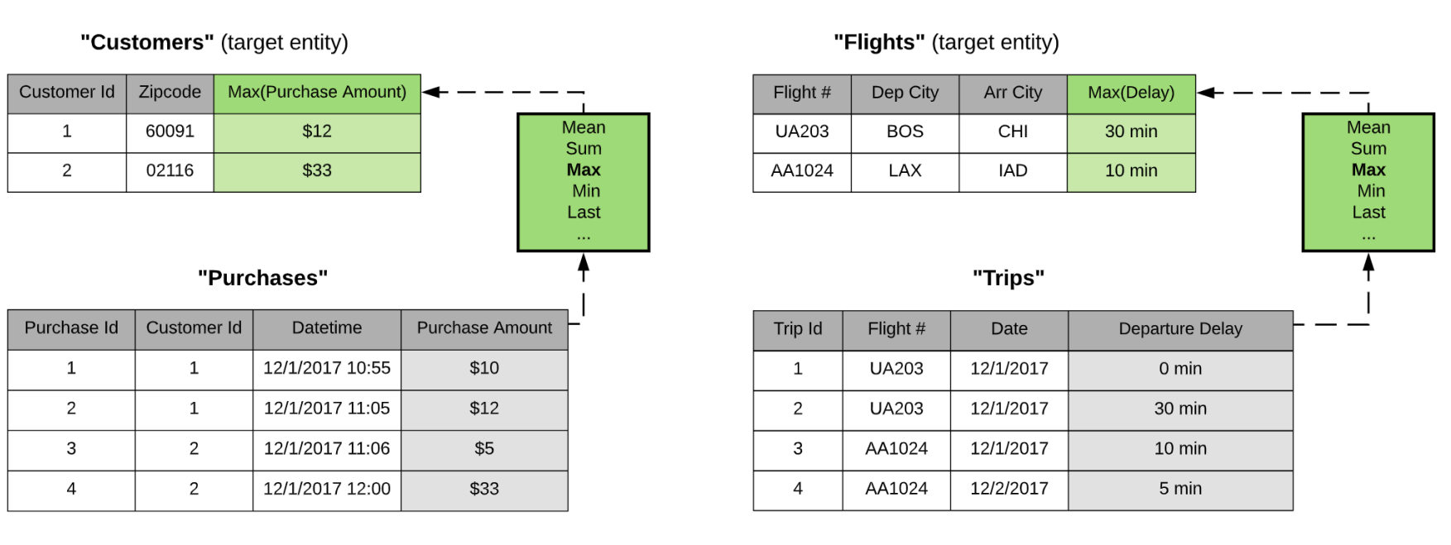
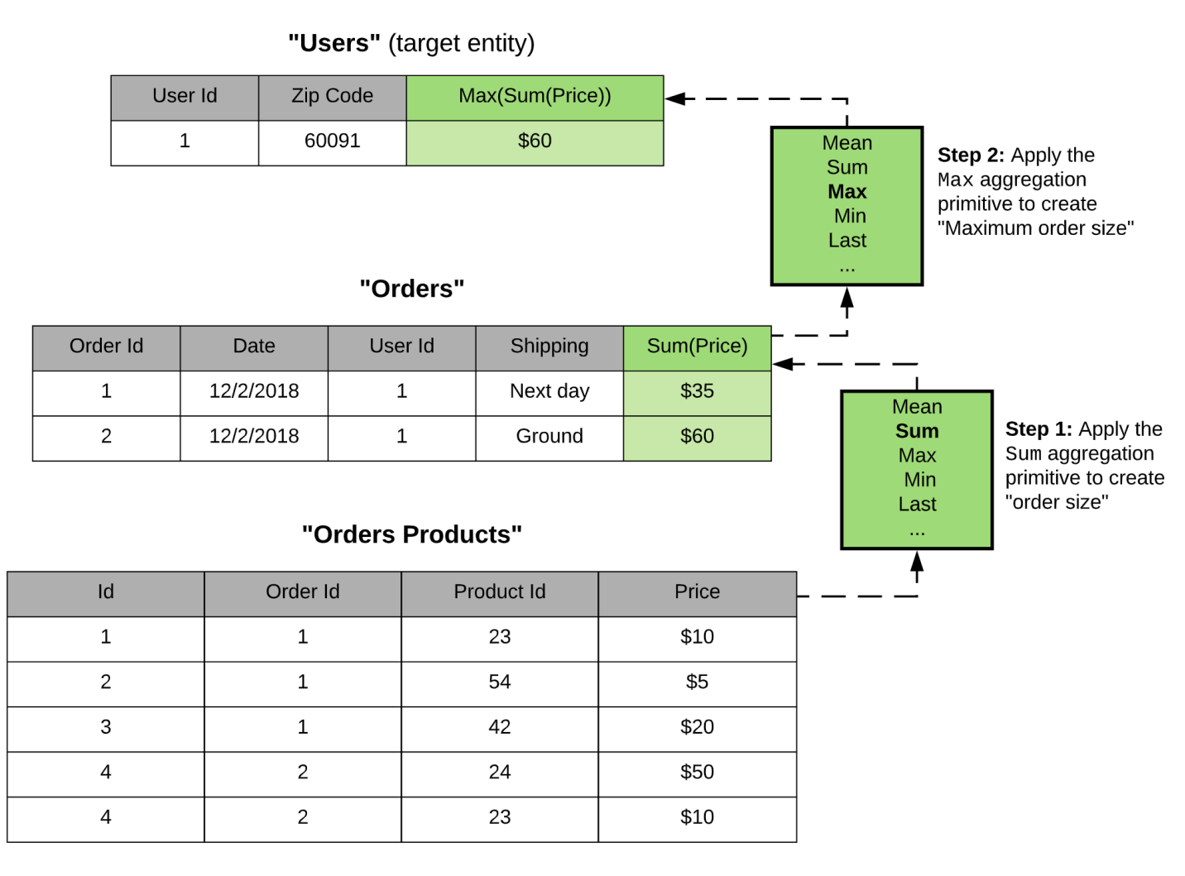
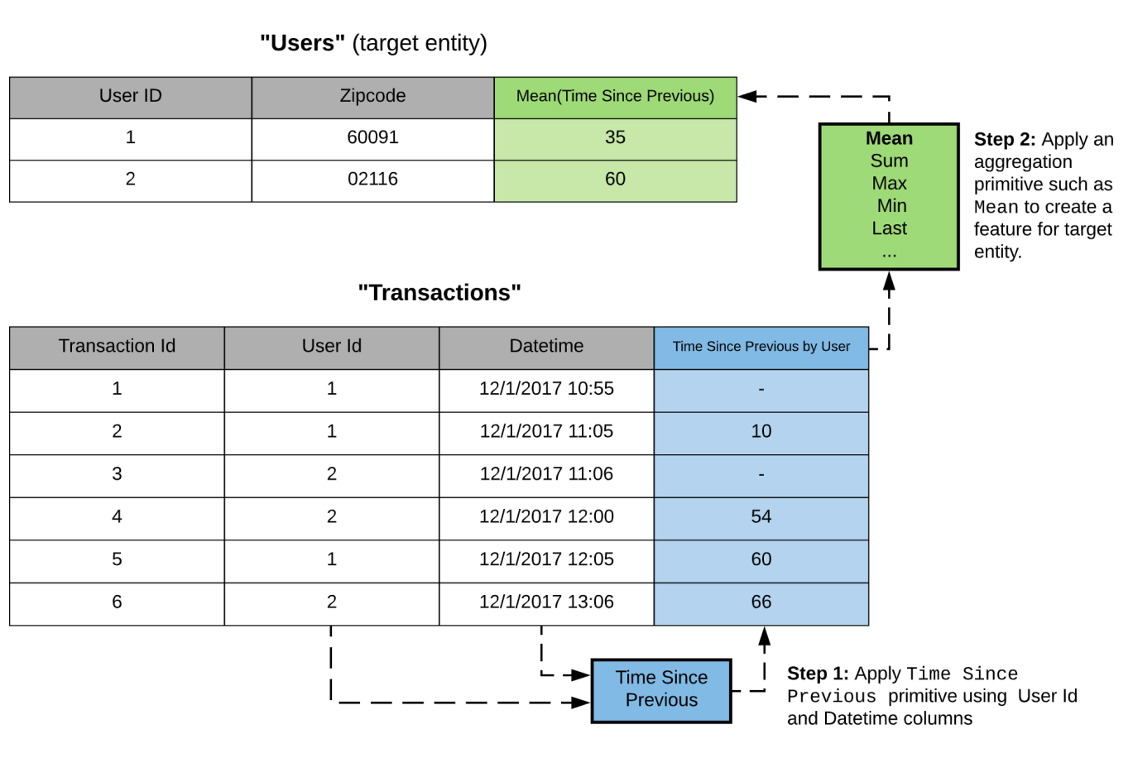
출처

- https://blog.featurelabs.com/deep-feature-synthesis/
- https://docs.featuretools.com/en/stable/index.html

3가지 key point

- 데이터 point들의 관계로부터 feature를 유도한다.
    - 대부분의 현실 데이터는 여전히 관계형 데이터
- 도메인이 다른 데이터라 할지라도, 유사한 수학적인 operation을 통해서 유사한 feature가 유도된다.
    - 

      To calculate a customer's most expensive purchase, we apply the `Max` primitive to the purchase amount field in all related purchases. When we perform the same steps to a dataset of airplane flights, we calculate "the longest flight delay".
- 도출된 기존 feature들의 조합(compositional)을 통해서 새로운 feature가 유도될 수 있다.
    - 

    - 

Entity와 Relation 정의

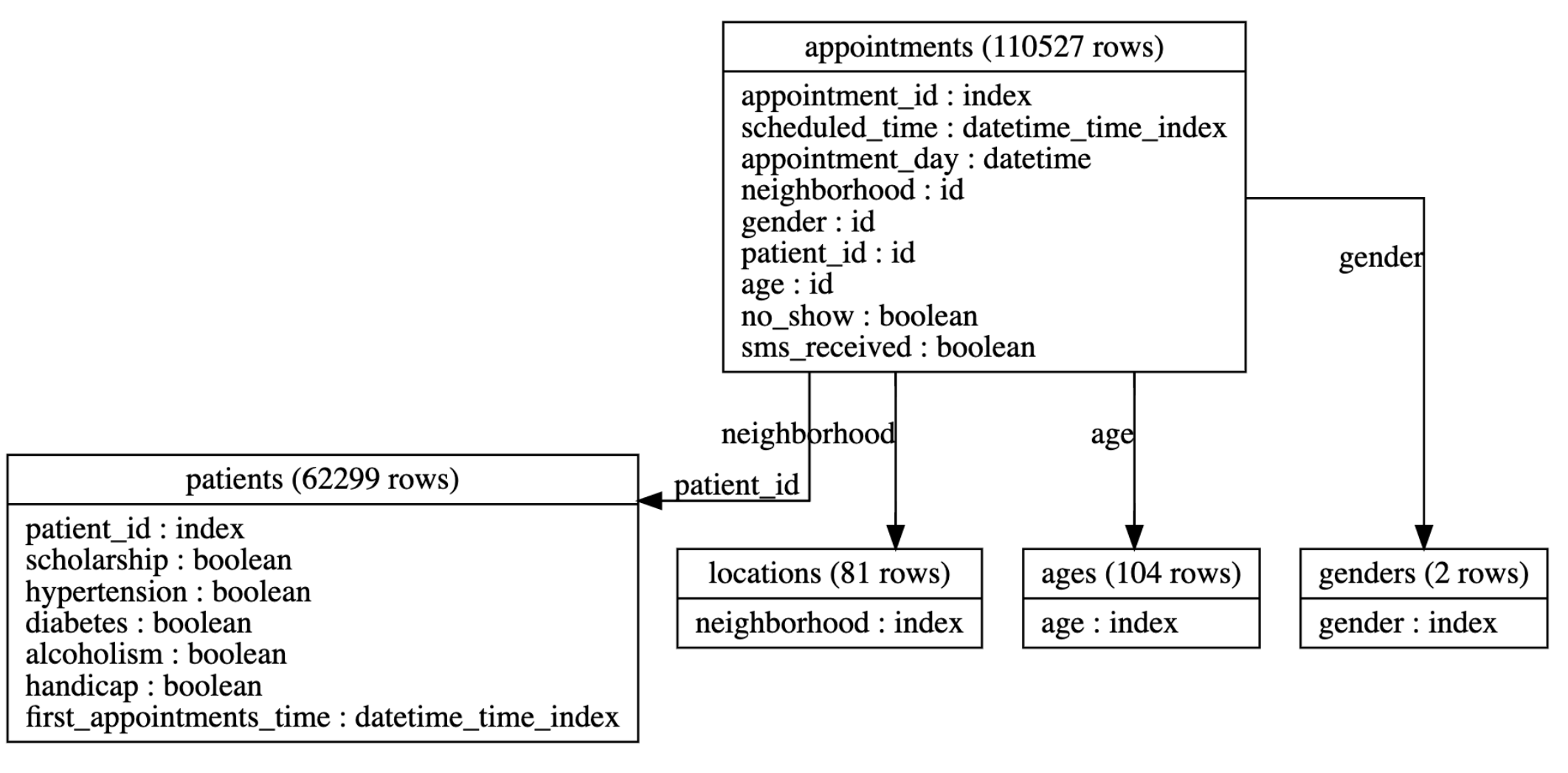
- Entity
    - 하나의 relation table에 대응, ex) customer table
    - 보통은 uuid column이 존재해서 다른 table과 관계 형성, ex) customer id
- Relationship
    - Foreign key column을 매개체로 해서 두 relation table이 관계 형성
    - Ex) customer entity, customer id => session entity, customer id
- EntitySet API
    - 복수 개의 entity와 relationship의 집합체를 소유한 class로써 새로운 entity, relationship factory 역할도 한다.

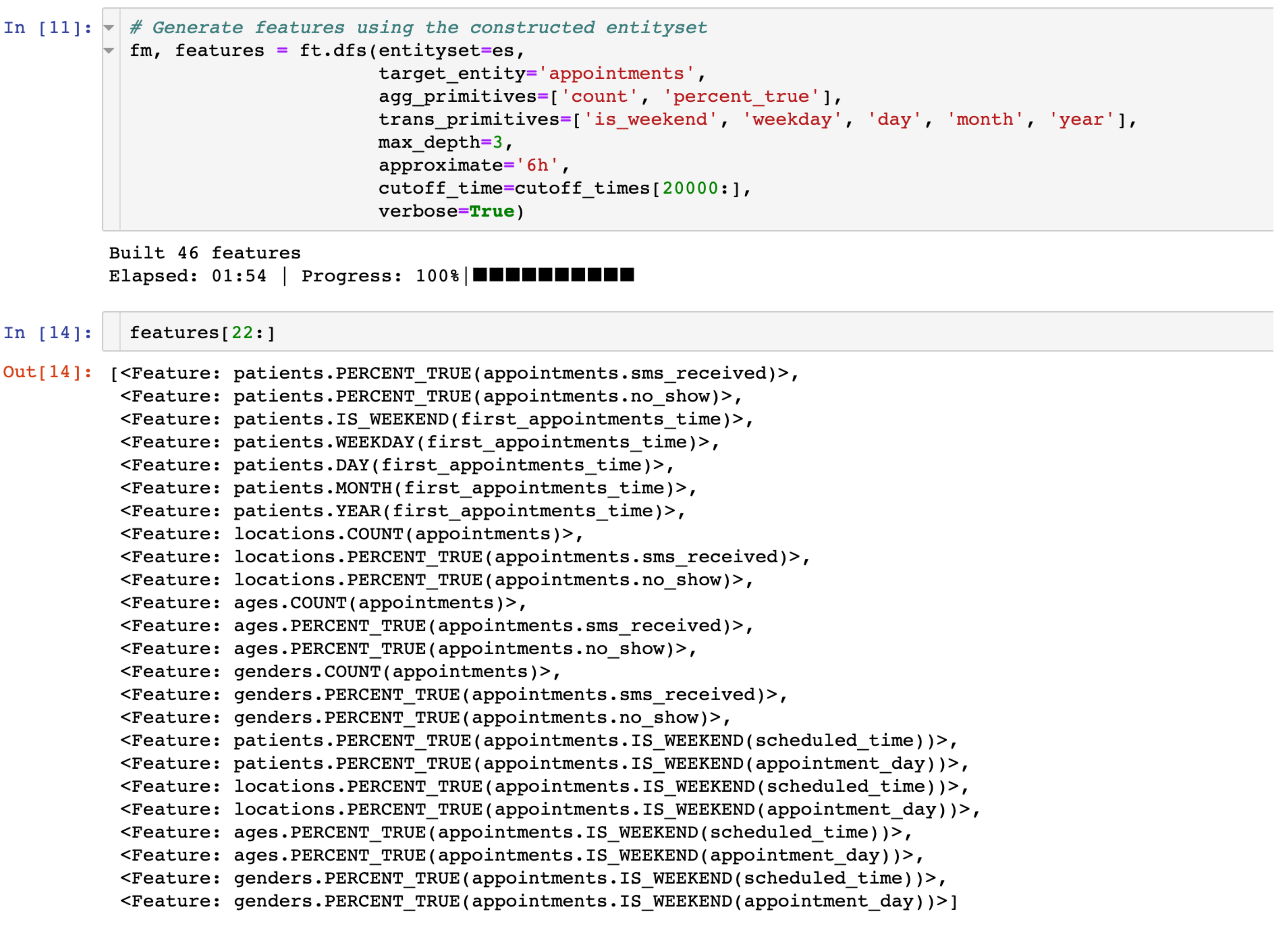
Feature primitives

- featuretools의 building block에 해당
- 이들을 쌓아올려서 new feature가 만들어진다.
- Data type에만 constraint가 있으므로, 2차,3차 등의 깊은 feature가 형성된다.
- Ex) `SUM(sessions.MEAN(transactions.amount))`
- 종류
    - `aggregation`
        - 복수 개의 인스턴스들(행)을 받아들여서 하나의 값을 출력한다. `Ex) sum, mean`
    - `Transformative`
        - 하나의 값을 입력으로 하나의 출력을 만든다. `Ex) absolute`
- 사용자 정의 `primitive`
    - `Ex) absolute transformation`
    - `Ex) WordCount transformation = Text`를 받아들여 `word` 개수 `count`

생성된 feature의 예시

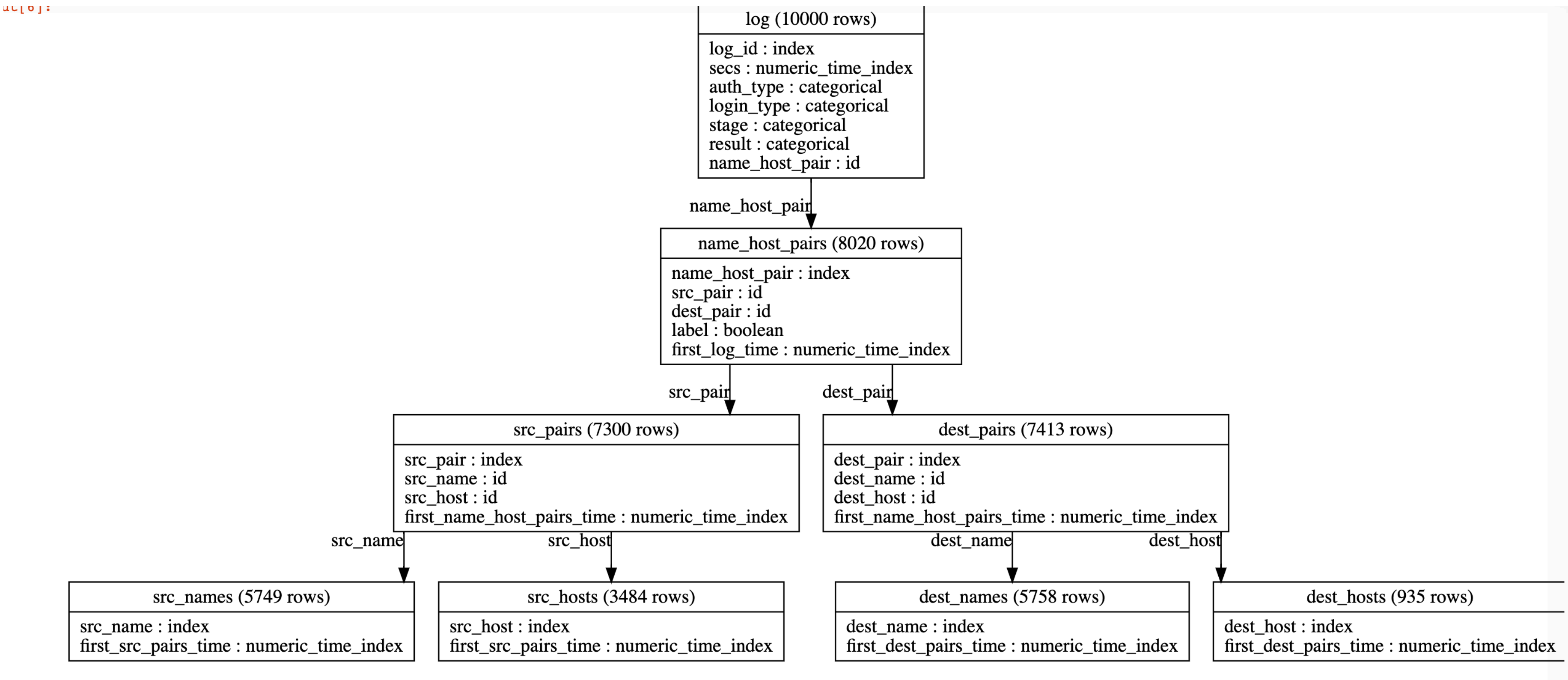
병원 no-show 문제

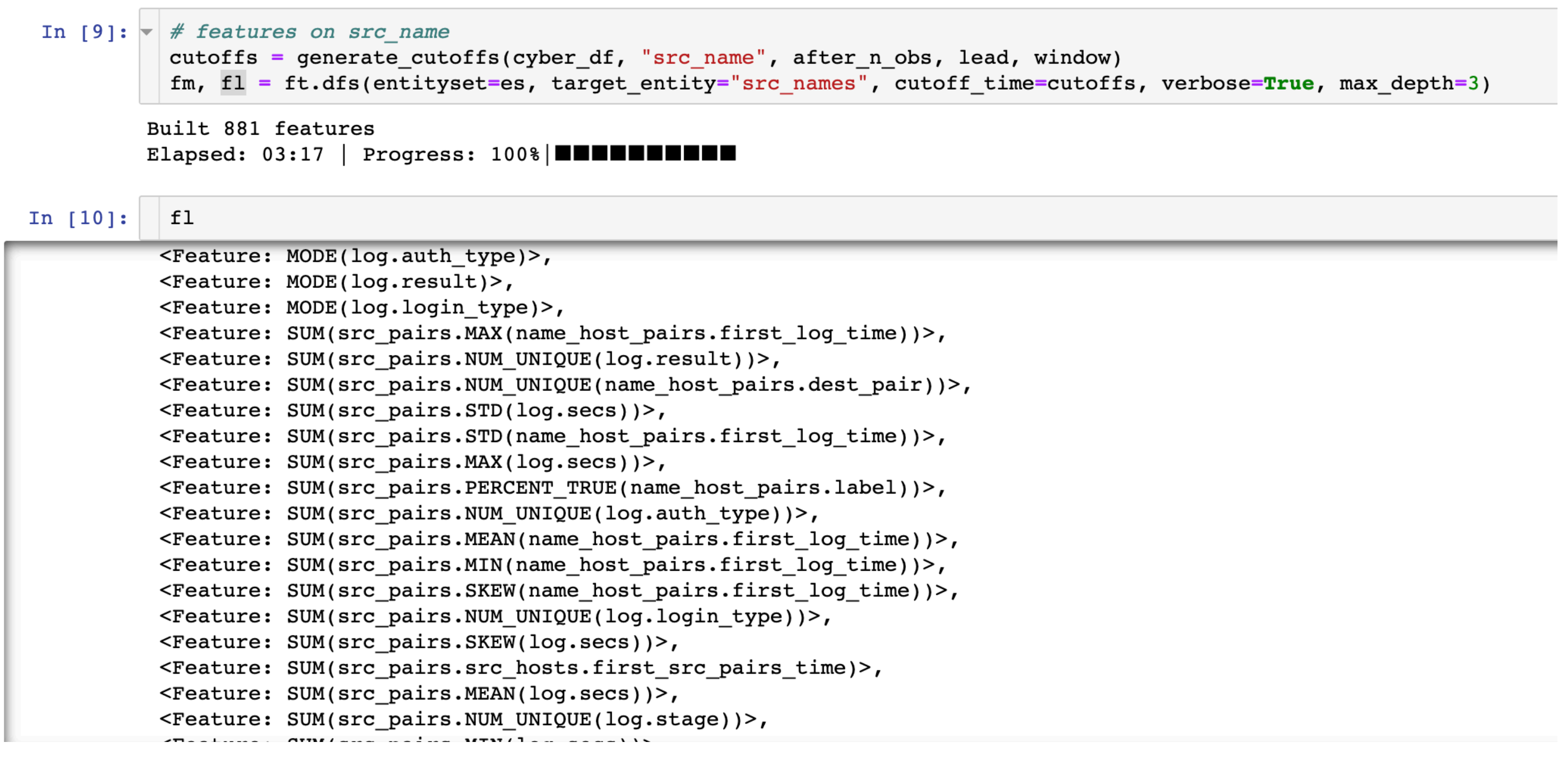




```
In [11]: # Generate features using the constructed entityset
         fm, features = ft.dfs(entityset=es,
                               target_entity='appointments',
                               agg_primitives=['count', 'percent_true'],
                               trans_primitives=['is_weekend', 'weekday', 'day', 'month', 'year'],
                               max_depth=3,
                               approximate='6h',
                               cutoff_time=cutoff_times[20000:],
                               verbose=True)

         Built 46 features
         Elapsed: 01:54 | Progress: 100%|██████████

In [14]: features[22:]

Out[14]: [<Feature: patients.PERCENT_TRUE(appointments.sms_received)>,
          <Feature: patients.PERCENT_TRUE(appointments.no_show)>,
          <Feature: patients.IS_WEEKEND(first_appointments_time)>,
          <Feature: patients.WEEKDAY(first_appointments_time)>,
          <Feature: patients.DAY(first_appointments_time)>,
          <Feature: patients.MONTH(first_appointments_time)>,
          <Feature: patients.YEAR(first_appointments_time)>,
          <Feature: locations.COUNT(appointments)>,
          <Feature: locations.PERCENT_TRUE(appointments.sms_received)>,
          <Feature: locations.PERCENT_TRUE(appointments.no_show)>,
          <Feature: ages.COUNT(appointments)>,
          <Feature: ages.PERCENT_TRUE(appointments.sms_received)>,
          <Feature: ages.PERCENT_TRUE(appointments.no_show)>,
          <Feature: genders.COUNT(appointments)>,
          <Feature: genders.PERCENT_TRUE(appointments.sms_received)>,
          <Feature: genders.PERCENT_TRUE(appointments.no_show)>,
          <Feature: patients.PERCENT_TRUE(appointments.IS_WEEKEND(scheduled_time))>,
          <Feature: patients.PERCENT_TRUE(appointments.IS_WEEKEND(appointment_day))>,
          <Feature: locations.PERCENT_TRUE(appointments.IS_WEEKEND(scheduled_time))>,
          <Feature: locations.PERCENT_TRUE(appointments.IS_WEEKEND(appointment_day))>,
          <Feature: ages.PERCENT_TRUE(appointments.IS_WEEKEND(scheduled_time))>,
          <Feature: ages.PERCENT_TRUE(appointments.IS_WEEKEND(appointment_day))>,
          <Feature: genders.PERCENT_TRUE(appointments.IS_WEEKEND(scheduled_time))>,
          <Feature: genders.PERCENT_TRUE(appointments.IS_WEEKEND(appointment_day))>]
```

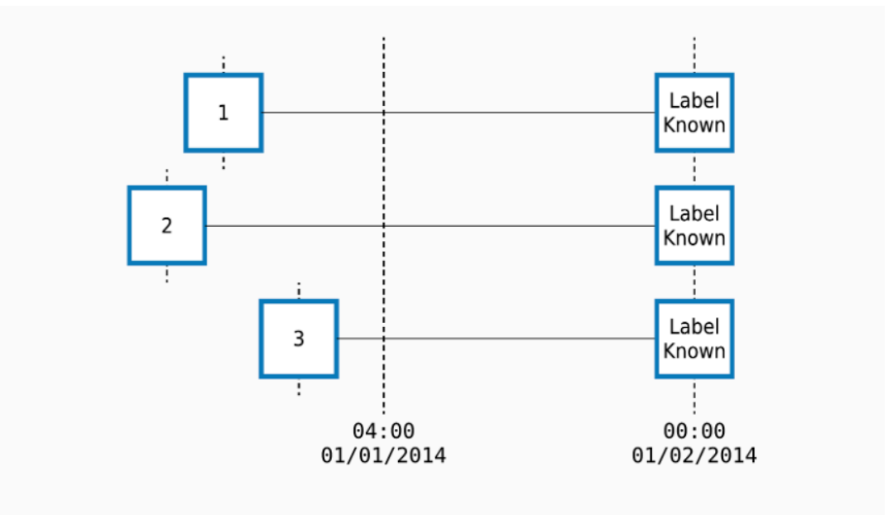
Malicious cyber connection





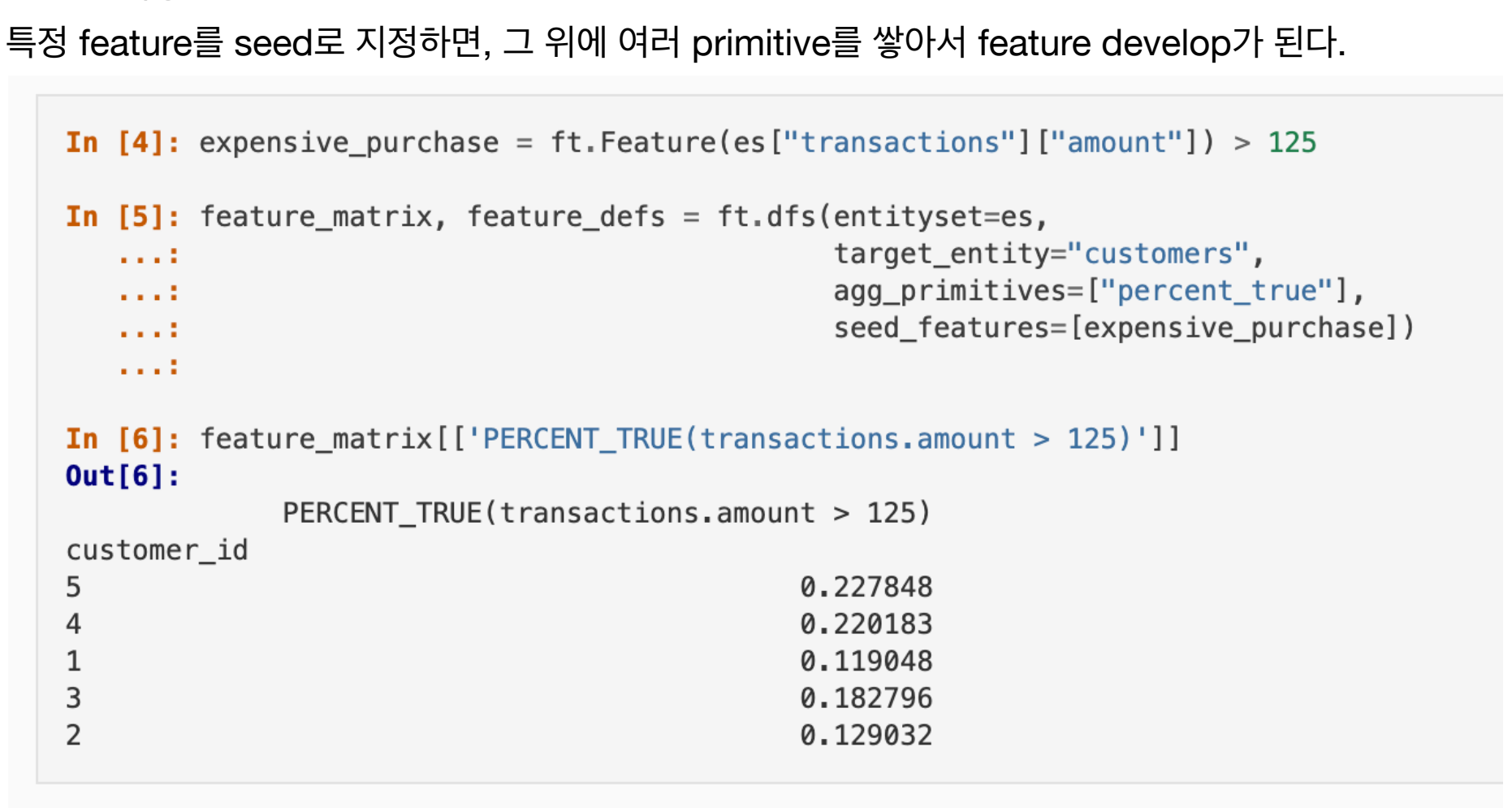
```
In [9]: # features on src_name
        cutoffs = generate_cutoffs(cyber_df, "src_name", after_n_obs, lead, window)
        fm, fl = ft.dfs(entityset=es, target_entity="src_names", cutoff_time=cutoffs, verbose=True, max_depth=3)

        Built 881 features
        Elapsed: 03:17 | Progress: 100%|██████████

In [10]: fl

          <Feature: MODE(log.auth_type)>,
          <Feature: MODE(log.result)>,
          <Feature: MODE(log.login_type)>,
          <Feature: SUM(src_pairs.MAX(name_host_pairs.first_log_time))>,
          <Feature: SUM(src_pairs.NUM_UNIQUE(log.result))>,
          <Feature: SUM(src_pairs.NUM_UNIQUE(name_host_pairs.dest_pair))>,
          <Feature: SUM(src_pairs.STD(log.secs))>,
          <Feature: SUM(src_pairs.STD(name_host_pairs.first_log_time))>,
          <Feature: SUM(src_pairs.MAX(log.secs))>,
          <Feature: SUM(src_pairs.PERCENT_TRUE(name_host_pairs.label))>,
          <Feature: SUM(src_pairs.NUM_UNIQUE(log.auth_type))>,
          <Feature: SUM(src_pairs.MEAN(name_host_pairs.first_log_time))>,
          <Feature: SUM(src_pairs.MIN(name_host_pairs.first_log_time))>,
          <Feature: SUM(src_pairs.SKEW(name_host_pairs.first_log_time))>,
          <Feature: SUM(src_pairs.NUM_UNIQUE(log.login_type))>,
          <Feature: SUM(src_pairs.SKEW(log.secs))>,
          <Feature: SUM(src_pairs.src_hosts.first_src_pairs_time)>,
          <Feature: SUM(src_pairs.MEAN(log.secs))>,
          <Feature: SUM(src_pairs.NUM_UNIQUE(log.stage))>,
```

Handling time

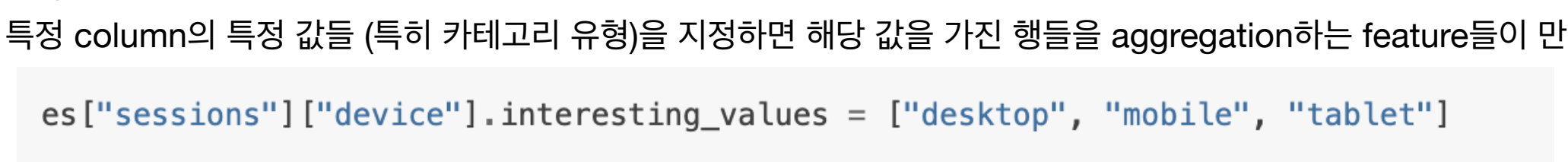
- `Temporal data`를 다루는 경우에 한함
- `time index column`이 지정되고 `cutoff time`이 지정된 경우 그 시간 이후의 정보들은 `feature` 형성에서 제외된다.
    - 해당 정보들은 일종의 `label leakage`로 간주되는 경우
- 즉 `cutoff time`이란 해당 정보가 `feature`로 사용될 수 있는 마지막 시간이다.
- 

- 각 행별로(`ex. Customer id`) 서로 다른 `cutoff-time`을 지정해 줄수도 있다.
- `Training window`
    - 기본적으로는 `cut-off time` 이전 인스턴스들은 `feature` 계산에 사용된다.
    - 이 기간을 더 줄이고 싶으면 `ex) within 2 hour, window`를 지정하면 `aggregation feature` 계산시 반영된다.
- Setting a Last Time Index
- Approximating Features by Rounding Cutoff Times
- Secondary Time Index
- Creating and Flattening a Feature Tensor

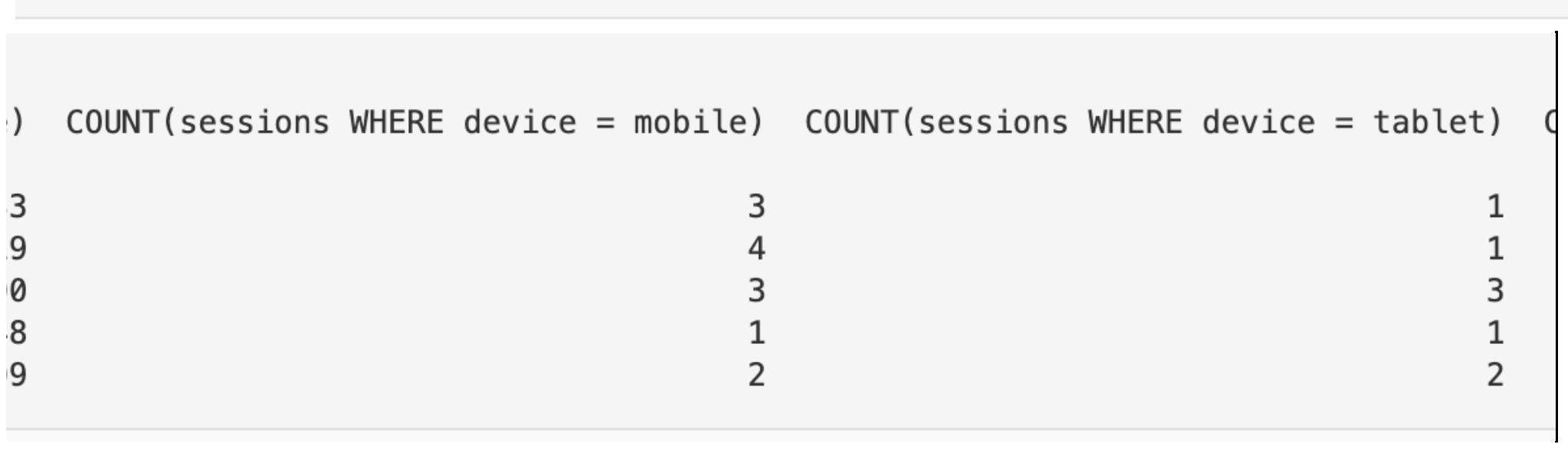
Tuning

- Seed feature 지정
    - 특정 feature를 seed로 지정하면, 그 위에 여러 primitive를 쌓아서 feature develop가 된다.
    - 

```
In [4]: expensive_purchase = ft.Feature(es["transactions"]["amount"]) > 125

In [5]: feature_matrix, feature_defs = ft.dfs(entityset=es,
   ...:                                       target_entity="customers",
   ...:                                       agg_primitives=["percent_true"],
   ...:                                       seed_features=[expensive_purchase])
   ...:

In [6]: feature_matrix[['PERCENT_TRUE(transactions.amount > 125)']]
Out[6]: 
             PERCENT_TRUE(transactions.amount > 125)
customer_id                                         
5                                           0.227848
4                                           0.220183
1                                           0.119048
3                                           0.182796
2                                           0.129032
```

- Interesting feature value 지정
    - 특정 column의 특정 값들 (특히 카테고리 유형)을 지정하면 해당 값을 가진 행들을 aggregation하는 feature들이 만들어진다.
    - 

```
es["sessions"]["device"].interesting_values = ["desktop", "mobile", "tablet"]
```

    - 

```
)  COUNT(sessions WHERE device = mobile)  COUNT(sessions WHERE device = tablet)  C

3                                       3                                      1
9                                       4                                      1
0                                       3                                      3
8                                       1                                      1
9                                       2                                      2
```

- 카테고리 유형의 경우 one-hot encoding 제공

Specify primitive options

- 별 지정 없으면 모든 entity, column에 대해서 모든 primitive의 조합이 적용된다.
- 이는 조합의 폭발을 일으킬수 있으므로 적절히 적용 대상을 선정할 수 있다. (Include or ignore)
- 방법 1)
    - primitive의 종류와 상관없이 global하게 ignore할 대상 선정 : ignore_entities, ignore_variables
- 방법 2)
    - 개별 primitive 별로 ignore/include 대상 선정
        - Ex) mode primitive에 대해서 특정 entity나 column들을 ignore/include

병렬 처리

- 원래 design은 single 머신에서 메모리 상에서 처리하는 것
- 단일 머신의 병렬 cpu 활용은 지원
    - calculate_feature_matrix의 n_jobs
- dask의 병렬 스케줄러 지원
    - calculate_feature_matrix의 dask_kwargs={'cluster': cluster}
- dask의 dashboard와 연동되서 수행 상태 모니터링
    - dask_kwargs={'diagnostics_port': 8787}
- Data를 특정 조건을 partitioning해서 dask, pyspark을 이용해서 병렬 처리
    - Predict Next Purchase demo notebook.
    - Featuretools on Dask notebook.
    - Parallelizing Feature Engineering with Dask article
    - Feature Engineering on Spark notebook
    - Featuretools on Spark article
- Featuretools enterprise 버젼은 모두 지원

Deployment

- Feature 생성 방식을 export해서 저장하고
- load해서 new data에 대해 적용하면 동일한 engineered feature가 만들어진다.
- Feature definition은 json 양식으로 저장

Advanced custom primitive guide

- 별도의 argument를 받아들이는 transformation primitive 작성 방법
- Multiple output을 산출하는 transformation primitive 작성 방법

Feature tools demo

- https://www.featuretools.com/demos/