System – predict a user's age and gender

1) **Before the user listens**
Before we have any interaction with the user we have (presumably) all previous historical information from existing users. This likely includes the three tables provided (**users, listens, artists**). Additionally we can correlate songs with artists (possibly more useful than genre, especially for artists with multi-generational appeal). This kind of information is external to iHeart but freely available. Assuming we have a table with the following information:

Schema(Artist, song title, year produced)

We can join with the existing **listens** table, and then with the **users** and **artists** tables. To bootstrap this model we'd want the following columns in the model table:

Schema(profile_id, age, gender, listen_date, songs_listened, genre, artist_id, song_title, year)

This can be loaded into a distributed database to perform queries on the age and gender of the userbase for a given song or genre. Suggestion would be distributed relational table like PostgreSQL. Other options would be key-row stores (keyed by genre/song_title) like Cassandra, which may support scalable writes better but have less flexible schema design if we acquire new information.

iHeart presumably contracts with or maintains a CDN to deliver songs to users. A unique DB of this type can be maintained at each CDN center separately as tastes likely vary by region. This may make the engine blind to users traveling out of their "home" region, or transplants/expatriates. Having a central DB may increase latency of the guess if used as input to recommendations or further songs in the stream. DBs with this schema will be referred to as GuessDB.

2) **At user listen**
User listens on digital app. Assume for the profile_id we don't already have their age/gender information. If the age/gender guess is being used to tune the user's song stream, most use cases will have the length of the first song (usually at least minutes) to make a determination. Latency will be some concern, but not a major one.

Some proposals for returning guesses:

- In the cold start case for a user, query GuessDB and return median age and most common gender for users who have previously listened to that song. This is simplistic and easy to implement but of course has a high chance of being very far off.
- Set up machine learning (i.e. Apache Mahout) where the input is a partial Schema from GuessDB: Schema(songs_listened, genre, artist_id, song_title, year) and expected output is age/gender. Test initially by training against a control group for a week's worth of listens and make forward prediction to next week's listens. Somewhat more difficult to maintain and construct, but can tell us which schema fields correlate more

highly with age/gender and can potentially inform other simple queries we might make. Downside is that it is often computationally expensive and shouldn't be run alongside any nodes that also manage or query the CDN, and requires continual retesting of the model and results to ensure accuracy.

- As an external addition to the dataset, we may consider GPS information (users may be more likely to listen to the same songs as other people near them in their circles of friends), but it's notoriously finicky, and location (particularly for mobile devices) information availability is not entirely correlated with Internet availability. GPS information would need to be cleaned for certain extremely common locations not tied to population centers, indicative of default GPS information.

3) After user listen

After a user's listen, we can record it to the DB. As before, this assumes the listen is recorded at the same time the application API makes a request for a song. The listen is recorded to GuessDB. We can follow up with the user:

- Ask the user's age/gender. Can provide feedback to the model prediction, but may be inaccurate or simply ignored.
- Ask for the user's opinion on the stream, if the prediction is being used to inform further songs in the stream

This enjoyment follow-up query can be added as a column to the GuessDB.