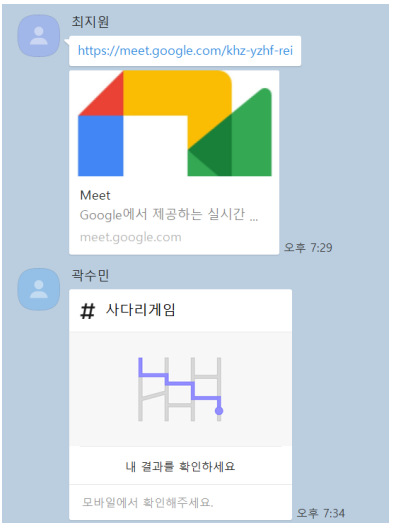
12~13강

상태	곽수민



▼ 시계열에 따른 사용자 전체의 상태 변화 찾기

- 서비스를 운영하는 입장에서는 사용자가 계속 사용하기를 원한다
- 휴면 사용자를 어떻게 하면 다시 사용할지 구상

1. 등록 수의 추이와 경향 보기

```
-- 등록 날짜별로 distinct 유저 수를 계산
select register_date, count(distinct user_id) register_count
from mst_users
group by 1
order by 1
```

- 월별 등록수 추이

```
-- mom으로 전월대비 비율까지
with mst_users_with_ym as (
select *, substr(register_date, 1, 7) as ym
from mst_users
)

select ym, count(distinct user_id) register_count
  , lag(count(distinct user_id)) over(order by ym)
  , 1.0 * count(distinct user_id) / lag(count(distinct user_id)) over (order by ym) mom
from mst_users_with_ym
group by 1
```

- 등록 디바이스별 추이

```
with mst_users_with_ym as (
select *, substr(register_date, 1, 7) as ym
from mst_users
)

select ym, count(distinct user_id) register_count
  , count(distinct case when register_device='pc' then user_id end) register_pc
  , count(distinct case when register_device='sp' then user_id end) register_sp
  , count(distinct case when register_device='app' then user_id end) register_app
```

```
from mst_users_with_ym
group by 1
```

2. 지속률과 정착률 산출하기

- 지속률 : 등록일 기준으로 이후 지정일 동안 사용자가 서비스를 얼마나 이용했는지 나타내는 지표
  - 6/12 등록한 유저가 13일까지 이용했다면 1일 지속자, 2일 후에도 이용한다면 2일 지속자
- 정착률(리텐션) : 등록일 기준으로 이후 7일동안 사용자가 서비스를 사용했는지 나타내는 지표
  - 7일 이내 한 번이라도 이용하면 언제 이용했냐와 다르게 모두 7일 정착자로 판단
- 지속률은 사용자가 매일 이용하길 기대하는 서비스(SNS 등), 정착률은 어떤 목적이 생겼을 때 사용했으면 하는 서비스 (리뷰 사이트 등등)일 때 주로 확인
- 날짜별 n일 지속률 추이

```
-- 유저별로 등록일 다음 날에 액션을 취했는지를 확인
with action_log_with_mst_users as (
select u.user_id, u.register_date, a.stamp::date action_date
  , max(a.stamp::date) over() latest_date
  , u.register_date::date + interval '1 day' next_day_1
from mst_users u
left join action_log a on u.user_id = a.user_id
)
, user_action_flag as (
select user_id, register_date,
  -- 등록일 다음날에 액션 했는지 아닌지
  sign(sum(case when next_day_1 <= latest_date
      then case when next_day_1 = action_date then 1 else 0 end end)) next_1_day_action
from action_log_with_mst_users
group by 1, 2
)

-- 유저별로 액션 유무 확인 후, 날짜별로 다음날 지속률을 계산
select register_date, avg(100.0 * next_1_day_action) repeat_rate_1_day
from user_action_flag
group by 1
order by 1
```

- 7일까지 1일 기준으로 지속률 관리하는 테이블 작성

```
with repeat_interval(index_name, interval_date) as (
values
  ('01 day repeat', 1),
  ('02 day repeat', 2),
  ('03 day repeat', 3),
  ('04 day repeat', 4),
  ('05 day repeat', 5),
  ('06 day repeat', 6),
  ('07 day repeat', 7)
)
, action_log_with_index_date as (
select u.user_id, u.register_date, a.stamp::date action_date
  , max(a.stamp::date) over() latest_date
  , r.index_name
  , u.register_date::date + interval '1 day' * r.interval_date as index_date
from mst_users u
left join action_log a on u.user_id = a.user_id
  -- cross join으로 1 ~ 7일까지의 interval을 벌린 날짜를 각각 집계
cross join repeat_interval r
)
, user_action_flag as (
select user_id, register_date, index_name
  , sign(sum(case when index_date <= latest_date then
       case when index_date = action_date then 1 else 0 end end)) index_date_action
from action_log_with_index_date
group by 1, 2, 3, index_date
)

select register_date, index_name, avg(100.0*index_date_action) repeat_date
from user_action_flag
group by 1, 2
order by 1, 2
```

- 정착률 계산

```
with repeat_interval(index_name, interval_begin_date, interval_end_date) as (
values
  ('07 day retention', 1, 7),
  ('14 day retention', 8, 14),
  ('21 day retention', 15, 21),
  ('28 day retention', 22, 28)
)
, action_log_with_index_date as (
select u.user_id, u.register_date, a.stamp::date action_date
  , max(a.stamp::date) over() latest_date
  , r.index_name
  , u.register_date::date + interval '1 day'*r.interval_begin_date index_begin_date
  , u.register_date::date + interval '1 day'*r.interval_end_date index_end_date
from mst_users u
left join action_log a on a.user_id = u.user_id
cross join repeat_interval r
)
, user_action_flag as (
select user_id, register_date, index_name
  , sign(sum(case when index_end_date <= latest_date then
        case when action_date between index_begin_date and index_end_date then 1 else 0 end end)) index_date_action
from action_log_with_index_date
group by 1, 2, 3, index_begin_date, index_end_date
)

select register_date, index_name, avg(100.0*index_date_action) index_rate
from user_action_flag
group by 1, 2
order by 1, 2
```

- n일 지속률과 n일 정착률 추이

```
with repeat_interval(index_name, interval_begin_date, interval_end_date) as (
values
  ('01 day repeat', 1, 1),
  ('02 day repeat', 2, 2),
  ('03 day repeat', 3, 3),
  ('04 day repeat', 4, 4),
  ('05 day repeat', 5, 5),
  ('06 day repeat', 6, 6),
  ('07 day repeat', 7, 7),
  ('07 day retention', 1, 7),
  ('14 day retention', 8, 14),
  ('21 day retention', 15, 21),
  ('28 day retention', 22, 28)
)
, action_log_with_index_date as (
select u.user_id, u.register_date, a.stamp::date action_date
  , max(a.stamp::date) over() latest_date
  , r.index_name
  , u.register_date::date + interval '1 day'*r.interval_begin_date index_begin_date
  , u.register_date::date + interval '1 day'*r.interval_end_date index_end_date
from mst_users u
left join action_log a on a.user_id = u.user_id
cross join repeat_interval r
)
, user_action_flag as (
select user_id, register_date, index_name
  , sign(sum(case when index_end_date <= latest_date then
        case when action_date between index_begin_date and index_end_date then 1 else 0 end end)) index_date_action
from action_log_with_index_date
group by 1, 2, 3, index_begin_date, index_end_date
)

select index_name, avg(100.0*index_date_action) index_rate
from user_action_flag
group by 1
order by 1
```

3. 지속과 정착에 영향을 주는 액션 집계하기

- 지속률과 정착률도 좋지만, 무엇이 그 지속률/정착률에 영향을 주는지 파악하는 것도 중요
- 따라서 액션에 따라 사용자/비사용자의 1일 지속률을 비교하면 됨

```
with repeat_interval(index_name, interval_begin_date, interval_end_date) as (
values
  ('01 day repeat', 1, 1)
)
, action_log_with_index_date as (
select u.user_id, u.register_date, a.stamp::date action_date
  , max(a.stamp::date) over() latest_date
  , r.index_name
  , u.register_date::date + interval '1 day'*r.interval_begin_date index_begin_date
  , u.register_date::date + interval '1 day'*r.interval_end_date index_end_date
from mst_users u
left join action_log a on a.user_id = u.user_id
cross join repeat_interval r
)
, user_action_flag as (
select user_id, register_date, index_name
  , sign(sum(case when index_end_date <= latest_date then
        case when action_date between index_begin_date and index_end_date then 1 else 0 end end)) index_date_action
from action_log_with_index_date
group by 1, 2, 3, index_begin_date, index_end_date
)
, mst_actions as (
select 'view' action
union all select 'comment'
union all select 'follow'
)
, mst_user_actions as (
select u.user_id, u.register_date, a.action
from mst_users u
cross join mst_actions a
)
, register_action_flag as (
select distinct m.user_id, m.register_date, m.action
  , case when a.action is not null then 1 else 0 end do_action
  , index_name
  , index_date_action
from mst_user_actions m
left join action_log a on m.user_id = a.user_id
          and m.register_date::date = a.stamp::date
          and a.action = m.action
left join user_action_flag f on m.user_id = f.user_id
where f.index_date_action is not null
)

select action, count(1) users, avg(100.0*do_action) usage_rate
  , index_name
  , avg(case do_action when 1 then 100.0*index_date_action end) idx_rate
  , avg(case do_action when 0 then 100.0*index_date_action end) no_action_idx_rate
from register_action_flag
group by index_name, action
order by index_name, action
```

4. 액션 수에 따른 정착률 집계하기

```
-- 등록일로부터 7일 이내에 comment, follow 액션을 취한 횟수 그룹에 따라 14일 리텐션율을 집계

with repeat_interval(index_name, interval_begin_date, interval_end_date) as (
values
  ('14 day retention', 8, 14)
)
, action_log_with_index_date as (
select u.user_id, u.register_date, a.stamp::date action_date
  , max(a.stamp::date) over() latest_date
  , r.index_name
  , u.register_date::date + interval '1 day'*r.interval_begin_date index_begin_date
  , u.register_date::date + interval '1 day'*r.interval_end_date index_end_date
from mst_users u
left join action_log a on a.user_id = u.user_id
cross join repeat_interval r
)
, user_action_flag as (
select user_id, register_date, index_name
  , sign(sum(case when index_end_date <= latest_date then
        case when action_date between index_begin_date and index_end_date then 1 else 0 end end)) index_date_action
from action_log_with_index_date
group by 1, 2, 3, index_begin_date, index_end_date
)
, mst_action_bucket(action, min_count, max_count) as (
values
```

```
  ('comment', 0, 0),
  ('comment', 1, 5),
  ('comment', 6, 10),
  ('comment', 11, 9999),
  ('follow', 0, 0),
  ('follow', 1, 5),
  ('follow', 6, 10),
  ('follow', 11, 9999)
)
, mst_user_action_bucket as (
select u.user_id, u.register_date, a.action, a.min_count, a.max_count
from mst_users u
cross join mst_action_bucket a
)
, register_action_flag as (
select m.user_id, m.action, m.min_count, m.max_count
  , count(a.action) action_count
  , case when count(a.action) between m.min_count and m.max_count then 1 else 0 end achieve
  , index_name
  , index_date_action
from mst_user_action_bucket m
left join action_log a on m.user_id = a.user_id
      and a.stamp::date between m.register_date::date and m.register_date::date + interval '7 days'
      and m.action = a.action
left join user_action_flag f on m.user_id = f.user_id
where f.index_date_action is not null
group by 1, 2, 3, 4, 7, 8
)

select action, min_count||'~'||max_count count_range
  , sum(case achieve when 1 then 1 else 0 end) achieve
  , index_name
  , avg(case achieve when 1 then 100.0*index_date_action end) achieve_index_rate
from register_action_flag
group by index_name, action, min_count, max_count
order by index_name, action, min_count
```

5. 사용 일수에 따른 정착률 집계하기

- 사용 일수 날짜별로 정착률 집계

```
with repeat_interval(index_name, interval_begin_date, interval_end_date) as (
values
  ('28 day retention', 22, 28)
)
, action_log_with_index_date as (
select u.user_id, u.register_date, a.stamp::date action_date
  , max(a.stamp::date) over() latest_date
  , r.index_name
  , u.register_date::date + interval '1 day'*r.interval_begin_date index_begin_date
  , u.register_date::date + interval '1 day'*r.interval_end_date index_end_date
from mst_users u
left join action_log a on a.user_id = u.user_id
cross join repeat_interval r
)
, user_action_flag as (
select user_id, register_date, index_name
  , sign(sum(case when index_end_date <= latest_date then
       case when action_date between index_begin_date and index_end_date then 1 else 0 end end)) index_date_action
from action_log_with_index_date
group by 1, 2, 3, index_begin_date, index_end_date
)
, register_action_flag as (
select m.user_id
  , count(distinct a.stamp::date) dt_count
  , index_name, index_date_action
from mst_users m
left join action_log a on m.user_id = a.user_id
      and a.stamp::date between m.register_date::date + interval '1 day' and m.register_date::date + interval '8 day'
left join user_action_flag f on m.user_id = f.user_id
where f.index_date_action is not null
group by 1, 3, 4
)

select dt_count dates
  , count(user_id) users
  , 100.0 * count(user_id) / sum(count(user_id)) over() user_ratio
  , 100.0 * sum(count(user_id)) over(order by index_name, dt_count
                    rows between unbounded preceding and current row)
                    / sum(count(user_id)) over() as cum_ratio
  , sum(index_date_action) achieve_users
```

```
  , avg(100.0 * index_date_action) achieve_ratio
from register_action_flag
group by index_name, dt_count
order by index_name, dt_count
```

6. 사용자의 잔존율 집계하기

- 월 단위로 서비스를 계속 이용하는지 잔존율(리텐션)을 계산

```
with mst_intervals(interval_month) as (
values (1), (2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12)
)
, mst_users_with_index_month as (
select u.user_id, u.register_date
  , u.register_date::date + interval '1 month'*i.interval_month index_date
  , substr(u.register_date, 1, 7) as register_month
  , substr(cast(u.register_date::date + interval '1 month'*i.interval_month as text), 1, 7) as index_month
from mst_users u
cross join mst_intervals i
)
, action_log_in_month as (
select distinct user_id, substr(stamp, 1, 7) action_month
from action_log
)

select u.register_month, u.index_month
  , sum(case when a.action_month is not null then 1 else 0 end) users
  , avg(case when a.action_month is not null then 100.0 else 0.0 end) retension_rate
from mst_users_with_index_month u
left join action_log_in_month a on a.user_id = u.user_id
            and u.index_month = a.action_month
group by 1, 2
order by 1, 2
```

7. 방문 빈도를 기반으로 사용자 속성을 정의하고 집계하기

- MAU : Monthly Active Users. 특정 월에 서비스를 이용한 사용자 수
- 하지만 MAU만 보는 것이 아니라, MAU 중 몇 명이 신규 사용자인지 기존 사용자인지 등에 대한 정보가 필요
- MAU를 3개로 나누어 분석하기
  - 신규 사용자 : 이번 달에 등록한 신규 사용자
  - 리피트(Repeat) 사용자 : 이전 달에도 사용했던 사용자
  - 컴백 사용자 : 이번 달의 신규 등록자가 아니고, 이전 달에도 사용하지 않았던 사용자
- 신규 사용자, 리피트 사용자, 컴백 사용자를 집계하는 쿼리

```
with monthly_user_action as (
select distinct u.user_id
  , substr(u.register_date, 1, 7) register_month
  , substr(l.stamp, 1, 7) action_month
  , substr(cast(l.stamp::date - interval '1 month' as text), 1, 7) action_month_priv
from mst_users u
left join action_log l on u.user_id = l.user_id
    -- 등록만 하고 액션 로그가 없는 경우 제거용 (추가)
where l.stamp is not null
)
, monthly_user_with_type as (
select action_month, user_id
  , case when register_month = action_month then 'new_user'
    when action_month_priv = lag(action_month) over(partition by user_id order by action_month) then 'repeat_user'
    else 'comeback_user' end c
  , action_month_priv
from monthly_user_action
)

select action_month, count(user_id) mau
  , count(case when c = 'new_user' then 1 end) new_users
  , count(case when c = 'repeat_user' then 1 end) repeat_users
  , count(case when c = 'comeback_user' then 1 end) comback_users
from monthly_user_with_type
group by 1
order by 1
```

- 리피트 사용자를 3가지로 분류하기
  - 신규 리피트 사용자 : 이전 달에는 신규 사용자였으며, 이번 달에는 리피트 사용자
  - 기존 리피트 사용자 : 이전 달에도 리피트 사용자였으며, 이번 달에도 리피트 사용자
  - 컴백 리피트 사용자 : 이전 달에는 컴백 사용자였으며, 이번 달에는 리피트 사용자

```
with monthly_user_action as (
select distinct u.user_id
  , substr(u.register_date, 1, 7) register_month
  , substr(l.stamp, 1, 7) action_month
  , substr(cast(l.stamp::date - interval '1 month' as text), 1, 7) action_month_priv
from mst_users u
left join action_log l on u.user_id = l.user_id
where l.stamp is not null
)
, monthly_user_with_type as (
select action_month, user_id
  , case when register_month = action_month then 'new_user'
    when action_month_priv = lag(action_month) over(partition by user_id order by action_month) then 'repeat_user'
    else 'comeback_user' end c
  , action_month_priv
from monthly_user_action
)
, monthly_users as (
select m1.action_month
  , count(m1.user_id) mau
  , count(case when m1.c = 'new_user' then 1 end) new_users
  , count(case when m1.c = 'repeat_user' then 1 end) repeat_users
  , count(case when m1.c = 'comeback_user' then 1 end) comeback_users

  , count(case when m1.c = 'repeat_user' and m0.c = 'new_user' then 1 end) new_repeat_users
  , count(case when m1.c = 'repeat_user' and m0.c = 'repeat_user' then 1 end) continuous_repeat_users
  , count(case when m1.c = 'repeat_user' and m0.c = 'comeback_user' then 1 end) comeback_repeat_users

from monthly_user_with_type m1
left join monthly_user_with_type m0 on m1.action_month_priv = m0.action_month
                and m1.user_id = m0.user_id
group by 1
)

select *
from monthly_users
order by action_month
```

- MAU 속성별 반복률 계산하기

```
with monthly_user_action as (
select distinct u.user_id
  , substr(u.register_date, 1, 7) register_month
  , substr(l.stamp, 1, 7) action_month
  , substr(cast(l.stamp::date - interval '1 month' as text), 1, 7) action_month_priv
from mst_users u
left join action_log l on u.user_id = l.user_id
where l.stamp is not null
)
, monthly_user_with_type as (
select action_month, user_id
  , case when register_month = action_month then 'new_user'
    when action_month_priv = lag(action_month) over(partition by user_id order by action_month) then 'repeat_user'
    else 'comeback_user' end c
  , action_month_priv
from monthly_user_action
)
, monthly_users as (
select m1.action_month
  , count(m1.user_id) mau
  , count(case when m1.c = 'new_user' then 1 end) new_users
  , count(case when m1.c = 'repeat_user' then 1 end) repeat_users
  , count(case when m1.c = 'comeback_user' then 1 end) comeback_users

  , count(case when m1.c = 'repeat_user' and m0.c = 'new_user' then 1 end) new_repeat_users
  , count(case when m1.c = 'repeat_user' and m0.c = 'repeat_user' then 1 end) continuous_repeat_users
  , count(case when m1.c = 'repeat_user' and m0.c = 'comeback_user' then 1 end) comeback_repeat_users

from monthly_user_with_type m1
left join monthly_user_with_type m0 on m1.action_month_priv = m0.action_month
                and m1.user_id = m0.user_id
```

```
group by 1
)

select *
  -- 이전 달에 신규 사용자면서 해당 월에 신규 리피트 사용자 사용자의 비율
  , 100.0 * new_repeat_users / nullif(lag(new_users) over(order by action_month), 0) priv_new_repeat_ratio
  -- 이전 달에 리피트 사용자면서 해당 월에도 리피트 사용자 사용자의 비율
  , 100.0 * continuous_repeat_users / nullif(lag(repeat_users) over(order by action_month), 0) priv_continuous_repeat_repeat_ratio
  -- 이전 달에컴백 사용자면서 해당 월에 리피트 사용자 사용자의 비율
  , 100.0 * comeback_users / nullif(lag(repeat_users) over(order by action_month), 0) priv_comeback_repeat_ratio
from monthly_users
order by 1
```

8. 방문 종류를 기반으로 성장지수 집계하기

- 그로스 해킹 : 사용자의 지속 사용, 리텐션 등을 높이고 서비스 성장을 가속화 시키기 위한 업무
- 성장지수 : 서비스 사용과 관련한 상태 변화를 수치화해서 서비스가 성장하는지 알려주는 지표
- 여기서는 성장지수 $= Signup + Reactivation - Deactivation - Exit$ 으로 계산
- 성장지수 집계하기

```
with unique_action_log as (
select distinct user_id, stamp::date action_date
from action_log
)
, mst_calendar as (
-- 집계하고 싶은 기간을 캘린더 테이블로 만들어두기
select '2016-10-01' dt
union all select '2016-10-02'
union all select '2016-10-03'
union all select '2016-10-04'
union all select '2016-10-05'
union all select '2016-10-06'
union all select '2016-10-07'
union all select '2016-10-08'
union all select '2016-10-09'
union all select '2016-10-10'
union all select '2016-10-11'
union all select '2016-10-12'
union all select '2016-10-13'
union all select '2016-10-14'
union all select '2016-10-15'
union all select '2016-10-16'
union all select '2016-10-17'
union all select '2016-10-18'
union all select '2016-10-19'
union all select '2016-10-20'
union all select '2016-10-21'
union all select '2016-10-22'
union all select '2016-10-23'
union all select '2016-10-24'
union all select '2016-10-25'
union all select '2016-10-26'
union all select '2016-10-27'
union all select '2016-10-28'
union all select '2016-10-29'
union all select '2016-10-30'
union all select '2016-10-31'
union all select '2016-11-01'
union all select '2016-11-02'
union all select '2016-11-03'
union all select '2016-11-04'
)
, target_date_with_user as (
select c.dt target_date
  , u.user_id
  , u.register_date
  , u.withdraw_date
from mst_users u
cross join mst_calendar c
)
, user_status_log as (
select u.target_date, u.user_id, u.register_date, u.withdraw_date
  , a.action_date
  , case when u.register_date::date = a.action_date then 1 else 0 end is_new
  , case when u.withdraw_date::date = a.action_date then 1 else 0 end is_exit
  , case when u.target_date::date = a.action_date then 1 else 0 end is_access
  , lag(case when u.target_date::date = a.action_date then 1 else 0 end)
```

```
  over(partition by u.user_id order by u.target_date) was_access
from target_date_with_user u
left join unique_action_log a on u.user_id = a.user_id
              and u.target_date::date = a.action_date
where u.register_date <= u.target_date
  and (u.withdraw_date is null or u.target_date <= u.withdraw_date)
)
, user_growth_index as (
select *
  -- 어떤 날에 신규 등록 또는 탈퇴한 경우 signup 또는 exit
  , case when is_new + is_exit = 1 then
    case when is_new = 1 then 'signup'
      when is_exit = 1 then 'exit'
    end
  when is_new + is_exit = 0 then
    case when was_access = 0 and is_access = 1 then 'reactivation'
    when was_access = 1 and is_access = 0 then 'deactivation'
    end
  end growth_index
from user_status_log
)


select target_date
  , sum(case growth_index when 'signup' then 1 else 0 end) signup
  , sum(case growth_index when 'reactivation' then 1 else 0 end) reactivation
  , sum(case growth_index when 'deactivation' then -1 else 0 end) deactivation
  , sum(case growth_index when 'exit' then -1 else 0 end) exit
  , sum(case growth_index
     when 'signup' then 1
     when 'reactivation' then 1
     when 'deactivation' then -1
     when 'exit' then -1 else 0 end) growth_index
from user_growth_index
group by 1
order by 1
```

9. 지표 개선 방법 익히기

- 지표를 향상 시키려면
  1. 달성하고 싶은 지표를 결정한다
  2. 사용자 행동 중에서 지표에 영향을 많이 줄 것으로 보이는 행동을 결정한다
  3. 위에서 결정한 행동 여부와 횟수를 집계하고, 설정한 지표를 만족하는 사용자의 비율을 비교한다
- 즉 **어떠한 액션이 영향을 주었는지를 확인하는 것**이 포인트!
- EX) 글의 업로드와 댓글 수를 늘리고 싶은 경우
  - 팔로우된 사람에 따라 차이가 있는가?
  - 팔로우하는 사람 수에 따라 차이가 있는가?
  - 프로필 사진을 등록한 사람에 따라 차이가 있는가?
- EX) 신규 사용자의 리피트율을 개선하고 싶은 경우
  - 등록한 달에 올린 글의 수에 따라 차이가 있는가?
  - 등록한 달에 팔로우한 사람 수에 따라 차이가 있는가?
  - 등록 다음날부터 7일 이내의 사용 일수에 따라 차이가 있는가?
- EX) CVR을 개선하고 싶은 경우
  - 구매 전에 상세 페이지를 본 횟수에 따라 차이가 있는가?
  - 구매 전에 관심 상품 기능 사용 여부에 따라 차이가 있는가?

▼ 시계열에 따른 사용자의 개별적인 행동 분석하기

1. 사용자의 액션 간격 집계하기

- 사용자의 특정 액션을 기반으로 다음 액션까지의 기간을 집계해 새로운 문제를 찾을 수 있음
- 예를 들어, `자료청구` → `방문예약` → `견적` → `계약`의 흐름으로 진행될 때, 실제 계약까지의 기간(리드타임)을 확인해 각각의 흐름에서의 문제를 찾을 수 있음

- 같은 레코드에 있는 두개의 날짜로 계산할 경우
  - 숙박 시설, 음식점 등의 예약 테이블은 저장하는 레코드에 신청일과 숙박일, 방문일을 한 번에 저장

```
with reservations(reservation_id, register_date, visit_date, days) as (
values
  (1, date '2016-09-01', date '2016-10-01', 3),
  (2, date '2016-09-20', date '2016-10-01', 2),
  (3, date '2016-09-30', date '2016-11-20', 2),
  (4, date '2016-10-01', date '2017-01-03', 2),
  (5, date '2016-11-01', date '2016-12-28', 3)
)

select reservation_id, register_date, visit_date
  , visit_date - register_date lead_time
from reservations
```

- 여러 테이블에 있는 여러 개의 날짜로 계산할 경우

```
-- 그냥 조인해서 앞에랑 똑같이 하면 된다
select r.user_id, r.product_id
  , e.estimate_date - r.request_date as estimate_lead_time
  , o.order_date - e.estimate_date as order_lead_time
  , o.order_date - r.request_date as total_lead_time
from requests r
left join estimates e on r.user_id = e.user_id and r.product_id = e.product_id
left join orders o on r.user_id = o.user_id and r.product_id = o.product_id
```

- 같은 테이블에 다른 레코드에 있는 날짜로 계산할 경우
  - 예) 이전 구매일로부터 다음 구매일까지의 리드타임을 알고 싶은 경우

```
with purchase_log(user_id, product_id, purchase_date) as (
values
  ('U001', '1', '2016-09-01'),
  ('U001', '2', '2016-09-20'),
  ('U002', '3', '2016-09-30'),
  ('U001', '4', '2016-10-01'),
  ('U002', '5', '2016-11-01')
)

select user_id, purchase_date,
  purchase_date::date - lag(purchase_date::date)
            over(partition by user_id order by purchase_date) lead_time
from purchase_log
```

2. 카트 추가 후에 구매했는지 파악하기

- 카트 탈락 : 카트에 넣은 상품을 구매하지 않고 이탈한 상황
  - 상품 구매까지의 절차에 어떤 문제가 있다
  - 예상하지 못한 비용 (배송비 또는 수수료)로 당황했다
  - 북마크 기능 대신 카트를 사용했다
- 카트에 추가된지 48시간 이내에 구매되지 않은 상품을 '카트 탈락'으로 정의하고, 카트탈락률을 집계

```
with row_action_log as (
select dt, user_id, action
  , regexp_split_to_table(products, ',') as product_id
  , stamp
from action_log
)
, action_time_stats as (
select user_id, product_id
  , min(case action when 'add_cart' then dt end) dt
  , min(case action when 'add_cart' then stamp end) add_cart_time
  , min(case action when 'purchase' then stamp end) purchase_time
  , extract(epoch from min(case action when 'purchase' then stamp::timestamp end)
      - min(case action when 'add_cart' then stamp::timestamp end)) as lead_time
from row_action_log
group by 1, 2
)
```

```
, purchase_leadtime_flag as (
select user_id, product_id, dt
  , case when lead_time <= 1*60*60 then 1 else 0 end purchase_1_hour
  , case when lead_time <= 6*60*60 then 1 else 0 end purchase_6_hour
  , case when lead_time <= 24*60*60 then 1 else 0 end purchase_24_hour
  , case when lead_time <= 48*60*60 then 1 else 0 end purchase_48_hour
  , case when lead_time is null or not(lead_time <= 48*60*60) then 1 else 0 end not_purchase
from action_time_stats
)

select dt, count(*) as add_cart
  , sum(purchase_1_hour) purchase_1_hour
  , avg(purchase_1_hour) purchase_1_hour_rate
  , sum(purchase_6_hour) purchase_6_hour
  , avg(purchase_6_hour) purchase_6_hour_rate
  , sum(purchase_24_hour) purchase_24_hour
  , avg(purchase_24_hour) purchase_24_hour_rate
  , sum(purchase_48_hour) purchase_48_hour
  , avg(purchase_48_hour) purchase_48_hour_rate

  , sum(not_purchase) not_purchase
  , avg(not_purchase) not_purchase_rate
from purchase_leadtime_flag
group by 1
```

3. 등록으로부터의 매출을 날짜별로 집계하기

- 등록 월 기준으로 n일 경과 시점의 1인당 매출 금액을 집계

```
with index_intervals(index_name, interval_begin_date, interval_end_date) as (
values
  ('30 day sales amount', 0, 30),
  ('45 day sales amount', 0, 45),
  ('60 day sales amount', 0, 60)
)
, mst_users_with_base_date as (
select user_id
  , register_date base_date
from mst_users
)
, purchase_log_with_index_date as (
select u.user_id, u.base_date, p.stamp::date action_date
  , max(p.stamp::date) over() latest_date
  , substr(u.base_date, 1, 7) as month
  , i.index_name
  , u.base_date::date + interval '1 day'*i.interval_begin_date index_begin_date
  , u.base_date::date + interval '1 day'*i.interval_end_date index_end_date
  , p.amount
from mst_users_with_base_date u
left join action_log p on u.user_id = p.user_id and p.action='purchase'
cross join index_intervals i
)
, user_purchase_amount as (
select user_id, month, index_name
  , sum(case when index_end_date <= latest_date then
     case when action_date between index_begin_date and index_end_date then amount else 0 end end) index_date_amount
from purchase_log_with_index_date
group by 1, 2, 3, index_begin_date, index_end_date
)


select month, count(index_date_amount) users
  , index_name, count(case when index_date_amount > 0 then user_id end) purchase_uu
  , sum(index_date_amount) total_amount
  , avg(index_date_amount) avg_amount
from user_purchase_amount
group by month, index_name
order by month, index_name
```

- 서비스의 사용자 수를 분모에 넣으면 ARPU(1인당 평균 매출 금액)
- 과금 사용자 수를 분모에 넣으면 ARPPU(과금 사용자 1인당 평균 매출 금액)
- 프리미엄 모델에서 과금/무과금 사용자를 구별할 필요가 있으면 ARPPU를 사용

LTV

- 고객 생애 가치(Life Time Value)

- $LTV =$ 연간거래액 $*$ 수익률 $*$ 지속연수