Exploratory Data Analysis of Farcaster Users

September 5, 2024

[1]:
```
from dotenv import load_dotenv
load_dotenv()
```

[1]: True

[2]:
```
import pandas
import pandas_gbq
import os
import numpy
import json
from collections import Counter
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from google.cloud import storage, bigquery
```

1 Load data from BigQuery

[3]:
```
project_id = os.environ['GCP_PROJECT_ID']
```

[4]:
```
sql = """
SELECT * FROM dsart_farcaster.fid_features where msg_messages>0 order by fid
"""
```

[5]:
```
df = pandas_gbq.read_gbq(sql, project_id=project_id, progress_bar_type=None)
```

[6]:
```
df[['fid', 'user_name', 'msg_num_days', 'msg_messages']]
```

[6]:

	fid	user_name	msg_num_days	msg_messages
0	2	v	29	811
1	3	dwr.eth	30	6509
2	5	mircea.eth	1	1
3	8	jacob	30	389
4	9	b-rad	9	25
…	…	…	…	…
245513	850352	interhev	1	11
245514	850353	clarenceaguilar	1	21
245515	850354	sonicmoonfang	1	1

```
245516   850355          maki1007              1             4
245517   850356            xtian1              1             4

[245518 rows x 4 columns]
```

2 Sample Size

[7]:
```python
len(df), len(df['fid'].unique()), len(df['user_name'].unique())
```

[7]:
```
(245518, 245518, 227073)
```

The dataset is extracted on Sept/5 2024, and has 245k unique fids who have submitted an activity during the last 30 days (cast, reply reaction or follow)

The last fid registered on Farcaster at this time is at 850k, so that's about 29% of all registered users.

3 Messages

msg_ features aggregate the messages submitted by the user during the last 30 days.

[8]:
```python
df[['msg_num_days', 'msg_messages', 'msg_messages_per_day']].describe().round()
```

[8]:

	msg_num_days	msg_messages	msg_messages_per_day
count	245518.0	245518.0	245518.0
mean	8.0	291.0	18.0
std	9.0	2715.0	108.0
min	1.0	1.0	1.0
25%	1.0	4.0	2.0
50%	3.0	15.0	4.0
75%	10.0	64.0	9.0
max	30.0	662375.0	22079.0

- 50% of users active over last 30 days were actually active only on 1 to 3 distinct days.
- The median number of messages submitted over 30 days is 15; but the average is much higher at 290, because it is pulled up by bots and spammy users who submit a very high number.

3.1 Top 10 message submitters

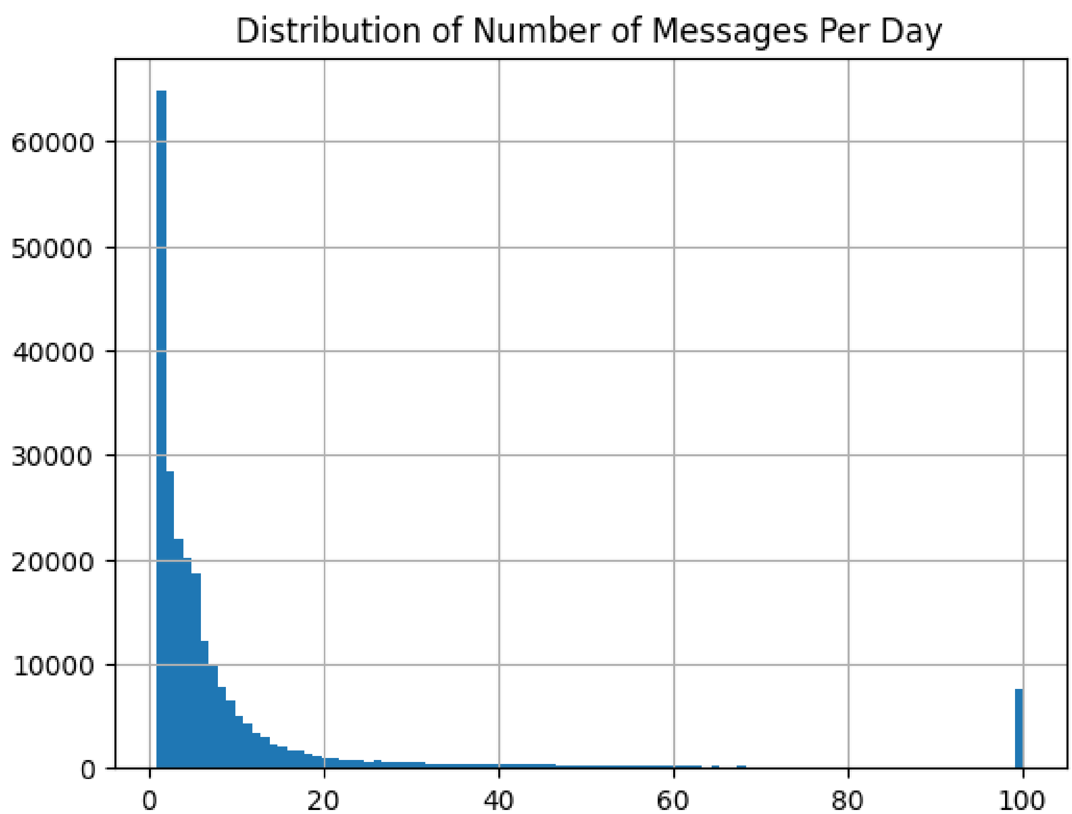
[9]:
```python
df.sort_values('msg_messages', ascending=False)[:10][['user_name',
 ↪'msg_num_days', 'msg_messages', 'msg_messages_per_day']].round()
```

[9]:

	user_name	msg_num_days	msg_messages	msg_messages_per_day
210149	masks-tipper	30	662375	22079.0
48185	degentipbot.eth	30	540788	18026.0
200674	None	30	311911	10397.0
59170	floatybot	30	235901	7863.0

41218	automod	30	168951	5632.0
65192	3dit	30	164037	5468.0
67620	3d1t	30	162729	5424.0
40188	l3mbda	30	112707	3757.0
68081	sandalwood	30	90174	3006.0
43497	qqsksk12	30	85395	2846.0

[10]:
```
_ = df['msg_messages_per_day'].clip(0,100).hist(bins=100)
_ = plt.title("Distribution of Number of Messages Per Day")
```



The distribution of messages/day metric is interesting:

- Most of them are packed in the left, with less than 15/day.
- On the right, there's a pack of about 4% spammy accounts with +100/day.

4 Spam

The **spam** features are a way to flag spammy accounts based on their number of messages per day, their ratio of deleted messages, or the speed at which they reply or react to others.

The thresholds were automatically chosen to flag the bottom or top 5% after filtering for num_messages>10 and num_active_days>1.

These thresholds are:

- Submits more than 93 messages per day.
- Deletes more than 7.4% of their actions.
- Replies or reacts to others in less than 83 minutes on average.

While there's no direct relationship between these flags and the fact that an account is a bot; these provide a good proxy to identify spammy behavior in general, which is probably correlated with bots.

Even if not, one could argue that bots that don't post more than a reasonable number of messages per day, don't delete often, and don't react too fast are somehow behaving properly, and therefore don't cause any harm to the platform.

So while these flags don't measure the real percentage of bots or spam in the platform, they do provide a good proxy to "toxic" accounts in general, and a directional read. For example, if one observes a higher rate of spammy accounts in a channel vs another; that's a sign that the latter is cleaner than the former.

We can also look at how the Farcaster volume evolved over time and when more spammy users got acquired.

[11]:
```
df['spam_features'] = df['spam_messages_per_day'] + df['spam_deletes'] +␣
↪df['spam_speed']
df['spam_any'] = (df['spam_features']>0).astype(int)
```

[12]:
```
df[['spam_messages_per_day', 'spam_deletes', 'spam_speed', 'spam_any']].
↪describe().round(3)
```

[12]:

	spam_messages_per_day	spam_deletes	spam_speed	spam_any
count	245518.0	245518.0	245518.0	245518.000
mean	0.027	0.027	0.025	0.073
std	0.161	0.161	0.156	0.261
min	0.0	0.0	0.0	0.000
25%	0.0	0.0	0.0	0.000
50%	0.0	0.0	0.0	0.000
75%	0.0	0.0	0.0	0.000
max	1.0	1.0	1.0	1.000

[13]:
```
df['spam_features'].value_counts()
```

[13]:
```
spam_features
0    227535
1     16704
2      1264
3        15
Name: count, dtype: Int64
```

[14]:
```python
print(f"{100 * (df['spam_features']>0).sum() / len(df):.2f}%")
```

```
7.32%
```

Overall, 7.32% of the dataset is flagged as spammy by at least one of the 3 criteria.

Now, let's look at different cohorts and compare with this number.

[15]:
```python
df['cohort'] = df['first_cast'].astype(str).str.slice(0,7)
df.loc[df['first_cast']<'2024', 'cohort'] = 'OG'
```

[16]:
```python
df[(df['first_cast']<'2024-08') | (df['first_cast']=='OG')].groupby('cohort').
↪agg({'spam_any': 'mean'})
```

[16]:

cohort	spam_any
2024-01	0.058512
2024-02	0.073962
2024-03	0.132943
2024-04	0.154744
2024-05	0.087145
2024-06	0.057879
2024-07	0.059263
OG	0.081782

Interestingly, the users onboarded during March and April have a +2x to 3x higher level of spaminess, which seemed to start calming down in May.

(Note that we excluded August cohort because they have been around less, so it won't be comparing apples to apples.)

[17]:
```python
df['no_user_name'] = df['user_name'].isnull()
```

[18]:
```python
df.groupby('no_user_name').agg({'spam_any': 'mean'})
```

[18]:

no_user_name	spam_any
False	0.074439
True	0.058552

Was just curious to see if there was a significant difference between users with vs without usernames. But it doesn't seem so.

We can also look at the spaminess levels by user language...

[19]:
```python
df.groupby('lang_1').agg({
    'fid': ['count'],
    'msg_messages_per_day': 'mean'
}).sort_values(('fid','count'), ascending=False)[:10]
```

[19]:
```
            fid msg_messages_per_day
          count                 mean
lang_1
en       179794            15.766310
tl        11469             8.316160
zh-cn      3291             4.142851
vi         3030           323.315423
nl         1406             4.534270
ja         1149            15.288979
ko          753            98.708065
es          630            17.534022
ru          447            26.400362
af          330            12.945200
```

Wow, that was unexpected!

While the average number of messages per day is around 15 in general, turns out the Vietnamese and Korean language users have 323 and 98.

Not sure how to interpret this.

Maybe they're simply smaller communities who actually use Farcaster more? Or maybe there's a bot or farming causing this data.

But definitely something intriguing.

What if we looked at the spam problem the other way:

[20]:
```
df.groupby('spam_any').agg({
    'fid': 'count',
    'msg_casts': 'sum',
    'msg_recasts': 'sum',
    'msg_likes': 'sum',
    'msg_replies': 'sum'
})
```

[20]:
```
             fid  msg_casts  msg_recasts  msg_likes  msg_replies
spam_any
0         227535    4310724      1188510    9600735      2607666
1          17983   12891808      5779266   27965057     11812764
```

Out of the 245k users who were active last 30 days, the spammiest 7% were the source of about 4x more activity volume.

This is definitely an issue that needs attention from the Farcaster community.

5 Follower and Following

[21]:
```
df[['followers_num', 'following_num']].describe()
```

[21]:

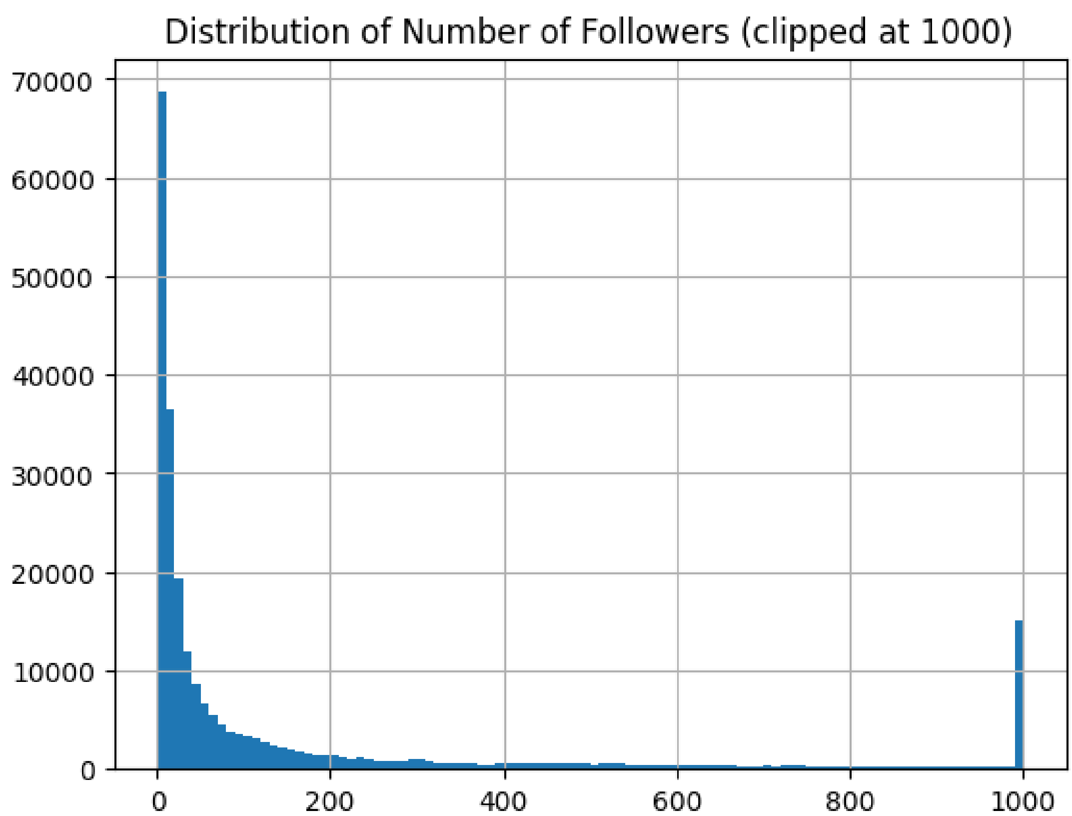
	followers_num	following_num
count	236281.0	232614.0
mean	391.093698	297.738584
std	5602.573225	788.818315
min	1.0	1.0
25%	9.0	69.0
50%	32.0	99.0
75%	167.0	263.0
max	467694.0	205633.0

[22]:

```
df[['followers_num', 'following_num']] = df[['followers_num', 'following_num']].
↪fillna(0)
```

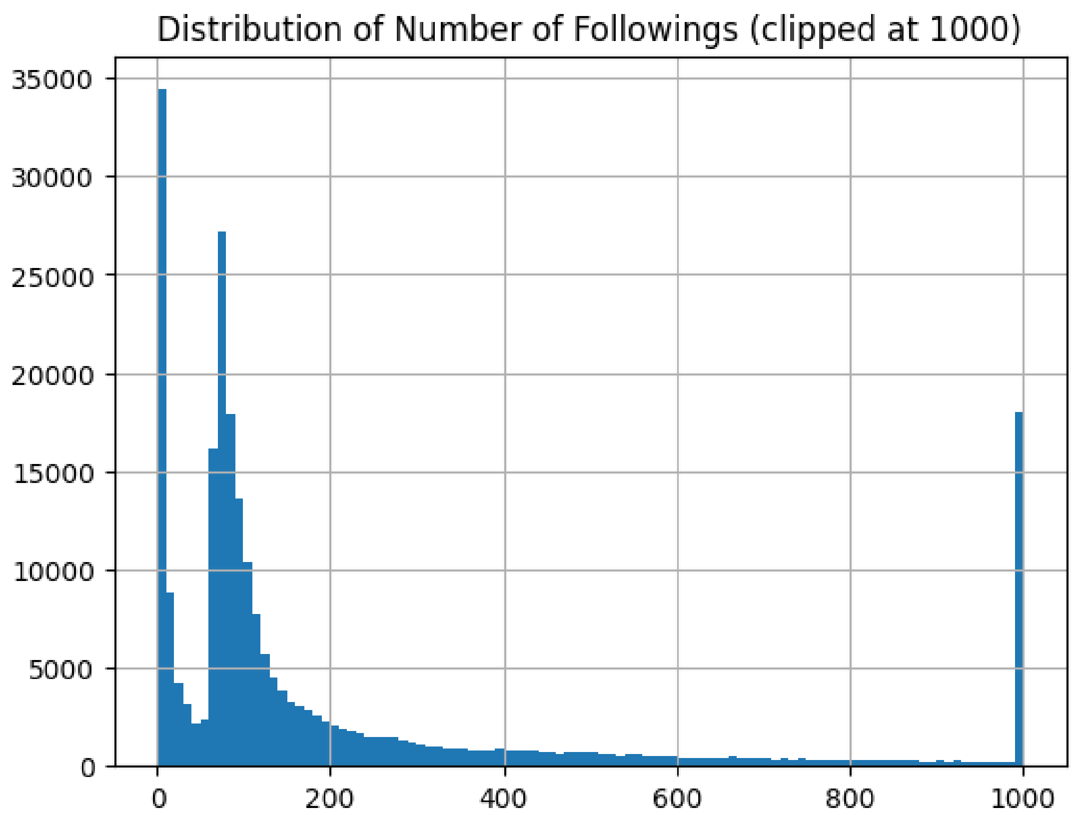
[23]:

```
_ = df['followers_num'].clip(0,1000).hist(bins=100)
_ = plt.title('Distribution of Number of Followers (clipped at 1000)')
```



[24]:

```
_ = df['following_num'].clip(0,1000).hist(bins=100)
_ = plt.title('Distribution of Number of Followings (clipped at 1000)')
```



The follower and follwing distributions look like expected, with the typical shape of a power distribution, where most users have between 0 and 100, and with the biggest accounts getting more than +100k, all the way to 467k for the biggest one. It's interesting that for the top10, almost all the active user base is following them.

Also interesting that the following distribution indicates 3 main modes: following 0 to 50 vs following around 100. vs following +1000.

6 Engagement Received

[25]:
```
df[['eng_num_days', 'eng_likes', 'eng_recasts', 'eng_replies']].describe()
```

[25]:

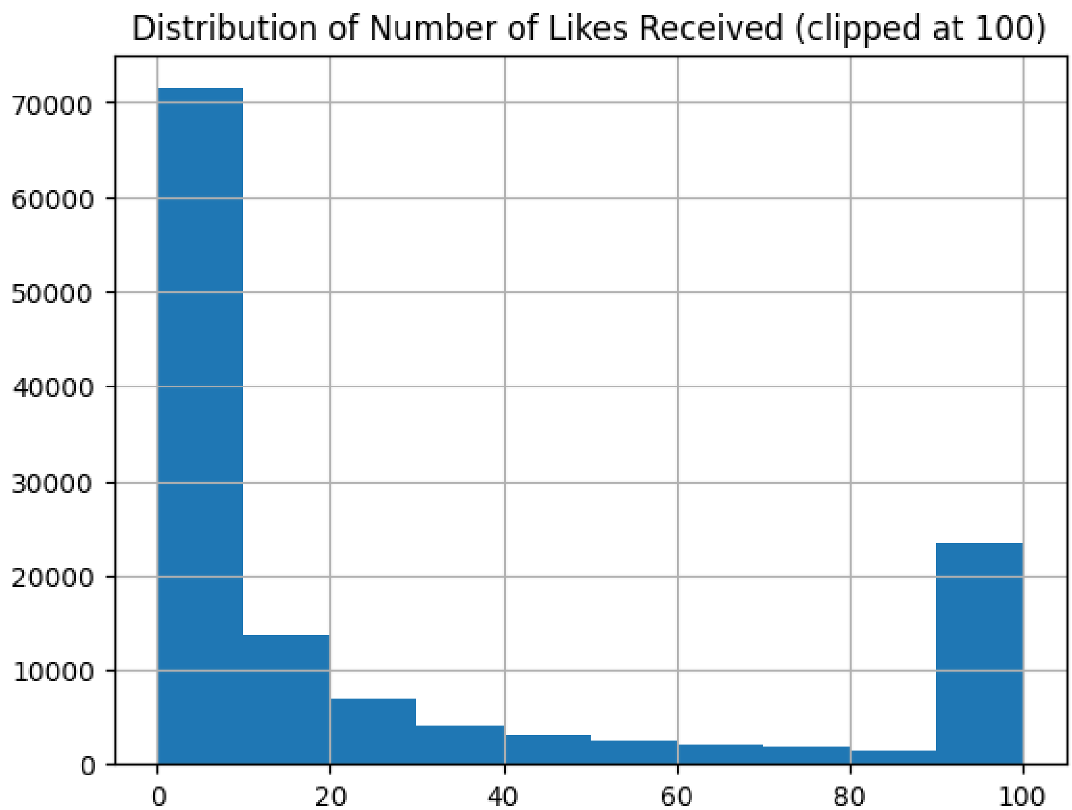
	eng_num_days	eng_likes	eng_recasts	eng_replies
count	130645.0	130645.0	130645.0	130645.0
mean	8.384355	286.477255	53.116575	110.000444
std	9.292669	1667.393242	365.177778	611.324632
min	1.0	0.0	0.0	0.0
25%	1.0	2.0	0.0	0.0
50%	4.0	7.0	0.0	1.0
75%	12.0	45.0	2.0	9.0
max	30.0	205161.0	11789.0	31235.0

Some engagement statistics to get a sense of what users get in 30 days:

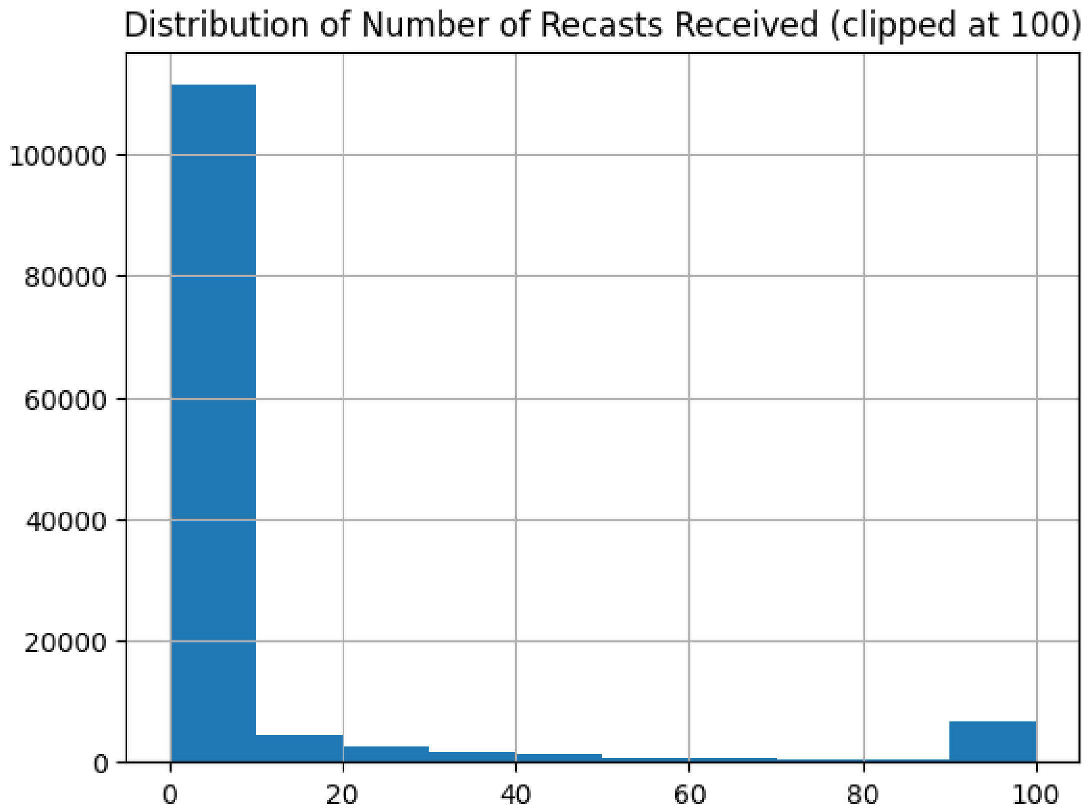
- 50% of users receive more than 7 likes.
- 25% of users get more than 2 recasts.
- 50% of users get at least one reply.

Again, averages are way higher because they are pulled up by the happy few who get much higher numbers.

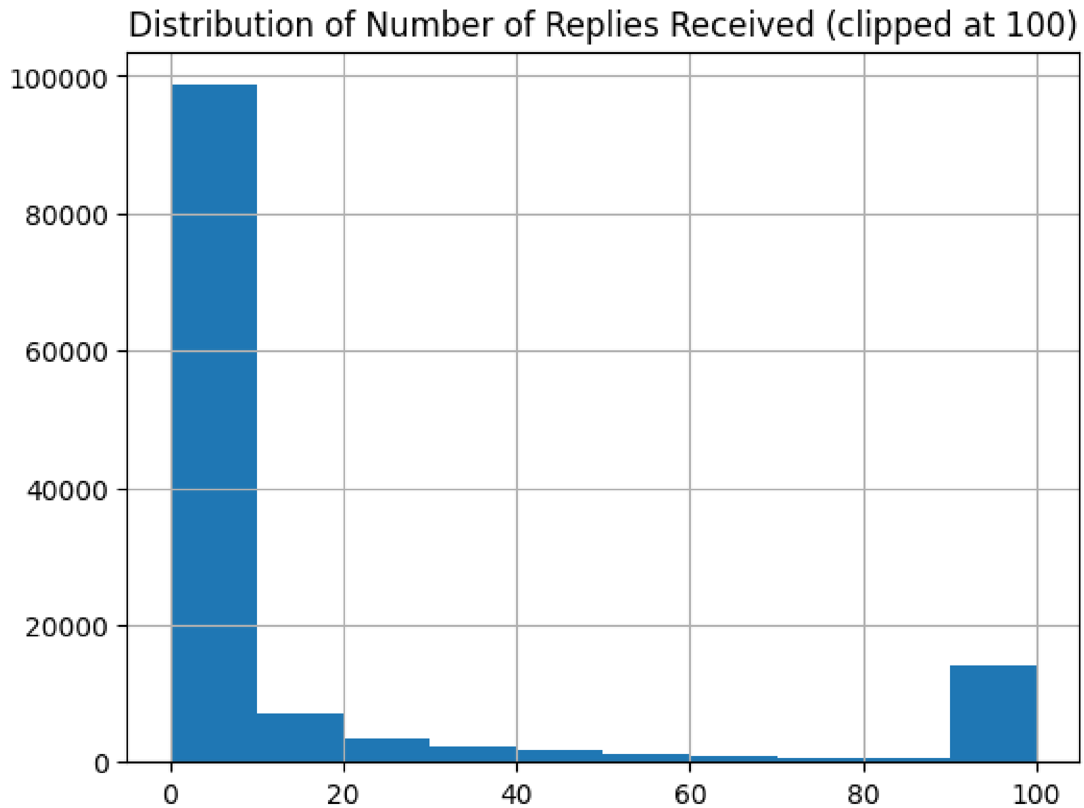
[27]:
```
_ = df['eng_likes'].clip(0,100).hist()
_ = plt.title('Distribution of Number of Likes Received (clipped at 100)')
```



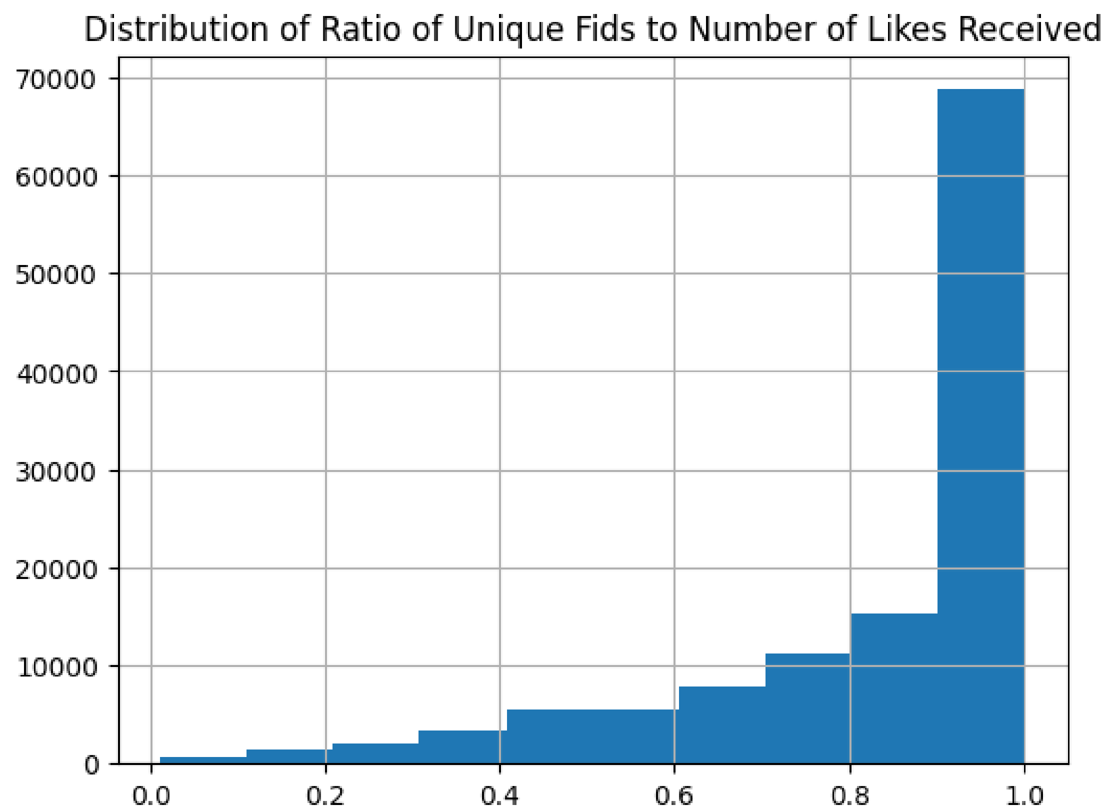
[28]:
```
_ = df['eng_recasts'].clip(0,100).hist()
_ = plt.title('Distribution of Number of Recasts Received (clipped at 100)')
```



[29]:
```
_ = df['eng_replies'].clip(0,100).hist()
_ = plt.title('Distribution of Number of Replies Received (clipped at 100)')
```



[31]:
```
_ = df['eng_ufids_likes_ratio'].hist()
_ = plt.title('Distribution of Ratio of Unique Fids to Number of Likes␣
↪Received')
```



[32]:
```
df.sort_values('eng_ufids_likes_ratio')[:10][['fid', 'num_casts', 'user_name',
    'eng_likes', 'eng_ufids_likes_ratio']]
```

[32]:

	fid	num_casts	user_name	eng_likes	eng_ufids_likes_ratio
56429	448322	283	yogy	411	0.012165
215403	816597	1012	niugts	651	0.014772
216191	817598	1067	coolpepe	683	0.016107
225621	828859	108	None	778	0.016642
222723	825736	326	ghyuna	2060	0.017957
225627	828866	106	None	754	0.018019
222734	825747	316	bleuok	1645	0.018748
222715	825728	340	truytna	1847	0.018819
225598	828832	108	liamharris	763	0.018879
222727	825740	311	suyna	1932	0.018952

Also noting that some users get very high number of recasts / likes / replies but from few accounts.

For example, the users with smallest unique fid to reaction ratios can get hundreds of reactions by a few users.

7 User Preferences

[33]:
```
df[['prefs_total_weight', 'prefs_q_info', 'prefs_q_funny',␣
↪'prefs_q_happiness']].describe().round()
```

[33]:

	prefs_total_weight	prefs_q_info	prefs_q_funny	prefs_q_happiness
count	212023.0	212023.0	212023.0	212023.0
mean	728.0	25.0	7.0	41.0
std	10696.0	21.0	6.0	19.0
min	1.0	0.0	0.0	0.0
25%	10.0	8.0	3.0	28.0
50%	29.0	21.0	6.0	40.0
75%	115.0	35.0	9.0	52.0
max	3081920.0	100.0	97.0	100.0

The user preferences are computed by aggregating the metrics from casts posted or reacted to, using the following weights:

- Casted: 4
- Recasted: 3
- Liked: 2
- Replied to: 1

The prefs_total_weight colum gives an indication on the quantity of data that was used to compute preferences.

First, let's filter for weights>10 to get better preferences prediction.

[34]:
```
df_prefs = df[df['prefs_total_weight']>10].copy()
df_prefs[['prefs_total_weight', 'prefs_q_info', 'prefs_q_funny',␣
↪'prefs_q_happiness']].describe()
```

[34]:

	prefs_total_weight	prefs_q_info	prefs_q_funny	prefs_q_happiness
count	154346.0	154346.000000	154346.000000	154346.000000
mean	998.068852	24.055602	7.260162	41.624948
std	12524.926095	17.496672	4.650543	15.817589
min	11.0	0.000000	0.000000	1.250000
25%	23.0	11.000000	4.103448	30.955181
50%	57.0	21.000000	6.513514	41.000000
75%	202.0	33.455816	9.350387	51.090909
max	3081920.0	99.000000	65.679245	99.666667

[35]:
```
cols_cats = [x for x in df.columns if x.startswith('prefs_c_')]
cols_cats
```

[35]:
```
['prefs_c_arts',
 'prefs_c_business',
 'prefs_c_crypto',
```

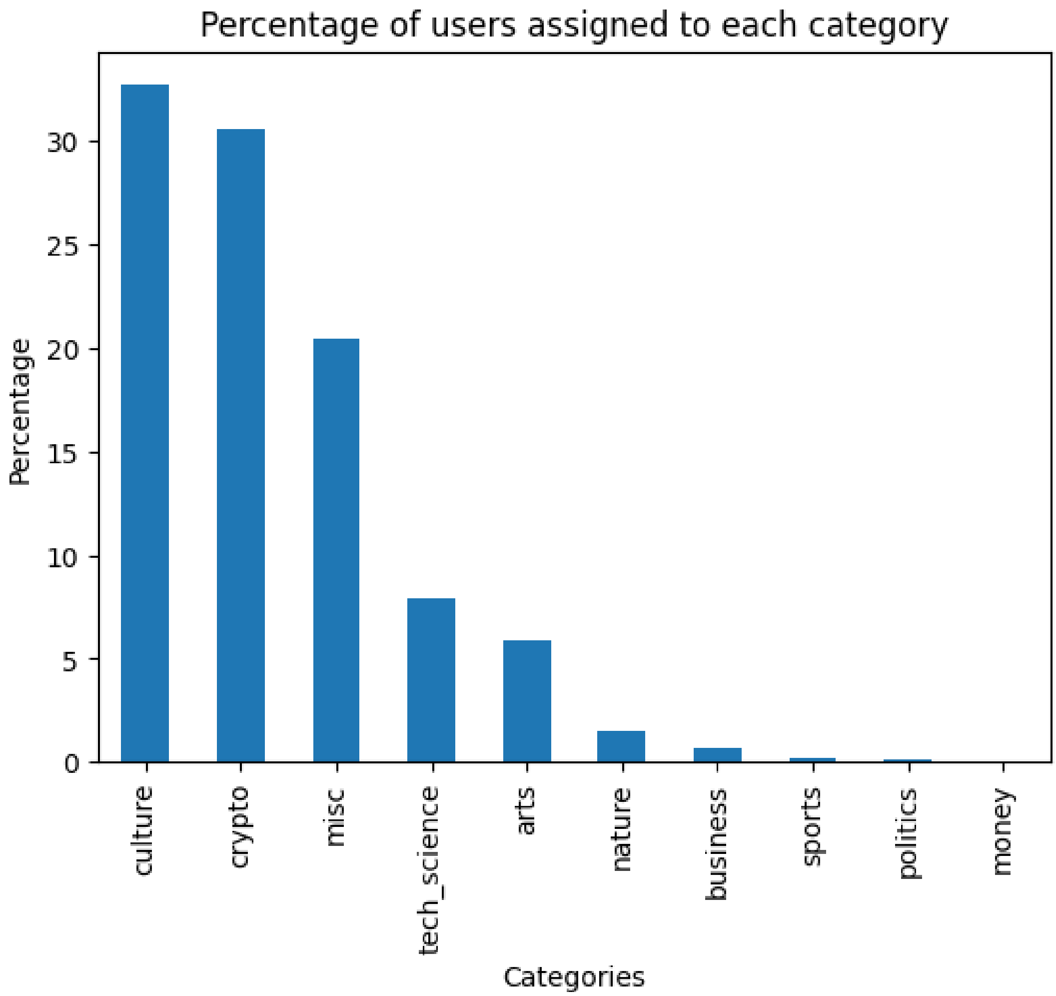
```
'prefs_c_culture',
'prefs_c_misc',
'prefs_c_money',
'prefs_c_na',
'prefs_c_nature',
'prefs_c_politics',
'prefs_c_sports',
'prefs_c_tech_science']
```

[36]:
```
df_prefs['prefs_category'] = numpy.argmax(df_prefs[cols_cats], axis=1)
df_prefs['prefs_category'] = df_prefs['prefs_category'].apply(lambda x:␣
↪cols_cats[x][8:])
categories = (100 * df_prefs['prefs_category'].value_counts() / len(df_prefs))
categories
```

[36]:
```
prefs_category
culture         32.677232
crypto          30.563798
misc            20.423594
tech_science     7.914685
arts             5.904267
nature           1.547821
business         0.672515
sports           0.175580
politics         0.112086
money            0.008423
Name: count, dtype: float64
```

[37]:
```
def plot_categories(data):
    data.plot(kind='bar')
    plt.xlabel('Categories')
    plt.ylabel('Percentage')
    plt.title('Percentage of users assigned to each category')
    plt.show()
```

[38]:
```
plot_categories(categories)
```



This is an interesting view about the preferred category per user.

Of course, actual user preferences are multidimensional and it's a really quick and dirty trick to simply assign a single category to each user with a basic max operator.

Moreover, the classification model is not 100% accurate, and there are lot of NFT related casts that end up sometimes in crypto, culture or arts, depending on the mood of the model :)

However, the model doesn't change, and it can provide interesting insights by comparing different populations.

For example, let's see if it works by comparing the categories obtained on the people following/followed by some known accounts.

7.1 Category Preferences - Example Users

[39]:
```
def get_followings(fid):
    sql = "select fid_followed as fid from dsart_farcaster.followers where␣
↪fid_follower={}".format(fid)
    return [int(x) for x in list(pandas_gbq.read_gbq(sql,␣
↪project_id=project_id, progress_bar_type=None)['fid'])]
```

[40]:
```
def get_followers(fid):
    sql = "select fid_follower as fid from dsart_farcaster.followers where␣
↪fid_followed={}".format(fid)
    return [int(x) for x in list(pandas_gbq.read_gbq(sql,␣
↪project_id=project_id, progress_bar_type=None)['fid'])]
```

[41]:
```
def rename_keys(d):
    return {x[8:]:y for x,y in d.items()}
```

[42]:
```
def get_user_prefs(user_name, user_fid):
    followings = get_followings(user_fid)
    print('followings', len(followings))
    followers = get_followers(user_fid)
    print('followers', len(followers))
    df_following = df_prefs[df_prefs['fid'].isin(followings)]
    df_follower = df_prefs[df_prefs['fid'].isin(followers)]
    df_others = df_prefs[(~df_prefs['fid'].isin(followings)) &␣
↪(~df_prefs['fid'].isin(followers))]
    print('following/follower/others samples', len(df_following),␣
↪len(df_follower), len(df_others))
    prefs_following = (100 * df_following['prefs_category'].value_counts() /␣
↪len(df_following))
    prefs_followers = (100 * df_follower['prefs_category'].value_counts() /␣
↪len(df_follower))
    prefs_others = (100 * df_others['prefs_category'].value_counts() /␣
↪len(df_others))
    combined_df = pandas.DataFrame({
        'Followed by '+user_name: prefs_following,
        'Following '+user_name: prefs_followers,
        'Other users': prefs_others}).fillna(0)
    return combined_df
```

[43]:
```
def plot_user_prefs(user_name, user_fid):
    combined_df = get_user_prefs(user_name, user_fid)
    combined_df.plot(kind='bar')
    plt.xlabel('Categories')
    plt.ylabel('Percentage')
    plt.title('Comparison of users assigned to each category')
    plt.show()
```

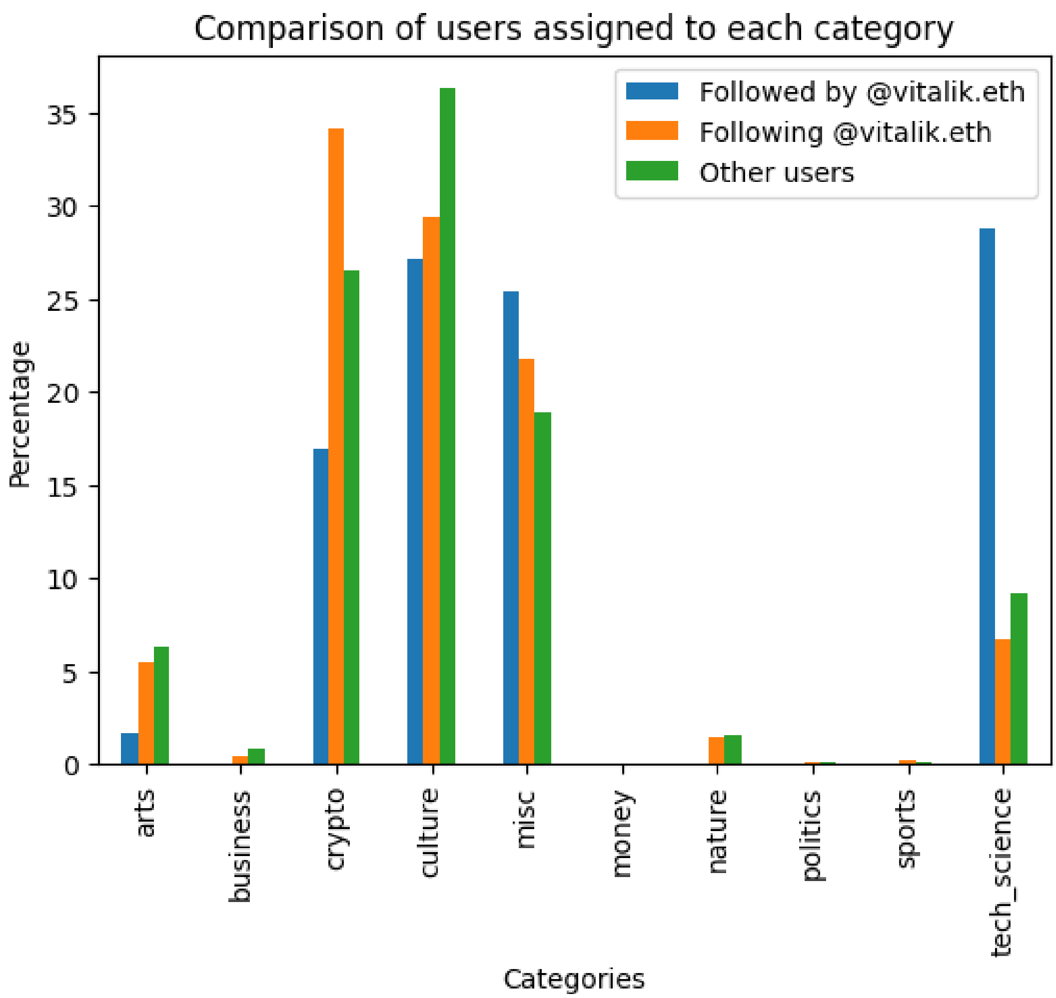
```
    return combined_df
```

7.1.1 @vitalik.eth

Let's start with Vitalik Buterin, I would expect his profile to be associated with more Ethereum/crypto people...

[44]:
```
df_vitalik = plot_user_prefs('@vitalik.eth', 5650)
```

```
followings 74
followers 384090
following/follower/others samples 59 81132 73214
```



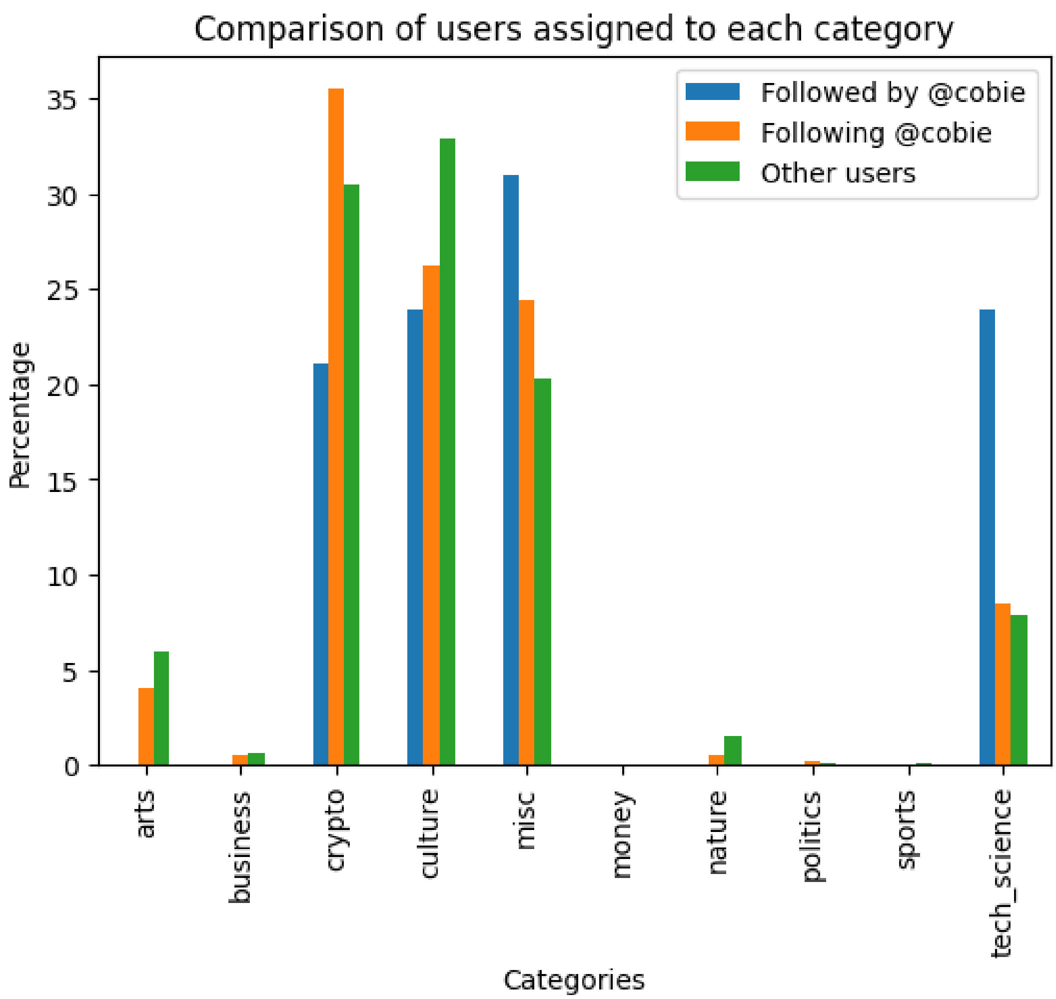
As expected, @vitalik.eth followers have a higher crypto bar than others, but interestingly, he actually follows more tech/science than crypto folks.

7.1.2 @cobie

Per his own profile description, @cobie is a “Total moron”; let’s see how it looks like in terms of user preferences.

[45]:
```
df_cobie = plot_user_prefs('@cobie', 264700)
```

```
followings 79
followers 13809
following/follower/others samples 71 4095 150222
```



@cobie’s chart looks very similar, he also follows more tech/science and followed by more crypto.

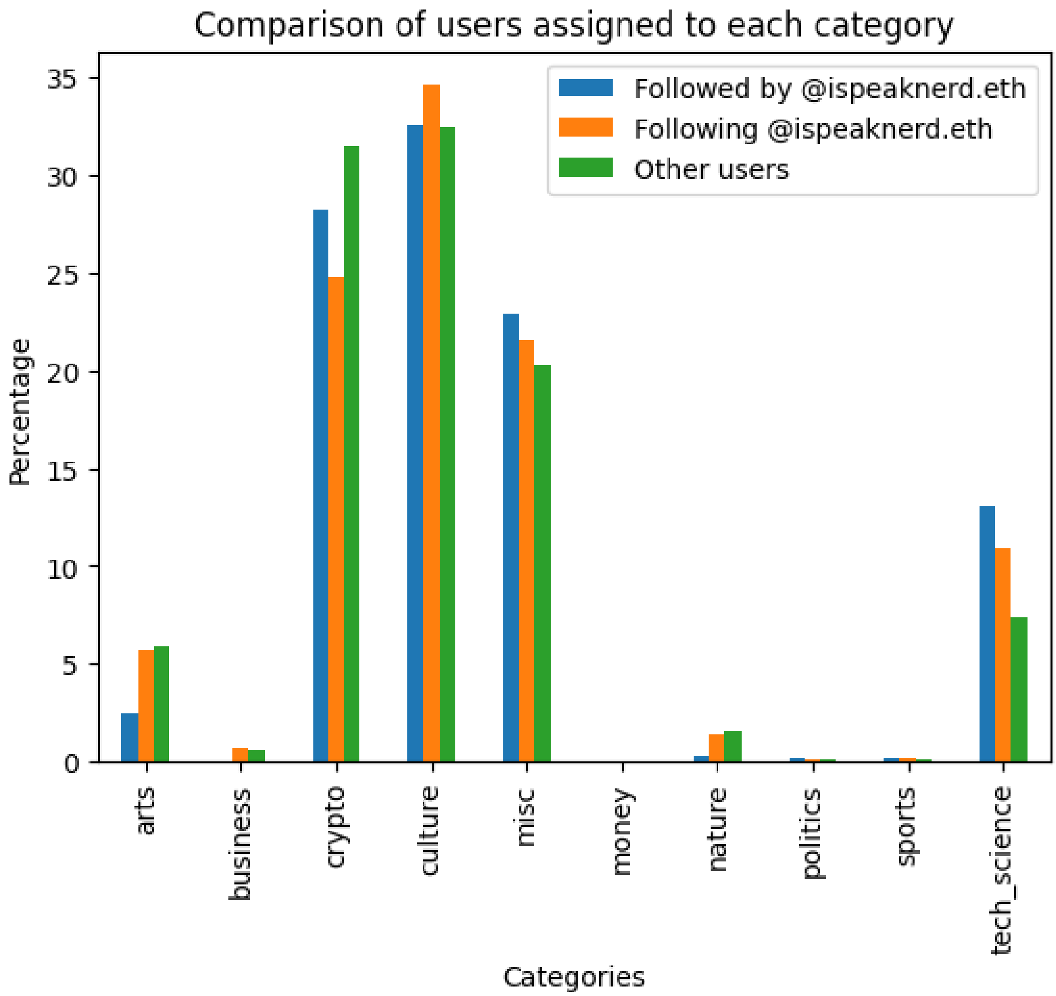
7.1.3 @ispeaknerd.eth

This is a gamer who runs the /tabletop channel. The gaming topics fall into Culture category so I would expect to see its bar higher here…

[46]:

```
df_gamer = plot_user_prefs('@ispeaknerd.eth', 9391)
```

```
followings 1390
followers 51165
following/follower/others samples 834 20469 133630
```



Turns out the distribution of categories amongst followers, following and others is pretty much similar.

At this point I am starting to think that the farcaster user base is still somehow homogeneous, and though we can observe little differences in clusters of users, they are still overall one big crypto/tech community.

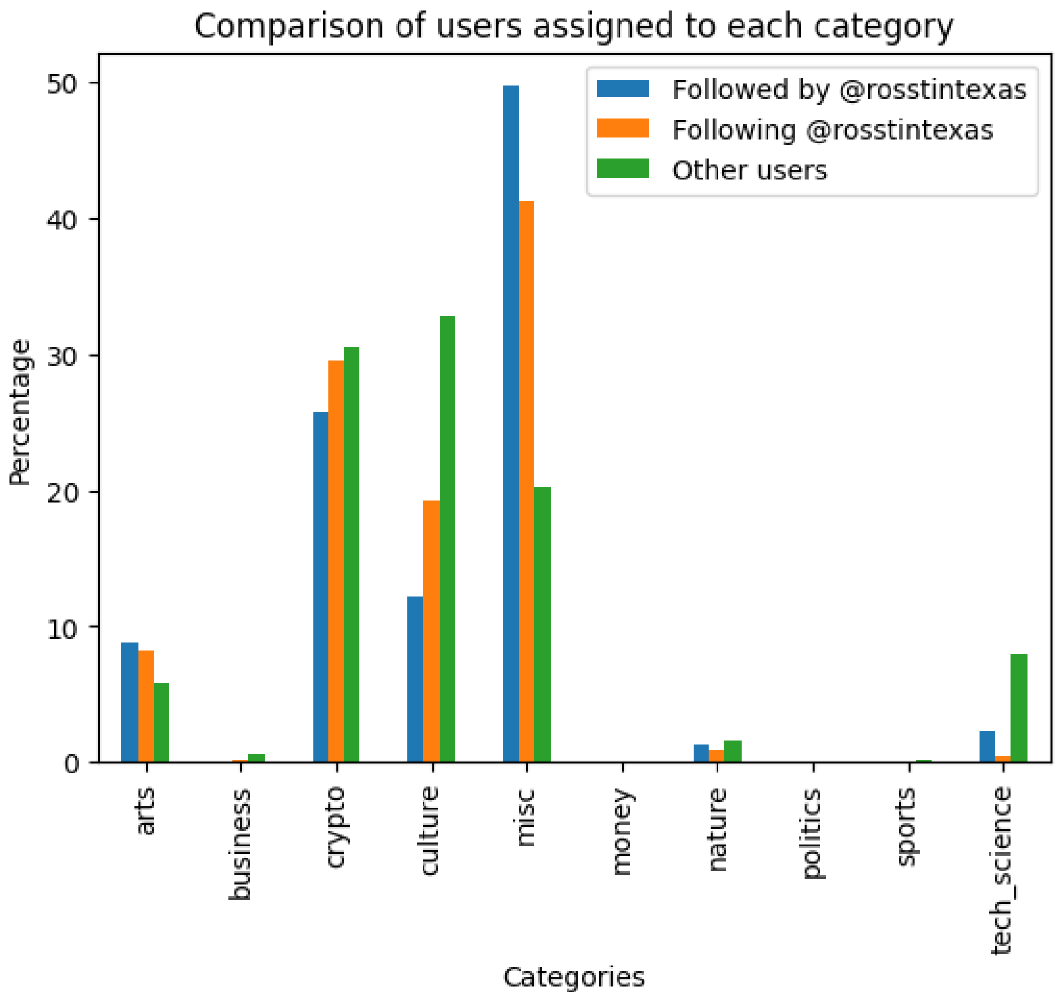
7.1.4 @rosstintexas

Trying with an account I didn't know, just picked one with +1000 followers after searching for photography.

Hopefully I see the Arts bar popping up…

[47]:
```
df_photo = plot_user_prefs('@rosstintexas', 317501)
```

```
followings 844
followers 1236
following/follower/others samples 602 855 153281
```



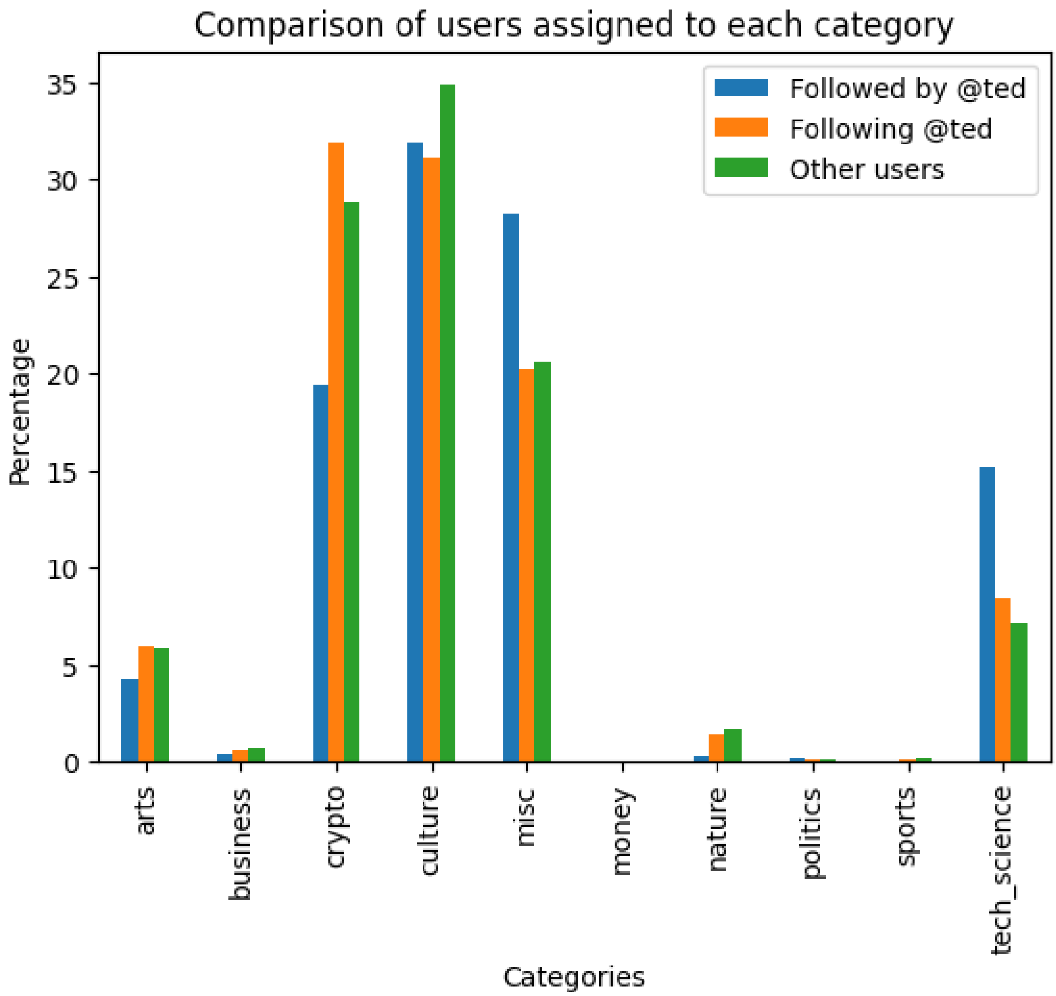
Well, there is a small bump up in Arts bar, but the one that increased most is the Misc category. This just revealed a weakness in my classification model: looking at the casts from the photography community, they often come with the artwork image, plus some gm, vague or cryptic text that the model classifies into Misc, because it's based on text only.

7.1.5 @ted

Finally, let's try with a Farcaster OG…

```
[48]: df_ted = plot_user_prefs('@ted', 239)
```

```
followings 1346
followers 377945
following/follower/others samples 928 89370 64931
```



Again, the distribution is very similar to the general population, except @ted follows more tech/science than average.

7.2 Category Preferences - Onboarding cohorts

Let's see if there differences between users by onboarding time.

```
[49]: # Ignore Aug as they don't all have 30 days of data, then group by quarter
df_cohorts = df_prefs[(df_prefs['first_cast']<'2024-08') |
 ↪(df_prefs['first_cast']=='OG')].copy()
df_cohorts['cohort_q'] = None
```

```
df_cohorts.loc[df_cohorts['cohort']=='OG', 'cohort_q'] = 'OG'
df_cohorts.loc[df_cohorts['cohort'].isin(['2024-01', '2024-02', '2024-03']),␣
↪'cohort_q'] = '2024-Q1'
df_cohorts.loc[df_cohorts['cohort'].isin(['2024-04', '2024-05', '2024-06']),␣
↪'cohort_q'] = '2024-Q2'
df_cohorts.loc[df_cohorts['cohort'].isin(['2024-07']), 'cohort_q'] = '2024-07'
df_cohorts['cohort_q'].value_counts()
```

[49]:

```
cohort_q
2024-Q2    69298
2024-07    25223
2024-Q1    22183
OG          5084
Name: count, dtype: int64
```

[50]:

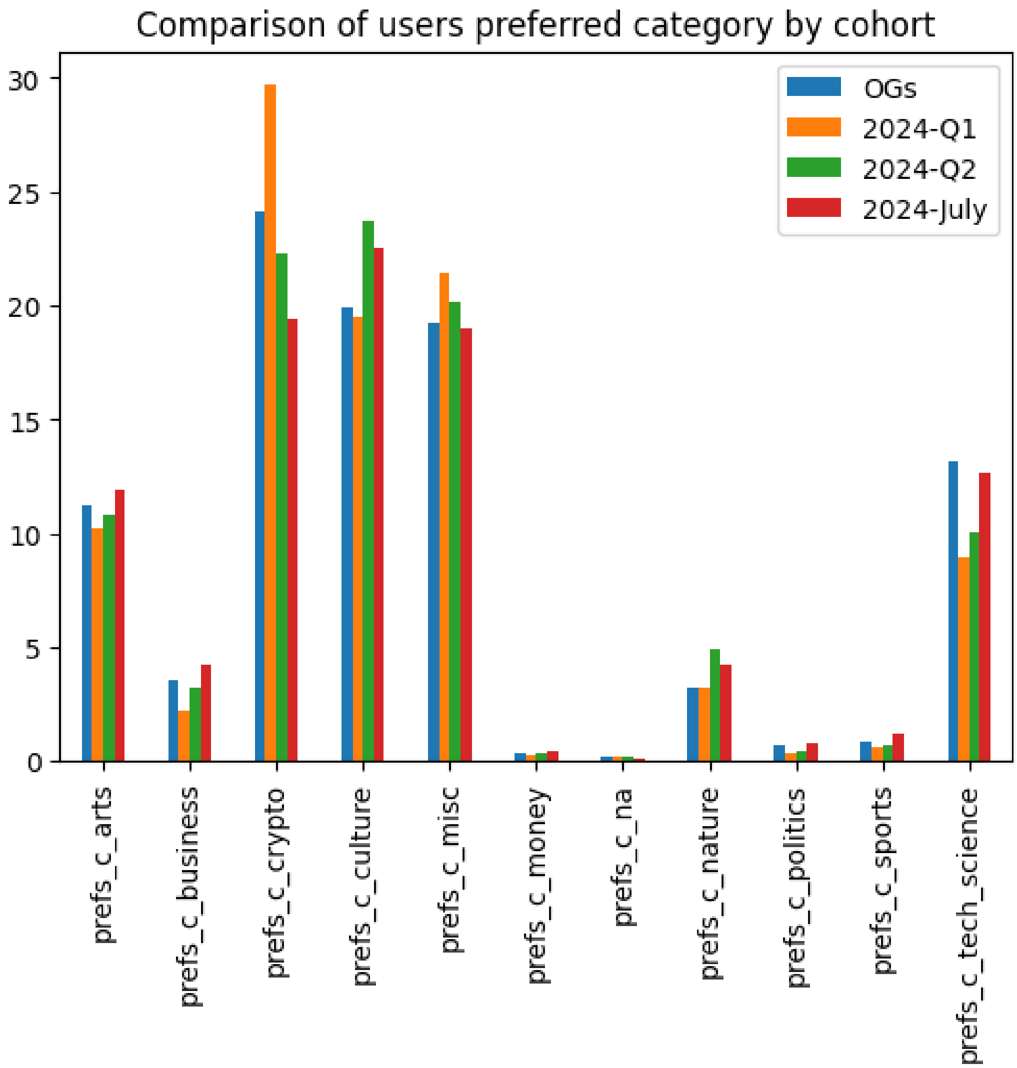
```
agg = df_cohorts.groupby('cohort_q').agg({
    x: 'mean' for x in cols_cats
})
df_cohorts_agg = pandas.DataFrame({
    'OGs': agg.loc['OG'],
    '2024-Q1': agg.loc['2024-Q1'],
    '2024-Q2': agg.loc['2024-Q2'],
    '2024-July': agg.loc['2024-07']
}).fillna(0)
df_cohorts_agg
```

[50]:

	OGs	2024-Q1	2024-Q2	2024-July
prefs_c_arts	11.232532	10.230246	10.803943	11.879539
prefs_c_business	3.535051	2.206162	3.247706	4.219832
prefs_c_crypto	24.100956	29.676703	22.289179	19.380258
prefs_c_culture	19.906334	19.527573	23.739692	22.526988
prefs_c_misc	19.269797	21.464598	20.211403	19.021131
prefs_c_money	0.391076	0.317838	0.349833	0.430527
prefs_c_na	0.162978	0.173824	0.150440	0.138019
prefs_c_nature	3.196947	3.246430	4.941872	4.226504
prefs_c_politics	0.724716	0.362856	0.419978	0.803999
prefs_c_sports	0.858297	0.603906	0.677775	1.222014
prefs_c_tech_science	13.152193	8.964758	10.094364	12.635168

[51]:

```
_ = df_cohorts_agg.plot(kind='bar')
_ = plt.title('Comparison of users preferred category by cohort')
```



Ok, so most categories don't show any significant differences, but there's a couple of interesting insights:

- Farcaster mainly onboarded crypto folks, with a peak in this category in 2024-Q1, but it's decreasing since then.
- The OGs were more of tech/science folks, but the new onboards are increasing again in this category.

7.3 Category Preferences - By Language

[52]:
```
agg_dict = {
    x: 'mean' for x in cols_cats
}
agg_dict['fid'] = ['count']
```

```
agg = df_cohorts.groupby('lang_1').agg(agg_dict).sort_values(('fid', 'count'),
    ascending=False)[:3]
del agg['fid']
agg.columns = [x[8:] for x in cols_cats]
agg
```

[52]:

```
             arts  business     crypto    culture       misc     money  \
lang_1
en      11.069021  3.566821  23.449594  21.473685  19.397397  0.380756
tl      13.045664  0.915483  16.441558  25.473875  31.854745  0.265970
vi       9.102647  1.534775  19.646789  39.216269  16.827170  0.089882

              na    nature  politics    sports  tech_science
lang_1
en      0.143631  4.619906  0.547380  0.843680     11.164734
tl      0.222878  3.082864  0.189849  0.512909      6.383762
vi      0.105221  2.394676  0.184990  0.213159      6.799715
```

[53]:

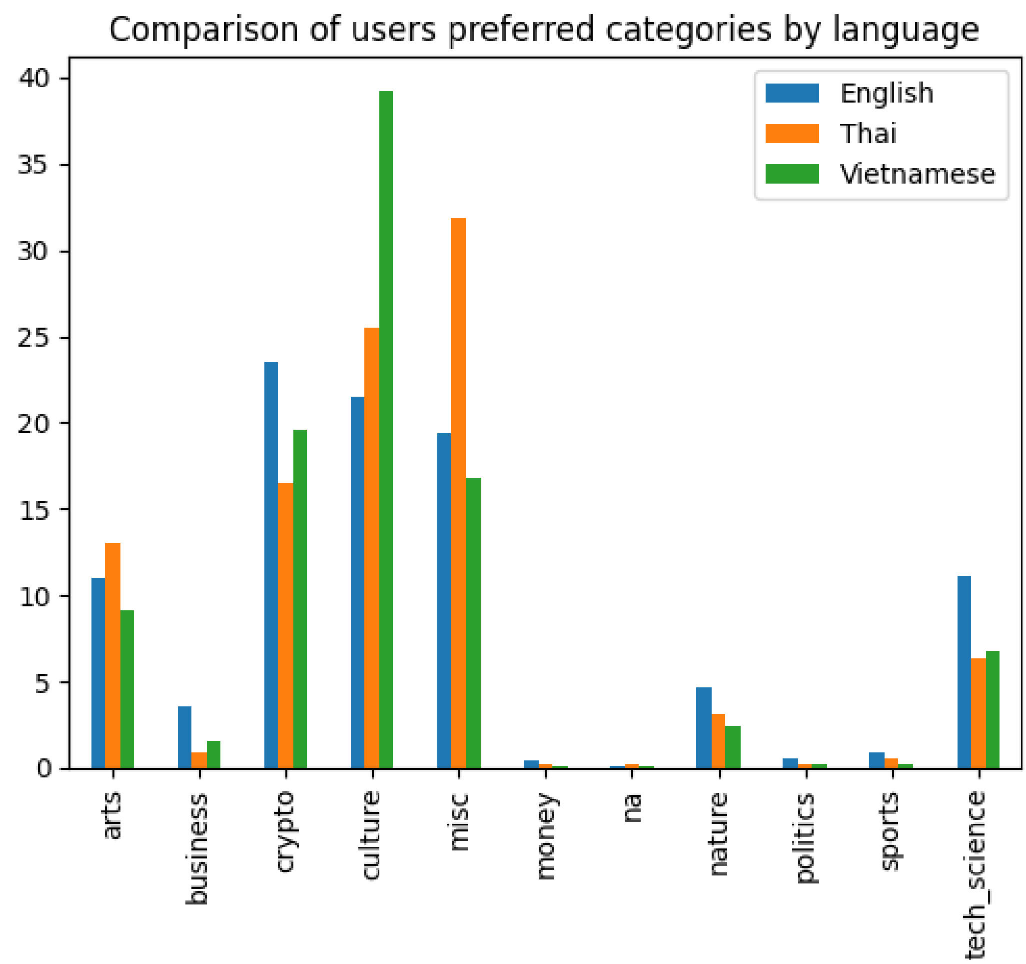
```
df_lang_agg = pandas.DataFrame({
    'English': agg.loc['en'],
    'Thai': agg.loc['tl'],
    'Vietnamese': agg.loc['vi']
}).fillna(0)
df_lang_agg
```

[53]:

```
                English       Thai  Vietnamese
arts          11.069021  13.045664    9.102647
business       3.566821   0.915483    1.534775
crypto        23.449594  16.441558   19.646789
culture       21.473685  25.473875   39.216269
misc          19.397397  31.854745   16.827170
money          0.380756   0.265970    0.089882
na             0.143631   0.222878    0.105221
nature         4.619906   3.082864    2.394676
politics       0.547380   0.189849    0.184990
sports         0.843680   0.512909    0.213159
tech_science  11.164734   6.383762    6.799715
```

[54]:

```
_ = df_lang_agg.plot(kind='bar')
_ = plt.title('Comparison of users preferred categories by language')
```



Ok, some interesting differences here:

- Thai community has a higher level of Misc, generally associated with GMs and non text casts.
- Vietnamese scores higher on culture.
- English scores higher on Tech/Science.

7.4 Take-aways

1) The Data hints at the fact that the Farcaster community is still relatively homogeneous.
2) While we can observe the beginning of a differentiation in clusters and communities, they remain relatively small.
3) OGs were more of Tech/Science folks, and zooming on some big accounts, they also seem to be biased towards following tech/science folks.
4) The insights have to be taken with a grain of salt given that the classification model is based on text only, and ignores the contexts and images of every cast.

8 Keywords

Finally, for sanity checks and for fun, let's make a bunch of word clouds…

Can also give a global taste of what's going on the platform.

[55]:
```
df_words = df[df['words_dict'].notnull()].copy()
def parse_words(x):
    x = x.replace("\\", "")
    return json.loads(x)
df_words['words_dict'] = df_words['words_dict'].apply(parse_words)
len(df_words)
```

[55]: 206848

[56]:
```
def make_word_cloud(df, n):
    counter = Counter()
    for _, row in df.iterrows():
        counter.update(row['words_dict'])
    freqs = dict(counter.most_common(n))
    wordcloud_config = WordCloud(width=800, height=400, background_color='white')
    wordcloud = wordcloud_config.generate_from_frequencies(freqs)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')  # Hide the axis
    plt.show()
    return freqs
```

[57]:
```
words_all = make_word_cloud(df_words, 50)
```

8.1 Word Clouds per category

[58]:
```
df_words['prefs_category'] = numpy.argmax(df_words[cols_cats], axis=1)
df_words['prefs_category'] = df_words['prefs_category'].apply(lambda x: cols_cats[x][8:])
categories = (100 * df_words['prefs_category'].value_counts() / len(df_words))
categories
```

[58]:
```
prefs_category
crypto          30.836170
culture         29.814163
misc            19.598933
tech_science     9.648631
arts             6.344756
nature           2.035794
business         1.154954
sports           0.285234
politics         0.227220
money            0.054146
Name: count, dtype: float64
```

[59]:
```
def make_word_cloud_category(c):
    df_tmp = df_words[df_words['prefs_category']==c]
    return make_word_cloud(df_tmp, 50)
```

8.1.1 Crypto

[60]:
```
words_crypto = make_word_cloud_category('crypto')
```

8.1.2 Culture

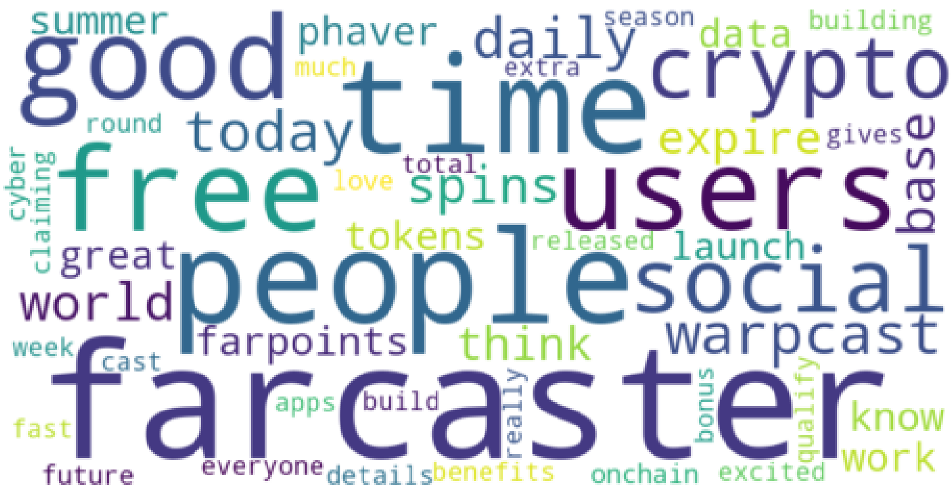
[61]:
```
words_culture = make_word_cloud_category('culture')
```

8.1.3 Misc

[62]:
```
words_misc = make_word_cloud_category('misc')
```

8.1.4 Tech / Science

[63]:
```
words_misc = make_word_cloud_category('tech_science')
```



8.1.5 Arts

[64]:
```python
words_arts = make_word_cloud_category('arts')
```

8.1.6 Nature

[65]:
```python
words_nature = make_word_cloud_category('nature')
```

8.1.7 Business

[66]:
```
words_business = make_word_cloud_category('business')
```

8.1.8 Sports

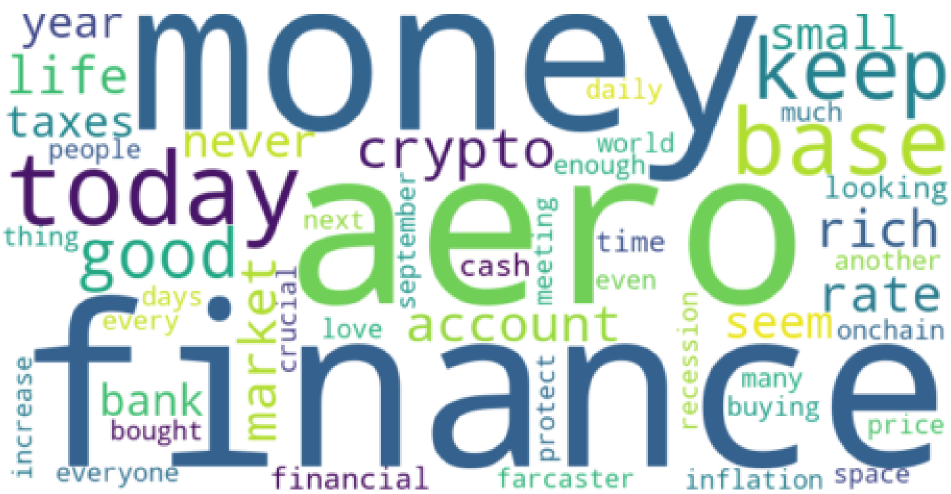
[67]:
```
words_sports = make_word_cloud_category('sports')
```

8.1.9 Politics

[68]:
```python
words_politics = make_word_cloud_category('politics')
```

8.1.10 Money

[69]:
```python
words_money = make_word_cloud_category('money')
```



8.2 Word Clouds per language

[70]:
```
df_words['lang_1'].value_counts()[:10]
```

[70]:
```
lang_1
en       179592
tl        10308
zh-cn      3288
vi         3026
nl         1405
ja         1149
ko          739
es          630
ru          447
af          329
Name: count, dtype: int64
```

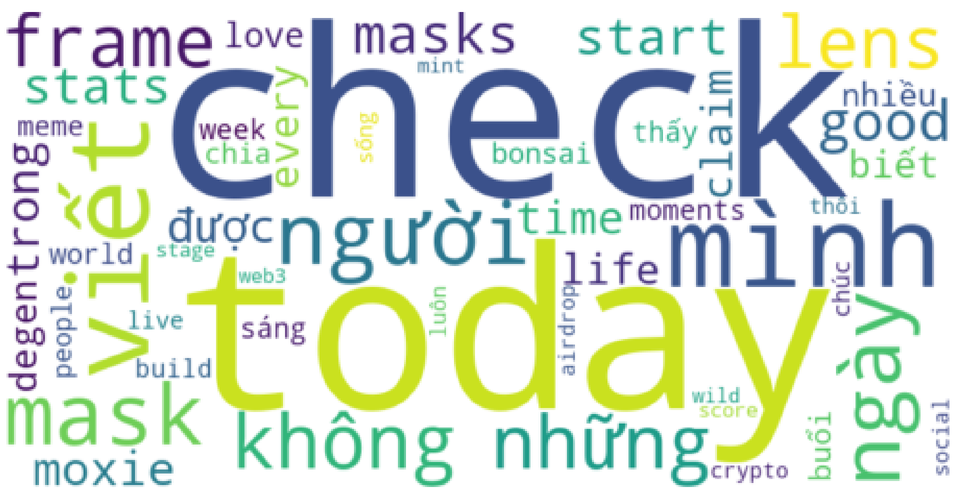
[71]:
```
def make_word_cloud_language(l):
    df_tmp = df_words[df_words['lang_1']==l]
    return make_word_cloud(df_tmp, 50)
```

[72]:
```
words_en = make_word_cloud_language('en')
```

[73]:
```
words_tl = make_word_cloud_language('tl')
```

[74]:
```
words_chinese = make_word_cloud_language('zh-cn')
```


```
[75]: words_vi = make_word_cloud_language('vi')
```

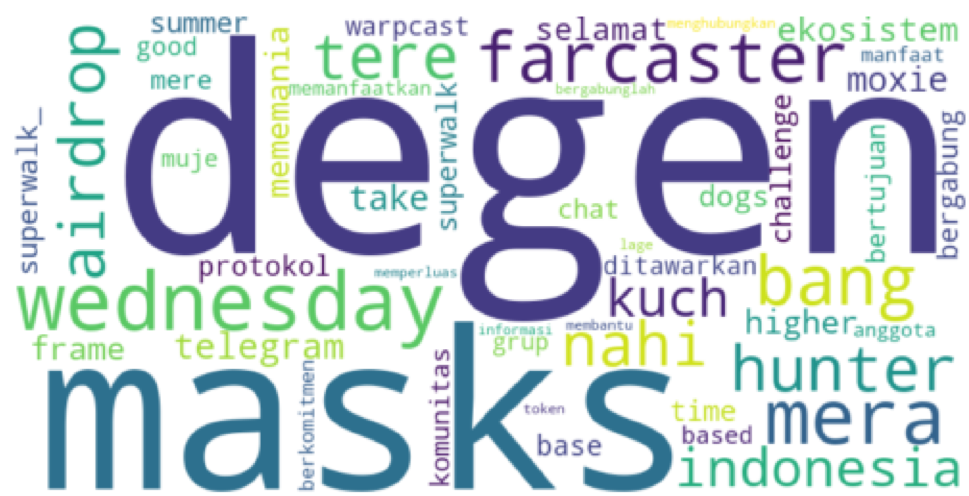



```
[76]: words_nl = make_word_cloud_language('nl')
```

[77]:
```
words_es = make_word_cloud_language('es')
```

[78]:
```
words_ru = make_word_cloud_language('ru')
```

[79]:
```
words_id = make_word_cloud_language('id')
```



Sorry for spamming my own notebook with a bunch of word clouds, just wanted to test the models and make sure the outputs made sense…

Funny how Degen is everywhere :)

9 Summary

There is a lot of discussion about Farcaster issues, criticizing the dominance of OG accounts and bots, emphasizing the spam probem, and questioning the warpcast algorithms.

Deep diving into the data, my conclusion is that there are definitely some symptoms, but they are not as bad as some casters say they are.

I have seen claims that “the bot problem” could be as huge as 50% or more of the user base, but the data indicates that the actual number of toxic is clearly under 10%, probably less than 5%. The problem is not much their number, but their impact on the user experience.

I also observed that the platform onboarded a high ratio of spammy accounts during Q2 2024, but it is now evolving towards better quality users, which is an optimistic sign. There’s probably some learnings to take-away from this cohort in terms of incentives and their consequences on the platform trajectory, like growth hack with token gamification is probably a bad idea, unless it’s deeply aligned on content quality and long term objectives.

Farcaster is still in early stage, relatively healthy, where the main weakness is actually the user base itself: it is still revolving around crypto and tech folks, and therefore doesn’t generate enough value to attract other communities.

As a Farcaster enthusiast, I wish the platform would attract more diverse user profiles, and hope to see some improvement in the clients algorithms to favor users who engage less, but meaningfully, rather than hyperactive accounts.