# rtweet パッケージのあれこレシピ

@anemptyarchive*

2020/05/11-2020/05/28

## Contents



## ・ツイート収集

まずは `rtweet` パッケージを利用して、ツイートを集めます。

```
# 利用パッケージ
library(rtweet)

# アカウントを指定
screen_name <- "anemptyarchive"
```

*https://www.anarchive-beta.com/

```
# アカウント指定でツイートを収集
tw_data <- get_timeline(screen_name, n = 10000, include_rts = FALSE)
```

get_timeline() にアカウント (@*** の ***) を指定して、ツイートを取得します。引数 n は収集するツイート数、include_rts は取得ツイートにリツイートを含めるかです (デフォルトは TRUE)。

特定の単語を含むツイートを収集するのであれば、search_tweets("検索ワード") を使います。

どちらも取得できるツイートの数に制限があったりします。

## ・ツイート数を棒グラフで可視化

rtweet パッケージを利用して、指定したアカウントのタイムラインインを収集し、設定した期間 (年・月・日・時) ごとのツイート数を集計し、棒グラフによる可視化を行います。

```
library(rtweet) # ツイート収集:get_timeline(), search_tweets()
library(dplyr) # データフレーム操作
library(lubridate) # 時間データ操作:floor_date(), as_date()
library(ggplot2) # 作図
```

利用するパッケージを読み込みます。

### ・前処理

ツイート日時の情報があればいいので、取得したツイートデータのうち created_at 列のみを使います。それでは created_at 列のデータを確認してみましょう。

```
tw_data[["created_at"]][1:10]
```

```
##  [1] "2020-05-24 15:48:56 UTC" "2020-05-24 01:33:57 UTC"
##  [3] "2020-05-23 14:55:20 UTC" "2020-05-23 14:45:54 UTC"
##  [5] "2020-05-23 14:35:50 UTC" "2020-05-23 14:15:49 UTC"
##  [7] "2020-05-23 14:12:26 UTC" "2020-05-23 13:59:04 UTC"
##  [9] "2020-05-23 10:33:35 UTC" "2020-05-23 10:26:21 UTC"
```

取得したツイート日時のタイムゾーンが協定世界時 (UTC) です。これを日本標準時 (JST) に変換にする必要があります。

```
# ツイート日時データを変換
tw_time <- tw_data[["created_at"]] %>% # ツイート日時を抽出
  as.POSIXct(tz = "Etc/GMT") %>% # POSIXct 型の協定世界時を明示
  as.POSIXlt(tz = "Asia/Tokyo") # POSIXlt 型の日本標準時に変換
```

as.POSIXlt() で POSIXct 型から POSIXlt 型に変換します。また tz 引数に"Japan" あるいは"Asia/Tokyo" を指定して、タイムゾーンを日本標準時に変更します。(多分 2 行目はいらない)

変換後のデータを確認しておきましょう。

```
tw_time[1:10]
```

```
##  [1] "2020-05-25 00:48:56 JST" "2020-05-24 10:33:57 JST"
##  [3] "2020-05-23 23:55:20 JST" "2020-05-23 23:45:54 JST"
##  [5] "2020-05-23 23:35:50 JST" "2020-05-23 23:15:49 JST"
##  [7] "2020-05-23 23:12:26 JST" "2020-05-23 22:59:04 JST"
##  [9] "2020-05-23 19:33:35 JST" "2020-05-23 19:26:21 JST"
```

これでツイートデータの取得と前処理は完了です。次からは可視化のための処理を行っていきます。

### ・年・月・日別に可視化

ここからは、指定した期間 (年・月・日) ごとにツイート数を集計し、棒グラフを作図していきます。

```
# セルの単位（区切る期間）を指定
term <- "year"
term <- "mon"
term <- "day"
```

ツイート数をカウントする際に区切る間隔を指定します。「年」単位であれば year、「月」単位なら mon、「日」単位なら day とします。これは主に floor_date() の unit 引数に指定するためのものなので、このままの文字列を使用してください。

指定した期間ごとのツイートを数を集計します。

```
# 指定した期間ごとにツイート数を集計
tw_count <- tw_time %>%
  floor_date(unit = term) %>% # 指定した単位に切り捨て
  as_date() %>% # Date 型に変換
  tibble(terms = .) %>% # データフレームに変換
  count(terms) # ツイート数をカウント
```

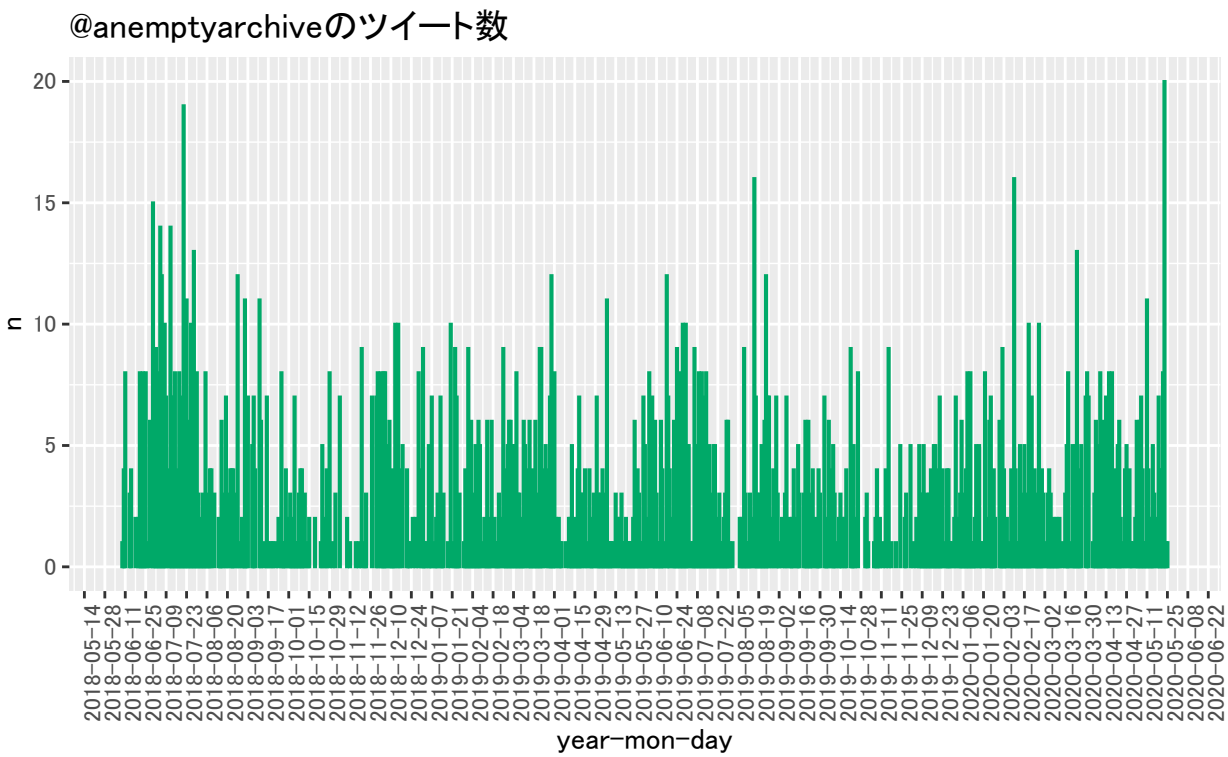
floor_date() で指定した期間ごとに日時を丸め (切り捨て) ます。
また時刻情報も不要なので、as_date() で POSIXlt 型から Date 型に変換することで落とします。
次に tibble() でデータフレームに変換して、count() で各期間のツイート数を集計します。

```
tail(tw_count, 10)
```

```
## # A tibble: 10 x 2
##    terms          n
##    <date>     <int>
##  1 2020-05-15     5
##  2 2020-05-16     3
##  3 2020-05-18     2
##  4 2020-05-19     7
##  5 2020-05-20     4
##  6 2020-05-21     4
##  7 2020-05-22     8
##  8 2020-05-23    20
##  9 2020-05-24     1
## 10 2020-05-25     1
```

このデータフレームを用いて作図します。作図の処理は指定した期間によって異なるため、場合分けしておきます。

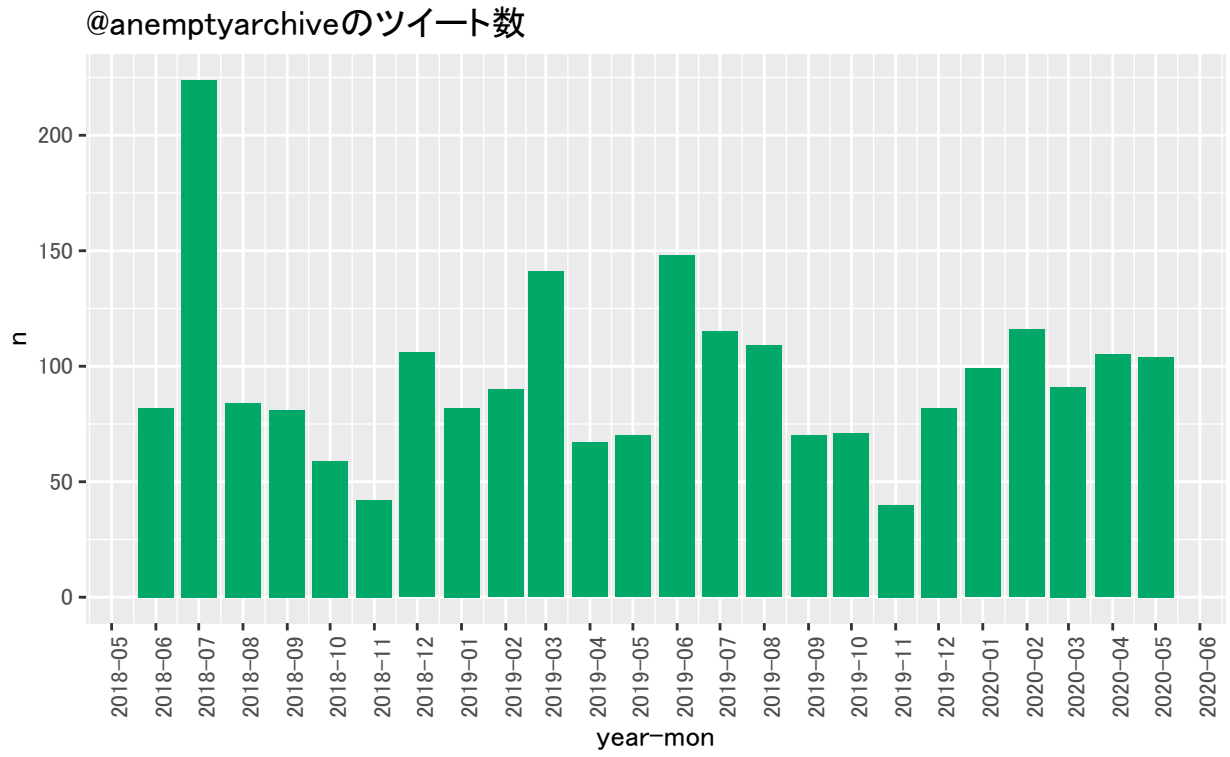
```
# 棒グラフを作図
if(term == "day") {

  ggplot(tw_count, aes(x = terms, y = n)) +
    geom_bar(stat = "identity", fill = "#00A968", color = "#00A968") + # 棒グラフ
    scale_x_date(date_breaks = "2 weeks", date_labels = "%Y-%m-%d") + # x軸目盛（日付）
    theme(axis.text.x = element_text(angle = 90)) + # x軸目盛の傾き
    labs(title = paste0("@", screen_name, "のツイート数"),
         x = "year-mon-day") # ラベル

} else if(term == "mon") {

  ggplot(tw_count, aes(x = terms, y = n)) +
    geom_bar(stat = "identity", fill = "#00A968") + # 棒グラフ
    scale_x_date(date_breaks  = "1 month", date_labels = "%Y-%m") + # x軸目盛（日付）
    theme(axis.text.x = element_text(angle = 90)) + # x軸目盛の傾き
    labs(title = paste0("@", screen_name, "のツイート数"),
         x = "year-mon") # ラベル

} else if(term == "year") {

  ggplot(tw_count, aes(x = terms, y = n)) +
    geom_bar(stat = "identity", fill = "#00A968", color = "#00A968") + # 棒グラフ
    scale_x_date(date_breaks = "1 year", date_labels = "%Y") + # x軸目盛（日付）
    labs(title = paste0("@", screen_name, "のツイート数"),
         x = "year") # ラベル

}
```



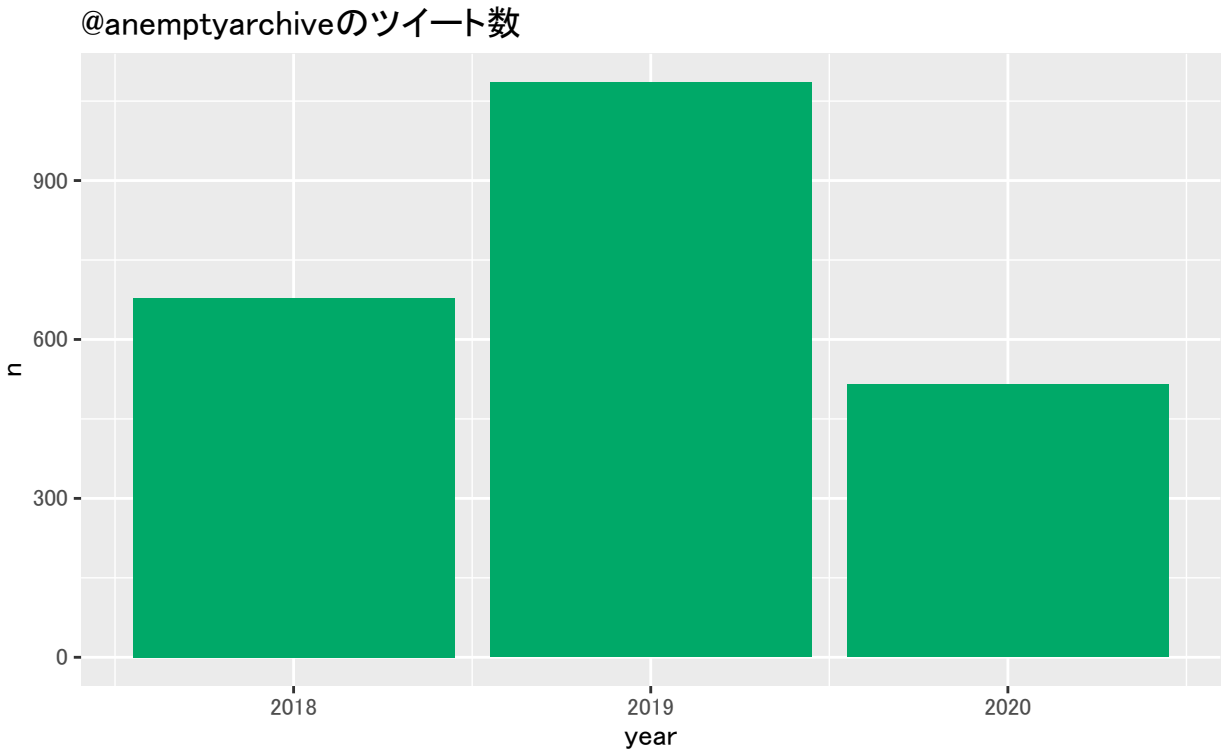
`geom_bar()` で棒グラフを作図します。

x 軸に表示するデータ数が多くなるため、必要に応じて x 軸目盛を表示する位置を間引きます。
Date 型のデータの場合は、`scale_x_date()` で x 軸目盛を調整できます。`date_breaks` 引数に表示する位置を、`date_labels` 引数に表示するテキストの形式を指定します。

・月別




・年別




### ・時別に可視化

続いて、1 時間ごとにツイート数を棒グラフ化します。
基本的な処理は同じです。こちらは可視化に時刻データを用いるため、ツイート日時を POSIXct 型で扱うところが異なります。

```
# セルの単位（区切る期間）を指定
term <- "hour"
```

区分けするときに用いる変数を"hour" としておきます。

```
# 指定した期間ごとにツイート数を集計
tw_count <- tw_time %>%
  floor_date(unit = term) %>% # 指定した単位に切り捨て
  as.POSIXct() %>% # POSIXct 型に変換
  tibble(terms = .) %>% # データフレームに変換
  count(terms) %>% # ツイート数をカウント
  filter(terms >= as.POSIXct("2020-01-01")) %>% # から
  filter(terms <= as.POSIXct("2020-01-31")) # まで
```

floor_date(., unit = "hour") で 1 時間単位で丸めて (切り捨て)、as.POSIXct() で POSIXct 型に戻します。
また tibble() でデータフレームに変換して、count() で各期間のツイート数を集計します。
描画するデータ量が多くなると思われるので、描画する範囲を絞っておきます。

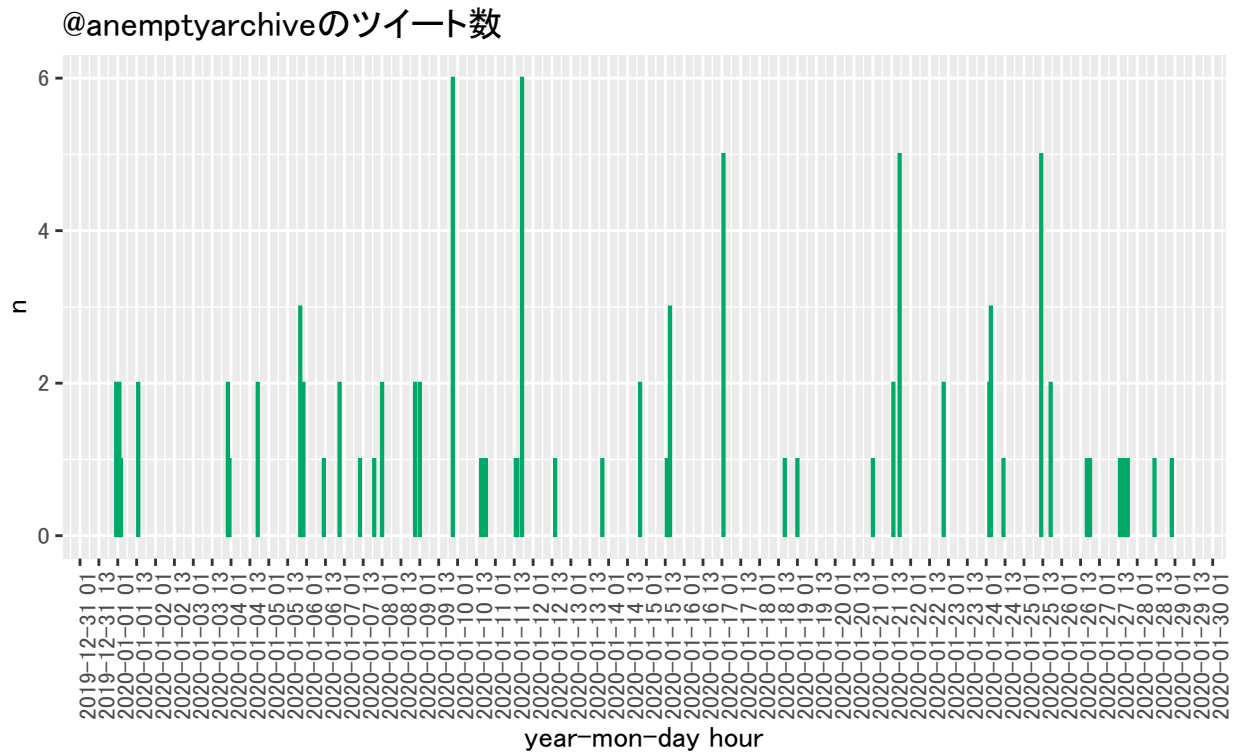
では、完成したデータフレームを確認しておきましょう。

```
tail(tw_count, 10)
```

```
## # A tibble: 10 x 2
##    terms                   n
##    <dttm>              <int>
##  1 2020-01-25 12:00:00     5
##  2 2020-01-25 18:00:00     2
##  3 2020-01-26 17:00:00     1
##  4 2020-01-26 19:00:00     1
##  5 2020-01-27 14:00:00     1
##  6 2020-01-27 16:00:00     1
##  7 2020-01-27 18:00:00     1
##  8 2020-01-27 19:00:00     1
##  9 2020-01-28 12:00:00     1
## 10 2020-01-28 23:00:00     1
```

これを用いて作図します。

```
# 棒グラフを作図
ggplot(tw_count, aes(x = terms, y = n)) +
  geom_bar(stat = "identity", fill = "#00A968", color = "#00A968") + # 棒グラフ
  scale_x_datetime(date_breaks = "12 hours",
                   date_labels = "%Y-%m-%d %H") + # x軸目盛（日時）
  theme(axis.text.x = element_text(angle = 90)) + # x軸目盛の傾き
  labs(title = paste0("@", screen_name, "のツイート数"),
       x = "year-mon-day hour") # ラベル
```




POSIXct 型のデータのときは、scale_x_datetime() で x 軸目盛を調整できます。引数は scale_x_date() と同じく、date_breaks 引数に表示する位置、date_labels 引数に表示するテキストの形式を指定します。

## ・ツイート数のヒートマップを作成

rtweet パッケージを利用して、指定したアカウントのタイムラインインを収集し、設定した期間 (年・月・日・時) ごとのツイート数を集計し、ヒートマップによる可視化を行います。

```
# 利用パッケージ
library(rtweet) # ツイート収集:get_timeline()
library(dplyr) # データフレーム操作
library(tidyr) # データフレーム操作:replace_na()
library(lubridate) # 時間データ操作:floor_date(), as_date(), now()
library(ggplot2) # 作図
```

利用するパッケージを読み込みます。

### ・前処理

ツイート数の推移をみるにはツイート日時の情報があればいいので、取得したツイートデータのうち created_at 列のみを使います。それでは created_at 列のデータを確認してみましょう。

```
tw_data[["created_at"]][1:10]
```

```
##  [1] "2020-05-24 15:48:56 UTC" "2020-05-24 01:33:57 UTC"
##  [3] "2020-05-23 14:55:20 UTC" "2020-05-23 14:45:54 UTC"
##  [5] "2020-05-23 14:35:50 UTC" "2020-05-23 14:15:49 UTC"
##  [7] "2020-05-23 14:12:26 UTC" "2020-05-23 13:59:04 UTC"
##  [9] "2020-05-23 10:33:35 UTC" "2020-05-23 10:26:21 UTC"
```

取得したツイート日時のタイムゾーンが協定世界時 (UTC) です。これを日本標準時 (JST) に変換にする必要があります。

```
# ツイート日時データを変換
tw_time <- tw_data[["created_at"]] %>% # ツイート日時を抽出
  as.POSIXct(tz = "Etc/GMT") %>% # POSIXct 型の協定世界時を明示
  as.POSIXlt(tz = "Asia/Tokyo") # POSIXlt 型の日本標準時に変換
```

as.POSIXlt() で POSIXct 型から POSIXlt 型に変換します。また tz 引数に"Japan" あるいは"Asia/Tokyo" を指定して、タイムゾーンを日本標準時に変更します。(多分 2 行目はいらない)

変換後のデータを確認しておきましょう。

```
tw_time[1:10]
```

```
##  [1] "2020-05-25 00:48:56 JST" "2020-05-24 10:33:57 JST"
##  [3] "2020-05-23 23:55:20 JST" "2020-05-23 23:45:54 JST"
##  [5] "2020-05-23 23:35:50 JST" "2020-05-23 23:15:49 JST"
##  [7] "2020-05-23 23:12:26 JST" "2020-05-23 22:59:04 JST"
##  [9] "2020-05-23 19:33:35 JST" "2020-05-23 19:26:21 JST"
```

これでツイートデータの取得と前処理は完了です。次からは可視化のための処理を行っていきます。

### ・年・月・日別に可視化

ここからは、指定した期間 (年・月・日) ごとにツイート数を集計し、ヒートマップを作図していきます。

```
# セルの単位（区切る期間）を指定
term <- "year"
term <- "mon"
term <- "day"
```

ヒートマップにする際のタイル (セル?) の単位 (期間) を指定します。「年」単位であれば year、「月」単位なら mon、「日」単位なら day とします。これは主に floor_date() の unit 引数に指定するためのものなので、このままの文字列を使用してください。

指定した期間ごとのツイートを数を集計します。

```
# 指定した期間ごとにツイート数を集計
tw_count1 <- tw_time %>%
  floor_date(unit = term) %>% # 指定した期間に切り捨て
  as_date() %>% # Date 型に変換
  tibble(terms = .) %>% # データフレームに変換
  count(terms) # ツイート数をカウント
```

floor_date() で指定した期間ごとに日時を丸めて (切り捨て) ます。
時刻情報も不要なので as_date() で POSIXlt 型から Date 型に変換することで落とします。
次に tibble() でデータフレームに変換して、count() で各期間のツイート数を集計します。

```
tail(tw_count1, 10)
```

```
## # A tibble: 10 x 2
##    terms          n
##    <date>     <int>
##  1 2020-05-15     5
##  2 2020-05-16     3
##  3 2020-05-18     2
##  4 2020-05-19     7
##  5 2020-05-20     4
##  6 2020-05-21     4
##  7 2020-05-22     8
##  8 2020-05-23    20
##  9 2020-05-24     1
## 10 2020-05-25     1
```

このままだと、ツイートがなかったタイミングが (ツイート数 0 ではなく) 抜け落ちてしまいます (5 月 2 日そのものがない)。あとで日時データを文字列型に変換して扱うため、ツイート数 0 の期間としておかないとグラフに表示されません。
その対処として、全ての期間が含まれるデータフレームを作成し、そこにツイート数の情報を移します。

```
# ツイートがない期間が欠落する対策
term_df <- seq(
  floor_date(tail(tw_time, 1), term), # 一番古い日時
```

```
  floor_date(now(), term), # 現在日時
  by = term
) %>% # 指定した期間刻みのベクトルを作成
  as_date() %>% # Date 型に変換
  tibble(terms = .)# データフレームに変換

# 集計結果を結合
tw_count2 <- left_join(term_df, tw_count1, by = "terms") %>%
  mutate(n = replace_na(n, 0)) # NA を 0 に置換
```

seq() で、一番古いツイート日時から現在の日時までのベクトルを作ります。このときベクトルの要素は、引数 by 指定した間隔となります。一番古いツイート日時は tw_time の最後の要素なので、tail() または tw_time[length(tw_time)] で取り出して渡します。現在時刻は now() で取得できます。
これを先ほどと同様に、日時データを Date 型に変換し、また全体をデータフレームに変換します。

これを受け皿として、left_join() で先ほどのデータフレームのツイート数と結合します。
このときツイートがなかった期間は NA となるので、それを mutate() と replace_na() を使って 0 に置換します。

集計結果を確認しましょう。

```
tail(tw_count2, 10)
```

```
## # A tibble: 10 x 2
##    terms          n
##    <date>     <dbl>
##  1 2020-05-19     7
##  2 2020-05-20     4
##  3 2020-05-21     4
##  4 2020-05-22     8
##  5 2020-05-23    20
##  6 2020-05-24     1
##  7 2020-05-25     1
##  8 2020-05-26     0
##  9 2020-05-27     0
## 10 2020-05-28     0
```

5 月 2 日のツイート数が 0 という行が含まれていることが確認できます。

次はこのデータフレームを作図用に加工します。ただし、指定した期間によって処理が多少変わるため、if() で場合分けしています。

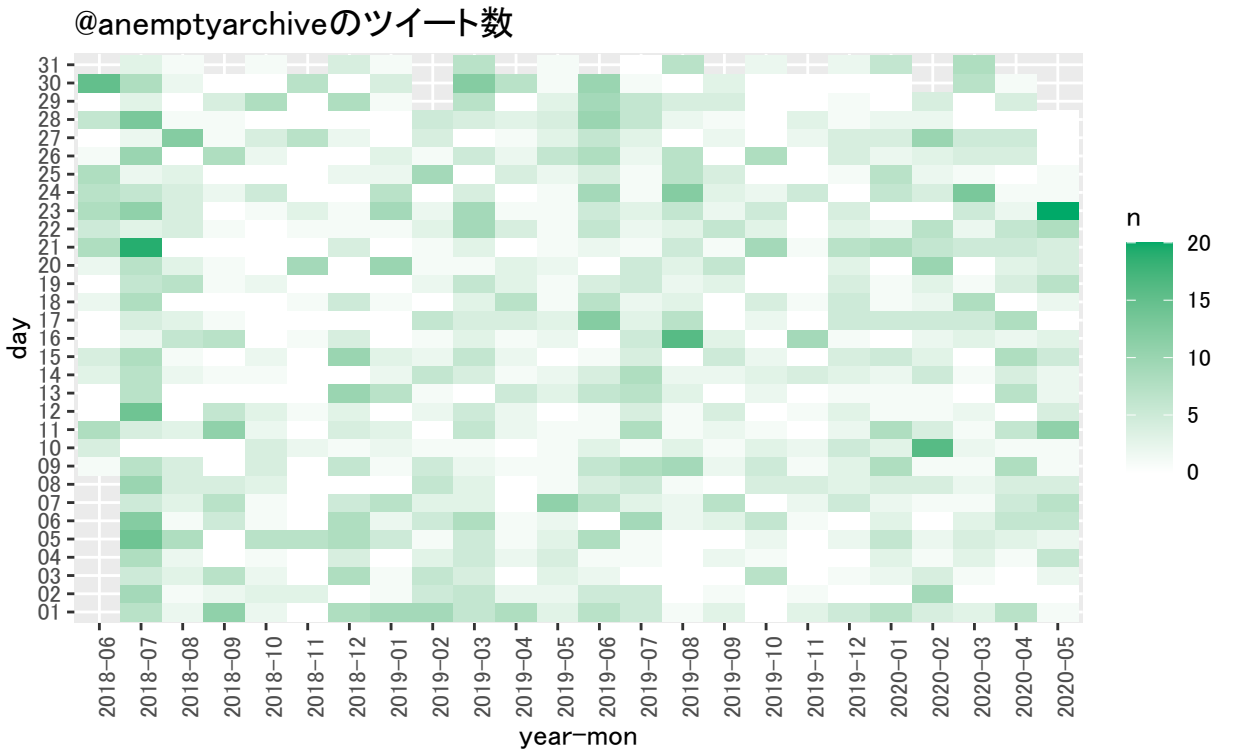
```
# 軸ラベル用にデータフレームを整形
if(term == "day") {

  tw_count2 <- tw_count2 %>%
    mutate(year_mon = format(terms, "%Y-%m")) %>% # 年月を抽出
    mutate(day = format(terms, "%d")) # 日を抽出

} else if(term == "mon") {

  tw_count2 <- tw_count2 %>%
    mutate(year = format(terms, "%Y")) %>% # 年を抽出
    mutate(mon = format(terms, "%m")) # 月を抽出
```

```
} else if(term == "year") {

  tw_count2 <- tw_count2 %>%
    mutate(year = format(terms, "%Y")) # 年を抽出

}
```

指定した期間を y 軸ラベル、それより大きな単位を x 軸目盛とするために、それぞれ format() でツイート日の列 (terms) から必要な情報を取り出し、mutate() で新しい列とします。

作図用のデータフレームができたので、これを用いて作図します。作図の処理も指定した期間によって異なるため、場合分けします。

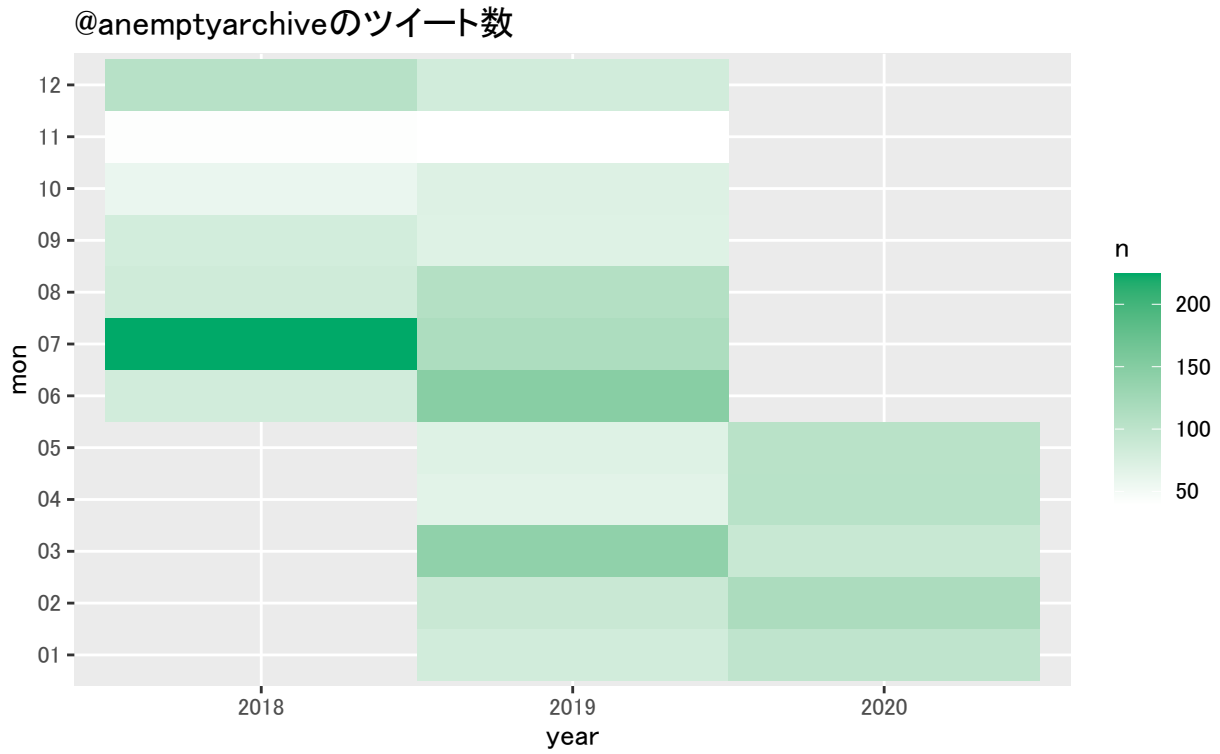
```
# ヒートマップを作図
if(term == "day") {

  ggplot(tw_count2, aes(x = year_mon, y = day, fill = n)) +
    geom_tile() + # ヒートマップ
    scale_fill_gradient(low = "white" , high = "#00A968") + # 塗りつぶしの濃淡
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) + # 軸目盛の傾き
    labs(title = paste0("@", screen_name, " のツイート数"),
         x = "year-mon", y = "day") # ラベル

} else if(term == "mon") {

  ggplot(tw_count2, aes(x = year, y = mon, fill = n)) +
    geom_tile() + # ヒートマップ
    scale_fill_gradient(low = "white" , high = "#00A968") + # 塗りつぶしの濃淡
    labs(title = paste0("@", screen_name, " のツイート数"),
         x = "year", y = "mon") # ラベル

} else if(term == "year") {

  ggplot(tw_count2, aes(x = year, y = 0, fill = n)) +
    geom_tile() + # ヒートマップ
    scale_fill_gradient(low = "white" , high = "#00A968") + # 塗りつぶしの濃淡
    ylim(c(-1, 1)) + # y 軸の範囲
    labs(title = paste0("@", screen_name, " のツイート数"),
         x = "year", y = "") # ラベル

}
```



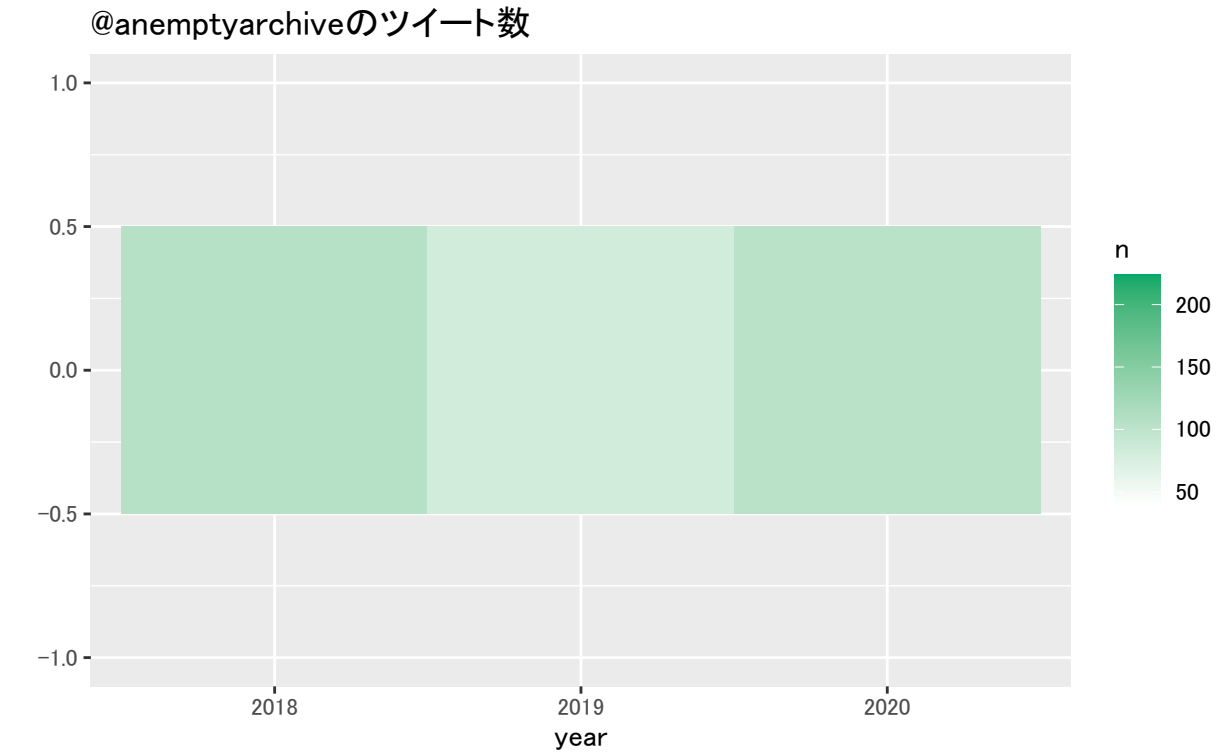

`geom_tile()` でヒートマップを作図します。

タイルの色は `scale_fill_gradient()` の `low` と `high` にそれぞれ色を指定することで、ツイート数 (`fill` 引数の値) に応じたグラデーションとなります。

・月ごとの場合


@anemptyarchiveのツイート数
12
11
10
09
08
07
06
05
04
03
02
01
mon
2018
2019
2020
year
n
200
150
100
50

・年ごとの場合


@anemptyarchiveのツイート数
1.0
0.5
0.0
−0.5
−1.0
2018
2019
2020
year
n
200
150
100
50

### ・時別に可視化

続いて、1 時間ごとのツイート数をヒートマップ化します。
基本的な処理は同様です。こちらは可視化に時刻データを用いるため、ツイート日時を POSIXct 型で扱うところが異なります。

```
# 1 時間ごとにツイート数を集計
tw_count1 <- tw_time %>%
  floor_date(unit = "hour") %>% # 1 時間単位で切り捨て
  as.POSIXct() %>% # POSIXct 型に変換
  tibble(terms = .) %>% # データフレームに変換
  count(terms) # ツイート数をカウント

# ツイートがない期間が欠落する対策
term_df <- seq(
  floor_date(tw_time[length(tw_time)], "hour"), # 一番古い時刻
  floor_date(now(), "hour"), # 現在時刻
  by = "hour"
) %>% # 1 時間刻みのベクトルを作成
  tibble(terms = .) # データフレームに変換

# 作図用にデータフレームを加工
tw_count2 <- left_join(term_df, tw_count1, by = "terms") %>% # 集計結果を結合
  mutate(n = replace_na(n, 0)) %>%  # NA を 0 に置換
  mutate(year_mon_day = as_date(terms)) %>% # 年月日を抽出
  mutate(hour = format(terms, "%H")) # 時間を抽出
```

floor_date(., unit = "hour") で 1 時間単位で丸めて (切り捨て)、as.POSIXct() で POSIXct 型に変換します。
また tibble() でデータフレームに変換して、count() で各期間のツイート数を集計します。

期間内にツイートがなかった場合に欠落する問題の対処や、作図用のデータフレームの加工も概ね同じ流れです。

期間を指定する際に"hour" とする点に注意しましょう。

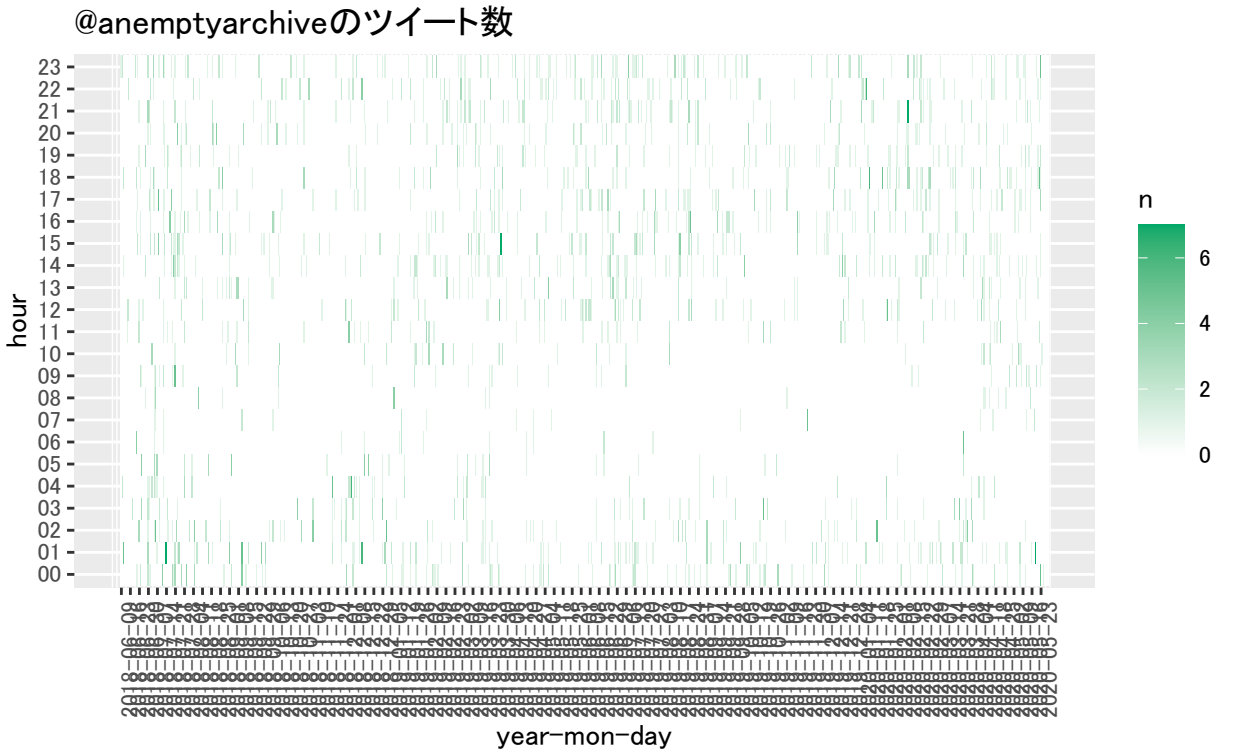
では、完成したデータフレームを確認しておきましょう。

```
tail(tw_count2, 10)
```

```
## # A tibble: 10 x 4
##    terms                   n year_mon_day hour
##    <dttm>              <dbl> <date>       <chr>
##  1 2020-05-28 14:00:00     0 2020-05-28   14
##  2 2020-05-28 15:00:00     0 2020-05-28   15
##  3 2020-05-28 16:00:00     0 2020-05-28   16
##  4 2020-05-28 17:00:00     0 2020-05-28   17
##  5 2020-05-28 18:00:00     0 2020-05-28   18
##  6 2020-05-28 19:00:00     0 2020-05-28   19
##  7 2020-05-28 20:00:00     0 2020-05-28   20
##  8 2020-05-28 21:00:00     0 2020-05-28   21
##  9 2020-05-28 22:00:00     0 2020-05-28   22
## 10 2020-05-28 23:00:00     0 2020-05-28   23
```

これを用いて作図します。

```
# ヒートマップを作図
ggplot(tw_count2, aes(x = year_mon_day, y = hour, fill = n)) +
  geom_tile() + # ヒートマップ
  scale_fill_gradient(low = "white", high = "#00A968") + # 塗りつぶしの濃淡
  scale_x_date(breaks = seq(tw_count2[["year_mon_day"]][1],
                            tw_count2[["year_mon_day"]][nrow(tw_count2)],
                            by = "1 week")) + # x軸目盛（日付）
  theme(axis.text.x = element_text(angle = 90)) + # x軸目盛の傾き
  labs(title = paste0("@", screen_name, "のツイート数"),
       x = "year-mon-day", y = "hour") # ラベル
```




x 軸に表示するデータ数が多くなるため、必要に応じて scale_x_date() で、x 軸ラベルを表示する位置を間引きます (ここに表示されるラベルデータは文字列型なので自動調整されない)。 scale_x_date() の breaks 引数に、表示する位置のベクトルを渡すことで表示数を絞ることができます。そのベクトルは、seq() に tw_count2 の year_mon_day 列の 1 行目と最終行の要素を渡して作成します。by 引数に指定する間隔を変更して調整してください。

## ・ツイートのネガポジ分析

ツイートテキストに対して感情分析を行います。
感情分析とは (ざっくり言うと)、テキストに含まれている単語がそれぞれネガティブな表現なのかポジティブな表現なのかを評価し、テキスト全体がネガ・ポジどちらなのかを判定するものです。

【分析手順】

1. rtweet パッケージを使ってツイートを拾ってきます
2. 取得したツイートテキストを、指定した期間 (日 or 月) ごとに区分けして 1 つのテキストとします
3. 各 (期間の) テキストに対して、MeCab で形態素解を行います
4. **単語感情極性対応表**を利用して、単語ごとにネガポジスコアを付与します
5. 期間ごとにネガティブスコア・ポジティブスコアを合算してその期間のネガ度・ポジ度とします
6. その結果を ggplot2 で可視化します

```
## 利用パッケージ
# ツイート収集:get_timeline()
library(rtweet)

# 形態素解析:docDF()
library(RMeCab)

# 時間データの処理:floor_date(), as_date()
library(lubridate)

# データフレーム操作
library(dplyr)
library(tidyr)

#文字列操作:str_remove_all()
library(stringr)

# 作図
library(ggplot2)
```

利用するパッケージを読み込みます。

取得したツイートデータの内、ツイート日時 (created_at 列) とツイートテキスト (text 列) を使います。

```
# ツイート日時の抽出と POSIXlt 型・タイムゾーンの変換
tw_time <- tw_data[["created_at"]] %>%
  as.POSIXct(tz = "Etc/GMT") %>%
  as.POSIXlt(tz = "Japan")
```

取得したツイート日時は POSIXct 型の世界共通時 (UTC) なので、これを as.POSIXlt() で POSIXlt 型の日本標準時 (JST) に変換します。引数 tz はタイムゾーンのことで JST に変更するなら"Japan" あるいは"Asia/Tokyo" を指定します (2 行目の Etc/GMT は UCT を明示しています。多分なくても動く)

```
tw_time[1:10]
```

```
##  [1] "2020-05-25 00:48:56 JST" "2020-05-24 10:33:57 JST"
##  [3] "2020-05-23 23:55:20 JST" "2020-05-23 23:45:54 JST"
##  [5] "2020-05-23 23:35:50 JST" "2020-05-23 23:15:49 JST"
##  [7] "2020-05-23 23:12:26 JST" "2020-05-23 22:59:04 JST"
##  [9] "2020-05-23 19:33:35 JST" "2020-05-23 19:26:21 JST"
```

こんな感じになっていれば OK です。

```
# ツイートテキストの抽出と文字列処理
tw_text <- tw_data[["text"]] %>%
  str_remove_all("^@.*?\\s") %>% # リプライ先の除去
  str_remove_all(("https?://[\\w/:%#\\$&\\?\\(\\)~\\.=\\+\\-]+")) %>% # url の除去
  str_remove_all("[\U0001F004-\U000207BF]") # 絵文字の除去
```

ツイートテキストから、分析に不要な部分を str_remove_all() で取り除きます。

これで必要なツイートデータを取得できました。次からは分析のための整形処理を行っていきます。

### ・データハンドリング

```
# 単位（期間）を指定
term <- "mon"
term <- "day"
```

ツイートテキストをひとまとめにする単位 (期間) を指定します。
「月」単位なら"mon"、「日」単位なら"day" とします。これは主に floor_date() の unit 引数に指定するためのものなので、このままの文字列を使用してください。

```
# 期間ごとにツイートテキストをまとめる
text_df <- data.frame(
  terms = as.Date(floor_date(tw_time, term)), # 指定した期間で丸める
  texts = tw_text
) %>%
  group_by(terms) %>% # 期間ごとにグループ化
  summarise(texts = paste(texts, collapse = "\n")) # 同一期間のテキストを結合
```

floor_date() で指定した期間 (単位) に丸めて、as_date() に Date 型の日付データに変換します (時刻データを落とします)。

また丸めて同一期間となったツイートテキストをまとめます。各ツイートの区切りは改行 (\n) としておきます。

データを確認しましょう。

```
tail(text_df)
```

```
## # A tibble: 6 x 2
##   terms      texts
##   <date>     <chr>
## 1 2020-05-20 "ゼロつく1終わったー。早速2周目をより丁寧にやってく\n時々こっち向くの凄くかわいい///\n別
## 2 2020-05-21 "何の推定もできませんでした！！はてなブログに投稿しました #はてなブログ\n\n【R】3.3.8：I
## 3 2020-05-22 "放心ちゅう #おうちでBuono!\nハロプログループの中で一番Buono!が好きなんだけど、嵌まる2か
## 4 2020-05-23 "まぁよく頑張ったと思う。\nやり切ると燃え尽きるので、既に始めた\n次の本をやっていき\n1ペ
## 5 2020-05-24 "楽しみにしててください！(これはしっかりやらねば宣言ですね)"
## 6 2020-05-25 "きょーのところはこれくらいにしといてやる"
```

これでデータハンドリングは完了です。次からはネガポジ分析を行っていきます。

### ・感情分析

まずはネガポジ度の配点データとなる辞書データを用意します。

```
# 単語感情極性対応表の取得
np_dic_original <- read.table(
  "http://www.lr.pi.titech.ac.jp/~takamura/pubs/pn_ja.dic",
  sep = ":", stringsAsFactors = FALSE
)
```

東京工業大学高村研究室で公開されている単語感情極性対応表を、単語のネガポジ度を測定するための辞書として利用します。詳しくは「R+RMeCab で感情分析」をご参照ください。

```
head(np_dic_original)
```

```
##         V1       V2     V3       V4
## 1   優れる すぐれる   動詞 1.000000
## 2     良い     よい 形容詞 0.999995
## 3     喜ぶ よろこぶ   動詞 0.999979
## 4   褒める   ほめる   動詞 0.999979
## 5 めでたい めでたい 形容詞 0.999645
## 6     賢い かしこい 形容詞 0.999486
```

これを RMeCab::docDF() の出力結果と結合するために形式を揃えます。

```
# ネガポジ辞書の作成
np_dic <- np_dic_original %>%
  select(TERM = V1, POS1 = V3, allocation = V4) %>% # 列の選択
  distinct(TERM, .keep_all = TRUE) # 重複の除去
head(np_dic)
```

```
##       TERM   POS1 allocation
## 1   優れる   動詞   1.000000
## 2     良い 形容詞   0.999995
## 3     喜ぶ   動詞   0.999979
## 4   褒める   動詞   0.999979
## 5 めでたい 形容詞   0.999645
## 6     賢い 形容詞   0.999486
```

これで辞書データを用意できました。続いて、上で作成した日 (あるいは月) ごとのテキスト 1 つずつに対して次の処理を行います。

1. docDF() で単語に切り分けます。
2. テキストに含まれる単語一覧とネガポジ辞書の単語を left_join() で (マッチして) 結合することで、配点データを付与します。
3. 各単語の配点と出現回数を掛けてスコアを計算します。
4. スコアがマイナスなら「ネガティブ」、プラスなら「ポジティブ」、0(または NA) なら「ニュートラル」のラベルを付与します。

テキストを MeCab にかけるには、一時的に txt ファイルとして書き出す必要があります。

```
# 一時テキストファイルの保存先を指定
folder_name <- "tmp_data"

# 分析結果の受け皿を初期化
score_df <- data.frame()
for(i in 1:nrow(text_df)) {

  # 一時ファイルパスを作成
  tmp_file_name <- paste(screen_name, "_", text_df[["terms"]][i], ".txt", sep = "")
  tmp_path <- paste(folder_name, tmp_file_name, sep = "/")

  # テキストファイルを書き出し
  write(text_df[["texts"]][i], tmp_path)

  # MeCab による形態素解析
  mecab_df <- docDF(tmp_path, type = 1, pos = c(" 動詞", " 形容詞", " 副詞", " 助動詞"))

  if(!is.null(mecab_df)) { ## (NULL でないときのみ)

    # ネガポジ配点を結合
    tmp_score_df <- mecab_df %>%
      left_join(np_dic, by = c("TERM", "POS1")) %>% # 各単語に配点を付与
      mutate(allocation = replace_na(allocation, 0)) %>% # NA を 0 に置換
      select(TERM, FREQ = tmp_file_name, allocation) # docDF 仕様の列名に変更

    # ネガポジスコアを計算
    tmp_score_df <- tmp_score_df %>%
      mutate(terms = text_df[["terms"]][i]) %>% # 日付情報列の追加
      mutate(allocation = replace_na(allocation, 0)) %>% # 配点が NA の場合 0 に置換
      mutate(score = allocation * FREQ) %>% # (スコア) = (配点) * (語数)
      mutate(
        np_label = case_when(
          score < 0 ~ "negative", # (スコア) < 0 ならネガ
          score > 0 ~ "positive", # (スコア) > 0 ならポジ
          score == 0 ~ "neutral"  # (スコア) = 0 ならニュート
        )
      ) # ネガポジラベルを付与

    # 全期間の結果を結合
    score_df <- rbind(score_df, tmp_score_df)
  }
}
```

辞書に含まれない単語はデータフレームの結合時に配点が NA となります。これを replace_na() で 0 に置換します。

単語ごとに配点 (allocation 列) と (その単語の) 語数 (FREQ 列) を掛けてスコア (score 列) とします。

また単語ごとにスコアの正負によってラベルデータを与えます。case_when() を使って score 列が、、0 未満なら"negative"0 より大きければ"positive"、0 なら"neutral" とします。

結果はこのようになります。

```
tail(score_df)
```

```
##      TERM FREQ allocation      terms     score np_label
## 8060   ぬ    1  -0.998545 2020-05-24 -0.998545 negative
## 8061 やる    1   0.000000 2020-05-24  0.000000  neutral
## 8062 くる    1   0.000000 2020-05-25  0.000000  neutral
## 8063 する    1   0.000000 2020-05-25  0.000000  neutral
## 8064 とく    1   0.000000 2020-05-25  0.000000  neutral
## 8065 やる    1   0.000000 2020-05-25  0.000000  neutral
```

単語ごとに得点を与えられたので、次は期間ごとに合計します。

```
# 期間ごとにネガスコア・ポジスコアの合計
result_df <- score_df %>%
  select(terms, np_label, score, FREQ) %>%
  group_by(terms, np_label) %>%
  summarise(score = sum(score), FREQ = sum(FREQ)) # スコアと頻度を合算
```

期間ごとにそのまま合計するのでなく、ネガポジラベルの情報も使ってポジティブ・ネガティブ・ニュートラル (0 だけど) ごとに合計しておきます (相殺されるのはもったいないので)。

```
tail(result_df)
```

```
## # A tibble: 6 x 4
## # Groups:   terms [3]
##   terms      np_label   score  FREQ
##   <date>     <chr>      <dbl> <dbl>
## 1 2020-05-23 negative -14.3      21
## 2 2020-05-23 neutral    0       103
## 3 2020-05-23 positive   4.82      5
## 4 2020-05-24 negative  -0.999     1
## 5 2020-05-24 neutral    0         6
## 6 2020-05-25 neutral    0         4
```

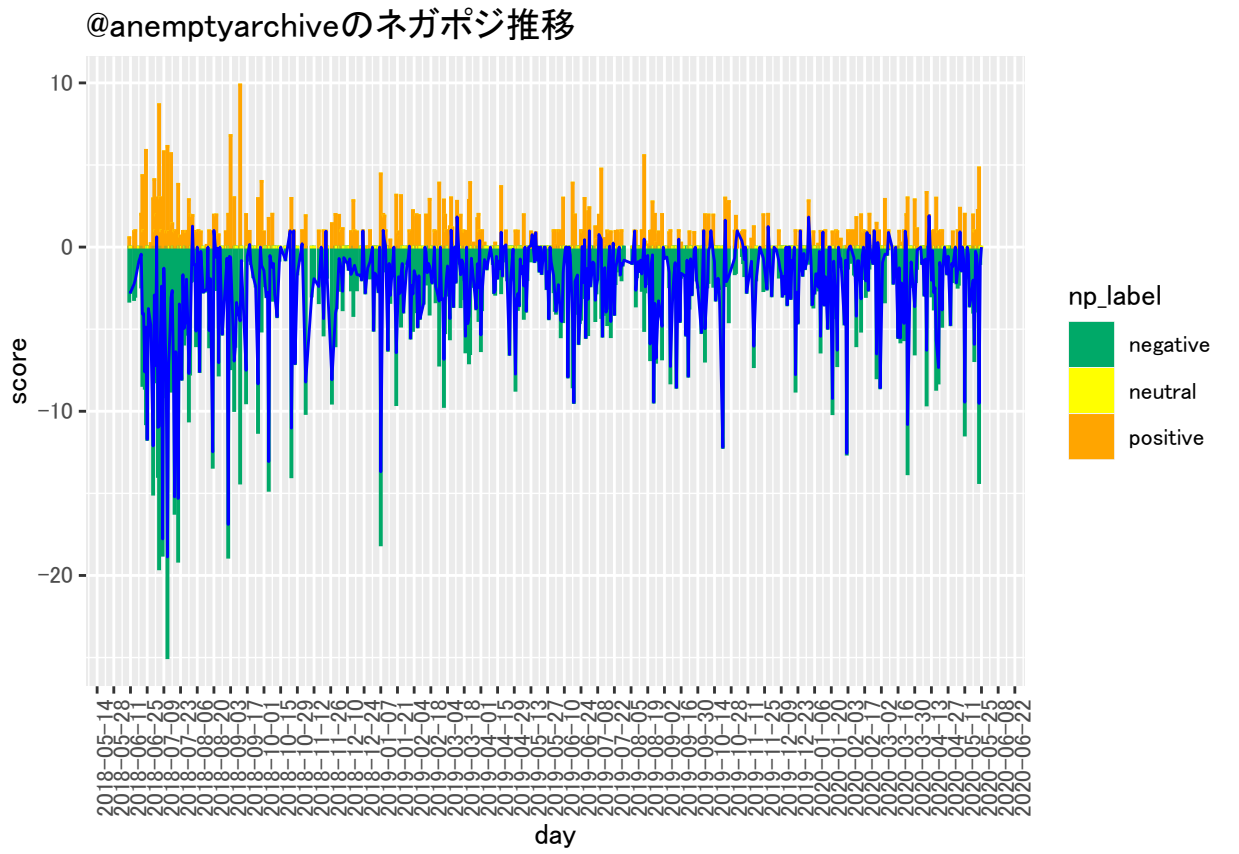
以上でネガポジ評価は完了です。では可視化しましょう。

### ・可視化

・ネガポジ推移

```
# ネガポジ推移
ggplot(result_df, aes(x = terms, y = score)) +
  geom_bar(mapping = aes(fill = np_label, color = np_label), stat = "identity") + # 棒グラフ
  scale_fill_manual(values = c("#00A968", "yellow", "orange")) + # 塗りつぶしの色
  scale_color_manual(values = c("#00A968", "yellow", "orange")) + # 枠の色
  geom_line(stat = "summary", fun = "sum", color = "blue") + # 折れ線グラフ
  scale_x_date(date_breaks = "2 weeks") + # x軸目盛 (day)
  #scale_x_date(date_breaks = "1 month", date_labels = "%Y-%m") + # x軸目盛 (mon)
  theme(axis.text.x = element_text(angle = 90)) + # x軸目盛の傾き
```

```
  labs(title = paste0("@", screen_name, " のネガポジ推移"),
       x = term) # ラベル
```




積み上げ棒グラフで、ポジティブスコアとネガティブスコアを表します。
　またポジ・ネガスコアの合計を折れ線グラフで描き、その期間のポジ・ネガを表します。

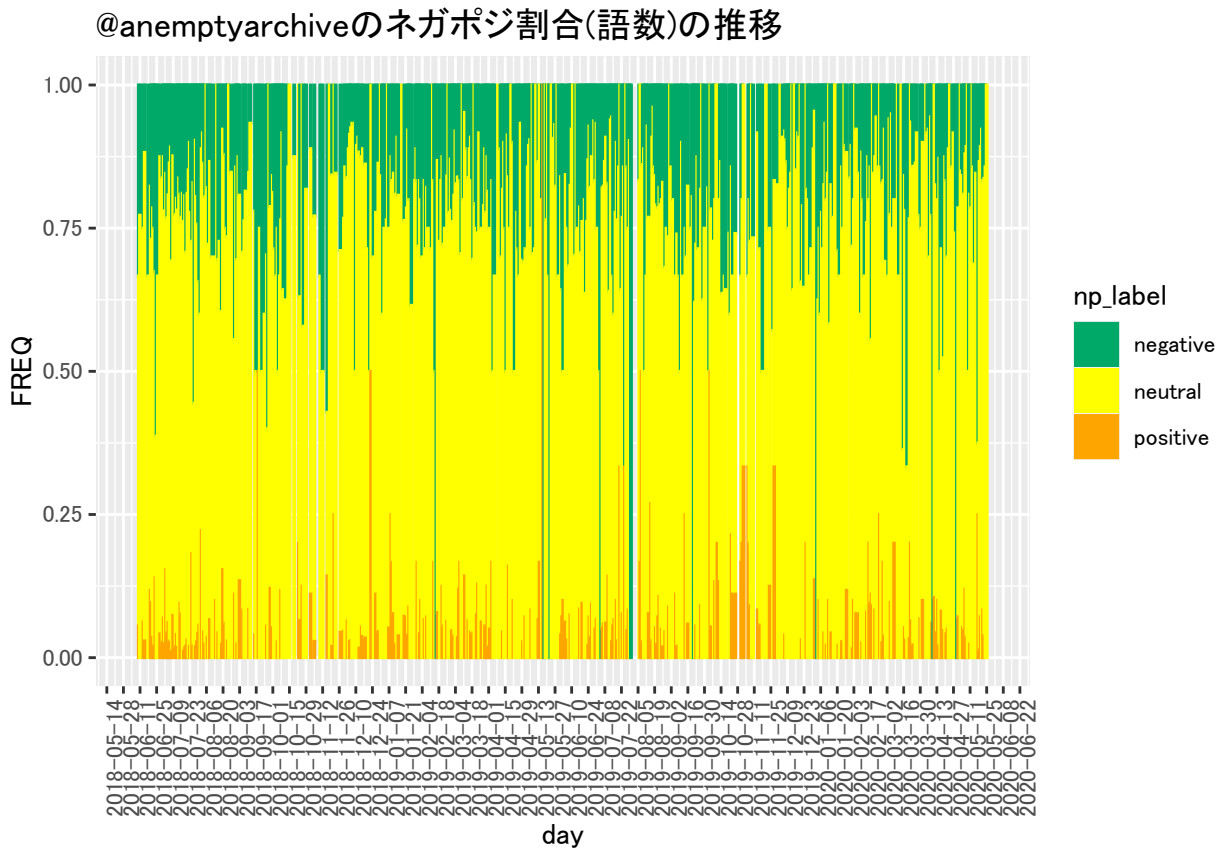
ツイート期間が多いと、x 軸目盛が潰れてしまうので、scale_x_date() の date_breaks 引数に表示する位置をしていして間引きます。(日別か月別でかなり変わるので、両方用意して片方をコメントアウトしています。)

このグラフだと主にニュートラル単語の情報が欠けているので、総単語数におけるネガポジ語数の割合も見ておきましょう。

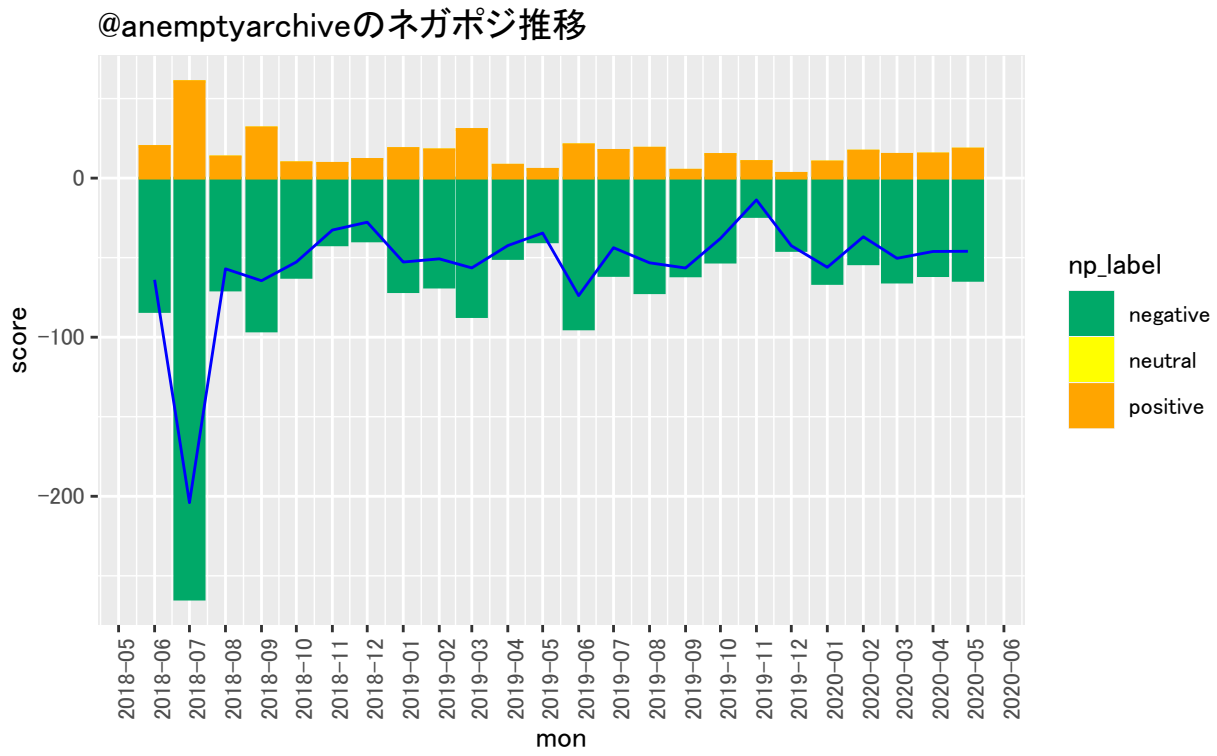
・ネガポジ割合の推移

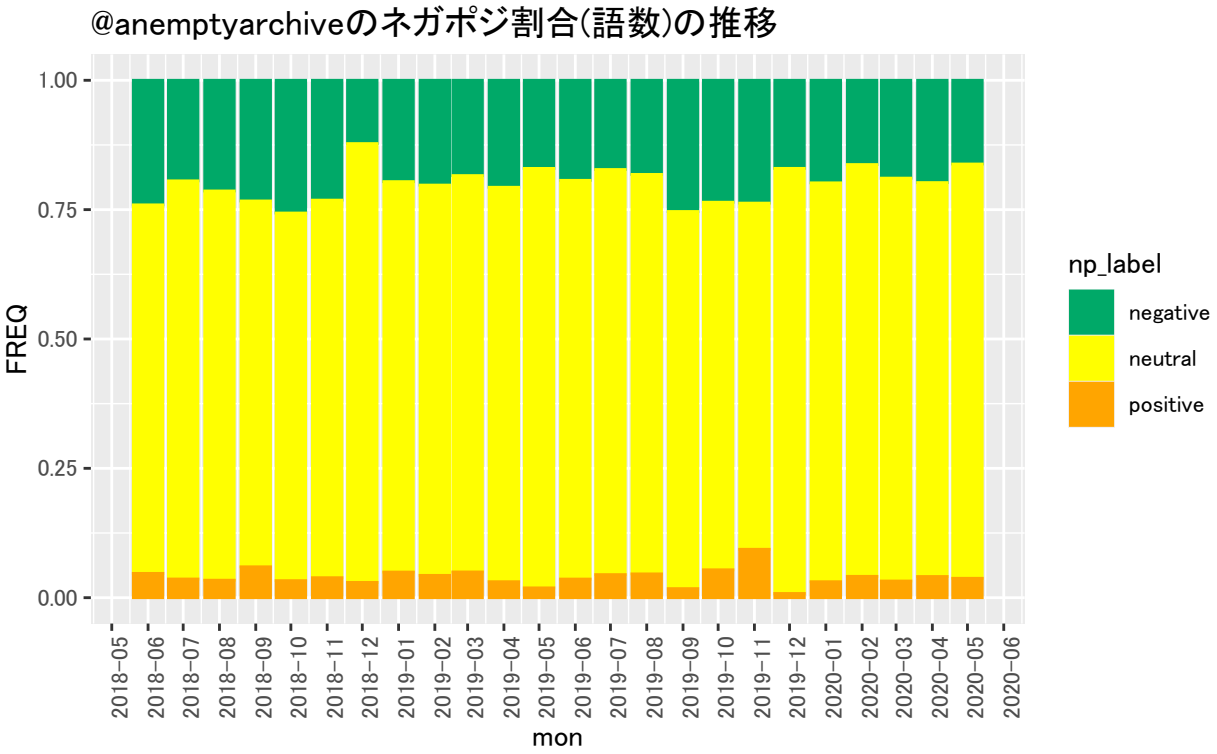
```
# ネガポジ割合の推移
ggplot(result_df, aes(x = terms, y = FREQ, fill = np_label, color = np_label)) +
  geom_bar(stat = "identity", position = "fill") + # 棒グラフ
  scale_fill_manual(values = c("#00A968", "yellow", "orange")) + # 塗りつぶしの色
  scale_color_manual(values = c("#00A968", "yellow", "orange")) + # 枠の色
  scale_x_date(date_breaks = "2 weeks") + # x 軸目盛 (day)
  #scale_x_date(date_breaks = "1 month", date_labels = "%Y-%m") + # x 軸目盛 (mon)
  theme(axis.text.x = element_text(angle = 90)) + # x 軸目盛の傾き
```

```
labs(title = paste0("@", screen_name, " のネガポジ割合（語数）の推移"),
     x = term) # ラベル
```




・月別の場合




ニュートラルな (ネガポジどちらでもない) 単語というよりも、得点が与えられて (ネガポジが判定されて) いないが単語多すぎますね。ネガ・ポジ判定のできた単語だけ見てもいくら何でもネガティブが過ぎますね、これは使い物になら・・・

## ・ツイート頻度によるクラスタリング

ツイートする時間帯から生活サイクルの近い人が分かるのでは？という思い付きを R 言語でやってみます。

【処理の流れ】

1. rtweet パッケージを使って、ツイート収集
2. ggplot2 パッケージを使って、ツイート頻度をヒートマップ化
3. ggdendro パッケージを使って、ツイート頻度のクラスタリングおよび樹形図作成

```
# 利用パッケージ
library(rtweet) # ツイート収集:get_timeline()
library(dplyr) # データフレーム操作
library(tidyr) # データフレーム操作:pivot_longer()
library(lubridate) # 時間データ操作:floor_date()
library(ggplot2) # 作図
library(ggdendro) # 樹形図:dendro_data(), ggdendrogram()
```

利用するパッケージを読み込みます。

### ・ツイート収集

まずは rtweet パッケージを利用して、ツイートを集めます。

アカウントのスクリーンネーム (@*** の ***) を指定します。そこから 1 アカウントずつ get_timeline() に渡して、ツイートデータを取得します (取得できるツイートに制限があったりします)。

```
# アカウントを指定
screen_names <- c(
  "MorningMusumeMg", "angerme_upfront", "JuiceJuice_uf",
  "tsubakifac_uf", "BEYOOOOONDS_", "kenshusei_uf"
)

# ツイート収集と集計
tw_count <- tibble(terms = floor_date(Sys.time(), "hour")) ## (本当は列名だけを持つ空の df を作りたい)
for(i in seq_along(screen_names)) {

  # ツイートを収集
  tw_data <- get_timeline(screen_names[i], n = 10000, include_rts = FALSE)

  # 指定した期間ごとにツイート数を集計
  tmp_tw_count <- tw_data[["created_at"]] %>% # ツイート日時を抽出
    as.POSIXct(tz = "Asia/Tokyo") %>% # 日本標準時に変換
    floor_date(unit = "hour") %>% # 1 時間ごとに切り捨て
    tibble(terms = .) %>% # データフレームに変換
    group_by(terms) %>% # グループ化
```

```
    summarise(!!screen_names[i] := n()) # ツイート数をカウント

  # 集計結果を結合
  tw_count <- full_join(tw_count, tmp_tw_count, by = "terms")

  # おまじない
  Sys.sleep(1)
  print(paste0(screen_names[i], "...", round(i / length(screen_names) * 100, 1), "%"))
}
```

取得したツイートデータの内、ツイート日時 (created_at 列) を利用します。

取得したツイート日時は世界共通時 (UTC) なので、as.POSIXlt() の tz 引数に"Asia/Tokyo" を指定することで日本標準時 (JST) に変換します。
また 1 時間ごとのツイート数を集計するために、日時データを floor_date(., unit = "hour") で分と秒の情報を切り捨てます。
ここまではベクトルとして扱っているので、これを tibble() でデータフレームとします。

あとは、グループ化してカウントしたものを全てのアカウント分結合していきます。

```
head(tw_count)
```

```
## # A tibble: 6 x 7
##   terms               MorningMusumeMg angerme_upfront JuiceJuice_uf
##   <dttm>                        <int>           <int>         <int>
## 1 2020-05-26 01:00:00              NA              NA            NA
## 2 2019-11-04 11:00:00               1               1            NA
## 3 2019-11-04 12:00:00               4              NA             6
## 4 2019-11-04 13:00:00               3               1             1
## 5 2019-11-05 08:00:00               2              NA             2
## 6 2019-11-05 09:00:00               1              NA             3
## # ... with 3 more variables: tsubakifac_uf <int>, BEYOOOOONDS_ <int>,
## #   kenshusei_uf <int>
```

ツイートがない日時は NA になります。

アカウントによって取得できたツイートの期間にばらつきがあるかもしれないため、利用する期間を絞っておきます。

```
# 期間を指定してツイート数を抽出
tw_count2 <- seq(
  as.POSIXct("2020/04/01", tz = "Japan"), # から
  as.POSIXct("2020/05/01", tz = "Japan"), # まで
  by = "hour"
) %>%
  tibble(terms = .) %>% # 指定した範囲の df を作成
  left_join(tw_count, by = "terms") # 範囲内のツイート数を結合

# ツイートがないと値が NA となるので 0 に置換
tw_count2[is.na.data.frame(tw_count2)] <- 0
```

seq() に始まり (第 1 引数) と終わり (第 2 引数) の日時を指定して by 引数を"hour" とすることで、1

時間ごとに日時データが並んだベクトルとなります。
それをデータフレームに変換し、left_join() で先ほどの頻度データと結合します。

ここでもツイートがない日時が NA となるので、0 に置換しておきます。NA のままだとヒートマップとしたときに表示されません。

```
tail(tw_count2)
```

```
## # A tibble: 6 x 7
##   terms               MorningMusumeMg angerme_upfront JuiceJuice_uf
##   <dttm>                        <dbl>           <dbl>         <dbl>
## 1 2020-04-30 10:00:00               1               1             1
## 2 2020-04-30 11:00:00               1               2             3
## 3 2020-04-30 12:00:00               5               3             2
## 4 2020-04-30 13:00:00               3               2             0
## 5 2020-04-30 14:00:00               0               0             0
## 6 2020-04-30 15:00:00               0               0             0
## # ... with 3 more variables: tsubakifac_uf <dbl>, BEYOOOOONDS_ <dbl>,
## #   kenshusei_uf <dbl>
```

こんな感じになっていれば OK です。

これで必要なツイート頻度データを用意できました。次はこれをヒートマップにします。

### ・ヒートマップ

ggplot2 で作図するために、pivot_longer() で縦型のデータフレームに変換します。

```
# データフレームを long 型に変換
tw_count_long <- pivot_longer(
  tw_count2,
  cols = -terms, # 変換しない列
  names_to = "screen_name", # 現列名を格納する列の名前
  values_to = "n" # 現セルを格納する列の名前
)
```

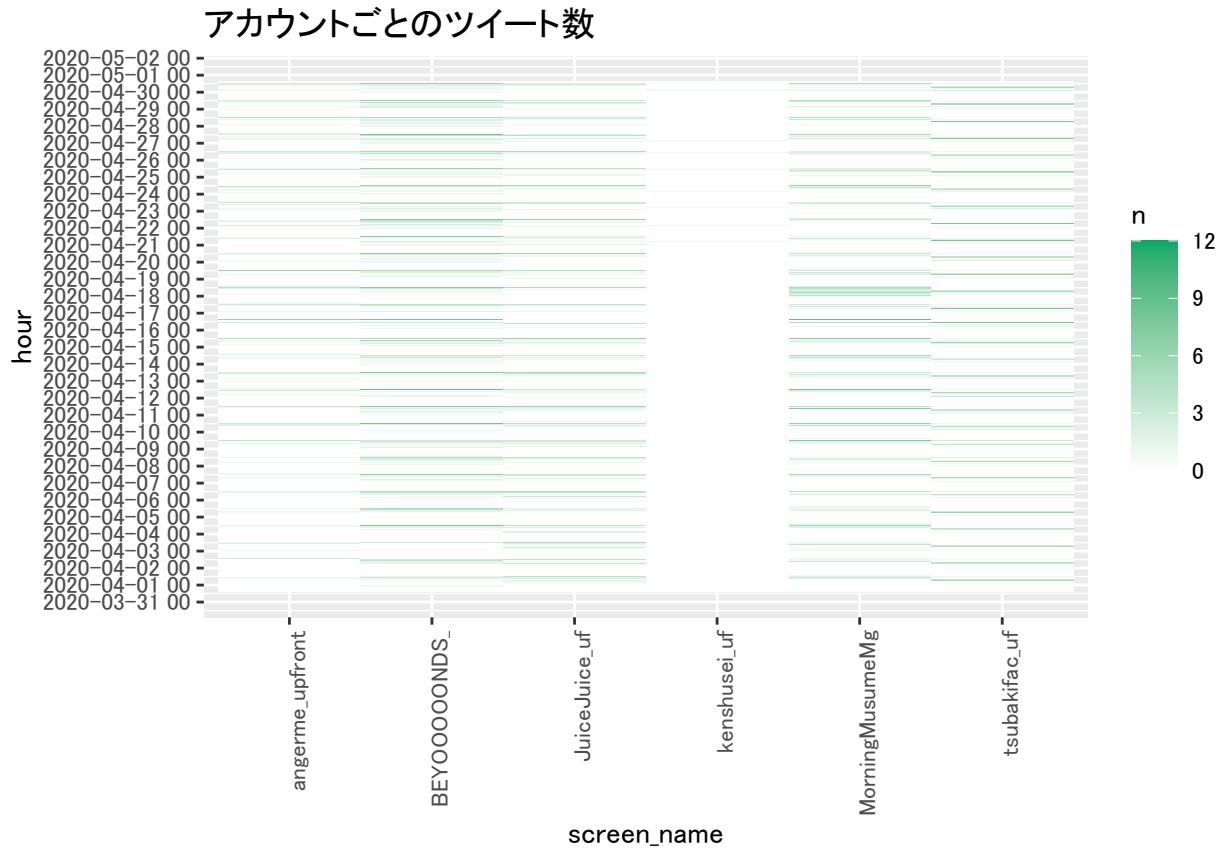
期間を行・アカウントを列・頻度をセルとするデータフレームを、期間・アカウント・頻度の 3 列からなるデータフレームに変換します。それぞれ y 軸・x 軸・値 (グラデーション) に対応します。

```
head(tw_count_long)
```

```
## # A tibble: 6 x 3
##   terms               screen_name         n
##   <dttm>              <chr>           <dbl>
## 1 2020-03-31 15:00:00 MorningMusumeMg     0
## 2 2020-03-31 15:00:00 angerme_upfront     0
## 3 2020-03-31 15:00:00 JuiceJuice_uf       0
## 4 2020-03-31 15:00:00 tsubakifac_uf       0
## 5 2020-03-31 15:00:00 BEYOOOOONDS_        0
## 6 2020-03-31 15:00:00 kenshusei_uf        0
```

この long 型のデータフレームを使って作図します。

```
# ヒートマップを作図
ggplot(tw_count_long, aes(x = screen_name, y = terms, fill = n)) +
  geom_tile() + # ヒートマップ
  scale_fill_gradient(low = "white", high = "#00A968") + # 塗りつぶし色のグラデーション
  scale_y_datetime(date_breaks = "1 day",
                   date_labels = "%Y-%m-%d %H") + # y 軸目盛（日時）
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) + # x 軸目盛の角度
  labs(title = " アカウントごとのツイート数",
       y = "hour") # ラベル
```




`geom_tile()` でヒートマップを作成します。
タイル (?) の色の設定は `scale_fill_gradient()` で行えます。引数 `low` に頻度が少ない場合の色を、引数 `high` に多い場合の色を指定すると、いい感じのグラデーションになります。

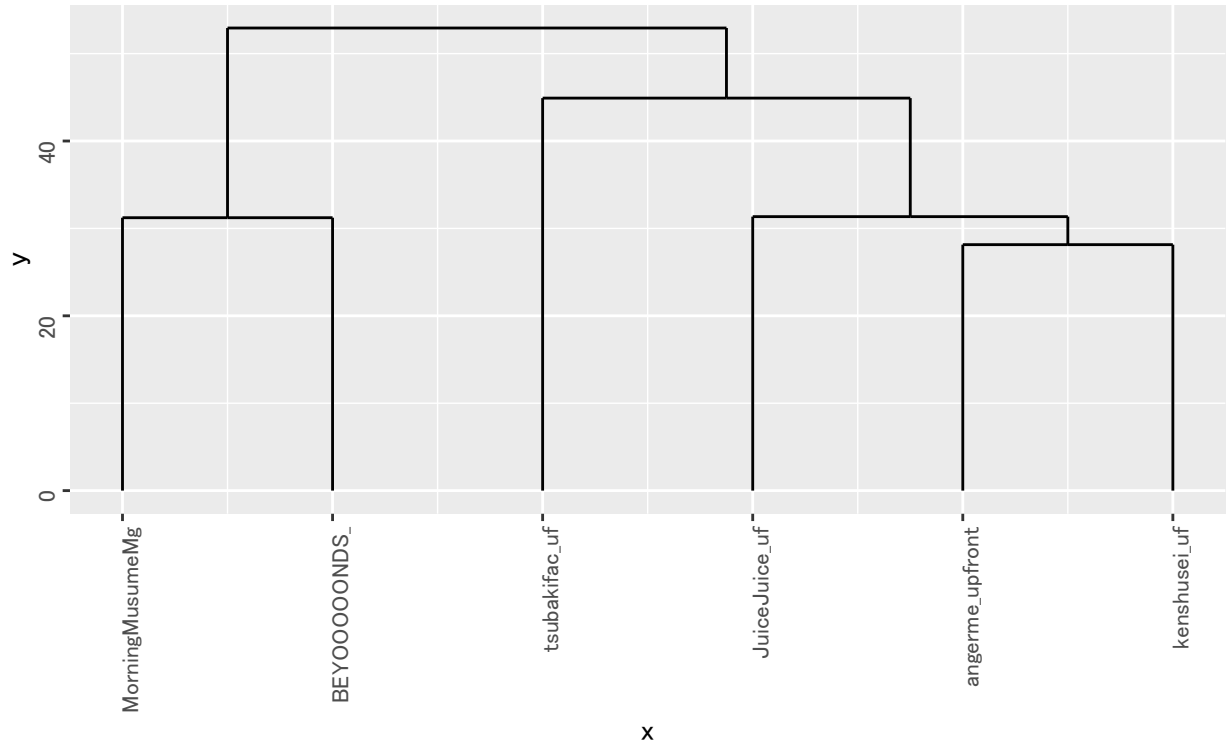
データが多いと y 軸目盛が潰れてしまいます。そんなときは `scale_y_datetime()` で調整できます。`date_breaks` 引数に表示する間隔を指定し、`date_labels` 引数に表示する文字列のフォーマットを指定します。

(これ自体はクラスタリングに必須ではないですが) アカウントごとのツイート頻度を視覚的に確認できました。ではクラスタリングを行いましょう。

・クラスタリング

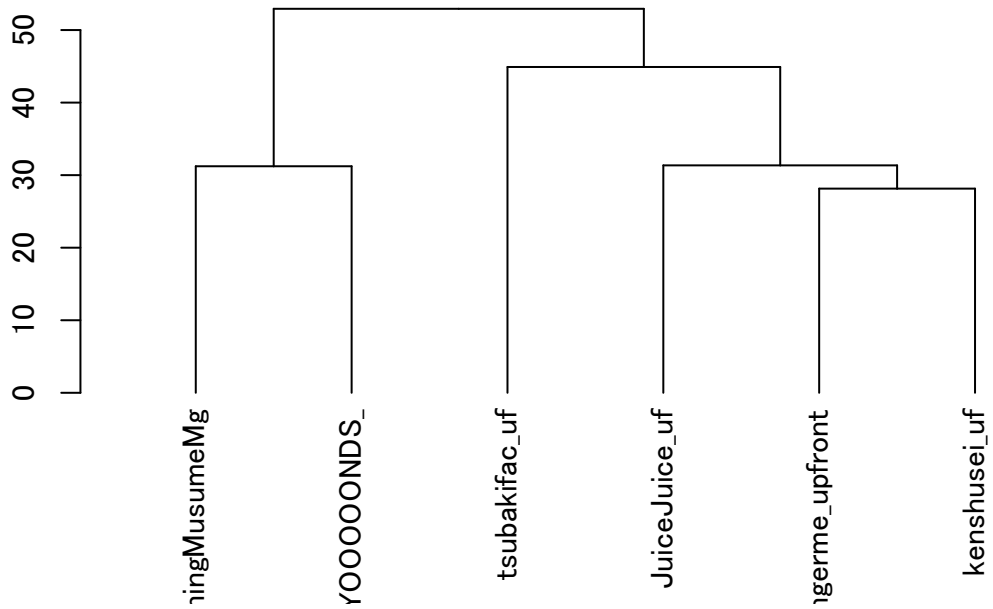
ggplot2 風と組み込み plot() で作図します (どちらもいまいち上手く扱えていないです…)。

```
# クラスタリング
res_dendrogram <- tw_count2[, -1] %>% # 非数値の列を落とす
  t() %>% # 転置
  dist() %>% # 距離 (類似度) を測る
  hclust("ward.D2") %>% # クラスタリング
  dendro_data() # 作図用にデータを変換

# 描画
ggdendrogram(res_dendrogram, theme_dendro = FALSE)
```




dist() で各アカウント間の距離 (非類似度) を測り、hclust() でクラスター (グループ) を作っていきます。最後に dendro_data() で ggdendrogram() 用のデータ型に変換します。

```
# クラスタリング
res_dendrogram <- tw_count2[, -1] %>% # 非数値の列を落とす
  t() %>% # 転置
  dist() %>% # 距離 (類似度) を測る
  hclust("ward.D2") %>% # クラスタリング
  as.dendrogram() # 作図用にデータを変換

# 描画
plot(res_dendrogram)
```




plot() で作図する場合は最後のデータ変換に使う関数が as.dendrogram() になります。

これで廃人クラスタとかニートクラスタなんぞが見えてくるわけですね…

## ・ツイート画像を取得する

特定の単語と共にツイートされた画像を取得して保存します。

```
# 利用パッケージ
library(rtweet) # ツイートの取得:search_tweets()
library(magick) # 画像の取得:image_read(), image_write()
library(dplyr) # データ操作
library(tidyr) # unnest()
```

利用するパッケージを読み込みます。

### ・ツイートの収集

まずは rtweet パッケージを利用して、ツイートを集めます。

```
# 検索ワードを指定してツイートを取得
tw_data <- search_tweets("#鞘師里保誕生祭", n = 1000, include_rts = FALSE)
```

search_tweets() に検索ワードを指定してツイートを拾ってきます。引数 n には取得するツイート数を指定します。include_rts = FALSE で、リツイートを除くことができます (デフォルトは TRUE)。

取得したツイートデータから、画像の URL を抽出します。URL 情報は ext_media_url 列です。

```
# 画像の url を抽出
tw_pic_url <- tw_data %>%
  unnest(ext_media_url) %>% # 画像 url のネストを解除
  filter(!is.na(media_url)) %>%  # 画像のあるもののみ抽出
  select(created_at, screen_name, text, ext_media_url)
```

1 つのツイートに複数枚の画像が含まれる場合は URL がネスト (入れ子) されているので、unnest() で解除します。

逆に画像がないツイートは NA となっているので、filter() で NA でないもののみ取り出します。

最後に必要な情報を取り出します。必要なのは ext_media_url だけですが、重複等の確認のためにここではツイート日時・ユーザーネーム・ツイートテキストも取り出すことにします。

```
tail(tw_pic_url)
```

```
## # A tibble: 6 x 3
##   created_at          text                          ext_media_url
##   <dttm>              <chr>                         <chr>
## 1 2020-05-27 15:00:10 "誕生日おめでとう！！\n一生大好きです！\nイン~ http://pbs.twimg.com/media/EZC~
## 2 2020-05-27 15:00:08 "鞘師里保ちゃんHAPPY BIRTHDAY<U+2764><U+FE0F>\~ http://pbs.twimg.com/media/EZC~
## 3 2020-05-27 15:00:05 "#鞘師里保 #鞘師里保誕生祭\nりほりほお誕生日~ http://pbs.twimg.com/media/EZC~
## 4 2020-05-27 15:00:03 "鞘師里保ちゃん22歳のお誕生日おめでとう\U00~ http://pbs.twimg.com/media/EZC~
## 5 2020-05-27 15:00:02 "鞘師里保ちゃんお誕生日おめでとう\U0001f3~ http://pbs.twimg.com/media/EZC~
## 6 2020-05-27 13:09:12 "もう直ぐりほりほお誕生日だよー！\nみんなでお祝~ http://pbs.twimg.com/ext_tw_v
```

(ユーザーネームの情報は落としています。) ツイート日時は協定世界時なので、日本時間とはズレています。またネストを解除した影響で、画像枚数分複製されているものがあります。
問題がなければ、この URL から画像を保存していきます。

### ・画像の取得

magick を利用して、画像を取得します。

```
# 画像を取得
for(i in 1:nrow(tw_pic_url)) {

  # url を取り出す
  pic_url <- tw_pic_url[["ext_media_url"]][i]
```

```
  # 画像を取得
  pic_data <- image_read(pic_url)

  # 画像を保存
  image_write(pic_data, path = paste0("pic_data/rihoriho/rihoriho_", i, ".png"), format = "png")

  # おまじない
  Sys.sleep(1)
}
```

for() ループで、順番に画像を取得し保存します。

URL を 1 つ取り出し image_read() に渡して、画像を取得します。

取得した画像データを image_write() で書き出します。

サーバーに負荷がかかりすぎないように、Sys.sleep() で処理を一時停止しましょう。

指定したフォルダに画像が 1 つずつ保存されていきます。