Finding New Dataset

Criteria for the dataset that applicable to our work motivation are as follows:

1. The dataset should has interesting views (the importance score above 0.5)
2. The top-k interesting views must have the same attributes and measures but different aggregate function

After looking all datasets on DeepEyes's paper such as Happy Countries, US baby names, Flights, Service statistics, Average Food Price and etc. There are no dataset that can be used for our motivation include two new datasets: Airbnb and Zomato. Especially the second criterion is hard to find.

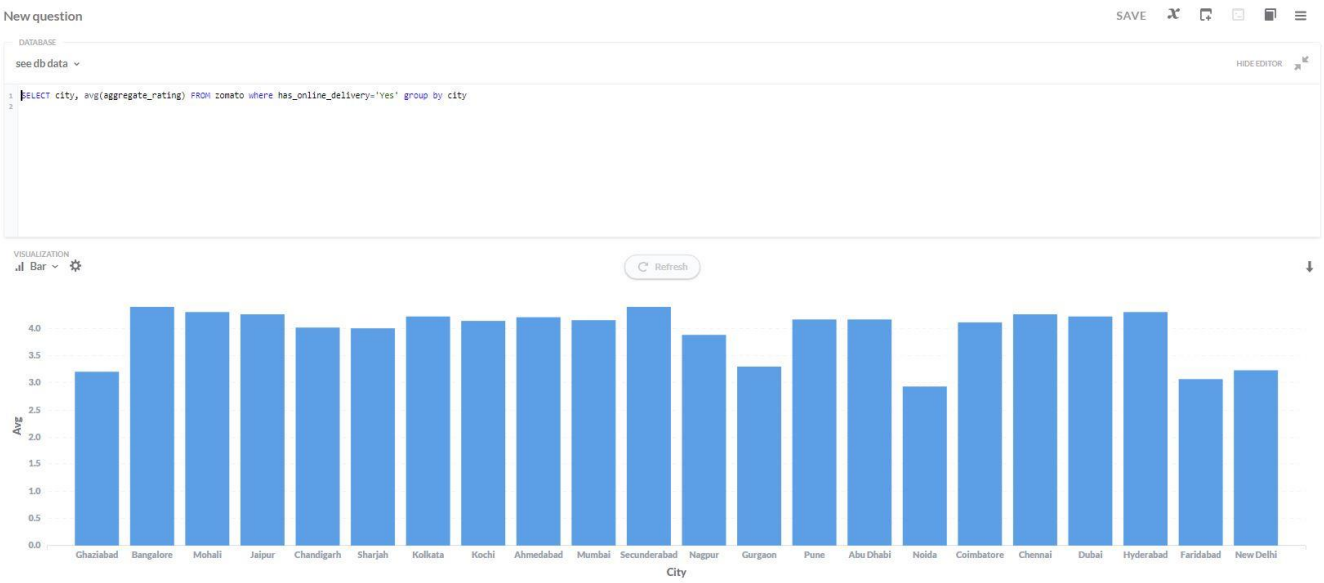
Moreover, the main issue in some datasets is while all attributes are used, mostly the top-k views that have high deviation are not good looking views. For instance, Zomato dataset has attribute 'country' and this attributes always in top-k set because it generates so many bars and some of bars are missing on the reference subset. That's why it has high deviation between target and reference subset.

For instance, the analyst wants to compare between restaurants which provide online delivery service and no delivery service in zomato dataset as follows:

Target Query = select A, F(M) from zomato where has_online_delivery = 'Yes' group by A
Ref Query = select A, F(M) from zomato where has_online_delivery = 'No' group by A

Here the results:

Attributes	Meassure	Function	Utility
city	aggregate_rating	avg	0.294822
city	aggregate_rating	max	0.28153
city	price_range	avg	0.27806
has_table_booking	votes	sum	0.270455
city	votes	avg	0.264461
city	price_range	max	0.256341
rating_text	price_range	sum	0.236017
city	votes	max	0.218148
city	votes	sum	0.209013
rating_text	votes	sum	0.208342
has_table_booking	votes	max	0.170685
city	aggregate_rating	sum	0.167206
rating_text	votes	max	0.152014
city	price_range	sum	0.15034
rating_text	aggregate_rating	sum	0.134693
has_table_booking	price_range	sum	0.107052
rating_text	votes	avg	0.0884
has_table_booking	aggregate_rating	sum	0.081758
has_table_booking	votes	avg	0.080846
has_table_booking	aggregate_rating	avg	0.07804
has_table_booking	price_range	avg	0.064576
rating_text	price_range	max	0.052486
rating_text	price_range	avg	0.045048
has_table_booking	aggregate_rating	max	0.00729
rating_text	aggregate_rating	avg	0.002286
rating_text	aggregate_rating	max	0
has_table_booking	price_range	max	0



Target query: A = city, M = aggregate rating, F = AVG, subset: has_online_delivery = 'Yes'

Reference query: A = city, M = aggregate rating, F = AVG, subset: has_online_delivery = 'No'

This issue happens to another dataset as well. I tried to not include attribute such as 'city', however, again the second condition (i.e., top-k views should have same attribute and measure but different aggregate function) is hard to be satisfied. This second condition should be satisfied due to our motivation for diversity. This task is seized a lot of my time and I will not give up to find the new dataset.

Rectifying Maximum Bound

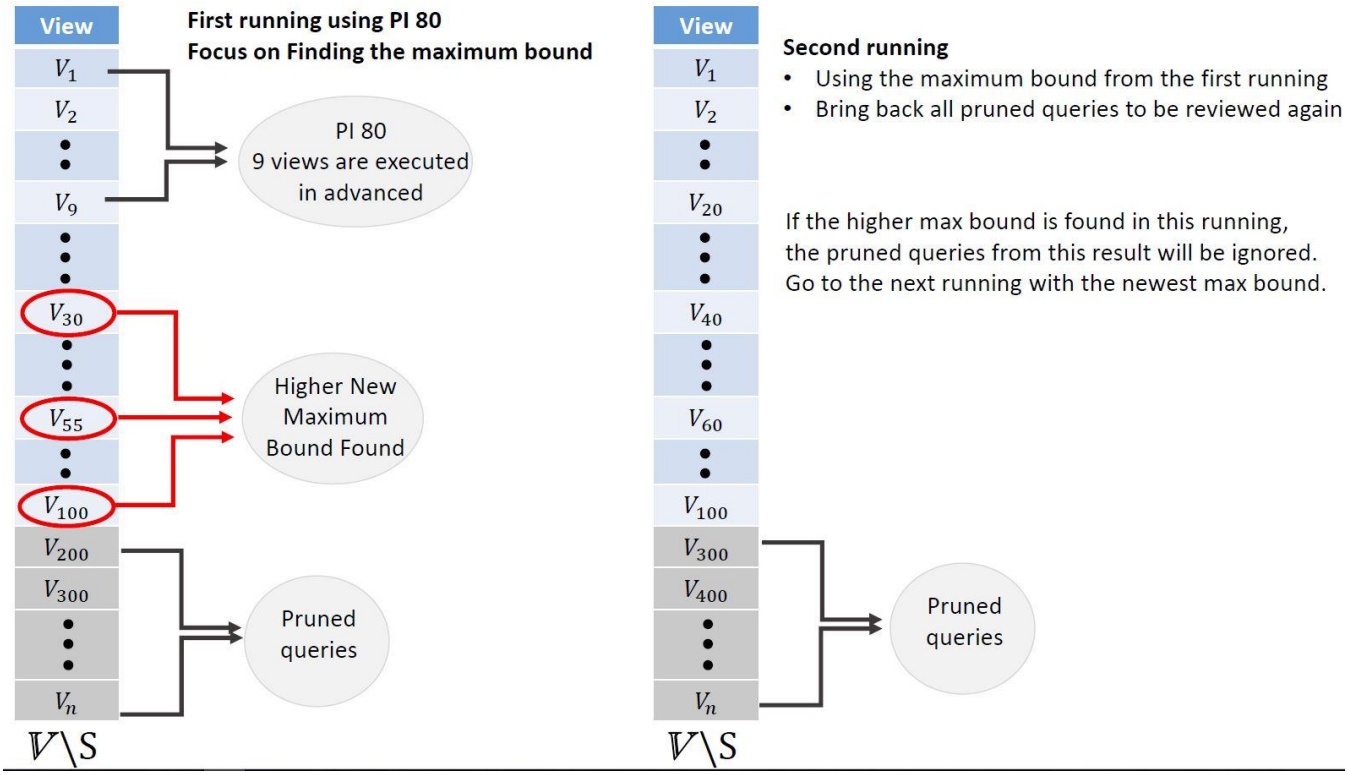
Below is the algorithm that currently I have. In the previous report, I wrote that by using this algorithm, it seems the program runs twice. That is because based on the experiments, the actual max bound is found in the first running. In the second running, the program only focus to review the pruned queries from the first running.

There is an issue that need to be consider to rectifying maximum bound on adaptive scheme as follows:

For instance, case of adaptive pruning scheme on DiVE-Greedy, two most distant views are used as the initialization set *S.* If *k* = 5, to generate the result three greedy iterations are needed. In each Greedy iteration, one view will be added to set *S*. Moreover, in each iteration, all pruned queries will be returned and diversity score will be re-calculated. The problem is if the actual maximum bound is found in the last Greedy iteration. If this happens, then the view which has been added in the first iteration may wrong and it needs to be recalculate from the first which means re-run the Greedy from the first iteration again.

Because of this reason, I proposed this algorithm as follows:

1. First running, the algorithm focus to find the maximum bound. (the program does not care about the *F(S)* and the number of pruned queries), the program runs until the end of iterations.
2. Second running, use the maximum bound from the first running to execute the algorithm. In this second running, all pruned views in first running are returned. If there is no view that has importance score more than the current maximum bound, the *F(S)* and pruned queries will be presented as the final result. However, if there is a higher importance score than the current maximum bound, then the result will be ignored and go to the next running using the newest maximum bound.



For instance, there are 500 views in *X* that sorted by diversity score to *S.* Several views are executed in advanced (e.g., 9 views are executed) and find the maximum importance score from those executed views which this importance score will be used as the maximum bound (e.g., 0.3). Then, the current maximum bound is 0.3 and there are 400 views are pruned by using this maximum bound 0.3 and the remaining (i.e., unpruned) views will be executed. While executing the remaining views there are three views that have higher maximum bound from the current maximum bound ($V_{30} = 0.45$, $V_{55} = 0.5$, and $V_{100} = 0.6$) and now there is no remaining views anymore.

In the second running, all pruned queries from the first running will be returned and the current maximum bound 0.6 is used. There may some returned views will not be considered as pruned queries while the bound is 0.6. That's why in this running, more view queries may be executed and the number of pruned queries is lower than in the first running. If after executing the unpruned views using maximum bound 0.6, there is no importance score that more than this value, then the result (i.e., the pruned queries) from the second running is correct. However, if there is a higher importance score than the current one, then the result will be ignored and go to the next running and run the program using the newest maximum bound.

This algorithm seems re-run the scheme to find the actual maximum bound and ignore the result since the higher maximum bound is found. This algorithm will stop running while there is no higher maximum bound, then the *F(S)* and the number of pruned queries will be presented to the user.

This algorithm is quite different with the logic that we discussed in the last meeting which need parameters to decide how many steps that we need to go backward to fix the wrong bound and else. However, this logic is quite simple and we don't need to use any parameters. How do you think?