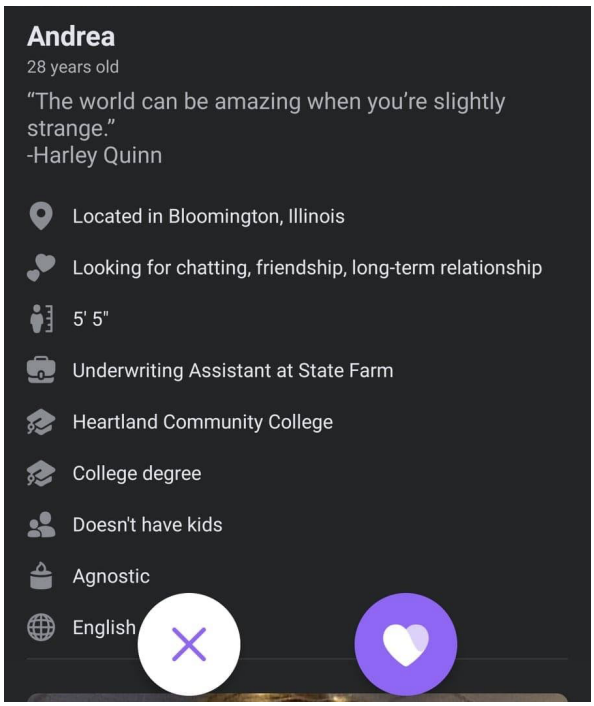
User Attribute Engagement: a glance

Summary

The users engage with candidates daily and have an inherent set of desired attributes that they are looking for. When a user looks at a profile and determines from the shown attributes, other than the profile pictures, whether to like or pass a certain candidate there is a lot of information to be gained from these attributes which can be used to improve a users' candidate quality and overall dating experience.



An example of a user's attributes:
- Age
- Bio
- Dating Intent
- Height
- Hometown
- Job and Employer
- Education Level
- Children Info
- Religious Info
- Languages
- Hobby
- IG connected
- Fun-fact answer
- Distance

Hypothesis

Profile attributes are important factors to help user make decision of like or pass. This assumption leads us to the next that the attributes that are present/not present on a profile that a user likes/passes may present a pattern that can be picked up on and used to better rank candidates.

A more detailed hypothesis

- Users have preference on profile attributes. the existence and the value of the profile attributes plays an important role when viewer make the decision of like or pass.
- For some of these profile attributes, users can explicitly set preferences to filter out un-wanted candidates. The filtering applies at the candidate retrieval stage.
- However,
    - filters are usually used to filter out un-wanted candidates. users could still have preference among the candidates passed the filters.
    - user does not always use all filters, and users set some filters as soft.
    - not all attributes have filters, e.g. number of photos, occupation filled, has hobby, etc

    - some user does not know or specify their attribute preference but the preference may shown in engagement data
- So there is opportunities for ranking to personalize the ranking through learning users preference on profile attributes.
- When users interact with dating profiles they are creating a set of implicit ratings for every profile attribute they see and we can capitalize on this by being able to record these interactions and transform them into statistics to have our models learn from them.

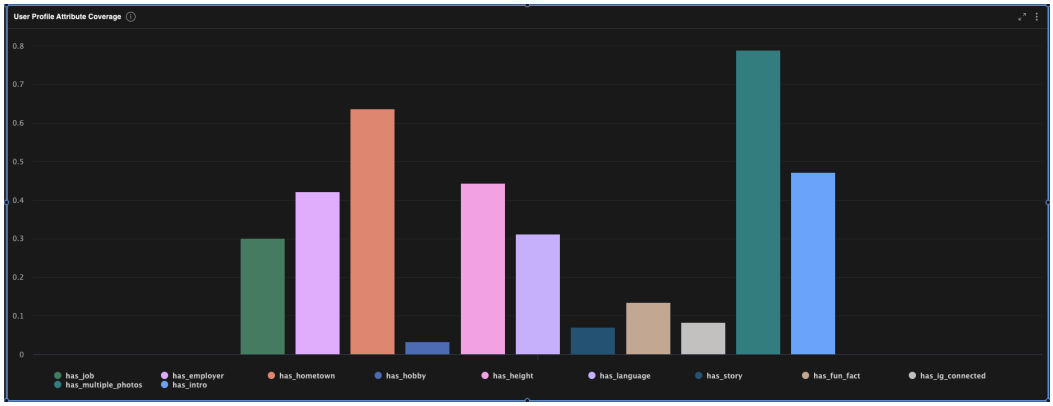
For exemplar purposes

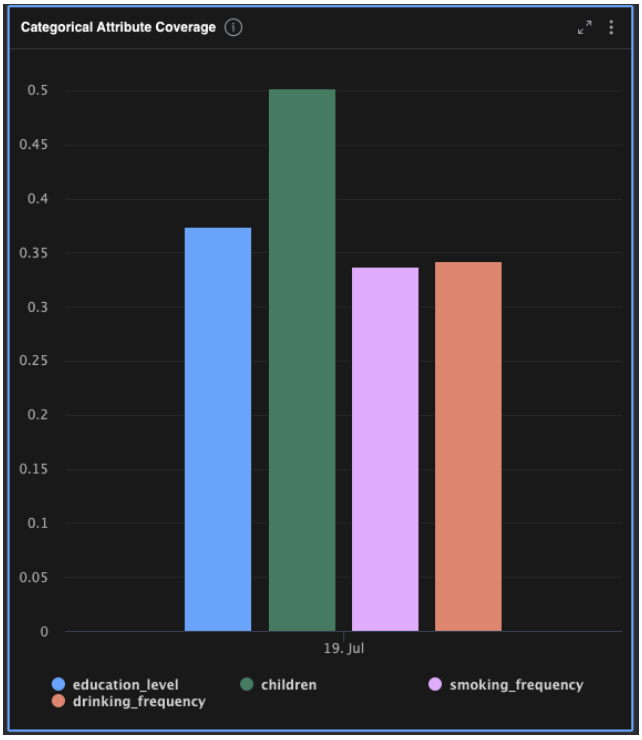
- Let us consider person A:
    - They are a picky dater and they want their partner to be as invested into the dating scene as them and they are more likely to pick candidates that have a bio, list their smoking and drinking preferences and have at least one hobby on their profile.
    - Now this person can't set these filters explicitly so they have to go through a list of profiles that weren't ranked with these attributes in mind before they can choose candidates that fit closer to their actual preference. If we were to pick up on these implicit ratings for attributes present/absent on candidate profiles that are liked we would be able to better present candidates to our users.

Attribute Preference Dense Feature Set

1. Identify and create (if needed) features and meta attributes:

- We take a look at the profile attributes and then we try to come up with features that would sufficiently be able to encapsulate their properties and be helpful to the model. We came up with 3 categories for this classification of attributes: **Existence, Categorical and Continuous**.
- **Existence Attributes** (can also be seen as categorical since yes and no can be categories of response) that denote whether a certain attribute has been filled or not. In this case we added new attributes that could capture attribute existence preferences as well since some people might not engage with profiles which don't have attributes filled out; we extended the dataswarm pipeline to add these in `dating_engagement_user_attribute` table
  For example:
    - has_intro
    - has_job
    - has_employer
    - has_hometown
    - has_hobby
    - has_height
    - has_language
    - has_story
    - has_fun_fact
    - has_ig_connected
    - has_multiple_photos

- 

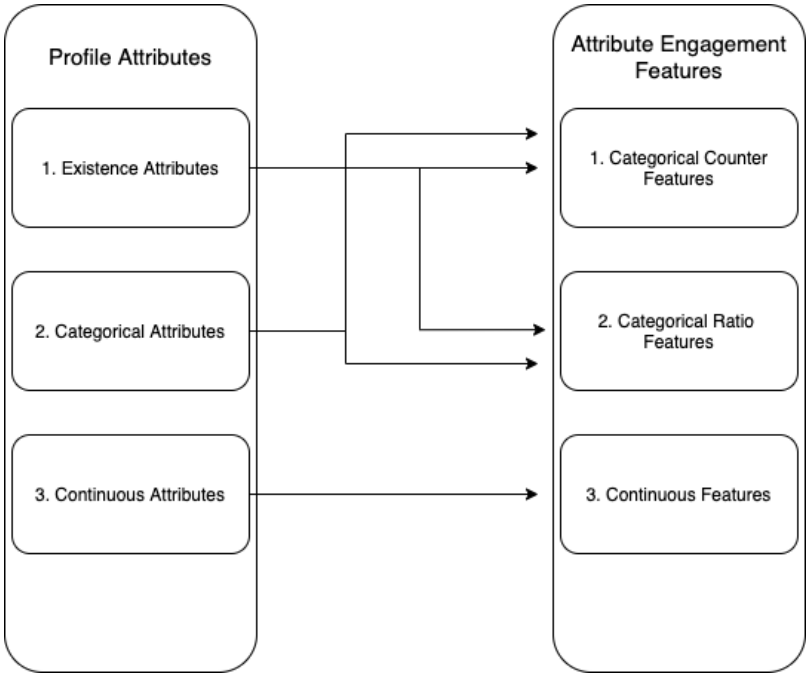
- **Categorical Attributes** from the low cardinality features (limited attribute option). For example:
  - Education Level: [high school, college, grad school, unknown] → [1,2,3,0]
  - Children Info: [has, no, unknown] → [1,2,0]
  - Smoking Preference: [never, often, always, occasionally] → [1,2,3,4,0]
  - Drinking Preference: [never, often, always, occasionally] → [1,2,3,4,0]
  - 

- **Continuous Attribute**. these attributes are real numbers. For example:
  - Age
  - Height
  - Intro_text_length
  - Num_photos
  - Num_dating_questions_answered
  - Num_hobbies
  - Distance

- Code Pointer for extended dataswarm table and the diff:
  - https://fburl.com/diffusion/9o2svy76
  - D21863638, D21854736

2. Capturing users attribute preferences

After we have established these attributes, we would like to gain engagement statistics in order to get user preferences regarding these said attributes.



- We decided on metrics that made the most sense to us to capture a user's attribute engagement preferences in a meaningful manner.
  - For the **Existence and Categorical attribute types** we used:
    - **Counter features** to keep a count of the number of likes/passes sent for a certain attribute's presence/absence and the separate categories it can take
      - These features are a count of the number of outbound likes/passes a user sends to a profile with/without the attribute in the past 14 days.
      - They are categorized based on the Dating Subsurface, Category of the attribute and Engagement Type.
      - eg: `VIEWER_ATTRIBUTE_ENGAGEMENT_DRINKING_FREQUENCY_DH_NEVER_LIKE_14D` is a categorical user attribute engagement counter feature which counts the number of likes a viewer sent in the past 14 days from the dating home subsurface on profiles that had drinking frequency as "NEVER".
    - **Ratio features** to keep the like rates for these attributes.
      - These features capture the like rate of the user's engagement for the specific subsurface and feature category in the past 14 days.
      - eg: `VIEWER_ATTRIBUTE_ENGAGEMENT_SMOKING_FREQUENCY_DH_NEVER_LIKE_RATE_14D` is a categorical user attribute engagement ratio feature which has the ratio of likes sent in the past 14 days form the dating home subsurface to candidates that have the smoking frequency as never. (feature::subsurface::attribute_val::like / feature::subsurface::attribute_val::(like+pass))
  - For the Continuous attribute types we used the following statistics:

        - avg
        - std
        - p50
        - count
        - min
        - max
        - e.g: `VIEWER_ATTRIBUTE_ENGAGEMENT_INTRO_TEXT_LENGTH_ALL_AVG_LIKE_14D` is a continuous user attribute engagement feature which measures the average length of intro text bios of all the candidates that the user liked in the past 14 days from all subsurfaces.

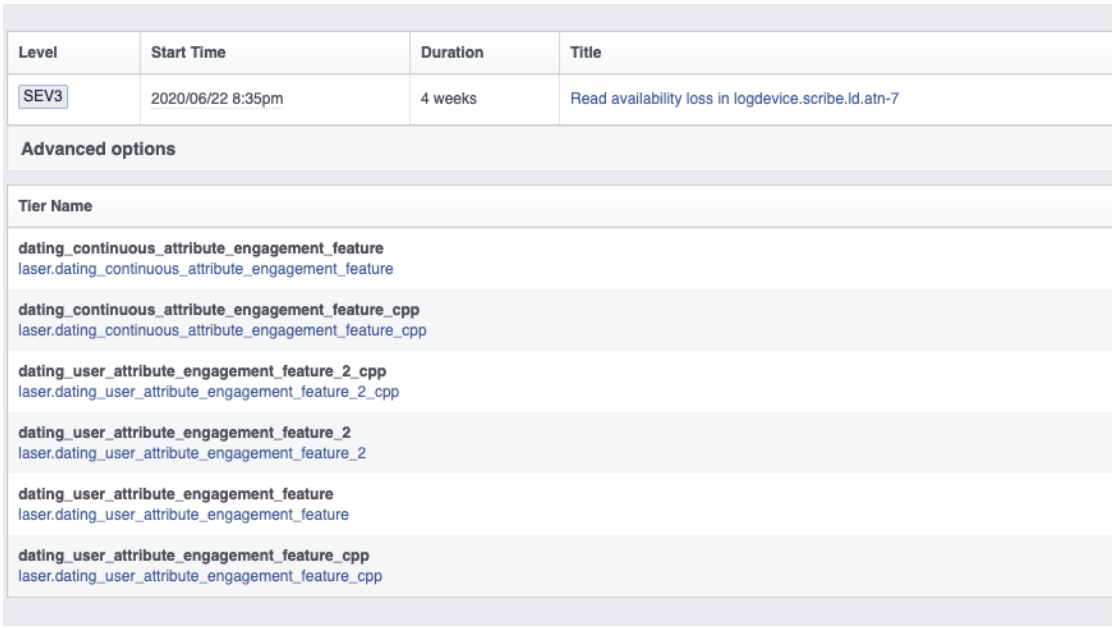
Dense Feature Set Implementation

1. Converting the counters/ratio of existence/categorial types and stats of continuous types into hive table schema of <user id, map<feature id, feature value>> for building laser tier

- We start converting these features that we just created in dataswarm into laser retrievable format or turning them into maps with the format `map<feature_ids, feature_vals>`
    a. We deterministically generate our feature ids in the pipeline itself by using a base_id
    b. To this we add specific offsets that are fixed for a certain feature and subsurface and then increment it for engagement type and attribute values, thus guaranteeing unique feature ids as long as the offsets are set with enough width.
- The idea was to make it really easy to scale this algorithm for other features sets by reusing our feature id generation logic and make it even easier to add new attributes by simply adding a feature block like this to the attribute set:

```
    "education_level": {
        "col_name": "education_level",
        "type": "VARCHAR",
        "candidate_transform": """
            CASE
                WHEN education_level IS NOT NULL AND education_level != 'prefer_not_to
                    THEN education_level
                ELSE 'unknown'
            END
        """,
        "is_category": True,
        "categories": ["unknown", "grad_school", "college", "high_school"],
        "is_used_in_engagement_counter": True,
        "feature_id_offset": 0,
    },
```

    - We have accounted for enough gap between the ids that in the future if we wish to add more features, subsurfaces or continuous statistics we would just need to add those to our original keysets and the algorithm would generate ids and insert them into the tables on its own. Hence it is super easy to scale and can be reused for other feature sets as well since we define a general function that takes these keysets and base ids as arguments.
    - We also guarantee deterministic dictionary access hence the ids won't spuriously change.
- After creating our laser source table in hive we can create a laser tier for our features.



Level	Start Time	Duration	Title
SEV3	2020/06/22 8:35pm	4 weeks	Read availability loss in logdevice.scribe.ld.atn-7

Advanced options

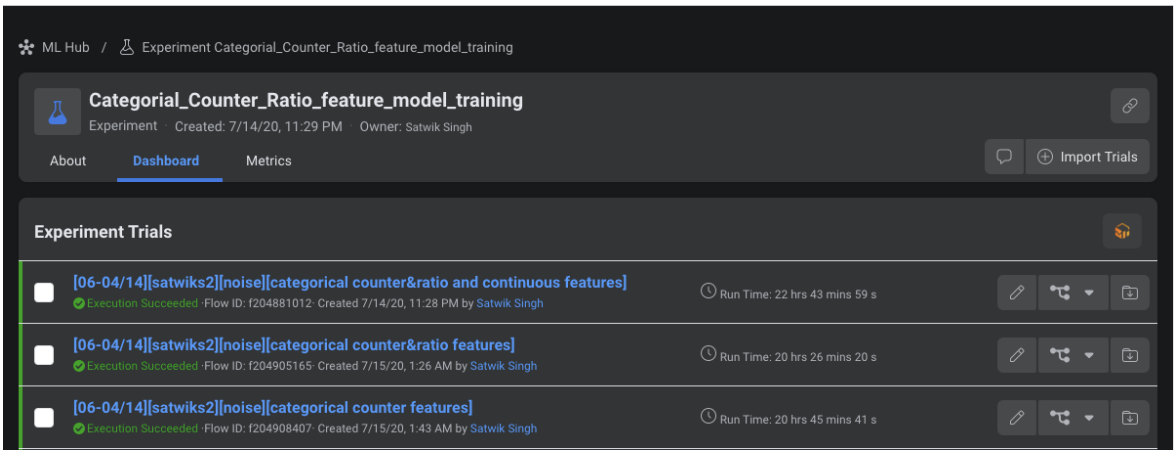
Tier Name
dating_continuous_attribute_engagement_feature laser.dating_continuous_attribute_engagement_feature
dating_continuous_attribute_engagement_feature_cpp laser.dating_continuous_attribute_engagement_feature_cpp
dating_user_attribute_engagement_feature_2_cpp laser.dating_user_attribute_engagement_feature_2_cpp
dating_user_attribute_engagement_feature_2 laser.dating_user_attribute_engagement_feature_2
dating_user_attribute_engagement_feature laser.dating_user_attribute_engagement_feature
dating_user_attribute_engagement_feature_cpp laser.dating_user_attribute_engagement_feature_cpp

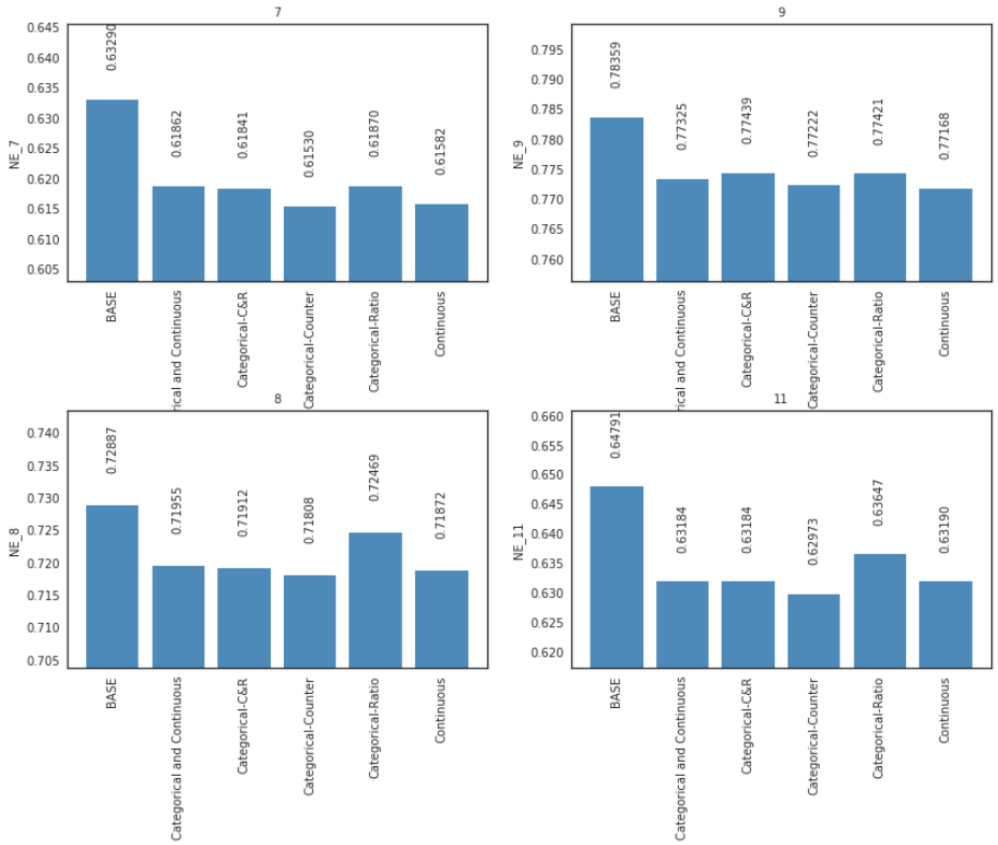
2. Create feature fetchers

- Create Feature Fetchers in cpp, php and mocha
- Add the required unit tests and local testing using these guides Dating Live Feature Fetcher Design Feature Fetcher Comparison
- Side task worked on: Adding a new feature id retrieval logic T69281096

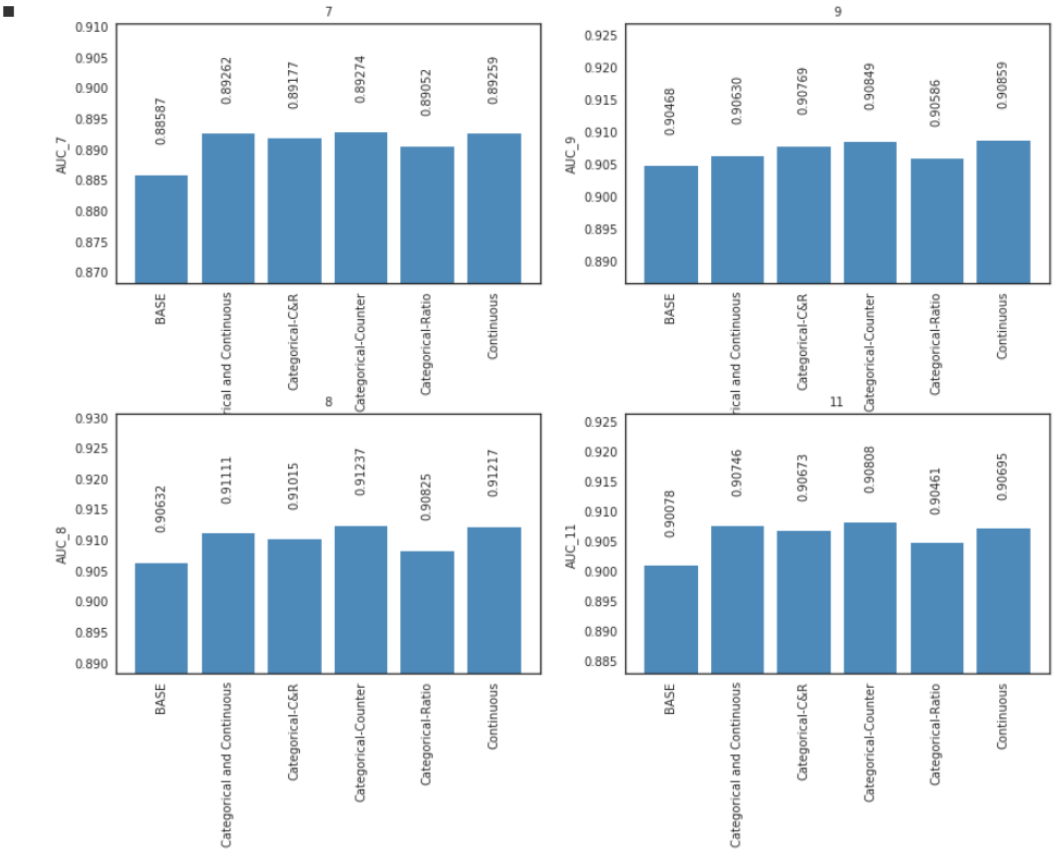
Training models and conducting offline evaluations of these new dense features:

- Now that we have our features ready we want to backfill our laser tables with enough data for our model training.
- After backfilling the laser tables we inject those features into the `mtml_model_training_random_noise` and table `dating_home_mtml_training`, using the `offline_feature_injection_with_feature_id.py` script Xiaoye wrote.
- After we've injected the features into out training and eval datasets we can launch model training from the fblearner UI https://fburl.com/mlhub/0jfwsvio

- 

- Offline Results :
  - NE
    - 

  - AUC



- Online Results till now:

Profile Attribute Sparse Feature Set

To capture the attribute preference in another way, we also create attribute id as sparse features.

1. Creation of the features:

- We decided to create sparse features conforming to the 3 categories of user profile attributes that we had identified previously (Existence, Categorical and Continuous).
- We want to map the combination of these attributes's values to integers and to do that we have come up with 3 separate methods:
    - Existence attributes:
        - We can denote the existence of an attribute with a single bit, 0 for absence and 1 for presence
        - We can also assign each attribute's bit a certain index in the overall sparse id and then simply concat them to get the final attribute id
    - Categorical attributes:
        - For the low cardinality attributes we can use n bits to represent the 2^n categories that the attribute may have'
        - We can concat the individual sets of bits with each other to get our final attribute id
    - Continuous attributes:
        - We can distinguish the values by placing them in buckets such that each bucket has roughly the same number of users.
        - We index each bucket and denote the index of bucket that the attribute falls into using n bits
        - Finally we concat these sets of n bits with each other to get our final attribute id
    - Note: Concat takes place in a set manner and we have fixed shift values for the attributes to guarantee that the meanings of the bits in the final id don't change even if one of the features is removed or a new one is added.

2. Profile attribute engagement collection

- We collect the attribute ids of the users' engagement.
- Have the attribute id and liked/passed attribute ids for both viewer and candidate

Further steps include feature fetching, training and finally offline and online tests