ChatGPT

Refined Analysis Plan for r/gonewildaudio

1. Data Acquisition

1.1. Selecting an API

We will access Reddit data using a combination of the official Reddit API (via the Python **PRAW** library) and the Pushshift API. PRAW (Python Reddit API Wrapper) requires creating a Reddit developer app to obtain **client_id**, **client_secret**, and **user_agent** [1] . PRAW is a common tool for Python scraping of Reddit [2] , but it only retrieves *current* posts and enforces rate limits [3] . Pushshift, by contrast, is an open archive of Reddit posts/comments: it saves every post and comment with expanded metadata (enabling time-range queries) [3] . In effect, Pushshift “makes it much easier for researchers to query and retrieve historical Reddit data” by providing full-text search and large query limits [4] . Thus, for bulk retrieval over the past two years, using Pushshift (via a library like PSAW) is recommended.

1.2. Setting Up Access

First, register an app on Reddit (at https://www.reddit.com/prefs/apps) and record the **client_id**, **client_secret**, and **user_agent** (this setup is described in tutorials [1]). In Python we create a *read-only* PRAW instance like:

```
import praw
reddit = praw.Reddit(client_id='YOUR_ID', client_secret='YOUR_SECRET',
user_agent='my_agent')
```

No login (username/password) is needed since we only fetch public data [5] . For Pushshift, install the `psaw` package (`pip install psaw`) and initialize it:

```
from psaw import PushshiftAPI
api = PushshiftAPI()  # no auth needed
```

1.3. Fetching Posts (Last 24 Months)

Use the APIs to retrieve posts from **r/gonewildaudio** in the last 24 months. With PSAW, specify `after` and `before` timestamps. For example:

```
from datetime import datetime
import pandas as pd

start_ts = int(datetime(2023, 1, 1).timestamp())
```

```
end_ts    = int(datetime.now().timestamp())
results = api.search_submissions(subreddit='gonewildaudio',
                                 after=start_ts, before=end_ts,

filter=['id','title','score','num_comments','upvote_ratio','author','created_utc'])
df = pd.DataFrame([r.d_ for r in results])
```

This collects matching posts into a DataFrame [6]. If using PRAW instead, one might loop:

```
for post in reddit.subreddit("gonewildaudio").new(limit=None):
    # collect post.score, post.num_comments, etc.
```

However, PRAW is limited to the most recent ~1000 posts per query [3], so Pushshift is safer for retrieving all posts in a date range. From each post we will record metadata fields such as **id**, **title**, **score** (upvotes), **num_comments**, **upvote_ratio**, **created_utc**, **author**, etc. (for example, a Pandas data frame of top posts in a subreddit can include these fields [7]).

1.4. Collecting Comments (Optional)

Optionally, we can also fetch comments under each post to analyze audience feedback. With PRAW:

```
submission = reddit.submission(id=post_id)
submission.comments.replace_more(limit=None)
comments = [c.body for c in submission.comments.list()]
```

This yields all comment texts for sentiment or qualitative analysis. Pushshift also provides a comment search endpoint if needed. (For a pilot study, we may focus on post-level metrics first and add comment analysis later if useful.)

2. Data Preprocessing

2.1. Filtering Male-Performer Posts

Filter the collected posts to keep only those featuring a male performer. Practically, check the title tags and include posts whose titles contain `[M]`, `[M4F]`, `[M4M]`, `[M4A]`, etc. (match case-insensitively or normalize brackets). This yields a subset of posts where the performer is male (regardless of target audience).

2.2. Tag Extraction and Normalization

Each retained post's title contains multiple bracketed tags. We will parse out all tags by extracting text between `[` and `]`. Then convert tags to a standardized format: - Lowercase all tags.
- Remove extra whitespace or punctuation.
- Merge obvious synonyms or variants (e.g. treat "kissing" and "lots of kisses" as one category; unify "DDLG" vs "DD/lg" to "ddlg").

We may start with a manual mapping for common tags. For more automation, one could embed tag strings (using word embeddings or sentence transformers) and cluster similar tags, but the results should be checked manually because these tags are often explicit slang. The goal is to reduce fragmentation. After normalization, create binary indicator features for each tag (or tag category) present in a post.

2.3. Additional Features

Include numeric features from each post's metadata: - **score** (upvote count) and **num_comments**.
- **upvote_ratio** (if informative).
- **age** (time elapsed since creation).
From these, we can derive rates (e.g. upvotes per day). Optionally, apply NLP on the post text: e.g. run a sentiment analyzer (NLTK's VADER or a transformer model) on the script description to get a positivity or intensity score. If needed, an LLM can **summarize** the post description to capture the scenario (though this step is advanced and may require careful handling of explicit content). All these will be assembled into our final analysis DataFrame (one row per post, with tag columns and numeric metrics).

3. Defining Success Metrics

We must define what makes a post "successful." Possible metrics include: - **Score (upvote count)**, reflecting popularity.
- **Number of comments**, indicating engagement.
- **Upvote ratio**, indicating general approval.
- **Composite index**: e.g. `0.7*score + 0.3*num_comments` (as the user suggested) to combine both.
- **Percentile threshold**: e.g. mark the top 10% (or 25%) of posts by score as "popular."

We will experiment with these. For example, we might label the top 20% of posts by score as high-engagement and analyze their tags. We should report which definition we use (or if multiple are explored). For robustness, we can check if trends hold under different metrics.

4. Exploratory Data Analysis (EDA)

With success labels in place, we perform EDA to uncover patterns:

- **Tag Frequency vs Success**: Plot bar charts of tag frequencies in the full male-posts dataset vs in the "successful" subset. This shows which tags/themes are over-represented among high-engagement posts.
- **Numeric Distributions**: Use histograms or boxplots for score and comments. For example, compare the distribution of scores for M4F vs M4M posts. Overlay or facet by success label to see differences.
- **Scatter Plots**: Plot score vs comments (and/or vs upvote_ratio) for all posts, coloring points by success label. This can reveal clusters or correlations.
- **Author/Creator Analysis**: Aggregate by Reddit username. Compute how many posts each male creator made and their average score. A bar chart or scatter (posts vs avg score) will identify prolific or consistently high-scoring creators.
- **Tag Co-occurrence**: Possibly a heatmap of tag co-occurrence in popular posts.
- **Temporal Trends**: If relevant, plot posting frequency or average score over time (e.g. monthly) to see growth or seasonality.

All charts will be made with Python's Matplotlib/Seaborn for clarity [7]. Each should have clear labels and a legend. For instance, a bar chart of "Top 10 tags in popular posts vs all posts" or a scatter of "Score vs Comments (successful in one color)". We will interpret any notable patterns (e.g. "tag X appears much more frequently in the top posts").

5. Advanced Analysis (Optional)

If simple EDA suggests interesting signals, we can apply more advanced techniques:

- **Statistical Tests**: e.g. Chi-square tests on tag vs success (categorical), or ANOVA/T-tests on score differences. We must adjust for multiple comparisons.
- **Machine Learning**:
- *Clustering*: Cluster posts by their tag features (k-means or hierarchical). See if clusters correspond to content types and if some clusters have higher average engagement.
- *Regression/Classification*: Train a model (e.g. linear regression or random forest) to predict post score or a binary popular/not based on features (tags, sentiment, etc.). Feature importance can highlight which tags or factors most influence success.
- *Topic Modeling*: Run LDA or a transformer-based topic model on the post text to identify common themes, and see if certain topics align with success.

We would use libraries like scikit-learn and gensim for these. Given the author's beginner level, we would rely on the AI assistant to generate code for these steps. Any findings should be validated (e.g. cross-validation for models, looking at p-values) and explained in plain terms ("Posts involving X have significantly higher scores").

6. Implementation and Tools

We will implement the analysis in **Python** within a Jupyter Notebook (or Google Colab) [8]. Key libraries: - **praw** – Reddit API wrapper [2].
- **psaw** – Pushshift API wrapper (for historical data).
- **pandas**, **NumPy** – data handling.
- **matplotlib**, **seaborn** – plotting.
- **scikit-learn**, **statsmodels** – statistical tests and ML.
- **NLTK/TextBlob/transformers** – NLP (tokenization, sentiment, summarization).

A sample workflow in the notebook:
1. **Authenticate and fetch data** (using PRAW/PSAW code as above).
2. **Clean and filter** the data (keep male posts, extract tags, normalize).
3. **Create features and label** "success" according to chosen metric.
4. **EDA visualizations** (bar charts, histograms, scatterplots).
5. **(Optional)** Statistical tests or ML modeling.

We'll interleave code cells with markdown explanations. At each stage, we'll verify results (e.g. print a few rows of the dataframe) to ensure correctness. If using Google Colab, we only need to install the necessary packages; if local, `pip install praw psaw pandas seaborn sklearn nltk` etc.

7. Ethical and Practical Notes

This analysis involves adult content. We will not process or display any explicit sexual material; only metadata, tags, and user-provided text. When using any language model or text analysis, be aware that general models may censor erotic content. If needed, one could use an open-source model (like GPT-2/GPT-J or fine-tuned LLaMA) or pre-process the text to remove or mask particularly graphic terms. We will anonymize any personal data – we use only public Reddit usernames (if at all) and will not expose any private information.

Conclusion

This roadmap turns the draft into a concrete plan: collect two years of gonewildaudio posts (filtering for male performers), parse tags and metrics into a dataframe, define popularity metrics, and then analyze patterns via charts and possible modeling. It emphasizes practical steps (API setup, code outline) and beginner-friendly methods (bar charts, pivot tables) while allowing for deeper analysis if warranted. Throughout, we provide references on Reddit data scraping and EDA best practices [1] [4] [7] to guide the coding implementation.

References: Key resources include guides on Reddit API usage [1] [6] and the use of Pushshift for historical Reddit data [4] [3], as well as tutorials on data analysis in Jupyter [8] [7], which informed this plan. These sources demonstrate how to authenticate, fetch Reddit data, and build dataframes for analysis.

---

[1] [2] [5] Scraping Reddit using Python - GeeksforGeeks
https://www.geeksforgeeks.org/python/scraping-reddit-using-python/

[3] Collecting Data from Reddit. With over 430 million active users per... | by Noelle Ferrari | Medium
https://noelleferrari.medium.com/collecting-data-from-reddit-e019d1dece6a

[4] [6] Reddit Data Collection and Analysis with PSAW — Introduction to Cultural Analytics & Python
https://melaniewalsh.github.io/Intro-Cultural-Analytics/04-Data-Collection/14-Reddit-Data.html

[7] [8] Reddit Web Scraping & Data Analysis Tutorial | by Ben Ward | Medium
https://medium.com/@BW.benward/reddit-web-scraping-data-analysis-walkthrough-bd8345f7fae3