# *Recommendation system plan:*

- **Data collection and visualisation:**

- Firstly, we have to analyse the frequency of visits for each user (how many times they visit the website each day or week, what artist they usually visit). These data are already available in the user_events file, so we could just create data visualisations to analyse this data, and the data collection is relatively simple (just recording the interaction of users each time they click play/plause/share a track or sharing artists, for example).
- Then, we will need to look into what artist does a user have positive interaction with (follow/share artists). In the current iteration of the dataset, we already have this data in user_events, so we have to make sure that the website can record interaction of users with artists in the same way.
- We could also look at other positive interactions from users with artists' tracks (play, like or share tracks). With this case, the duration of interaction is worthwhile to collect, but since they are only indirectly proved that users like an artist, their importance would be lower in our analysis (lower weights).

- **Data preprocessing:**

- The given dataset has already been cleared, so we just need to make sure that in the program we have, the visit details for each user (when, what interaction they have with artists/tracks, and what tracks they do with) could be collected in the same way. If the system somehow could not collect the data, we might need to consult with experienced people before deciding to correct the errors or remove the data out of our analysis.
- Then we have to create interaction matrices (e.g., user-artist engagement matrix). They should have two axes (user and artist), as well as the weight for each positive interaction (share/follow artists, or play/like/share tracks).
  (Note: we might need to add negative interaction (in the dataset, we only have unfollow artists as the negative one). This might be added to let us know about the actual sentiment that users have for each artist, but it could complicate our analysis.)

- **Choosing model:**

- We will need to know what artist an user would like to choose, given another artist already chosen, so we should use association rule learning models. We could deploy two types of models:
  + Unsupervised learning models: k-means clustering, DBSCAN for example. These models are simple to deploy, using less hardware (less need for CPU/GPU), and as such, are suitable for smaller companies (like in our case) to build recommendation systems (either to find similar users based on geographical area, which we already have, or on favorite artists/songs. We could also recommend artists liked by users who engaged with the same artists.
  + Deep learning frameworks (neural networks) for associative learning. These models could use a high amount of resources (GPU/CPU usage), could take longer time to train, and harder to deploy, so they are not recommended.
- Also, we could recommend users based on genres and tags of artists (to make our system more accurate), but since we do not have this metadata for artists in our

current dataset, we would need to record this data. Then with the new dataset we could combine this model with unsupervised learning models that we mentioned earlier.

- **System design:**

- Firstly, we could use historical data in user_events, users and artists files to train model offline to identify patterns. Data could be queried and stored in SQL databases, but then could be added using Apache Spark, Airflow or Kafka. (Some of the platforms like Kafka are free to use, since they are open-source).
- For the offline training, we could do it using Scikit-learn for K-Means clustering in Python. (If we decide to use deep learning frameworks, we will need to use PyTorch or TensorFlow to build our own model).
- After evaluating models based on a confusion matrix ( Precision, Recall, F1-Score) or other clustering metrics, we will choose a model to then create an inference model for suggestions. This should be lightweight and scalable, and we could use caching for this (by copying frequently accessed data and store it in our own database). The model should be trained periodically, and run using Flask in our website (to integrate with Python code).
- The front-end of our website would show the result in “Artists You May Like” in the UI, and personalize based on users' recent activity.

(Users might follow some artists too much, leaving others not recommended anytime. To resolve this problem, periodically we might need to collect data of artists that are not frequently followed, and showing it in another list for users (Hidden Gems, You Might Also Want To Listen).

- **Evaluation of models:**

- We will need to know relevancy of our model with user preference, whether users interact more with recommended content, and are our recommendations driving revenue and retention. Diversity of recommendations can also be added, to make sure that our model recommends lesser-known artists (though we will need to add this in the metadata).
- For the user’s engagement with recommended contents, we could track percentage of recommended artists clicked, how much time has a user spent with newly recommended artists, and percentage of recommendations leading to following and sharing artists. For the relevancy of our model, we could collect the same data as user_events and run the same relevancy tests to calculate the confusion matrix. As for the business impact, we will need to collect the number of subscriptions purchased (or advertisements added) for new artists and number of new followers.
- Data collection can be done in several ways:
    + Interactions (like, share, play tracks from artists, follow/share artists). This could combine with our analysis of average recommendations clicked by users, time users spent with an artist, etc.
    + Direct feedback to our system (thumbs up/thumbs down, how would you, user, rate the system (from 1 to 5 for example)).
    + A/B testing (dividing users into 2 groups: current system and new system, then calculate aforementioned metrics).

- Evaluation can proceed like this: After running models in the real world, we will collect users' interactions and compare it with performance in the past. We could also do A/B testing by dividing users as aforementioned, and calculate whether the performance of our new model is better, and whether the improvement is statistically significant or not. The model can then be upgraded based on our new insights, and should be fine-tuned periodically.