Jim Arnold

Data Challenge 2

Executive summary:

Analysis of weekly user engagement suggests there is a cohort of 256 users, primarily located in North America and Europe, who have stopped opening weekly digest emails. Given there remain active users in every country, this is unlikely to be caused by regional server connectivity issues.

Drop in user engagement does not appear to be isolated to a particular kind of device, suggesting it's not likely a software/hardware issue.

It also appears to involve at least 1 user from every company, although never 100% from a given company, suggesting the drop in user engagement is not likely due to changes in firewall / spam-filters.

Additionally, there is the weekly digest emails are being delivered as expected, suggesting the drop in user engagement is not likely product-related.

It appears that while emails are being delivered as expected, many users in North America and Europe are not engaging with the product. Given this anomaly is occuring at the end of July / early August, it may be possible we are seeing a seasonal effect, i.e. summer vacation.

Recommendations:

1) If possible, review prior years' data to confirm/ reject seasonality hypothesis.

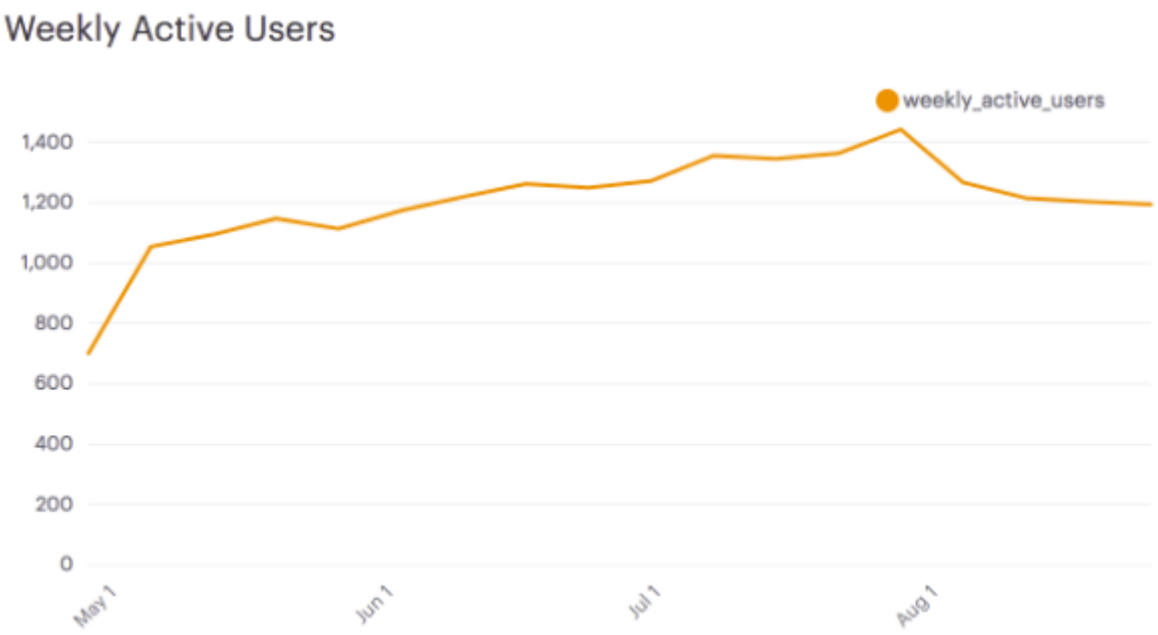
2) Contact product team to give heads up. Monitor weekly after users for rebound in 2-4 weeks.

3) Consider contingency plans if rebound not observed (methods to incentivize re-engagement).

Table of Contents

The Problem - Investigating a Drop in User Engagement

Tuesday morning, September 2, 2014. The head of the Product team walks over to your desk and asks you what you think about the latest activity on the user engagement dashboards. Here's what it looks like:



The above chart shows the number of engaged users each week.

Yammer defines engagement as having made some type of server call by interacting with the product (shown in the data as events of type “engagement”).

Any point in this chart can be interpreted as:

“the number of users who logged at least one engagement event during the week starting on that date.”

Important to understand... 'enagagement' is defined in the ***events*** table.

The head of product says “Can you look into this and get me a summary by this afternoon?”

"Sure thing!" I say.

But where to start?

Hypotheses and Approach:

I believe I'm being asked to summarize the data and find a potential explanation for the drop in 'active' users right around the end of July. From the dashboard chart, it looks like around 200 people stopped engaging, but this was after a period of growth.

200 users is pretty big. I think the best approach to summarizing this data is trying to figure out what is common among the users who stopped being 'active' around the end of July and August.

Given that 'active' is defined by 'engaging' with the product in some way during the following week, I can think of a few possible hypotheses to test.

Hypotheses

1) Users were prompted to signup, then stopped using it. - an adoption/ compliance issue.

2) Users changed how they interact with the platform - which could be specific to a company, language, location, or device.

3) Similar to 2, there could be an underlying technical issue, perhaps people can't connect and/ or interact with the product.

If none of these yield interesting results, I'll try thinking of others.

Approach

To approach this question, I think it's important to explore some details. Specifically:

WHO stopped using the product? If time remaining, address WHY.

To answer the question of WHO, I'll need to look at the users who specifically stopped engaging with the product between the end of July and the end of August.

To assess this, I'll need to utilize user engagement with the platform, which is in the events table. First thought - I'll try to group by a 'last used' date, and label those who engaged in July but not August vs. those who engaged in July and August.

Once I have that, I should be able to evaluate features of that group vs the currently active users (testing hypotheses 2 and 3).

I can also look at last active date vs. account creation to test hypothesis 1.

I'll focus on looking for differences in categorical features, namely country, language, device, or company.

Back to Table of Contents

Data Preparation - Loading Data and Assigning Labels

Because I have 4 tables of data with common features between them, I'm going to take a SQL approach.

Rationale: This will hopefully make querying between datasets easier, specifically once I identify my users of interest.

In [1]:
```
# Packages for postgres, sqlalchemy, and pandas.
from sqlalchemy import create_engine
from sqlalchemy_utils import database_exists, create_database
import psycopg2
import pandas as pd
import numpy as np
```

In [2]:
```
pwd = ""
```

In [3]:
```
# omitted my password in the final version
dbname = 'DC2'
username = 'jim'
pswd = pwd
```

In [4]:
```
# 'engine' is a connects to the local DC2 db I created for this.
engine = create_engine('postgresql://%s:%s@localhost/%s'%(username,ps
wd,dbname))

# create a database (if it doesn't exist)
if not database_exists(engine.url):
    create_database(engine.url)
print(database_exists(engine.url))
print(engine.url)
```

```
True
postgresql://jim:Nint3nd0@localhost/DC2
```

In [5]:
```
# I'm summarizing here, but I noticed several of the datetime columns required 'parse_dates' for accurate dtyping
users = pd.read_csv('yammer_users.csv', parse_dates = ['created_at', 'activated_at'])
# repeat for all csv's provided
emails = pd.read_csv('yammer_emails.csv', parse_dates = ['occurred_at'])
# user_id and user_type should be an int to match with other table.
# user_type threw a 'safety' error - need to follow up for explanation
events = pd.read_csv('yammer_events.csv', dtype = {'user_id': np.int64}, parse_dates = ['occurred_at'])
# parse the dates in the rollup table
rollup = pd.read_csv('yammer_dimension_rollup_periods.csv',
                     parse_dates = ['time_id', 'pst_start', 'pst_end', 'utc_start', 'utc_end'])
```

In [6]:
```
events.head()
```

Out[6]:

	user_id	occurred_at	event_type	event_name	location	device	user_type
0	10522	2014-05-02 11:02:39	engagement	login	Japan	dell inspiron notebook	3.0
1	10522	2014-05-02 11:02:53	engagement	home_page	Japan	dell inspiron notebook	3.0
2	10522	2014-05-02 11:03:28	engagement	like_message	Japan	dell inspiron notebook	3.0
3	10522	2014-05-02 11:04:09	engagement	view_inbox	Japan	dell inspiron notebook	3.0
4	10522	2014-05-02 11:03:16	engagement	search_run	Japan	dell inspiron notebook	3.0

The 'event_type', and 'occurred_at' datetime features are going to be critical for answering my question.

I also like that this table has the 'user_id' feature, which will be useful for joining on the previous *emails* and *users* tables.

I'm not sure how to use a rollup table. If there's time, I'd should ask someone how to implement it.

Import the properly dtyped data to my local db

In [7]:
```
# insert data into database
users.to_sql('users', engine, if_exists='replace')
emails.to_sql('emails', engine, if_exists='replace')
events.to_sql('events', engine, if_exists='replace')
rollup.to_sql('rollup_periods', engine, if_exists='replace')
```

In [8]:
```
# check to see tables are loaded
engine.table_names()
```

Out[8]:
```
['users', 'emails', 'events', 'rollup_periods']
```

With data loaded, on to generating labels.

[Back to Table of Contents]

Data Preparation - Assign Labels

In [9]:
```
# first, I want to get a summary of the event data available
# what dates does this data cover, and how many users are we talking
 about?
#
con = None
con = psycopg2.connect(database = dbname, user = username, host='loca
lhost', password=pswd)
# query:
sql_query = """
SELECT *
FROM events
"""
data_from_sql = pd.read_sql_query(sql_query,con)
data_from_sql.head()
```

Out[9]:

	index	user_id	occurred_at	event_type	event_name	location	device	user_type
0	0	10522	2014-05-02 11:02:39	engagement	login	Japan	dell inspiron notebook	3.0
1	1	10522	2014-05-02 11:02:53	engagement	home_page	Japan	dell inspiron notebook	3.0
2	2	10522	2014-05-02 11:03:28	engagement	like_message	Japan	dell inspiron notebook	3.0
3	3	10522	2014-05-02 11:04:09	engagement	view_inbox	Japan	dell inspiron notebook	3.0
4	4	10522	2014-05-02 11:03:16	engagement	search_run	Japan	dell inspiron notebook	3.0

It looks like the engagement label in the 'event_type' column is what I need.

First, let's try to replicate the initial observation here.

Rationale: I want to get the count of *weekly unique users* who engaged with the product that week, ***If I can replicate the initial finding then I know I'm on the right track.

In SQL, this should be possible with the DATE_TRUNC function and a WHERE clause.

In [10]:
```
sql_query = """
SELECT COUNT(DISTINCT(user_id)) AS weekly_users,
       DATE_TRUNC('week', occurred_at) AS week
 FROM events
 WHERE event_type = 'engagement'
 GROUP BY 2
"""
data_from_sql = pd.read_sql_query(sql_query,con)
data_from_sql.head()
```

Out[10]:

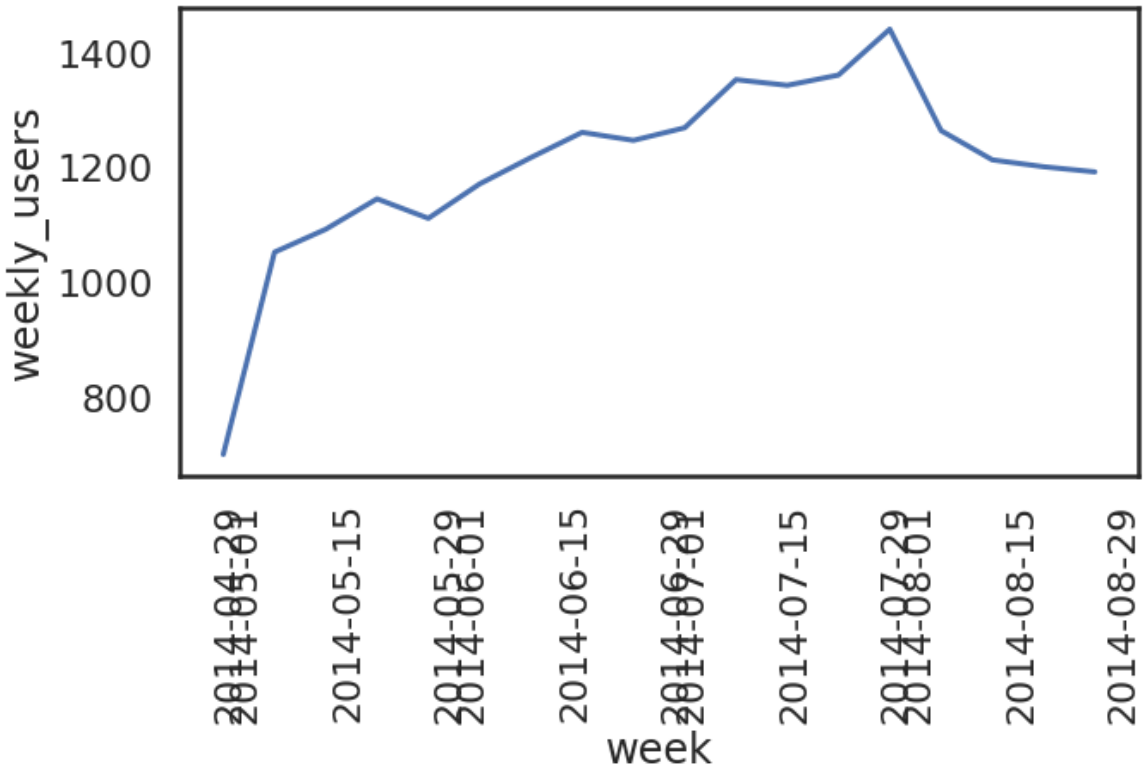
	weekly_users	week
0	701	2014-04-28
1	1054	2014-05-05
2	1094	2014-05-12
3	1147	2014-05-19
4	1113	2014-05-26

In [13]:
```
# That seems right, plot it to confirm.

# load in viz tools
import matplotlib.pyplot as plt
import seaborn as sns

# set conditions for plot
sns.set(context = 'poster', style = 'white')
plt.figure(figsize=(10,5))
ax = sns.lineplot(x="week", y="weekly_users", data=data_from_sql)
plt.xticks(rotation=90)
```

Out[13]:
```
(array([735352., 735354., 735368., 735382., 735385., 735399., 73541
3.,
        735415., 735429., 735443., 735446., 735460., 735474.]),
 <a list of 13 Text xticklabel objects>)
```



OK, managed to reproduce the plot. Now I know where that data came from.

The dates should be formatted, but worry about pretty later.

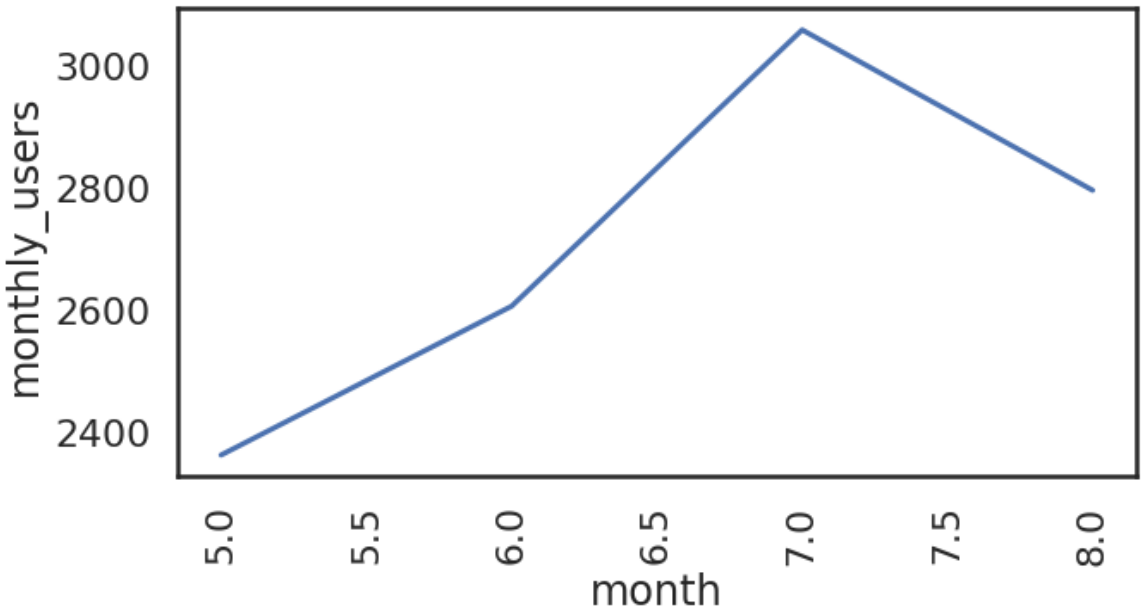
Now, I'd like to drill down on the specific time range where users dropped off.

It looks like the peak weekly users in July, which then drops off.

Given that, I can aggregate to 'monthly active users' instead of 'weekly active users'

In [14]:
```
# query monthly active users
sql_query = """
SELECT COUNT(DISTINCT(user_id)) AS monthly_users,
        EXTRACT('month'  FROM occurred_at) AS month
 FROM events
 WHERE event_type = 'engagement'
 GROUP BY 2
"""
data_from_sql = pd.read_sql_query(sql_query,con)
plt.figure(figsize=(10,5))
ax = sns.lineplot(x="month", y="monthly_users", data=data_from_sql)
plt.xticks(rotation=90)
```

Out[14]:
```
(array([4.5, 5. , 5.5, 6. , 6.5, 7. , 7.5, 8. , 8.5]),
 <a list of 9 Text xticklabel objects>)
```



In [15]:
```
# let's see the monthly user counts
data_from_sql.head()
```

Out[15]:

	monthly_users	month
0	2361	5.0
1	2605	6.0
2	3058	7.0
3	2795	8.0

In [16]:
```
# let's get the official drop off count for the boss.
print('active monthly user change between July and August 2014:',2795
-3058)
```

```
active monthly user change between July and August 2014: -263
```

From the monthly summary I see 263 less users engaged in August compared to July.

I'd like to label the users that stopped engaging between July and August.

This can be done using a SQL CASE statement.

Specifically, I want label user_id's that were active in July, BUT NOT active in August. Those will be my 'stopped' group. For comparison, I'll label the user_id's that were active in July, AND active in August. Those will be my 'engaged' group.

Then, I can do some comparisons to see what's distinct about the stopped group vs the engaged users.

To see how I built the following sub-sub-subquery, see the Appendix

In [17]:
```
# Perform a single SQL query to label my 'stopped' and 'engaged group
s'
sql_query = """
SELECT user_id,
    CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
         WHEN jul > 0 AND aug IS NULL THEN 'stopped'
         ELSE NULL END AS target_group
FROM (
    SELECT user_id,
           SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NULL E
ND) AS may,
           SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NULL E
ND) AS jun,
           SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NULL E
ND) AS jul,
           SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NULL E
ND) AS aug
     FROM(
           SELECT user_id,
                  COUNT(event_name) AS monthly_events,
                  EXTRACT('month'  FROM occurred_at) AS month
            FROM events
            WHERE event_type = 'engagement'
            GROUP BY 1, 3
            ) sub
     GROUP BY 1
     ) monthly_engagement
"""
user_labels = pd.read_sql_query(sql_query,con)
user_labels.head()
```

Out[17]:

	user_id	target_group
0	4	stopped
1	8	stopped
2	11	engaged
3	17	engaged
4	19	stopped

In [18]:
```
# quick pandas groupby to confirm these dates match my previous data
 (expect 263)
user_labels.groupby('target_group')[['user_id']].count()
```

Out[18]:

	user_id
target_group	
engaged	1401
stopped	1657

In [19]:
```
1657 - 1401
```

Out[19]: 256

That's weird. Maybe I lost a few users along the way? I was expecting 263 - the earlier unique monthly count via SQL.

Maybe there were some new users that signed up along the way? That is, they would be engaged only in August, not in July. Something to investigate later, for now 256 is still in the ballpark based on the initial chart.

Let's see if anything differentiates these groups.

Back to Table of Contents

User Events - Interactions with Weekly Email Digest

First, confirm that this isn't a technical glitch from the service.

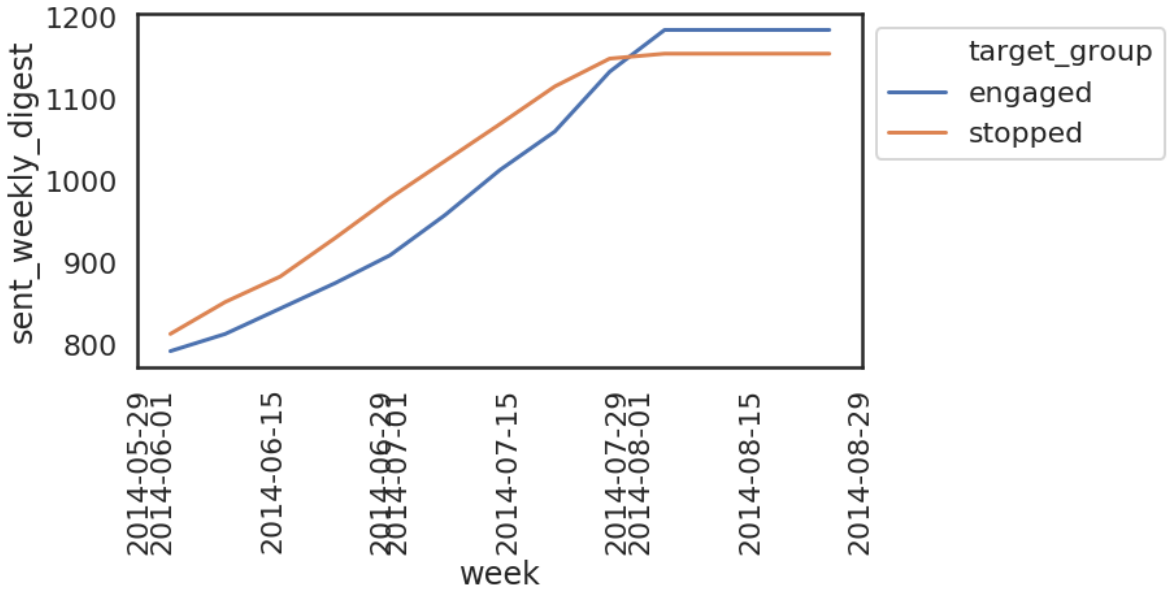
Is there any sign of a technial issue? Is the product getting delivered as expected? Where in the chain of events are the users not engaging? My understanding is that the chain of events should look like:

sent_weekly_digest -> email_open -> clickthrough

In [20]:
```
# join the labels to the events table, and generate summary of weekly
sent emails by group.

sql_query = """
SELECT labels.target_group,
    DATE_TRUNC('week', emails.occurred_at) AS week,
    COUNT(DISTINCT(emails.user_id)) AS sent_weekly_digest
FROM (
    SELECT user_id,
        CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
             WHEN jul > 0 AND aug IS NULL THEN 'stopped'
             ELSE NULL END AS target_group
    FROM (
        SELECT user_id,
                SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NU
LL END) AS may,
                SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NU
LL END) AS jun,
                SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NU
LL END) AS jul,
                SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NU
LL END) AS aug
         FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
         GROUP BY 1
         ) sub_sub
) labels
LEFT JOIN emails
 ON labels.user_id = emails.user_id
 WHERE target_group IS NOT NULL
 AND action = 'sent_weekly_digest'
 AND occurred_at > '2014-06-01'
 GROUP BY 1,2
"""
sent = pd.read_sql_query(sql_query,con)
# plot
plt.figure(figsize=(10,5))
ax = sns.lineplot(x="week", y="sent_weekly_digest", hue = 'target_gro
up', data=sent)
plt.xticks(rotation=90)
plt.legend(bbox_to_anchor=(1, 1))
```

Out[20]: <matplotlib.legend.Legend at 0x7f8fb6bd7160>



This chart is a little confusing- it looks like the 'engaged' got more emails than the 'stopped'. That might be true, but that's because the engaged group, as I've defined it, acquired additional unique users through August. That agrees with differences I saw in monthly unique users vs July AND August unique users.

The take away here is that the 'stopped' group didn't see a drop in email deliveries.

There isn't any indication to suggest this is a technical malfunction on the delivery side.

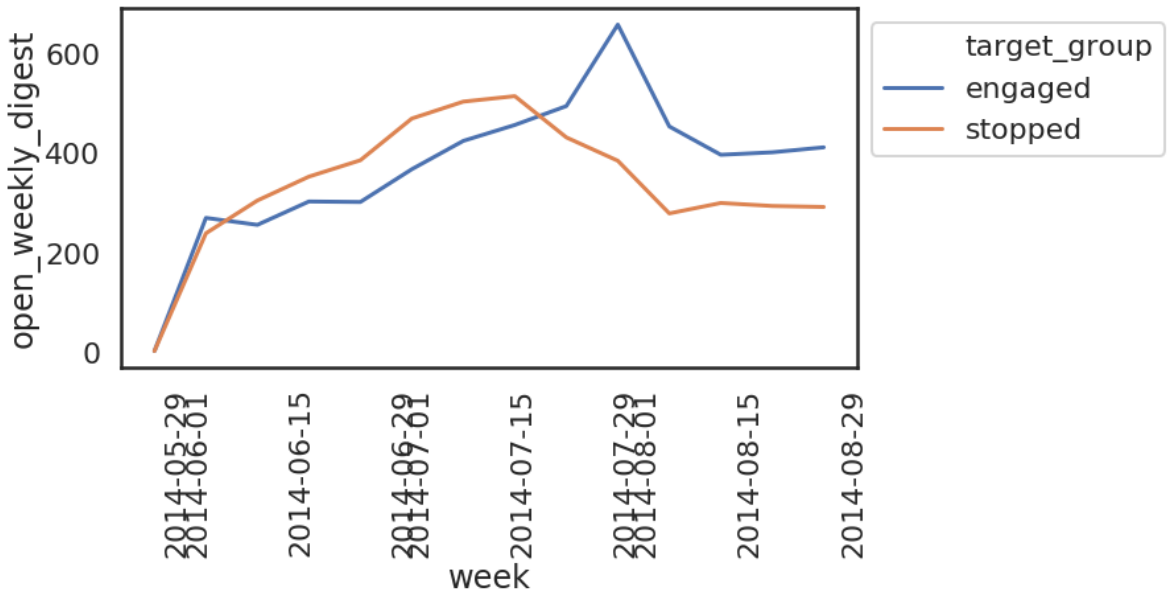
Let's see if both groups actually opened their emails.

In [21]:
```
# join the labels to the events table, and generate summary of weekly
open emails by group.

sql_query = """
SELECT labels.target_group,
    DATE_TRUNC('week', emails.occurred_at) AS week,
    COUNT(DISTINCT(emails.user_id)) AS open_weekly_digest
FROM (
    SELECT user_id,
        CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
             WHEN jul > 0 AND aug IS NULL THEN 'stopped'
             ELSE NULL END AS target_group
    FROM (
        SELECT user_id,
                SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NU
LL END) AS may,
                SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NU
LL END) AS jun,
                SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NU
LL END) AS jul,
                SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NU
LL END) AS aug
         FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
         GROUP BY 1
         ) sub_sub
) labels
LEFT JOIN emails
 ON labels.user_id = emails.user_id
 WHERE target_group IS NOT NULL
 AND action = 'email_open'
 AND occurred_at > '2014-06-01'
 GROUP BY 1,2
"""
device = pd.read_sql_query(sql_query,con)

plt.figure(figsize=(10,5))
ax = sns.lineplot(x="week", y="open_weekly_digest", hue = 'target_gro
up', data=device)
plt.xticks(rotation=90)
plt.legend(bbox_to_anchor=(1, 1))
```

Out[21]: <matplotlib.legend.Legend at 0x7f8fb6c59320>



Well, this looks interesting.

it definitely matches the pattern I see in the initial problem. I think this is it.

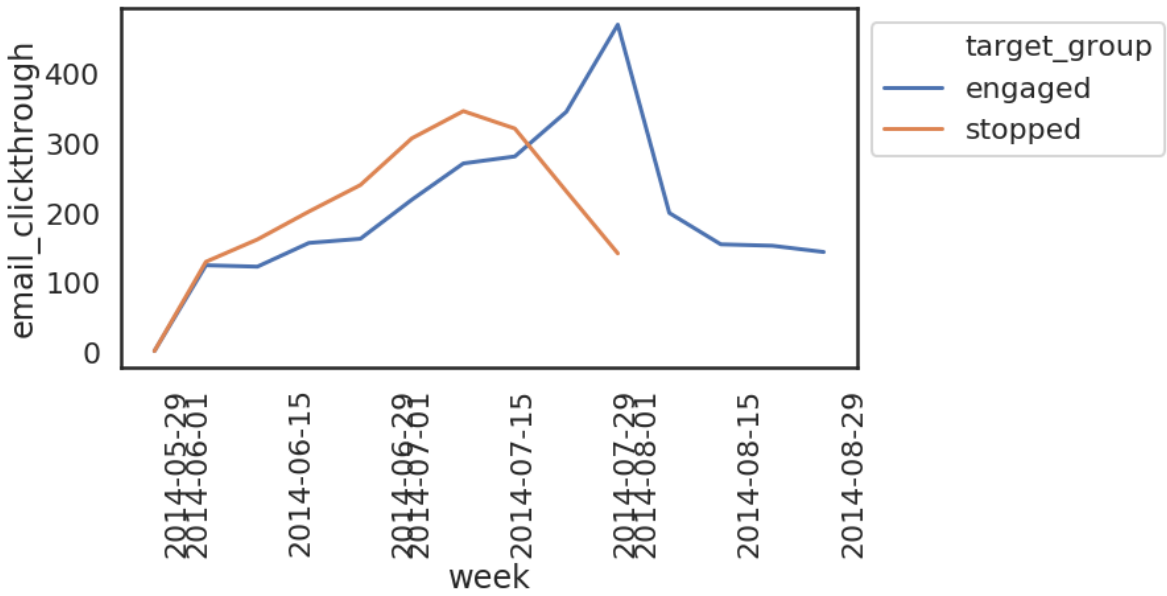
The 'stopped' group isn't opening the weekly digest. I can look at the click through rate, but it will probably look like this.

In [22]:
```
# join the labels to the events table, and generate summary of weekly
clickthroughs by group.

sql_query = """
SELECT labels.target_group,
    DATE_TRUNC('week', emails.occurred_at) AS week,
    COUNT(DISTINCT(emails.user_id)) AS email_clickthrough
FROM (
    SELECT user_id,
        CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
             WHEN jul > 0 AND aug IS NULL THEN 'stopped'
             ELSE NULL END AS target_group
    FROM (
        SELECT user_id,
                SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NU
LL END) AS may,
                SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NU
LL END) AS jun,
                SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NU
LL END) AS jul,
                SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NU
LL END) AS aug
         FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
         GROUP BY 1
         ) sub_sub
) labels
LEFT JOIN emails
 ON labels.user_id = emails.user_id
 WHERE target_group IS NOT NULL
 AND action = 'email_clickthrough'
 AND occurred_at > '2014-06-01'
 GROUP BY 1,2
"""
device = pd.read_sql_query(sql_query,con)

plt.figure(figsize=(10,5))
ax = sns.lineplot(x="week", y="email_clickthrough", hue = 'target_gro
up', data=device)
plt.xticks(rotation=90)
plt.legend(bbox_to_anchor=(1, 1))
```

Out[22]: <matplotlib.legend.Legend at 0x7f8fb59c0160>



Clickthrough

I think this is more of a product of how I picked my groups rather than being causal. Still, that's nice to see my SQL commands worked as expected.

I think the story might go like this:

Users in the 'stopped' group are still receiving emails as expected, but they are not opening them compared to the 'engaged' group.

And if they do open them they aren't clicking through to the platform.

This answers WHAT is happening, but not the WHO or WHY.

Who are they, where are they, and what are they using to engage?

I'll answer those next.

[Back to Table of Contents]

User Profile - Engagement comparison by Location, Device, and Company

I have an idea of what's happening, but I don't know why. To answer this, I'll need to explore some of the other user-related features available.

In [23]:
```
# first, let's keep it contained to the events table.
# I'm going to self-join with the labels I made, generate a df and do
some EDA.

sql_query = """
SELECT labels.*, events.*
FROM (
    SELECT user_id,
        CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
             WHEN jul > 0 AND aug IS NULL THEN 'stopped'
             ELSE NULL END AS target_group
    FROM (
        SELECT user_id,
                SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NU
LL END) AS may,
                SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NU
LL END) AS jun,
                SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NU
LL END) AS jul,
                SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NU
LL END) AS aug
         FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
         GROUP BY 1
         ) sub_sub
) labels
LEFT JOIN events
 ON labels.user_id = events.user_id
 WHERE target_group IS NOT NULL
"""
target_groups = pd.read_sql_query(sql_query,con)
target_groups.head()
```

Out[23]:

	user_id	target_group	index	user_id	occurred_at	event_type	event_name	location
0	10612	engaged	6	10612	2014-05-01 09:59:46	engagement	login	Netherlands
1	10612	engaged	7	10612	2014-05-01 10:00:18	engagement	like_message	Netherlands
2	10612	engaged	8	10612	2014-05-01 10:00:53	engagement	send_message	Netherlands
3	10612	engaged	9	10612	2014-05-01 10:01:24	engagement	home_page	Netherlands
4	10612	engaged	10	10612	2014-05-01 10:01:52	engagement	like_message	Netherlands

Awesome, from here I can test my hypotheses 2 and 3 to get an idea of what distinguishes the users who stopped.

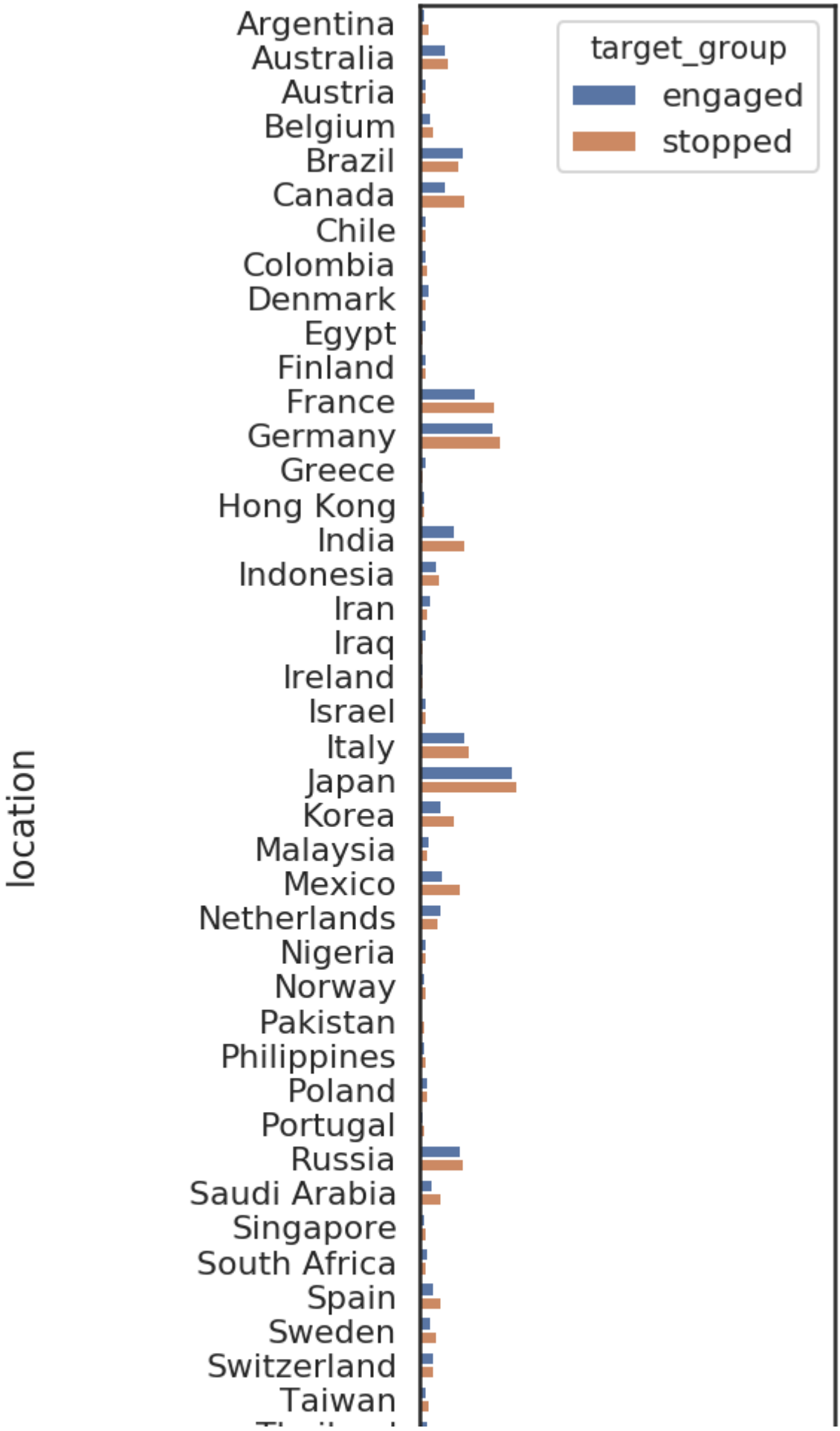
My initial thoughts are to explore location and device.

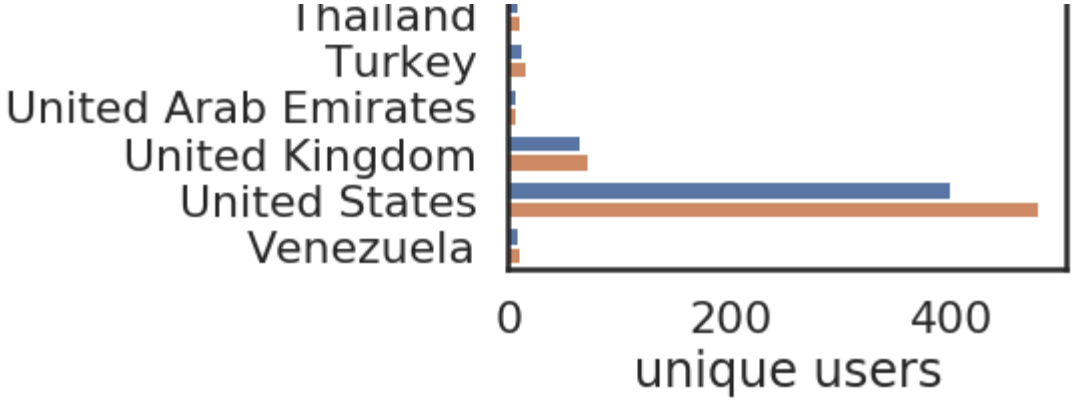
In [24]:
```
# Check engagement by user location (country)

sql_query = """
SELECT labels.target_group, COUNT(DISTINCT events.user_id), events.lo
cation
FROM (
    SELECT user_id,
        CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
             WHEN jul > 0 AND aug IS NULL THEN 'stopped'
             ELSE NULL END AS target_group
    FROM (
        SELECT user_id,
                SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NU
LL END) AS may,
                SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NU
LL END) AS jun,
                SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NU
LL END) AS jul,
                SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NU
LL END) AS aug
         FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
         GROUP BY 1
         ) sub_sub
) labels
LEFT JOIN events
 ON labels.user_id = events.user_id
 WHERE target_group IS NOT NULL
 GROUP BY labels.target_group, events.location
"""
loc = pd.read_sql_query(sql_query,con)

# plot it
plt.figure(figsize=(5,20))
ax = sns.barplot(x="count", y="location", hue="target_group", data=lo
c)
ax.set(xlabel='unique users')
```

Out[24]: [Text(0.5, 0, 'unique users')]


target_group
engaged
stopped
location
Argentina
Australia
Austria
Belgium
Brazil
Canada
Chile
Colombia
Denmark
Egypt
Finland
France
Germany
Greece
Hong Kong
India
Indonesia
Iran
Iraq
Ireland
Israel
Italy
Japan
Korea
Malaysia
Mexico
Netherlands
Nigeria
Norway
Pakistan
Philippines
Poland
Portugal
Russia
Saudi Arabia
Singapore
South Africa
Spain
Sweden
Switzerland
Taiwan



To me, it looks like a lot of people in the northern hemisphere stopped engaging.

US, Canada, France, Germany, Mexico stand out, but there's others too.

It would be summer time in those countries. Maybe they went on vacation?

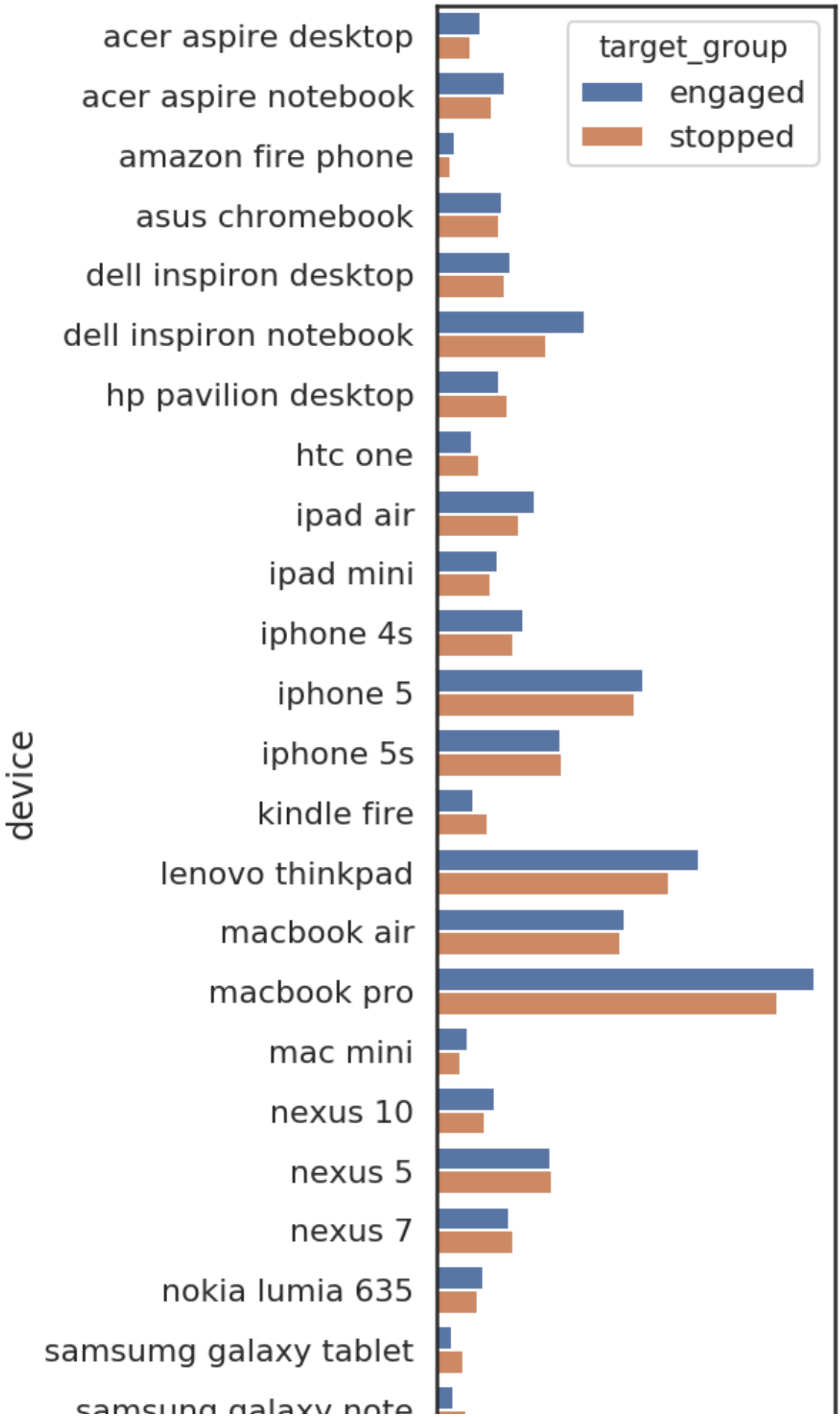
Let's check the device breakdown. Maybe it's a technical issue (app update, website crash, etc).

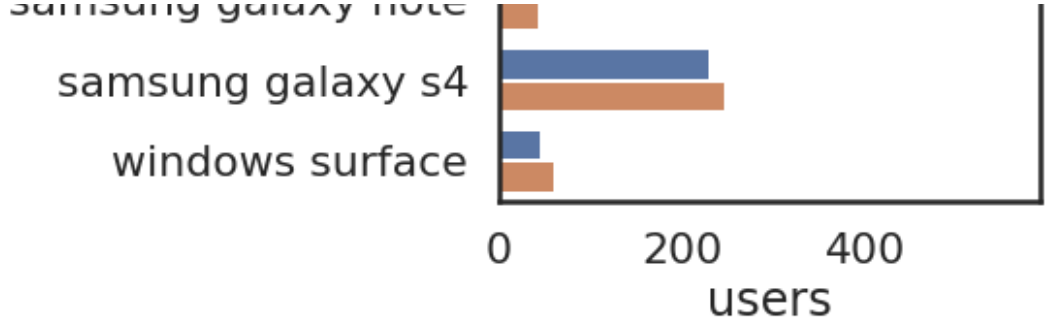
In [25]:
```
# Check engagement by device

sql_query = """
SELECT labels.target_group, COUNT(DISTINCT events.user_id), events.de
vice
FROM (
    SELECT user_id,
        CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
             WHEN jul > 0 AND aug IS NULL THEN 'stopped'
             ELSE NULL END AS target_group
    FROM (
        SELECT user_id,
                SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NU
LL END) AS may,
                SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NU
LL END) AS jun,
                SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NU
LL END) AS jul,
                SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NU
LL END) AS aug
         FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
         GROUP BY 1
         ) sub_sub
) labels
LEFT JOIN events
 ON labels.user_id = events.user_id
 WHERE target_group IS NOT NULL
 GROUP BY labels.target_group, events.device
"""
device = pd.read_sql_query(sql_query,con)

# plot it
plt.figure(figsize=(5,20))
ax = sns.barplot(x="count", y="device", hue="target_group", data=devi
ce)
ax.set(xlabel='users')
```

Out[25]: [Text(0.5, 0, 'users')]


target_group
engaged
stopped
device
acer aspire desktop
acer aspire notebook
amazon fire phone
asus chromebook
dell inspiron desktop
dell inspiron notebook
hp pavilion desktop
htc one
ipad air
ipad mini
iphone 4s
iphone 5
iphone 5s
kindle fire
lenovo thinkpad
macbook air
macbook pro
mac mini
nexus 10
nexus 5
nexus 7
nokia lumia 635
samsumg galaxy tablet



I don't see a large difference by device between stopped and engaged users.

This suggests to me there's not a technical issue with the platforms (website, iphone app, android app, etc).

In general, there seems to a large number of macbook pro, macbook air, and lenovo thinkpad users who stopped engaging. There's also a drop in iphone and galaxy s4 users.

Those are more modern devices... I want to test whether this drop in engagement is specific to a single or group of companies.

In [26]:
```
# Check engagement by company

sql_query = """
SELECT labels.target_group, users.company_id, COUNT(DISTINCT(users.us
er_id))
FROM (
    SELECT user_id,
        CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
             WHEN jul > 0 AND aug IS NULL THEN 'stopped'
             ELSE NULL END AS target_group
    FROM (
        SELECT user_id,
                SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NU
LL END) AS may,
                SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NU
LL END) AS jun,
                SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NU
LL END) AS jul,
                SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NU
LL END) AS aug
         FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
         GROUP BY 1
         ) sub_sub
) labels
LEFT JOIN users
 ON labels.user_id = users.user_id
 WHERE target_group IS NOT NULL
 GROUP BY 1,2
"""
company = pd.read_sql_query(sql_query,con)
company.head()
```

Out[26]:

	target_group	company_id	count
0	engaged	1	78
1	engaged	2	37
2	engaged	3	25
3	engaged	4	21
4	engaged	5	14

In [27]:
```
# I tried plotting this and my computer almost crashed. Let's see how
many companies we have
print('unique companies:', company.company_id.nunique())
```

```
unique companies: 2301
```

In [28]:
```
# OK, I'm going to take a different approach. I'm going to pivot the
 users/co by target_group.
# then I'm going to sum to get the total, and then calc the % stopped
per company.
# I want to know if this is specific to a single company, or distribu
ted.

company = company.pivot(index = 'company_id', columns='target_group',
values='count')
company = company.reset_index(drop=True)
company['total'] = company.engaged + company.stopped
company['pct_stopped'] = 100*company.stopped / company.total
company.sort_values(by=['pct_stopped'], inplace=True, ascending=False
)
company.head()
```

Out[28]:

target_group	engaged	stopped	total	pct_stopped
11	2.0	9.0	11.0	81.818182
30	1.0	4.0	5.0	80.000000
38	1.0	4.0	5.0	80.000000
37	1.0	3.0	4.0	75.000000
35	1.0	3.0	4.0	75.000000

In [29]:
```
# let's look at the stats on the per company % stopped.
company.pct_stopped.describe()
```

Out[29]:
```
count    82.000000
mean     52.218206
std      13.896200
min      25.000000
25%      50.000000
50%      50.000000
75%      60.833333
max      81.818182
Name: pct_stopped, dtype: float64
```

Summary stats indicate that within the companies that saw some loss of engagement, the lowest is 25%, and the highest is 82%.

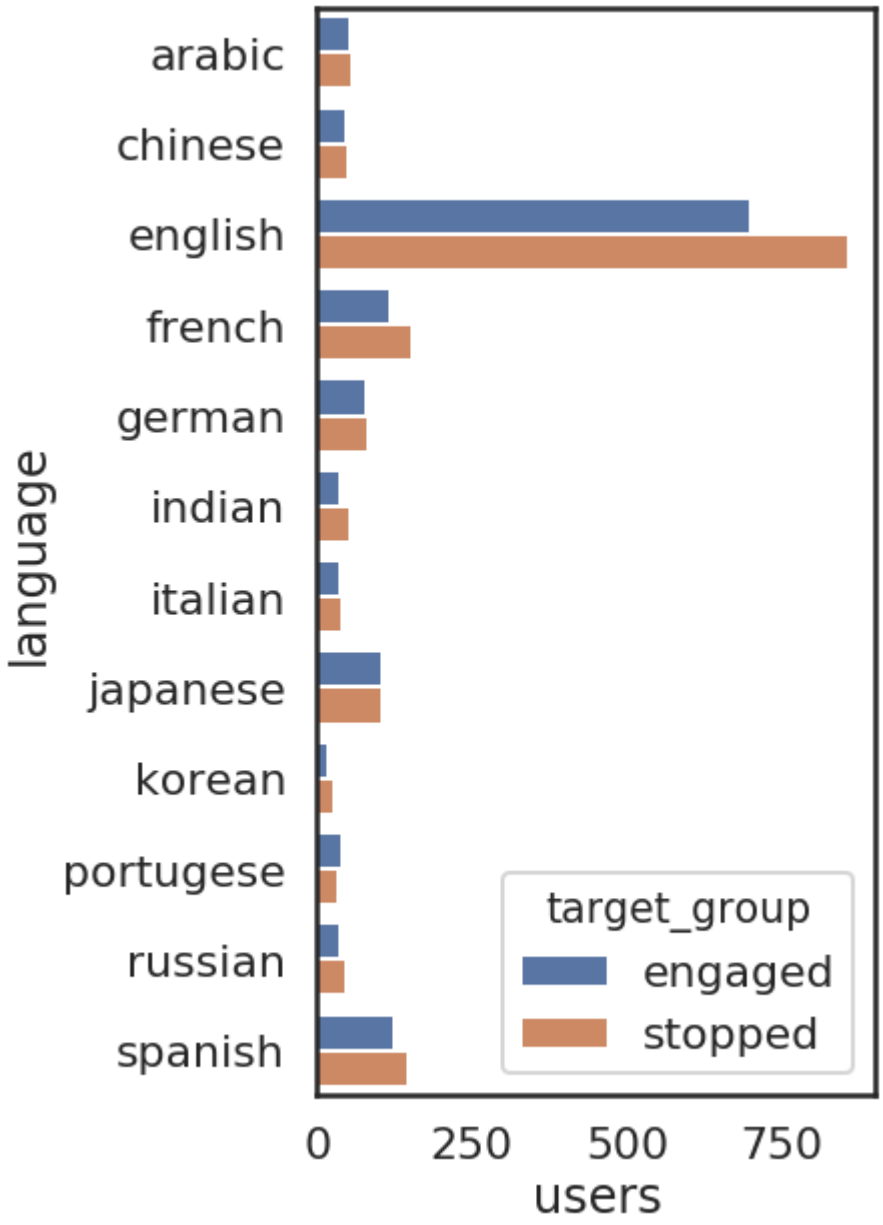
My take home from this is the issue is not specific to a single company / group of companies.

In [30]:
```
# Check engagement by language

sql_query = """
SELECT labels.target_group, users.language, COUNT(DISTINCT(users.user
_id))
FROM (
    SELECT user_id,
        CASE WHEN jul > 0 AND aug IS NOT NULL THEN 'engaged'
             WHEN jul > 0 AND aug IS NULL THEN 'stopped'
             ELSE NULL END AS target_group
    FROM (
        SELECT user_id,
                SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NU
LL END) AS may,
                SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NU
LL END) AS jun,
                SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NU
LL END) AS jul,
                SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NU
LL END) AS aug
         FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
         GROUP BY 1
         ) sub_sub
) labels
LEFT JOIN users
 ON labels.user_id = users.user_id
 WHERE target_group IS NOT NULL
 GROUP BY 1,2
"""
lang = pd.read_sql_query(sql_query,con)

# plot it
plt.figure(figsize=(5,10))
ax = sns.barplot(x="count", y="language", hue="target_group", data=la
ng)
ax.set(xlabel='users')
```

Out[30]: [Text(0.5, 0, 'users')]



It's predominately English speakers who have stopped engaging, although there's also a large proportion of french and spanish, and japanese. This aligns with the country data seen earlier.

To summarize:

1) a majority of the 'stopped' users are in northern hemisphere countries. 2) the 'stopped' users aren't specific to any specific device. 3) the 'stopped' users are receiving the weekly digest, but they're not opening it.

These users are likely on vacation.

Recommendations:

1) check historical records to test if this is a seasonal trend. 2) give heads up to product team that emails aren't being opened. Might be caused by a change in inbox spam filters.

Table of Contents

Appendix

I built my big sub-sub-subquery iterately. Here's how:

In [31]:
```
# first, let's get the number of events, per user, per month
sql_query = """
SELECT user_id,
        COUNT(event_name) AS monthly_events,
        EXTRACT('month'  FROM occurred_at) AS month
 FROM events
 WHERE event_type = 'engagement'
 GROUP BY 1, 3
"""
data_from_sql = pd.read_sql_query(sql_query,con)
data_from_sql.head()
```

Out[31]:

	user_id	monthly_events	month
0	4	41	5.0
1	4	38	6.0
2	4	14	7.0
3	8	29	5.0
4	8	7	7.0

In [32]:
```
# next, I'm going to pivot the monthly events to columns doing a sub-
query
# there's a great example of this in the advanced SQL section on MODE
analytics.

sql_query = """
SELECT user_id,
        SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NULL END)
 AS may,
        SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NULL END)
 AS jun,
        SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NULL END)
 AS jul,
        SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NULL END)
 AS aug
 FROM(
        SELECT user_id,
                COUNT(event_name) AS monthly_events,
                EXTRACT('month'  FROM occurred_at) AS month
         FROM events
         WHERE event_type = 'engagement'
         GROUP BY 1, 3
         ) sub
 GROUP BY 1
"""
data_from_sql = pd.read_sql_query(sql_query,con)
data_from_sql.head()
```

Out[32]:

	user_id	may	jun	jul	aug
0	4	41.0	38.0	14.0	NaN
1	8	29.0	NaN	7.0	NaN
2	11	NaN	64.0	37.0	25.0
3	17	NaN	NaN	23.0	32.0
4	19	NaN	56.0	15.0	NaN

In [33]:
```
# OK, I can use this table to create the labels for the 'stopped' and
'engaged' groups,
# but first make sure the sub-subquery works

sql_query = """
SELECT *
FROM (
    SELECT user_id,
            SUM(CASE WHEN month = 5.0 THEN monthly_events ELSE NULL E
ND) AS may,
            SUM(CASE WHEN month = 6.0 THEN monthly_events ELSE NULL E
ND) AS jun,
            SUM(CASE WHEN month = 7.0 THEN monthly_events ELSE NULL E
ND) AS jul,
            SUM(CASE WHEN month = 8.0 THEN monthly_events ELSE NULL E
ND) AS aug
     FROM(
            SELECT user_id,
                    COUNT(event_name) AS monthly_events,
                    EXTRACT('month'  FROM occurred_at) AS month
             FROM events
             WHERE event_type = 'engagement'
             GROUP BY 1, 3
             ) sub
     GROUP BY 1
     ) monthly_engagement
 WHERE jul > 0 AND aug IS NULL
"""
stopped = pd.read_sql_query(sql_query,con)
stopped.head()
```

Out[33]:

	user_id	may	jun	jul	aug
0	4	41.0	38.0	14.0	None
1	8	29.0	NaN	7.0	None
2	19	NaN	56.0	15.0	None
3	171	2.0	NaN	10.0	None
4	172	65.0	32.0	21.0	None

last steps is to do the sub-sub-subquery, which is done in the

User Events and User Profiles sections