Data Challenge Approaches and Solutions:

I have created 3 features matrices with 3 different group of features as explained below:

1. Feature Engineering Approach 1:

- Combined the user information of user_assesment_scores and user_interests for the courses the user has expressed interest and taken an assesment.
- On analysing the data, although all the given 10000 users have expressed atleast one interest in a course, not all have taken the assesment. Hence there was loss of information in combining the information.
- Applied log transformation to the user_assesment_scores to achieve normal distribution of scores and discretized the scores as 'ratings' using unsupervised discretization with k-means algorithm. Used elbow's method to find the optimal value of K
- Though the information was not present for all the users, the similarity of users picked provided meaningful observations.
- For example, consider the users (user_handle: 12 and 4343), both the users have expressed the intrest in Java, taken assesments and scored close scores of 170 and 164 respectively.

2. Feature Engineering Approach 2:

- Combined the user information of user_course_views and course_tags for the courses the user has viewed
- Joining user_course_views with course_tags on course_id to get the course_tags the user has viewed
- Creating a new Feature combining Course_tag and Level of the course. Deleting the unwanted features after creating the new feature
- Aggreating(sum) the number of seconds an user has viewed a course described using the newly created feature of updated_tag which is a combination of course_tag and level.
- Removing the entries where a user a viewed a course for '0' seconds¶
    - f.Applying log transformation to the aggregated number of seconds to convert the distribution of the feature close to normality
- Creating a new feature 'Rating':
    - 0 - if the user has not viewed a course
    - 1 - if the log_transformed_seconds in the range (0 - 1st quantile),
    - 2 - if the log_transformed_seconds in the range (1st quantile - 2nd quantile)
    - 3 - if the log_transformed_seconds in the range (2nd quantile - 3rd quantile)
    - 4 - if the log_transformed_seconds in the range (3rd quantile - max)

- Although there was little loss in information the similarity was consistent with the findings for example, consider the top 3 similar users of user_handle : 10000 - user_handles [4197,9349,464] they have all commonly viewed the course 'apex-absolute-beginner-guide-coding-salesforce' with 'beginner level' for almost equal amount of time and the combination of users (10000, 4197) has literally the viewed same courses.

3. Feature Engineering Approach 3:

- The user_interest was the only with the information of all the given 10000 users, where every user has expressed interest in a course at least once.
- Used the interest_tag as the features for this model.

Solution for the questions given:

1. The metric chosen for similarity calculation is Cosine Similarity with Mean Normalization (Adjusted Cosine). The reasons are:

- Adjusted Cosine is invariant to scaling of the vector. Hence it is robust to while increasing the number of users or for more expressive features to capture better association.
- On calculating the coefficient based on all the elements in the vector produces a self-damping effect too attenuate the similarity between users that may not have many features in common. Hence it takes care of 2 main issues, dealing with uncommon features between 2 users, and having very few features in common between 2 users.
- Adjusted Cosine is statistically similar to Pearson Correlation robust way of describing how one vector is similar to another.
- Projection wise cosine similarity is dependent on only the direction of the vector.
- The cosine similarities of a subset of the original data are the same as that of the original data, which is not true for the Pearson correlation.

2. I would consider the following steps to scale the product.

- Starting with storing the data, I have used SQLite for solving the problem, as the scale increases shifting to distributed database or file storage like HDFS, Azure Data lake could be efficient.
- Considering the data is stored on a distributed environment (Databricks), implementing SPARK jobs to retrieve data, to implement parallel processing in the pipelines. For example consider discretization of continuous features, the task could be accomplished in parallel which is highly efficient.
- Rendering the product to end-user should be a more containerized version ex: using Kubernetes to deploy Tensor Flow models or deploying the model on the cloud will provide substantial increase in efficiency.

- Using DAG engines like Apache Spark or Tensor Flow for numerical computations will increase the efficiency of the process and computations internally through lazy evaluation. DAG engine will have access to the entire computation graph assisting performance improvement. This could prove incredibly efficient when the associations and the users increase which increases the computation at scale.
- Perform computation on GPU
- Perform random or stratified sampling (to ensure users from all groups) to perform initial EDA analysis rather than using the whole data.

3. Considering the scale and diversity of the users the company serves, for any given context similar to this:

- Build a more robust API by changing the DB storage to distributed storage or Data Lake
- As mentioned, definitely gather more data on the users (Job/student, proficiency level, Completed the Course/Not Completed/ Assessment Scores/ User Reviews/ etc) to improve the performance by engineering more expansive and expressive features. Also in the case of a Supervised Learning Problem it reduces variance and hence overfitting
- Track user journey through clickstream data to identify friction points
- Develop end-to-end pipelines by covering more features like mentioned above.
- Careful consideration of the points mentioned above for scaling the solutions created.