters like emojis are removed from the strings, stop-words are also filtered out during the process.</td>
</tr>
</table>

<pre># Ensure all values in the 'Content' column are strings by filling NaN and converting non-string values data['Content'] = data['Content'].fillna('').astype(str) # Remove punctuation, numbers, and special characters data['Content'] = data['Content'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x)) # Tokenization data['Tokenized'] = data['Content'].apply(word_tokenize) # Get English stopwords english_stopwords = set(stopwords.words('english')) # Combine both Tagalog and English stopwords combined_stopwords = set(english_stopwords).union(set(tagalog_stop_words_list)) # Stop-word removal using combined stopwords data['No_Stopwords'] = data['Tokenized'].apply(lambda x: [word for word in x if</pre>		

```
word.lower() not in
combined_stopwords])


# Lemmatization
lemmatizer = WordNetLemmatizer()
data['Lemmatized'] =
data['No_Stopwords'].apply(lambd
a x: [lemmatizer.lemmatize(word)
for word in x])

# Join tokens back into strings
data['Processed_Content'] =
data['Lemmatized'].apply(lambda
x: ' '.join(x))

# Vectorization using TF-IDF
tfidf_vectorizer =
TfidfVectorizer()
data_tfidf =
tfidf_vectorizer.fit_transform(d
ata['Processed_Content'])

# Vectorization using
CountVectorizer (alternative)
count_vectorizer =
CountVectorizer()
data_vectorized =
count_vectorizer.fit_transform(d
ata['Processed_Content'])

# Display the processed content
print(data['Processed_Content'])
```

<table>
<tr>
<td>

Python

```python
# Drop rows where all elements are NaN (optional)
data.dropna(how='all', inplace=True)

# Check the number of rows
num_rows = data.shape[0]  # Get number of rows
print(f"Number of rows in the Google Sheet: {num_rows}")
```

</td>
<td>Number of rows in the Google Sheet:
511</td>
<td>The following block of code removes rows wherein all attributes are null, which is part of the data cleaning process.</td>
</tr>
<tr>
<td>

Python

```python
# Data processing

# Apply Topic Modeling
# A. Latent Dirichlet Allocation (LDA)

from sklearn.decomposition import LatentDirichletAllocation

# Number of topics to extract
num_topics = 3

# LDA Model
lda_model = LatentDirichletAllocation(n_components=num_topics, random_state=42)
```

</td>
<td>LDA Topics:

Topic 1:
['budget', 'english', 'sara', 'wala', 'skill', 'deped', 'teacher', 'science', 'matatag', 'curriculum']

Topic 2:
['program', 'year', 'subject', 'grade', 'teacher', 'matatag', 'school', 'deped', 'education', 'curriculum']

Topic 3:
['wala', 'teaching', 'bata', 'education', 'hour', 'school', 'deped', 'curriculum', 'matatag', 'teacher']</td>
<td>This block of code applies topic modeling to the dataset by using the LDA model, extracting 3 topics.</td>
</tr>
</table>

<pre>lda_model.fit(data_vectorized) # Get the topic distribution def print_topics(model, vectorizer, top_n=10): for idx, topic in enumerate(model.components_): print(f"Topic {idx + 1}:") print([vectorizer.get_feature_na mes_out()[i] for i in topic.argsort()[-top_n:]]) print("LDA Topics:") print_topics(lda_model, count_vectorizer)</pre>		
Python <pre># Apply Topic Modeling # Latent Semantic Analysis (LSA) from sklearn.decomposition import TruncatedSVD # LSA Model lsa_model = TruncatedSVD(n_components=num_to pics, random_state=42) lsa_model.fit(data_tfidf)</pre>	LSA Topics: Topic 1: ['new', 'science', 'subject', 'grade', 'education', 'school', 'teacher', 'deped', 'matatag', 'curriculum'] Topic 2: ['stay', 'mamamatay', 'true', 'estudyante', 'grabe', 'deped', 'phase', 'yes', 'curriculum', 'matatag'] Topic 3: ['kid', 'history', 'social', 'makabansa',	This block of code applies topic modeling to the dataset by using the LSA model, extracting 3 topics.

`` # Get the topic distribution print("LSA Topics:") print_topics(lsa_model, tfidf_vectorizer) ```	'matatag', 'grade', 'curriculum', 'gmrc', 'subject', 'science']	
Python ``` # Evaluate the Models # A. Evaluate LDA: Coherence Score import gensim from gensim import corpora from gensim.models import CoherenceModel # Ensure 'Lemmatized' tokens are used for dictionary and corpus data_lemmatized_tokens = data['Lemmatized'] # Create a dictionary from the lemmatized tokens id2word = corpora.Dictionary(data_lemmatized_tokens) ```	LDA Coherence Score: 0.2939349977512377	This block of code is for evaluating the LDA model using coherence scores. The LDA model got a coherence score of around 0.29, suggesting that the topics produced have a moderate interpretability, though the model can be improved by tuning the number of topics or the preprocessing.

``# Filter out words that appear in less than 5 documents or more than 50% of the documents id2word.filter_extremes(no_below=5, no_above=0.5) # Create the bag-of-words corpus corpus = [id2word.doc2bow(text) for text in data_lemmatized_tokens] # Set number of topics, passes, and iterations passes = 20 iterations = 100 # Build the LDA model lda_gensim_model = gensim.models.ldamodel.LdaModel(corpus=corpus, id2word=id2word, num_topics=num_topics, passes=passes, iterations=iterations, random_state=42) # Compute Coherence Score using 'c_v' coherence_model_lda = CoherenceModel(model=lda_gensim_model, texts=data_lemmatized_tokens,```		

<pre>dictionary=id2word, coherence='c_v') coherence_lda = coherence_model_lda.get_coherenc e() print(f'LDA Coherence Score: {coherence_lda}')</pre>		
Python <pre>from sklearn.cluster import KMeans from sklearn.metrics import silhouette_score import numpy as np # Number of clusters (we'll use the same as the number of topics for LSA) num_clusters = num_topics # Apply K-means clustering to the LSA components (topic-term matrix) kmeans_model = KMeans(n_clusters=num_clusters, random_state=42) lsa_topic_matrix = lsa_model.transform(data_tfidf) # Transform data into LSA topic space</pre>	K-means Silhouette Score for LSA: 0.4839 Cluster centers (top words per cluster): Cluster 1: aabutin, aadjust, aabot Cluster 2: aadjust, aabutin, aabot Cluster 3: aabutin, aadjust, aabot	This block of code is for evaluating the performance of the LSA model. K-means clustering was used to get the top words per cluster, and compute for a score of 0.48, suggesting that though the cluster is reasonable, similar top words may mean that they are not well separated.

`` # Fit the K-means model kmeans_model.fit(lsa_topic_matrix) # Predict cluster labels cluster_labels = kmeans_model.predict(lsa_topic_matrix) # Compute the Silhouette Score silhouette_avg = silhouette_score(lsa_topic_matrix, cluster_labels) print(f"K-means Silhouette Score for LSA: {silhouette_avg:.4f}") # Optionally, if you want to analyze cluster centers print("Cluster centers (top words per cluster):") for idx, topic_center in enumerate(kmeans_model.cluster_centers_): top_word_indices = topic_center.argsort()[-10:] # Top 10 words in each cluster top_words = [tfidf_vectorizer.get_feature_names_out()[i] for i in top_word_indices] print(f"Cluster {idx + 1}: {', '.join(top_words)}") ```		

4 Topics		
LDA Topics	LDA Topics: Topic 1: ['wala', 'year', 'science', 'grade', 'school', 'deped', 'subject', 'teacher', 'matatag', 'curriculum'] Topic 2: ['teacher', 'school', 'matatag', 'ma', 'grade', 'subject', 'bata', 'curriculum', 'deped', 'science'] Topic 3: ['student', 'wala', 'sana', 'guro', 'subject', 'science', 'deped', 'teacher', 'curriculum', 'matatag'] Topic 4: ['matatag', 'day', 'year', 'education', 'deped', 'school', 'subject', 'curriculum', 'grade', 'science']	Topic 1 and 4 show similarity as both discuss the matatag curriculum as a whole and its major imposed changes such as the later deployment of the science subject. Topic 2 and 3 also show similarity as these topics focus more on the effects and sentiments of the matatag curriculum upon students and teachers.
LDA topics coherence score	0.3621500406774457	A score of 0.36 indicates that there is a moderate level of coherence between topics; however, there is a certain level of specificity and definition that is not achieved.
LSA Topics	LSA Topics: Topic 1: ['phase', 'yes', 'school', 'subject', 'grade', 'teacher', 'science', 'deped', 'curriculum', 'matatag'] Topic 2: ['reading', 'skill', 'filipino', 'english', 'makabansa', 'school', 'gmrc', 'grade', 'subject', 'science']	Topic 1 discusses general facts that the matatag curriculum imposes, while Topic 2 becomes more specific as it discusses the subjects that will be changed or added such as makabansa, GMRC, science, and reading. Topic 3 and 4 show similarity as these topics focus more on the effects and sentiments of the matatag curriculum upon students and teachers.

	Topic 3: ['critical', 'mamamatay', 'thinking', 'social', 'included', 'yes', 'matatag', 'phase', 'curriculum', 'science'] Topic 4: ['feel', 'matatag', 'social', 'kid', 'included', 'basic', 'secretary', 'school', 'deped', 'science']	
LSA K-Means Silhoutte score	K-means Silhouette Score for LSA: 0.3965 Cluster centers (top words per cluster): Cluster 1: aalisin, aangkop, aadjust, aabutin Cluster 2: aangkop, aalisin, aabutin, aadjust Cluster 3: aadjust, aangkop, aalisin, aabutin Cluster 4: aalisin, aangkop, aabutin, aadjust	A silhouette score of 0.3965 may suggest moderate performance and clusters may not be as well-defined and separated. Top words per cluster are same all throughout clusters which may indicate overlap or redundancy within themes.
5 Topics		
LDA Topics	LDA Topics: Topic 1: ['new', 'sara', 'wala', 'subject', 'school', 'bata', 'deped', 'teacher', 'matatag', 'curriculum'] Topic 2: ['wala', 'sana', 'secretary', 'ma', 'education', 'student', 'science', 'curriculum', 'matatag', 'deped'] Topic 3: ['student', 'science', 'subject', 'sana', 'guro', 'wala', 'deped', 'teacher',	Topic 1 and 2 show similarity as these topics focus more on the reform of the curriculum itself as well as on key figures such as former DepEd secretary Sara Duterte. Topic 3 and 4 also show similarity as both discuss the matatag curriculum as a whole and its major imposed changes such as the later deployment of the science subject. Lastly, topic 5 becomes more specific as it discusses the subjects that will be changed or added such as makabansa, GMRC, science, and

	'curriculum', 'matatag'] Topic 4: ['tapos', 'wala', 'subject', 'teacher', 'school', 'makabansa', 'deped', 'curriculum', 'grade', 'science'] Topic 5: ['mapeh', 'math', 'english', 'curriculum', 'school', 'filipino', 'gmrc', 'grade', 'subject', 'science']	reading.
LDA topics coherence score	LDA Coherence Score: 0.3516777353222008	A score of 0.35 indicates that there is a moderate level of coherence between topics; however, there is a certain level of specificity and definition that is not achieved.
LSA Topics	LSA Topics: Topic 1: ['phase', 'yes', 'school', 'subject', 'grade', 'teacher', 'science', 'deped', 'curriculum', 'matatag'] Topic 2: ['reading', 'skill', 'filipino', 'english', 'makabansa', 'school', 'gmrc', 'grade', 'subject', 'science'] Topic 3: ['nalesson', 'critical', 'thinking', 'social', 'included', 'yes', 'matatag', 'phase', 'curriculum', 'science'] Topic 4: ['kid', 'high', 'social', 'included', 'sana', 'basic', 'secretary', 'school', 'deped', 'science'] Topic 5:	Topic 1 discusses the Matatag curriculum as a whole and its major imposed changes, such as the later deployment of the science subject. Topic 2 becomes more specific as it discusses the subjects that will be changed or added, such as makabansa, GMRC, science, and reading. Topic 3 and 4 show similarity, as these topics focus more on the effects and sentiments of the Matatag curriculum upon students and teachers. Topic 5 focuses more on the reform of the curriculum itself as well as on key figures such as former DepEd secretary Sara Duterte.

<table>
<tr>
<td></td>
<td>['true', 'magresign', 'science', 'everyone', 'grade', 'matatag', 'secretary', 'gmrc', 'makabansa', 'deped']</td>
<td></td>
</tr>
<tr>
<td>LSA K-Means Silhoutte score</td>
<td>K-means Silhouette Score for LSA: 0.3653
Cluster centers (top words per cluster):
Cluster 1: aangkop, aalisin, aanhin, aabutin, aadjust
Cluster 2: aanhin, aalisin, aangkop, aadjust, aabutin
Cluster 3: aadjust, aangkop, aanhin, aalisin, aabutin
Cluster 4: aanhin, aalisin, aangkop, aabutin, aadjust
Cluster 5: aalisin, aadjust, aangkop, aanhin, aabutin</td>
<td>A silhouette score of 0.3653 may suggest moderate performance and clusters may not be as well-defined and separated. Top words per cluster are same all throughout clusters which may indicate overlap or redundancy within themes.</td>
</tr>
<tr>
<td colspan="3">Evaluation of the LSA and LDA 3 topics</td>
</tr>
<tr>
<td>Python<pre><code># Evaluate the Models
Evaluate LSA: Explained Variance

Explained variance for LSA
explained_variance = lsa_model.explained_variance_ratio_
print(f'Explained Variance Ratio for LSA: {explained_variance}')</code></pre></td>
<td>Explained Variance Ratio for LSA:
[0.01531701 0.0168826 0.01127805]</td>
<td>The low outputs for the variance ratio of the Latent Semantic Analysis model may suggest that the components captures a small amount in the total variance of the data, which in turn denotes that the LSA may not be effectively summarizing the whole structure of the corpus.</td>
</tr>
</table>

Python
```python
# Analyze and Discuss

# LDA Analysis
print("LDA Topic Analysis:")
for idx, topic in lda_gensim_model.print_topics(num_words=10):
    print(f"Topic {idx + 1}:")
    print("Words:", [word.split('*')[1] for word in topic.split(' + ')])
    print("Interpretation: Topic likely represents ...")

# LSA Analysis
# Assuming lsa_model and tfidf_vectorizer are defined
print("\nLSA Topic Analysis:")
for idx, topic in enumerate(lsa_model.components_):
    print(f"Topic {idx + 1}:")
    top_words = [tfidf_vectorizer.get_feature_names_out()[i] for i in topic.argsort()[-10:]]
    print("Words:", top_words)
    print("Interpretation: Topic likely represents ...")
```

```
LDA Topic Analysis:
Topic 1:
Words: ['"science"', '"subject"', '"curriculum"', '"grade"', '"teacher"', '"bata"', '"wala"', '"school"', '"makabansa"', '"matatag"']
Interpretation: Topic likely represents ...
Topic 2:
Words: ['"teacher"', '"deped"', '"education"', '"curriculum"', '"student"', '"school"', '"program"', '"matatag"', '"teaching"', '"learning"
Interpretation: Topic likely represents ...
Topic 3:
Words: ['"curriculum"', '"matatag"', '"school"', '"education"', '"deped"', '"grade"', '"year"', '"teacher"', '"k"', '"learner"']
Interpretation: Topic likely represents ...

LSA Topic Analysis:
Topic 1:
Words: ['new', 'science', 'subject', 'grade', 'education', 'school', 'teacher', 'deped', 'matatag', 'curriculum']
Interpretation: Topic likely represents ...
Topic 2:
Words: ['stay', 'mamamatay', 'true', 'estudyante', 'grabe', 'deped', 'phase', 'yes', 'curriculum', 'matatag']
Interpretation: Topic likely represents ...
Topic 3:
Words: ['kid', 'history', 'social', 'makabansa', 'matatag', 'grade', 'curriculum', 'gmrc', 'subject', 'science']
Interpretation: Topic likely represents ...
```

LDA Topic Analysis:
Topic 1:
Words: ['"science"', '"subject"', '"curriculum"', '"grade"', '"teacher"', '"bata"', '"wala"', '"school"', '"makabansa"', '"matatag"']
Interpretation: Topic likely represents ...
Topic 2:
Words: ['"teacher"', '"deped"', '"education"', '"curriculum"', '"student"', '"school"', '"program"', '"matatag"', '"teaching"', '"learning"']
Interpretation: Topic likely represents ...
Topic 3:
Words: ['"curriculum"', '"matatag"', '"school"', '"education"', '"deped"', '"grade"', '"year"', '"teacher"', '"k"', '"learner"']
Interpretation: Topic likely represents ...

LSA Topic Analysis:
Topic 1:
Words: ['new', 'science', 'subject', 'grade', 'education', 'school', 'teacher', 'deped', 'matatag', 'curriculum']
Interpretation: Topic likely represents ...
Topic 2:
Words: ['stay', 'mamamatay', 'true',

Both LDA and LSA models identify themes that are discussed within the context of the launch of the matatag curriculum. LDA topics, however, focused more on general descriptions made on the curriculum itself and the Philippine education system, seeming a bit more formal. Meanwhile, the LSA model was able to capture the mass reactions and major sentiments of students and teachers upon the launch of the curriculum. The developed models were able to output unique insights on the topic.

	'estudyante', 'grabe', 'deped', 'phase', 'yes', 'curriculum', 'matatag'] Interpretation: Topic likely represents ... Topic 3: Words: ['kid', 'history', 'social', 'makabansa', 'matatag', 'grade', 'curriculum', 'gmrc', 'subject', 'science'] Interpretation: Topic likely represents ...	

SCRIPT	OUTPUT	REMARKS

SCRIPT

Python
```python
# Machine-Based Model: Logistic Regression & Random Forest for Sentiment Analysis

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from imblearn.over_sampling import SMOTE

# Load data (assuming 'data' is already preprocessed as in earlier steps)
```

OUTPUT

```
Logistic Regression Results:
Accuracy: 0.6990291262135923
Classification Report:
              precision    recall  f1-score   support

    negative       0.68      0.98      0.80        64
     neutral       0.90      0.29      0.44        31
    positive       0.00      0.00      0.00         8

    accuracy                           0.70       103
   macro avg       0.53      0.42      0.41       103
weighted avg       0.69      0.70      0.63       103
```

```
Random Forest Results:
Accuracy: 0.7281553398058253
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_
  _warn_prf(average, modifier, msg_start, len(result))
Classification Report:
              precision    recall  f1-score   support

    negative       0.70      0.98      0.82        64
     neutral       0.92      0.35      0.51        31
    positive       1.00      0.12      0.22         8

    accuracy                           0.73       103
   macro avg       0.87      0.49      0.52       103
weighted avg       0.79      0.73      0.68       103
```

REMARKS

The following block code performs sentiment analysis using logistic regression and random forest classifiers, with preprocessing and performance metrics included. Based on the output accuracy, the random forest model achieved a higher accuracy with 72.81%, as compared to the logistic regression model that achieved 69.90%. Both models struggled to predict the 'positive' class, which also explains the minimal accuracy difference between the two models.

``X = data['Processed_Content'] y = data['Sentiment'] #TODO Smote data # Split dataset into training, validation, and testing sets (80-20 partition for train-test) X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Take 10% of the training data as validation set X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=42) # TF-IDF Vectorization tfidf_vectorizer = TfidfVectorizer(max_features=100 0) X_train_tfidf = tfidf_vectorizer.fit_transform(X _train) X_val_tfidf = tfidf_vectorizer.transform(X_val) X_test_tfidf = tfidf_vectorizer.transform(X_tes t)```		

```python
# Logistic Regression Model
log_reg =
LogisticRegression(random_state=
42)
log_reg.fit(X_train_tfidf,
y_train)

# Random Forest Model
rf_model =
RandomForestClassifier(random_st
ate=42)
rf_model.fit(X_train_tfidf,
y_train)

# Model Evaluation: Logistic
Regression
log_reg_pred =
log_reg.predict(X_test_tfidf)
print("Logistic Regression
Results:")
print("Accuracy:",
accuracy_score(y_test,
log_reg_pred))
print("Classification
Report:\n",
classification_report(y_test,
log_reg_pred))

# Model Evaluation: Random
Forest
rf_pred =
rf_model.predict(X_test_tfidf)
print("Random Forest Results:")
```

<table>
<tr>
<td>print("Accuracy:",
accuracy_score(y_test, rf_pred))
print("Classification
Report:\n",
classification_report(y_test,
rf_pred))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python
Apply SMOTE to balance the
classes in the training set
smote = SMOTE(random_state=42)
smoteX_train_tfidf_resampled,
smotey_train_resampled =
smote.fit_resample(X_train_tfidf
, y_train)
smoteX_test_tfidf_resampled,
smotey_test_resampled =
smote.fit_resample(X_test_tfidf,
y_test)

Logistic Regression Model for
smoted
log_reg_smote =
LogisticRegression(random_state=
42)
log_reg_smote.fit(smoteX_train_t
fidf_resampled,
smotey_train_resampled)

Random Forest Model for smoted</td>
<td>Logistic Regression Results with smote:
Accuracy: 0.5989583333333334
Classification Report:
precision recall f1-score support

negative 0.48 0.70 0.57 64
neutral 0.68 0.47 0.56 64
positive 0.73 0.62 0.67 64

accuracy 0.60 192
macro avg 0.63 0.60 0.60 192
weighted avg 0.63 0.60 0.60 192

Random Forest Results with smote:
Accuracy: 0.5364583333333334
Classification Report:
precision recall f1-score support

negative 0.39 0.66 0.49 64
neutral 0.79 0.52 0.62 64
positive 0.67 0.44 0.53 64

accuracy 0.54 192
macro avg 0.61 0.54 0.55 192
weighted avg 0.61 0.54 0.55 192</td>
<td>The following code block also develops random forest and logistic regression models for sentiment analysis, however, a noticeable difference is the application of SMOTE to balance the training set. Both of the models' accuracy was reduced, with the random forest model achieving 53.64% and the logistic regression model having 59.89%. This indicates that resampling and balancing the distribution of the training dataset did not necessarily improve the performance of the model, deeming it unnecessary.</td>
</tr>
</table>

```python
rf_model_smote = RandomForestClassifier(random_state=42)
rf_model_smote.fit(smoteX_train_tfidf_resampled, smotey_train_resampled)

# Model Evaluation: Logistic Regression for smote
log_reg_pred_smote = log_reg_smote.predict(smoteX_test_tfidf_resampled)
print("Logistic Regression Results with smote:")
print("Accuracy:", accuracy_score(smotey_test_resampled, log_reg_pred_smote))
print("Classification Report:\n", classification_report(smotey_test_resampled, log_reg_pred_smote))

# Model Evaluation: Random Forest for smote
rf_pred_smote = rf_model_smote.predict(smoteX_test_tfidf_resampled)
print("Random Forest Results with smote:")
print("Accuracy:", accuracy_score(smotey_test_resampled, rf_pred_smote))
```

<table>
<tr>
<td>

```
print("Classification
Report:\n",
classification_report(smotey_tes
t_resampled, rf_pred_smote))
```

</td>
<td></td>
<td></td>
</tr>
<tr>
<td>

```python
# Evaluate the Logistic
Regression model on the
validation set
log_reg_val_pred =
log_reg.predict(X_val_tfidf)
print("Logistic Regression
Results on Validation Set:")
print("Accuracy:",
accuracy_score(y_val,
log_reg_val_pred))
print("Classification
Report:\n",
classification_report(y_val,
log_reg_val_pred))

# Evaluate the Random Forest
model on the validation set
rf_val_pred =
rf_model.predict(X_val_tfidf)
print("Random Forest Results on
Validation Set:")
print("Accuracy:",
accuracy_score(y_val,
rf_val_pred))
```

</td>
<td>

```
Logistic Regression Results on Validation Set:
Accuracy: 0.6341463414634146
Classification Report:
               precision    recall  f1-score   support

    negative       0.62      1.00      0.77        25
     neutral       1.00      0.09      0.17        11
    positive       0.00      0.00      0.00         5

    accuracy                           0.63        41
   macro avg       0.54      0.36      0.31        41
weighted avg       0.65      0.63      0.51        41

Random Forest Results on Validation Set:
Accuracy: 0.6585365853658537
Classification Report:
               precision    recall  f1-score   support

    negative       0.64      1.00      0.78        25
     neutral       1.00      0.09      0.17        11
    positive       1.00      0.20      0.33         5

    accuracy                           0.66        41
   macro avg       0.88      0.43      0.43        41
weighted avg       0.78      0.66      0.56        41
```

</td>
<td>The following code block tests the random forest and logistic regression models on a validation set to evaluate their performance. The random forest model had an accuracy of 65.85%, whereas the logistic regression model achieved an accuracy of 63.41%. Both models had good recall of the ‘negative’ class; however, both also struggled with the remaining classes, ‘neutral’ and 'positive.’</td>
</tr>
</table>

<table>
<tr>
<td>

```
print("Classification
Report:\n",
classification_report(y_val,
rf_val_pred))
```

</td>
<td></td>
<td></td>
</tr>
<tr>
<td>

```python
# Evaluate the Logistic
Regression model with smote on
the validation set
log_reg_val_pred =
log_reg_smote.predict(X_val_tfid
f)
print("Logistic Regression
Results on Validation Set:")
print("Accuracy:",
accuracy_score(y_val,
log_reg_val_pred))
print("Classification
Report:\n",
classification_report(y_val,
log_reg_val_pred))

# Evaluate the Random Forest
model with smote on the
validation set
rf_val_pred =
rf_model_smote.predict(X_val_tfi
df)
print("Random Forest Results on
Validation Set:")
```

</td>
<td>

```
Logistic Regression Results on Validation Set:
Accuracy: 0.6097560975609756
Classification Report:
               precision    recall  f1-score   support

    negative       0.75      0.72      0.73        25
     neutral       0.56      0.45      0.50        11
    positive       0.25      0.40      0.31         5

    accuracy                           0.61        41
   macro avg       0.52      0.52      0.51        41
weighted avg       0.64      0.61      0.62        41

Random Forest Results on Validation Set:
Accuracy: 0.6585365853658537
Classification Report:
               precision    recall  f1-score   support

    negative       0.70      0.92      0.79        25
     neutral       0.60      0.27      0.37        11
    positive       0.33      0.20      0.25         5

    accuracy                           0.66        41
   macro avg       0.54      0.46      0.47        41
weighted avg       0.63      0.66      0.61        41
```

</td>
<td>The following code block tests the random forest and logistic regression models applied with SMOTE on a validation set to evaluate their performance. The random forest model maintained an accuracy of 65.85%, whereas the logistic regression model lowered with an accuracy of 60.97%. Similarly, both models had good recall of the ‘negative’ class; however, both also struggled with the remaining classes, ‘neutral’ and 'positive.’</td>
</tr>
</table>

`print("Accuracy:", accuracy_score(y_val, rf_val_pred))` `print("Classification Report:\n", classification_report(y_val, rf_val_pred))`		
Python `# Rule-Based Model: Using VADER for Sentiment Analysis` `from nltk.sentiment.vader import SentimentIntensityAnalyzer` `import numpy as np` `# Initialize VADER sentiment analyzer` `nltk.download('vader_lexicon')` `sid = SentimentIntensityAnalyzer()` `# Function to classify sentiment based on VADER scores` `def classify_sentiment_vader(text):` `    score = sid.polarity_scores(text)` `    compound = score['compound']` `    if compound >= 0.05:`	VADER Rule-Based Sentiment Analysis Results: Accuracy: 0.324853228962818 Classification Report: `              precision    recall  f1-score   support` `    negative       0.84      0.22      0.35       316` `     neutral       0.26      0.44      0.33       135` `    positive       0.18      0.60      0.28        60` `    accuracy                           0.32       511` `   macro avg       0.43      0.42      0.32       511` `weighted avg       0.61      0.32      0.34       511`	In order to perform rule-based analysis, the VADER sentiment analysis tool was utilized and was evaluated against sentiment labels. It only achieved an accuracy of 32.48% which indicates low performance overall. Similarly, it struggles with the ‘positive’ and ‘neutral’ classes but has better precision for ‘negative’ sentiments.

```python
        return 'positive'
    elif compound <= -0.05:
        return 'negative'
    else:
        return 'neutral'

# Apply VADER rule-based classification
data['Vader_Sentiment'] = data['Processed_Content'].apply(classify_sentiment_vader)

# Evaluate Rule-Based Model (comparing with true 'Sentiment' values)
print("VADER Rule-Based Sentiment Analysis Results:")
print("Accuracy:", accuracy_score(data['Sentiment'], data['Vader_Sentiment']))
print("Classification Report:\n", classification_report(data['Sentiment'], data['Vader_Sentiment']))
```

Python

```python
# Evaluation Summary
```

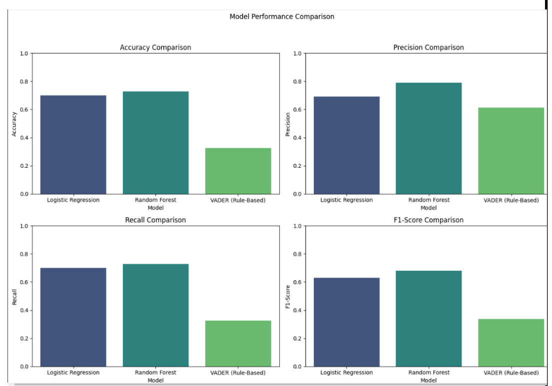
```
Comparison of Model Performance:
                  Model  Accuracy  Precision    Recall  F1-Score
0   Logistic Regression  0.699029   0.691795  0.699029  0.630804
1         Random Forest  0.728155   0.788511  0.728155  0.679630
2    VADER (Rule-Based)  0.324853   0.611805  0.324853  0.336605
```

The table summarizes the performance of all models developed in a comparative table. The output indicates that the random forest model performed the best out of the three, with an accuracy of

```
import pandas as pd

# Results for Logistic
Regression, Random Forest, and
VADER
evaluation_results =
pd.DataFrame({
    'Model': ['Logistic
Regression', 'Random Forest',
'VADER (Rule-Based)'],
    'Accuracy':
[accuracy_score(y_test,
log_reg_pred),
accuracy_score(y_test, rf_pred),
accuracy_score(data['Sentiment']
, data['Vader_Sentiment'])],
    'Precision':
[classification_report(y_test,
log_reg_pred,
output_dict=True)['weighted
avg']['precision'],

classification_report(y_test,
rf_pred,
output_dict=True)['weighted
avg']['precision'],

classification_report(data['Sent
iment'],
data['Vader_Sentiment'],
output_dict=True)['weighted
avg']['precision']],
```

72.81% while the other machine-based model, logistic regression, falls slightly behind with 69.90%. The only rule-based model, VADER, significantly underperformed, which may indicate that rule-based models have more limitations as compared to machine-based models.

<pre> 'Recall': [classification_report(y_test, log_reg_pred, output_dict=True)['weighted avg']['recall'], classification_report(y_test, rf_pred, output_dict=True)['weighted avg']['recall'], classification_report(data['Sent iment'], data['Vader_Sentiment'], output_dict=True)['weighted avg']['recall']], 'F1-Score': [classification_report(y_test, log_reg_pred, output_dict=True)['weighted avg']['f1-score'], classification_report(y_test, rf_pred, output_dict=True)['weighted avg']['f1-score'], classification_report(data['Sent iment'], data['Vader_Sentiment'], output_dict=True)['weighted avg']['f1-score']] })</pre>		

print("Comparison of Model Performance:") print(evaluation_results)		
Python import matplotlib.pyplot as plt import seaborn as sns # Set up the figure and axes for plotting fig, axes = plt.subplots(2, 2, figsize=(14, 10)) fig.suptitle('Model Performance Comparison') # Create individual bar plots for Accuracy, Precision, Recall, and F1-Score sns.barplot(x='Model', y='Accuracy', data=evaluation_results, ax=axes[0, 0], palette='viridis') axes[0, 0].set_title('Accuracy Comparison') axes[0, 0].set_ylim(0, 1)	 	The table summarizes the performance of all models developed in a comparative graph. Once more, both machine-based models, logistic regression and random forest, outperform the only rule-based model, VADER, indicating that rule-based models have more limitations as compared to machine-based models.

<pre>sns.barplot(x='Model', y='Precision', data=evaluation_results, ax=axes[0, 1], palette='viridis') axes[0, 1].set_title('Precision Comparison') axes[0, 1].set_ylim(0, 1) sns.barplot(x='Model', y='Recall', data=evaluation_results, ax=axes[1, 0], palette='viridis') axes[1, 0].set_title('Recall Comparison') axes[1, 0].set_ylim(0, 1) sns.barplot(x='Model', y='F1-Score', data=evaluation_results, ax=axes[1, 1], palette='viridis') axes[1, 1].set_title('F1-Score Comparison') axes[1, 1].set_ylim(0, 1) # Adjust layout for better spacing plt.tight_layout(rect=[0, 0.03, 1, 0.95]) # Show the plot plt.show()</pre>		

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td colspan="3">Updated Script, Screenshots and Remarks</td></tr>
<tr><td>

```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import re
from google.colab import auth, drive
import gspread
from gspread_dataframe import get_as_dataframe
import langdetect
```

</td><td></td><td>Importing Libraries</td></tr>
<tr><td>

```python
from sklearn.metrics import cohen_kappa_score
import pandas as pd
```

</td><td>{'annotation1_vs_annotation2': 0.8108515331324365, 'annotation1_vs_annotation3': 0.694174184798632, 'annotation2_vs_annotation3': 0.8416674298623195}</td><td>The following block of code computes the Kappa statistic within the three annotations. With scores ranging from 69.41% to 84.16%, it indicates that there is a moderate to substantial level of agreement between the annotators while still maintaining variability.</td></tr>
</table>

```
# Load data from the three
sheets
sheet_names = ['Annotation1',
'Annotation2', 'Annotation3']
dfs =
[get_as_dataframe(spreadsheet.wo
rksheet(name),
evaluate_formulas=True).dropna(h
ow='all') for name in
sheet_names]

# Assign DataFrames to variables
df1, df2, df3 = dfs

# Ensure the Sentiment columns
have consistent data types
(strings)
df1['Sentiment'] =
df1['Sentiment'].astype(str)
df2['Sentiment'] =
df2['Sentiment'].astype(str)
df3['Sentiment'] =
df3['Sentiment'].astype(str)

# Remove any rows where
'Sentiment' is NaN
df1 =
df1.dropna(subset=['Sentiment'])
df2 =
df2.dropna(subset=['Sentiment'])
df3 =
df3.dropna(subset=['Sentiment'])

# Compute Kappa statistics
```

`` kappa_scores = { 'annotation1_vs_annotation2': cohen_kappa_score(df1['Sentiment '], df2['Sentiment']), 'annotation1_vs_annotation3': cohen_kappa_score(df1['Sentiment '], df3['Sentiment']), 'annotation2_vs_annotation3': cohen_kappa_score(df2['Sentiment '], df3['Sentiment']) } print(kappa_scores) ```		
Python ``` # Authenticate and create the client auth.authenticate_user() # Initialize gspread client from google.auth import default creds, _ = default() gc = gspread.authorize(creds) # Open the Google Sheets file ```	Number of duplicate rows: 9 Total Number of SocMed comments: 442 Number of SocMed comments containing emojis: 86 Number of SocMed comments without emojis: 356	Loads dataset, checks for duplicates, replaces shortcuts/slang, and returns comments with and without emojis

<pre>spreadsheet = gc.open('Summative Activity 1 Prelim Data Mining') # Replace with your actual spreadsheet name # Function to load data from Google Sheets def load_data_from_sheet(): #worksheet = spreadsheet.worksheet("Sheet1") # or you can select by name or index if topic modelling worksheet = spreadsheet.worksheet("FinalSentiment") # Load the data into a Pandas DataFrame df = get_as_dataframe(worksheet, evaluate_formulas=True) # Drop any rows where all elements are NaN df.dropna(how='all', inplace=True) return df # Fetch the data data = load_data_from_sheet() # New: Check for duplicates duplicate_count = data.duplicated().sum()</pre>		

<pre>print(f"Number of duplicate rows: {duplicate_count}") data.drop_duplicates(inplace=Tru e) # Function to replace shortcuts/slang def replace_shortcuts(text): shortcuts = { 'u': 'you', 'r': 'are', 'y': 'why', 'idk': "I don't know", 'kuno': 'sabi daw', 'd': 'hindi', 'di': 'hindi', 'yong': 'iyong', 'yung': 'iyong', 'hs': 'highschool', 'ganun': 'ganon', 'gnun': 'ganon', 'smh': 'Shake my head', 'wag': 'huwag', 'confi': 'confidential', 'ff': 'following', 'socmed': 'social media', 'dun': 'doon', 'gen': 'generation', 'ung': 'iyong', 'eto': 'ito', 'ding': 'din', 'gawing': 'gawin',</pre>		

```
        'orayt': 'alright',
        'wdym': 'What do you
mean',
        'lng': 'lang',
        'tho': 'though',
        'nmp': 'National
Mathematics Program'
    }
    for short, full in
shortcuts.items():
        text = re.sub(r'\b' +
short + r'\b', full, text,
flags=re.IGNORECASE)
    return text

# Function to clean garbage text
def clean_garbage_text(text):
    # Remove HTML entities
    text =
re.sub(r'&amp;|&lt;|&gt;|&quot;|
&#x200B;', '', text)
    # Remove URLs
    text =
re.sub(r'http\S+|www.\S+', '',
text)
    return text

# Function to process text
def process_text(text):
    if isinstance(text, str):
        text = text.lower()  #
Convert to lowercase
```

<pre> text = replace_shortcuts(text) # Replace shortcuts/slang text = clean_garbage_text(text) # Clean garbage text return text # Makes objects in 'Content' column be read as str data['Content'] = data['Content'].astype(str) # Apply the text processing function to the 'Content' column data['Content'] = data['Content'].apply(process_te xt) # Counts total SocMed comments total = data['Content'].value_counts().s um() # Count SocMed comments containing emojis (this should be zero now after removing emojis) emoji_rows = data['Content'].apply(lambda x: bool(emoji_pattern.search(x))) emoji_count = emoji_rows.sum()</pre>		

<table>
<tr>
<td>

```python
# Count SocMed comments without emojis
no_emoji_count = len(data) - emoji_rows.sum()

print(f'Total Number of SocMed comments: {total}')
print(f'Number of SocMed comments containing emojis: {emoji_count}')
print(f'Number of SocMed comments without emojis: {no_emoji_count}')
```

</td>
<td></td>
<td></td>
</tr>
<tr>
<td>

Python

```python
# Machine-Based Model: Logistic Regression & Random Forest for Sentiment Analysis

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

</td>
<td>

```
Logistic Regression Results:
Accuracy: 0.797752808988764
Classification Report:
              precision    recall  f1-score   support

    negative       0.80      1.00      0.89        71
    positive       0.00      0.00      0.00        18

    accuracy                           0.80        89
   macro avg       0.40      0.50      0.44        89
weighted avg       0.64      0.80      0.71        89

Random Forest Results:
Accuracy: 0.797752808988764
Classification Report:
              precision    recall  f1-score   support

    negative       0.80      1.00      0.89        71
    positive       0.00      0.00      0.00        18

    accuracy                           0.80        89
   macro avg       0.40      0.50      0.44        89
weighted avg       0.64      0.80      0.71        89
```

</td>
<td>Splits, trains and evaluates data using Random Forest and Logistic Regression for Sentiment Analysis</td>
</tr>
</table>

<pre>from sklearn.metrics import classification_report, accuracy_score, confusion_matrix from imblearn.over_sampling import SMOTE # Drop rows where 'Content' or 'Sentiment' contains NaN data.dropna(subset=['Content', 'Sentiment'], inplace=True) # Load data (assuming 'data' is already preprocessed as in earlier steps) X = data['Content'] y = data['Sentiment'] #TODO Smote data # Split dataset into training, validation, and testing sets (80-20 partition for train-test) X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Take 10% of the training data as validation set X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=42)</pre>		

```python
# TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_val_tfidf = tfidf_vectorizer.transform(X_val)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Logistic Regression Model
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train_tfidf, y_train)

# Random Forest Model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_tfidf, y_train)

# Model Evaluation: Logistic Regression
log_reg_pred = log_reg.predict(X_test_tfidf)
print("Logistic Regression Results:")
```

<pre>print("Accuracy:", accuracy_score(y_test, log_reg_pred)) print("Classification Report:\n", classification_report(y_test, log_reg_pred)) # Model Evaluation: Random Forest rf_pred = rf_model.predict(X_test_tfidf) print("Random Forest Results:") print("Accuracy:", accuracy_score(y_test, rf_pred)) print("Classification Report:\n", classification_report(y_test, rf_pred))</pre>		

Python

```python
# Apply SMOTE to balance the classes in the training set
smote = SMOTE(random_state=42)
smoteX_train_tfidf_resampled, smotey_train_resampled = smote.fit_resample(X_train_tfidf, y_train)
smoteX_test_tfidf_resampled, smotey_test_resampled = smote.fit_resample(X_test_tfidf, y_test)

# Logistic Regression Model for smoted
log_reg_smote = LogisticRegression(random_state=42)
log_reg_smote.fit(smoteX_train_tfidf_resampled, smotey_train_resampled)

# Random Forest Model for smoted
rf_model_smote = RandomForestClassifier(random_state=42)
rf_model_smote.fit(smoteX_train_tfidf_resampled, smotey_train_resampled)

# Model Evaluation: Logistic Regression for smote
```

```
Logistic Regression Results with smote:
Accuracy: 0.6197183098591549
Classification Report:
              precision    recall  f1-score   support

    negative       0.58      0.89      0.70        71
    positive       0.76      0.35      0.48        71

    accuracy                           0.62       142
   macro avg       0.67      0.62      0.59       142
weighted avg       0.67      0.62      0.59       142

Random Forest Results with smote:
Accuracy: 0.5422535211267606
Classification Report:
              precision    recall  f1-score   support

    negative       0.52      0.99      0.68        71
    positive       0.88      0.10      0.18        71

    accuracy                           0.54       142
   macro avg       0.70      0.54      0.43       142
weighted avg       0.70      0.54      0.43       142
```

Applies SMOTE to the training dataset and returns performance metrics for eval

<table>
<tr>
<td>

```python
log_reg_pred_smote = log_reg_smote.predict(smoteX_test_tfidf_resampled)
print("Logistic Regression Results with smote:")
print("Accuracy:", accuracy_score(smotey_test_resampled, log_reg_pred_smote))
print("Classification Report:\n", classification_report(smotey_test_resampled, log_reg_pred_smote))

# Model Evaluation: Random Forest for smote
rf_pred_smote = rf_model_smote.predict(smoteX_test_tfidf_resampled)
print("Random Forest Results with smote:")
print("Accuracy:", accuracy_score(smotey_test_resampled, rf_pred_smote))
print("Classification Report:\n", classification_report(smotey_test_resampled, rf_pred_smote))
```

</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Evaluates Random Forest and Logistic Regression models’ metrics on validation sets</td>
</tr>
</table>

Python `# Evaluate the Logistic Regression model on the validation set` `log_reg_val_pred = log_reg.predict(X_val_tfidf)` `print("Logistic Regression Results on Validation Set:")` `print("Accuracy:", accuracy_score(y_val, log_reg_val_pred))` `print("Classification Report:\n", classification_report(y_val, log_reg_val_pred))` `# Evaluate the Random Forest model on the validation set` `rf_val_pred = rf_model.predict(X_val_tfidf)` `print("Random Forest Results on Validation Set:")` `print("Accuracy:", accuracy_score(y_val, rf_val_pred))` `print("Classification Report:\n", classification_report(y_val, rf_val_pred))`		
		Evaluates Random Forest and Logistic Regression models' metrics on the validation set after SMOTE was applied

Python `# Evaluate the Logistic Regression model with smote on the validation set` `log_reg_val_pred = log_reg_smote.predict(X_val_tfidf)` `print("Logistic Regression Results on Validation Set:")` `print("Accuracy:", accuracy_score(y_val, log_reg_val_pred))` `print("Classification Report:\n", classification_report(y_val, log_reg_val_pred))` `# Evaluate the Random Forest model with smote on the validation set` `rf_val_pred = rf_model_smote.predict(X_val_tfidf)` `print("Random Forest Results on Validation Set:")` `print("Accuracy:", accuracy_score(y_val, rf_val_pred))` `print("Classification Report:\n", classification_report(y_val, rf_val_pred))`		

```
# Custom lexicon for Tagalog/Taglish
words and phrases
custom_lexicon = {
   'saya': 2.0,
   'ganda': 2.0,
   'astig': 1.5,
   'ayos': 1.5,
   'sarap': 2.0,
   'buti': 1.5,
   'magaling': 2.0,
   'panalo': 2.0,
   'tagumpay': 2.0,
   'bait': 1.8,
   'love': 2.0,
   'wow': 2.0,
   'cute': 1.8,
   'salamat': 1.5,
   'tama': 1.5,
   'mabait': 1.8,
   'maligaya': 2.0,
   'thank you': 2.0,
   'agree': 2.0,

   '😊': 2.0,
   '😂': 2.0,
   '😁': 2.0,
   '😎': 1.5,
   '❤️': 2.0,
   '😍': 2.5,
   '🥹': 2.0,
   '💯': 2.0,
   '👌': 2.0,
   '👍': 2.0,

   # Negative words
   'lungkot': -2.0,
   'galit': -2.0,
   'pangit': -2.0,
```

The following block of code defines a custom lexicon that assigns sentiment scores to frequently found words and symbols in the dataset. It is designed to improve the performance of the developed rule-based classifier, which lessens the reliance of the model to pre-existing libraries.

```
    'grabe': -1.0,
    'badtrip': -1.5,
    'bwisit': -2.0,
    'nakakainis': -2.0,
    'loko': -1.8,
    'tanga': -2.5,
    'bobo': -2.5,
    'kupal': -2.5,
    'loko-loko': -2.0,
    'problema': -2.0,
    'pagod': -1.5,
    'takot': -1.8,
    'bastos': -2.0,
    'masama': -2.0,
    'hassle': -1.5,
    'pathetic': -2.0,
    'sakit': -2.0,
    '?': -1.5,
    'pota': -2.0,
    'hayop': -2.0,
    'wtf': -1.0,

    '😡': -2.0,
    '😠': -2.0,
    '😢': -2.0,
    '😭': -2.5,
    '😒': -1.5,
    '🤬': -2.5,
    '🤣': -2.0,
    '😩': -2.0,



    # Neutral or context-dependent
    'edi': 0.0,
    'lang': 0.0,
    'kasi': 0.0,
    'diba': 0.0,
```

```
    'naman': 0.0,
    'pare': 0.0,

    # Common Taglish expressions (contextual adjustments might be needed)
    'chill lang': 1.5,
    'walang kwenta': -2.5,
    'wala lang': -1.0,
    'hay naku': -1.5,
    'same lang': 0.0,
    'sana all': 1.0,

    '😐': 0.0,
    '🤔': 0.0
}
```

```
# Rule-Based Model: Using VADER for Sentiment Analysis
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import numpy as np

# Initialize VADER sentiment analyzer
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()

# Add custom lexicon to VADER's existing lexicon
sid.lexicon.update(custom_lexicon)

# Function to classify sentiment based on VADER scores
def classify_sentiment_vader(text):
    score = sid.polarity_scores(text)
```

```
Accuracy: 0.3310657596371882
Classification Report:
              precision    recall  f1-score   support

    negative       0.93      0.27      0.42       367
     neutral       0.00      0.00      0.00         0
    positive       0.29      0.62      0.40        74

    accuracy                           0.33       441
   macro avg       0.41      0.30      0.27       441
weighted avg       0.82      0.33      0.42       441
```

```python
    compound = score['compound']
    if compound >= 0.05:
        return 'positive'
    elif compound <= -0.05:
        return 'negative'
    else:
        return 'neutral'

# Apply VADER rule-based classification
data['Vader_Sentiment'] =
data['Content'].apply(classify_sentimen
t_vader)

# Evaluate Rule-Based Model (comparing
with true 'Sentiment' values)
print("VADER Rule-Based Sentiment
Analysis Results:")
print("Accuracy:",
accuracy_score(data['Sentiment'],
data['Vader_Sentiment']))
print("Classification Report:\n",
classification_report(data['Sentiment']
, data['Vader_Sentiment']))
```

```python
# Evaluation Summary

import pandas as pd

# Results for Logistic Regression,
Random Forest, and VADER
evaluation_results = pd.DataFrame({
    'Model': ['Logistic Regression',
'Random Forest', 'VADER (Rule-Based)'],
    'Accuracy': [accuracy_score(y_test,
log_reg_pred), accuracy_score(y_test,
rf_pred),
```

```
Comparison of Model Performance:
                 Model  Accuracy  Precision    Recall  F1-Score
0  Logistic Regression  0.797753   0.636410  0.797753  0.708006
1        Random Forest  0.797753   0.636410  0.797753  0.708006
2   VADER (Rule-Based)  0.335601   0.821475  0.335601  0.423323
```

The following block of code shows a tabular summary of the performance of each classification model. Machine-based models, logistic regression and random forest, both performed well with an accuracy of 79.78%. Meanwhile, the rule-based classifier VADER had much lower accuracy with 33.56%, which may indicate that rule-based classifiers may still have limitation especially when it comes to identifying sentiments in a much

`` accuracy_score(data['Sentiment'], data['Vader_Sentiment'])], 'Precision': [classification_report(y_test, log_reg_pred, output_dict=True)['weighted avg']['precision'], classification_report(y_test, rf_pred, output_dict=True)['weighted avg']['precision'], classification_report(data['Sentiment'] , data['Vader_Sentiment'], output_dict=True)['weighted avg']['precision']], 'Recall': [classification_report(y_test, log_reg_pred, output_dict=True)['weighted avg']['recall'], classification_report(y_test, rf_pred, output_dict=True)['weighted avg']['recall'], classification_report(data['Sentiment'] , data['Vader_Sentiment'], output_dict=True)['weighted avg']['recall']], 'F1-Score': [classification_report(y_test, log_reg_pred, output_dict=True)['weighted avg']['f1-score'], classification_report(y_test, rf_pred, ```		more diverse corpus.

<table>
<tr>
<td>
<pre>
output_dict=True)['weighted
avg']['f1-score'],

classification_report(data['Sentiment']
, data['Vader_Sentiment'],
output_dict=True)['weighted
avg']['f1-score']]
})

print("Comparison of Model
Performance:")
print(evaluation_results)
</pre>
</td>
<td></td>
<td></td>
</tr>
</table>

Discussion of Results

Topic Modelling

For the LDA model:

- The LDA model used 3 topics in order to determine trends. Once the model was trained, it returned 3 topics with differing relevant words: Topic 1 had teacher, deped, education, curriculum, and students as its most relevant terms. Topic 2 had curriculum, matatag, school, education, and deped as the most relevant terms. Topic 3 had science, subject, curriculum, grade, and teacher as the most relevant terms. After determining the most relevant terms for the topics, the model was evaluated, which after computing, returned a coherence score of around 0.29. This score is relatively low, which means the topics may not be that well-defined, suggesting that further tuning may be required, in particular, the number of topics or the preprocessing.
- Interpretations of the topics
- Topic 1 - Deped Curriculum implementation for the teachers and Students
- Topic 2 - Matatag Curriculum to be implemented by deped for school education
- Topic 3 - Science not being included as a subject in the new curriculum for lower levels in the curriculum.

For the LSA model:

- The LSA model also made use of 3 topics for topic modeling. And like the LDA model, it also returned 3 clusters with differing relevant terms. The first cluster had: aabutin, aadjust, and aabot as the most relevant terms. The second cluster had aadjust, aabutin, and aabot as its most relevant terms in that order. The third cluster had the exact same relevant terms and order as the first cluster. The model was evaluated using two different performance metrics: the k-means silhouette score, which was computed to be around 0.48, and the explained variance ratio, with cluster 1 having a score of 0.015, cluster 2 having 0.016, and cluster 3 having 0.011. The k-means score suggests that though the clusters are reasonably separated, there may be some overlap between the clusters, which is shown by the relevant terms produced. The explained variance ratio has low scores, and even taking the cumulative variance into consideration, it only accounts for a small part of the dataset.

Text Classification

For the machine-based model:

- Two models were used for the machine-based model, namely the Random Forest and Logistic Regression models. The results they produced are varied, with both models having performance metrics for the dataset with and without applying SMOTE and being validated using a validation set. Logistic Regression had an overall accuracy of 0.75 for the results without using SMOTE, with weighted averages of 0.56 for precision, 0.75 for recall, and 0.64 for the F1-score. The model performed well in regard to negative cases, with a 0.75 precision score, 1.00 recall, and 0.86 F1-score. However, for the positive cases, it scored 0 for precision, recall, and F1-score, meaning that it failed to classify any positive sentiment. For Random Forest, the model had an accuracy of 0.78, with weighted averages of 0.83 for precision, 0.78 for recall, and 0.7 for the F1-score. The model performed well for negative cases, with 0.77 precision, 1.00 recall, and 0.87 F1-score. While Random Forest also struggles with positive cases, this model managed to correctly classify some positive sentiments, having a precision score of 1.00, a recall of 0.11 and an F1-score of 0.20
- After applying SMOTE, the Logistic Regression model has lower performance metrics, having an overall accuracy of only 0.64 with weighted averages of 0.62 for precision, a 0.64 recall, and 0.63 F1-score. The model, after applying SMOTE, had performed well for negative cases, with scores of 0.75 precision, 0.78 recal and 0.76 F1-score. Positive cases however, returned poor results, with 0.25 precision, 0.22 recall, and 0.24 F1-score. For Random Forest, the overall accuracy was around 0.75, with weighted averages of 0.7 precision, 0.75 recall, and 0.68 F1-score. Negative cases again performed well, having scores of 0.76 for precision, 0.96 for recall, and 0.85 for F1-score. Positive cases on the other hand had a precision score of 0.5, recall of 0.11, and F1-score of 0.18.
-

For the rule-based model:

- The rule-based model used VADER for sentiment analysis returned scores of 0.33 for the overall accuracy, with weighted averages of 0.82 for the precision, 0.33 for the recall, and 0.42 for the F1-score. For individual cases, the model performed best with negative cases, having scores of 0.93 for precision but only 0.27 for recall and 0.41 for the F1 score. Positive cases, meanwhile, had scores of 0.29 for precision, 0.62 for recall, and 0.4 for the F1 score. Neutral cases performed the worst, with the model failing to classify any sentiments as neutral, with scores for precision, recall and F1-score all being 0.

When comparing performances between the models Random Forest performed the best out of the three having an accuracy of around 0.75-0.78, with Logistic Regression also performing well, having 0.64-0.75 overall accuracy. VADER performed the worst out of the three, only having an overall accuracy of 0.33. All of the models performed well when detecting negative sentiments, but struggled a lot with positive and neutral cases. These results suggest that the machine-based models are better suited for tasks involving classifying more nuanced sentiments, such as with the current dataset used.The results however could be improved, perhaps with better tuning of the dataset as the comments are more biased towards negative sentiments, or by further tuning the preprocessing of the data.

Conclusion

To conclude our activity, this project gave us useful insights into public discussions about the K-12 Matatag Curriculum across social media platforms. By employing topic modeling techniques like Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA), we were able to identify significant topics such as changes in the curriculum, teacher experiences, and discussions around specific subjects. However, we found that the topics generated were not as clear as we expected, as seen in our model results, LDA produced a low coherence score, indicating that the topics may need further refinement. Similarly, the LSA model showed some overlap between clusters, suggesting a need for adjustments to improve clarity.

In the area of sentiment analysis, our machine learning models, Logistic Regression and Random Forest outperformed the rule-based VADER model, particularly in detecting negative sentiments. Nevertheless, all models struggled with accurately classifying positive and neutral sentiments, likely due to a dataset that was heavily skewed toward negative opinions. While the models effectively captured key trends and sentiments related to the curriculum, the quality and balance of the data limited their performance. Overall, the findings underscore the importance of better data preparation and model tuning. With a more balanced dataset and refined models, we could achieve more accurate sentiment analysis and a clearer understanding of public perceptions regarding the K-12 Matatag Curriculum.

Link to the Dataset:

https://docs.google.com/spreadsheets/d/1C3_730jEUdhO-RAOXKwNBcGW6ibrY4GO8nuci98kwqM/edit?gid=0#gid=0