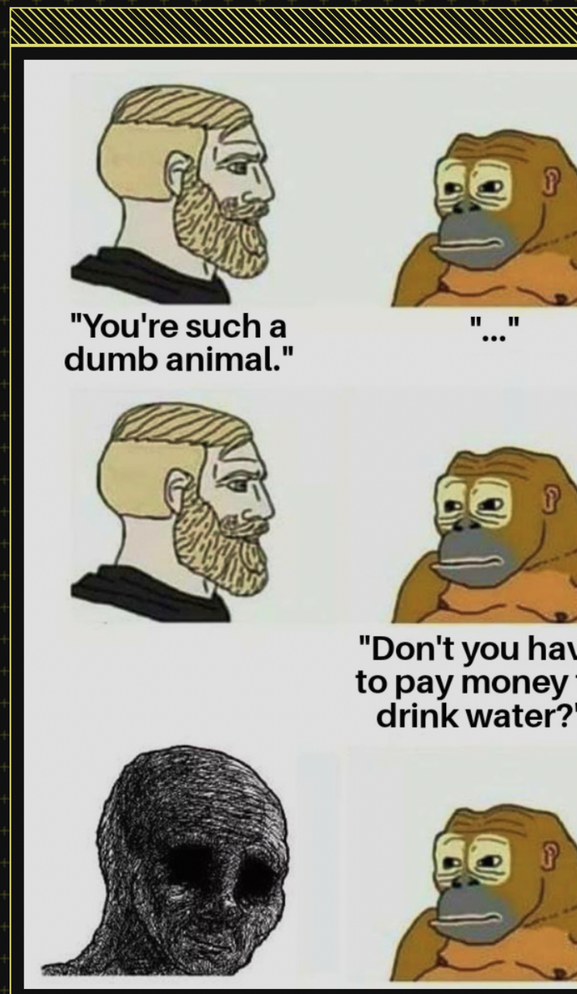
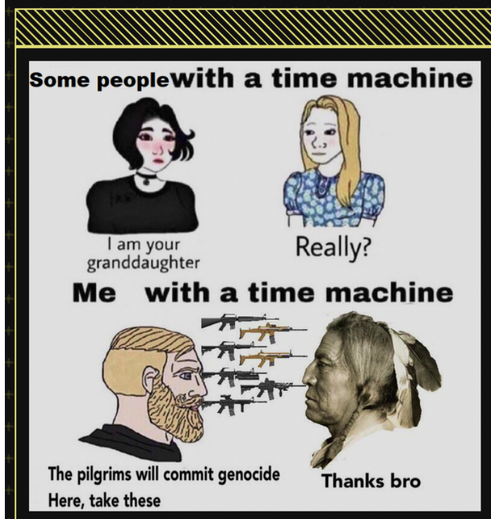
captious memes

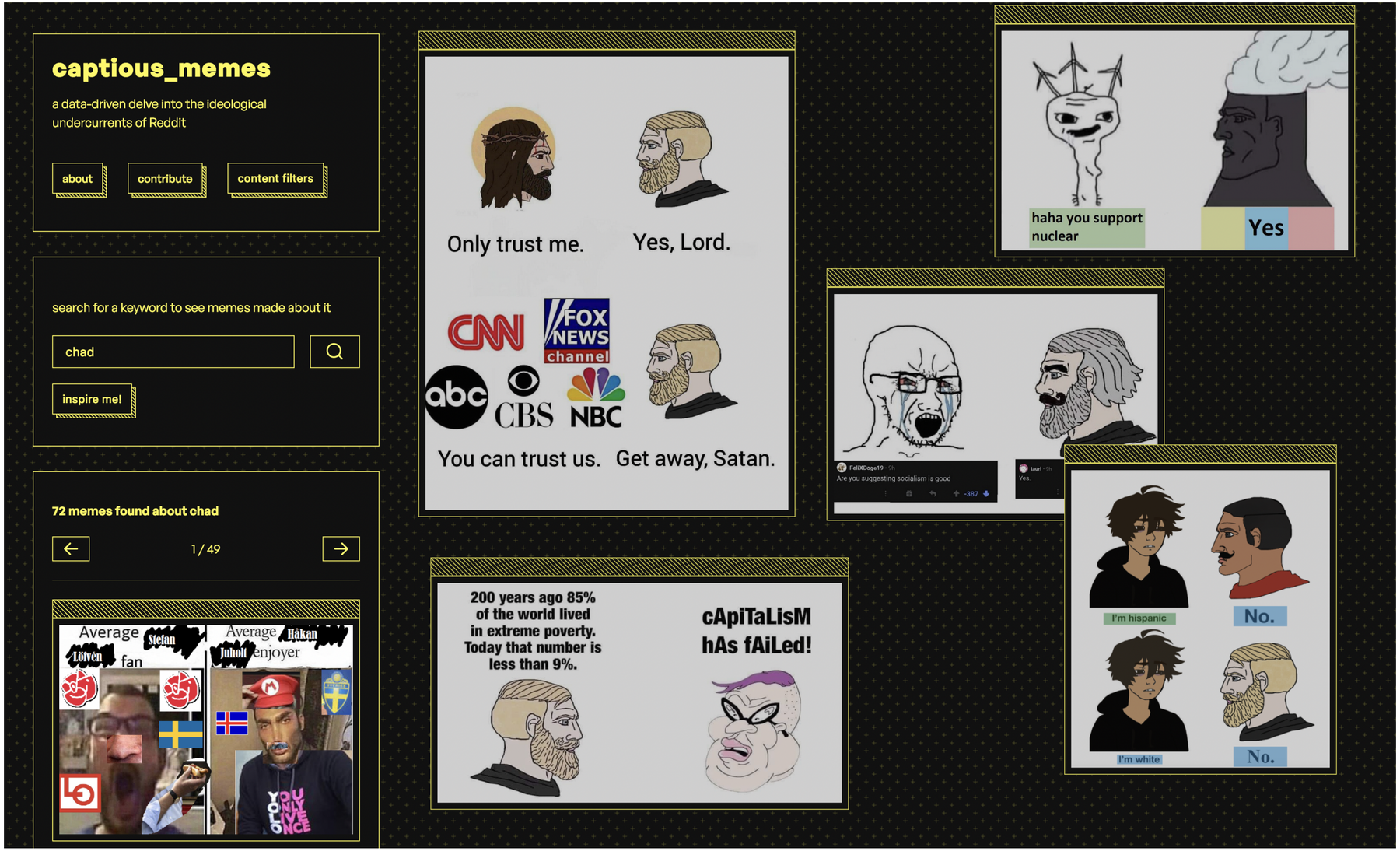
https://captious-memes.netlify.app

python scripts for scraping,
data collection and processing





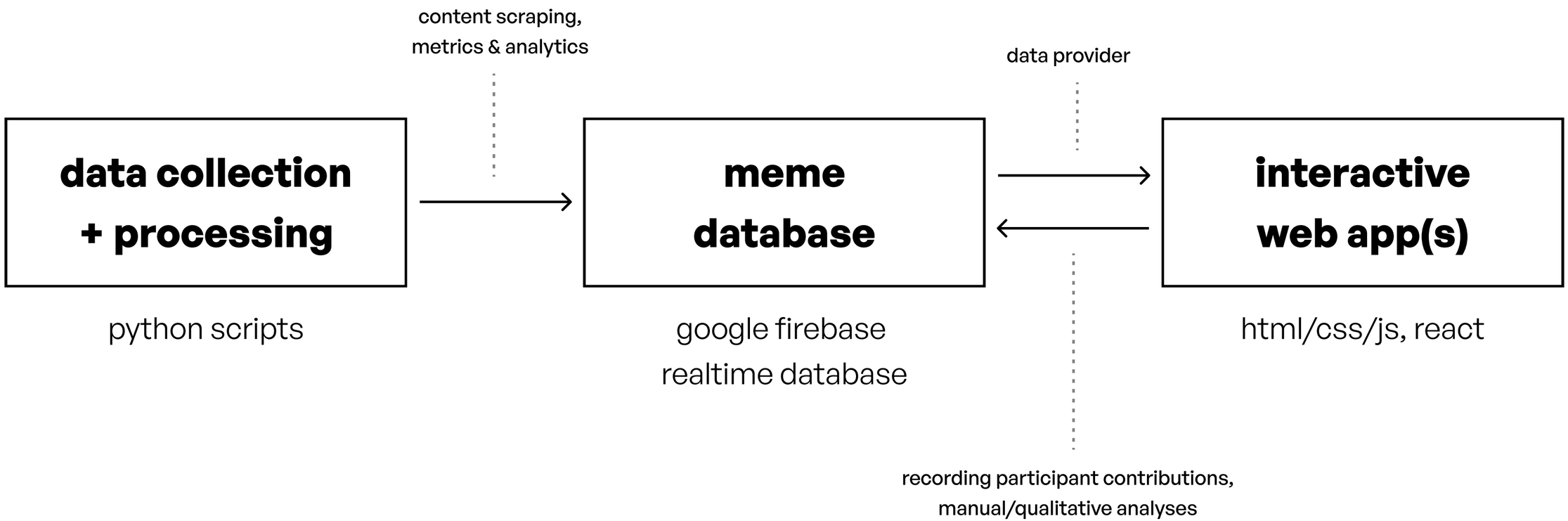
ege uz



a data-driven artistic research process + tool that delves into the ideological undercurrents of memetic media ecosystems (particularly my own)

in-depth computational + qualitative analysis of memes
presented in an open space for free association and juxtaposition

project structure



python scripts

1. reddit-meme-scraper
2. seed-initial-database
3. create-subreddit-index

reddit-meme-scraper

objective: create initial corpus in database
libraries used: praw, pandas (special thanks to re & time)

step 1:
create a list of subreddits to scrape from

filtering through a list of subreddits with +3 million entries, looking for meme pages

initial sample of 205 subreddits

```
#Creating a list of subreddits to sample from
subreddits = pd.read_csv('subreddits-2021-07-23.csv')
# we'll filter based on: "meme" in description/title + min. 10k subs
# then manually filter to remove pages about pop culture/intellectual products,
meme pages + political pages
meme_subs = subreddits[
    (subreddits['subs'] > 10000) &
    (
        subreddits['desc'].str.contains('meme') |
        subreddits['real_name'].str.contains('meme')
    )
]
#save the filtered dataset here, then review it
meme_subs.to_csv('meme-subreddits-2021.csv')
```

Press ^Enter to execute cell

Scraping images from subreddits

```
# initialize our reddit client
reddit = praw.Reddit(
    client_id=[redacted]
    client_secret=[redacted]
    user_agent=[redacted]
    username=[redacted]
    password=[redacted]
)
reddit.read_only = True
```

```
#now, collect images
subreddit_index = pd.read_csv('subreddit-index.csv')
subreddits = subreddit_index["real_name"]
```

reddit-meme-scraper

step 2:

using praw to scrape images from target subreddits, as well as basic information about them

collecting the top 1000 posts between 2020-2021 in each target subreddit

130,000 memes collected

took 150+ minutes to execute

```python
# mass post scraping method
def scrape_subreddits(subreddits):
    posts = []
    i = 0
    for sub in subreddits:
        sub_posts = scrape_posts_from_subreddit(sub)
        for post in sub_posts:
            posts.append(post)
        pd.DataFrame(posts).to_csv('scraped-posts-bulk.csv')
        time.sleep(15)
    return posts


def scrape_posts_from_subreddit(sub_name):
    posts = []
    try:
        query = reddit.subreddit(sub_name).top(time_filter="year", limit=None)
        for submission in query:
            submission_is_image = re.search(
                '([^\\s]+(\\.(?i)(jpe?g|png|gif|bmp|webp))$)', submission.url)
            if submission_is_image:
                post_data = {
                    "reddit_id": submission.id,
                    "image_url": submission.url,
                    "title": submission.title,
                    "author": submission.author,
                    "subreddit": sub_name,
                    "permalink": "https://reddit.com/r/" + sub_name + "/comments/" + submission.id,
                    "num_comments": submission.num_comments,
                    "num_upvotes": submission.score,
                    "upvote_ratio": submission.upvote_ratio
                }
                posts.append(post_data)
    except:
        print("an error occured :(")
    finally:
        return posts
```

Python

```python
# execute mass scrape
# WARNING: takes a while (approx. 2.5hrs) to finish running
posts = scrape_subreddits(subreddits)
```

Python

```python
raw_posts = pd.read_csv('scraped-posts-bulk.csv')
shuffled_posts = raw_posts.sample(frac=1)
shuffled_posts.to_csv('raw-posts.csv')
```

Python

create-subreddit-index

objective: create an index of contextual data about subreddits I've scraped memes from and save it to the database
libraries used: pandas, firebase_admin

```
subreddit_index = pd.read_csv('subreddit-index-proto.csv')
subreddits = subreddit_index["real_name"]
bulk_posts = pd.read_csv('scraped_posts_bulk.csv')
```

[3]

```
def generate_subreddit_index(subreddits):
    i = 0
    l = len(subreddits)
    index = []
    for sub_name in subreddits:
        time.sleep(10)
        index.append(
            analyze_subreddit(sub_name)
        )
        print(sub_name + " analyzed successfully [" + str(i) + "/" + str(l) + "]")
        i += 1

    return index


def analyze_subreddit(sub_name):
    # request information about subreddit
    try:
        subdata = reddit.subreddit(sub_name)
        sampled_posts = bulk_posts[bulk_posts["subreddit"] == sub_name]
        sub_data = {
            "name": sub_name,
            "permalink": "https://reddit.com/r/" + sub_name,
            "subscribers": subdata.subscribers,
            "posts_analyzed": len(sampled_posts),
            "upvote_data": get_upvote_data(sampled_posts),
            "comment_data": get_comment_data(sampled_posts),
            "content_creator_diversity_ratio": get_ccdr(sampled_posts)
        }
        return sub_data
    except:
        print(sub_name + ": sub not found :(")
        return { "error_message": "subreddit not found" }
```

create-subreddit-index

objective: create an index of contextual data about subreddits I've scraped memes from and save it to the database
libraries used: pandas, firebase_admin

```
subreddit_index = pd.read_csv('subreddit-index-proto.csv')
subreddits = subreddit_index["real_name"]
bulk_posts = pd.read_csv('scraped_posts_bulk.csv')
```

[3]

```
def generate_subreddit_index(subreddits):
    i = 0
    l = len(subreddits)
    index = []
    for sub_name in subreddits:
        time.sleep(10)
        index.append(
            analyze_subreddit(sub_name)
        )
        print(sub_name + " analyzed successfully [" + str(i) + "/" + str(l) + "]")
        i += 1

    return index


def analyze_subreddit(sub_name):
    # request information about subreddit
    try:
        subdata = reddit.subreddit(sub_name)
        sampled_posts = bulk_posts[bulk_posts["subreddit"] == sub_name]
        sub_data = {
            "name": sub_name,
            "permalink": "https://reddit.com/r/" + sub_name,
            "subscribers": subdata.subscribers,
            "posts_analyzed": len(sampled_posts),
            "upvote_data": get_upvote_data(sampled_posts),
            "comment_data": get_comment_data(sampled_posts),
            "content_creator_diversity_ratio": get_ccdr(sampled_posts)
        }
        return sub_data
    except:
        print(sub_name + ": sub not found :(")
        return { "error_message": "subreddit not found" }
```

create-subreddit-index

```
def get_upvote_data(post_data):
    upvote_scores = np.array(post_data["num_upvotes"])
    return {
        "avg_upvotes": float(upvote_scores.mean()),
        "min_upvotes": float(upvote_scores.min()),
        "max_upvotes": float(upvote_scores.max()),
        "std": float(upvote_scores.std())
    }


def get_comment_data(post_data):
    comment_scores = np.array(post_data["num_comments"])
    return {
        "avg_comments": float(comment_scores.mean()),
        "std": float(comment_scores.std()),
        "min_comments": float(comment_scores.min()),
        "max_comments": float(comment_scores.max()),
    }


def get_ccdr(post_data):
    authors = post_data["author"]
    num_posts = len(post_data)
    num_unique_authors = len(set(authors))
    return num_unique_authors / num_posts
```

[7] Press ^Enter to execute cell

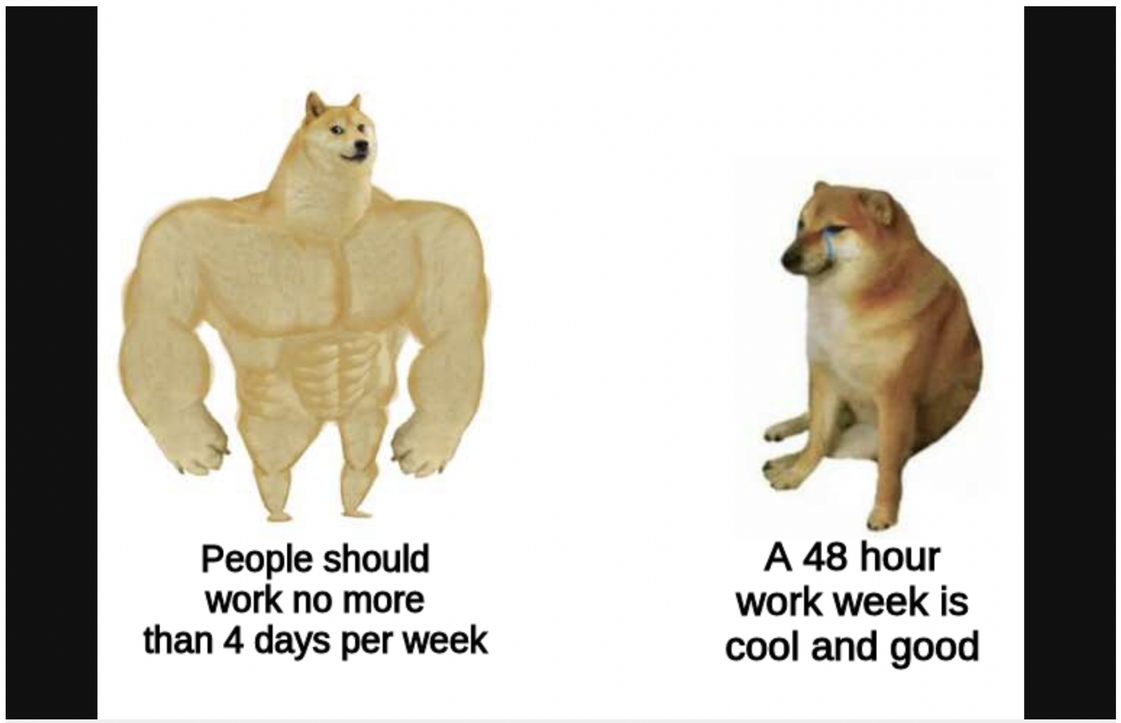
```
processed_index = generate_subreddit_index(subreddits)
```

[ ]

```
for sub_data in processed_index:
  try:
    sub_name = sub_data['name']
    db.reference('/subreddit_index_2/' + sub_name).set(sub_data)
  except:
    print('skipping this broken datapoint')
```

[11]

javascript intermission



Labour under Jez vs Labour under Keith

by toomuchgammon @ r/GreenAndPleasant
34 comments • 868 upvotes (0.99 upvote ratio)

<- prev | next -> | 11 / 122 | 11 | go >>

textual analysis

which memetic symbols are present in the meme? what do they represent? how do they characterize the thing they represent? what kinds of arguments are made through the symbol?

symbol	represents	adjectives	arguments
swole doge	labour party under jeremy corbyn		people should not work more than 4
cheems	labour party under keith starmer		a 48 hour (6 day) work week is cool and good

add another row

meta-textual analysis

check common memetic characteristics that can be found in this meme:

- [ ] shitpost
- [ ] troll post
- [ ] callout
- [ ] mockery
- [ ] ironic
- [ ] surreal
- [ ] meta
- [ ] edgy

save and continue

filtering through and labeling data

seed-initial-database

objective: consolidate and process filtered/analyzed memes and restructure them to be accessed by the web app

libraries used: pandas, firebase_admin, google cloud vision ai

step 1:
retrieve analyzed meme list (messy) and

```
#import reviewed images
image_data = db.reference('/reviewed-images').get()
```

[3]

prepare list of final images relevant images

```
# prepare list of final images filtered to be relevant
def filter_relevant_images(image_df):

    image_dict = image_df["labels"].to_dict()
    relevant_images = []

    for i, v in image_dict.items():
        try:
            if (v["relevance"] == "relevant"):
                row = image_df.iloc[i].to_dict()
                relevant_images.append(row)
        except TypeError:
            print("no labels found")

    return relevant_images
```

[4]

```
image_df = pd.DataFrame(list(image_data.values()))
relevant_images = filter_relevant_images(image_df)
```

[5]

seed-initial-database

step 2:
transcribe images using google vision ai
(ocr text recognition)

```
def transcribe_images(images):

  img_db = images.copy()

  for img_data in img_db:
    url = img_data['image_url']
    img_data['image_transcript'] = transcribe_image(url)
  return img_db



def transcribe_image(url):
  try:
    response = transcript_client.annotate_image({
      'image': {'source': {'image_uri': url}},
      'features': [{'type_': vision.Feature.Type.TEXT_DETECTION}]
    })

    transcript = response.text_annotations[0].description
    return transcript

  except:
    msg = "no transcript"
    return msg


[7]


image_dict_transcripts = transcribe_images(relevant_images)
[8]
```

seed-initial-database

step 3:

clean up and restructure meme datapoints

```
# flatten image labels
def flatten_image_labels(image_data):
  image_data_with_flat_labels = [];
  for post_data in image_data:
    pd = post_data.copy()
    labels = pd['labels']
    try:
      pd['symbols'] = labels['characters']
    except:
      pd['symbols'] = []
    try:
      pd['content_warning_tags'] = labels['content_warning_tags']
    except:
      pd['content_warning_tags'] = ""
    try:
      pd['meta_tags'] = labels['meta_tags']
    except:
      pd['meta_tags'] = {}
    image_data_with_flat_labels.append(pd)
  return image_data_with_flat_labels
```

[11] Press ^Enter to execute cell

```
image_dict_flat_labels = flatten_image_labels(image_dict_transcripts)
```

[12]

```
#clean up data so far and get it ready for final processing + assembling
idf = pd.DataFrame(image_dict_flat_labels)
idf.pop('idx')
idf.pop('idx_2')
idf.pop('labels')
idf.pop('category')
idf['post_author'] = idf['author']
idf.pop('author')
idf['post_title'] = idf['title']
idf.pop('title')
idf['post_id'] = idf['reddit_id']
idf.pop('reddit_id')
idf['post_url'] = idf['permalink']
idf.pop('permalink')
idf['post_num_comments'] = idf['num_comments']
idf['post_num_upvotes'] = idf['num_upvotes']
idf['post_upvote_ratio'] = idf['upvote_ratio']
idf.pop('num_comments')
idf.pop('num_upvotes')
idf.pop('upvote_ratio')
```

[13]

seed-initial-database

step 4:

add additional datapoints that are
derived from contextual subreddit data
related to the meme (from the subreddit index)

```
def remap(n, start1, stop1, start2, stop2):
  return ((n-start1)/(stop1-start1))*(stop2-start2)+start2

def map_popularity_score(score, avg_score, min_score, max_score):
  score = float(score)
  pop_score = 0
  if (score < avg_score):
    pop_score = remap(score, min_score, avg_score, -1, 0)
  elif (score > avg_score):
    pop_score = remap(score, avg_score, max_score, 0, 1)
  return pop_score
```

[48]

```
# get post popularity scores
subreddit_index = pd.read_json('./subreddit-index-final.json')

sources = idf['subreddit']
num_upvotes = idf['post_num_upvotes']
pop_scores = []
ccdvs = []

i = 0

for score in num_upvotes:
    source = sources[i]
    source_data = subreddit_index[subreddit_index["name"] == source]
    upvote_data = list(source_data['upvote_data'].to_dict().values())[0]
    popularity_score = map_popularity_score(
        score,
        upvote_data['avg_upvotes'],
        upvote_data['min_upvotes'],
        upvote_data['max_upvotes']
    )
    ccdv = source_data['content_creator_diversity_ratio']
    pop_scores.append(popularity_score)
    ccdvs.append(ccdv)
    i += 1
```

[51] Press ^Enter to execute cell

```
idf['post_popularity_score'] = pop_scores
```

[59]

https://captious-memes.netlify.app

https://memescope.netlify.app