```
LOAD CSV WITH HEADERS FROM 'file:/songs.csv' AS row
CREATE (:songs{songID: row.songID, title: row.title, year: row.year})

LOAD CSV WITH HEADERS FROM 'file:/users.csv' AS row
CREATE (:users {userID: row.userID, name: row.name})

LOAD CSV WITH HEADERS FROM 'file:/userRatedSong.csv' AS row
MATCH (s:songs {songID: row.songID})
MATCH (u:users {userID: row.userID})
MERGE (u)-[r:RATED]->(s)
ON CREATE SET r.userRating = toFloat(row.userRating)

MATCH(u1:users)-[x:RATED]->(s:songs)<-[y:RATED]-(u2:users)
WITH u1, u2, COUNT(s) AS intersection, COLLECT(s.songID) AS i
MATCH (u1)-[:RATED]->(u1s:RATED)
WITH u1,u2, intersection,i, COLLECT(u1s.songID) AS xLength
MATCH (u2)-[:RATED]->(u2s:RATED)
WITH u1,u2,intersection,i,xLength, COLLECT(u2s.songID) AS yLength

WITH u1,u2,intersection,xLength,yLength

WITH u1,u2,intersection,xLength+((x IN yLength) WHERE NOT (x in xLength)) AS union, xLength, yLength

RETURN u1.title, u2.title, xLength,yLength((1.0*intersection)/SIZE(union)) AS jaccard ORDER BY jaccard DESC
LIMIT 100

MATCH(u1:users {name: 'Steve'})-[s:SIMILARITY]-(u2:users)
WITH u2,s.value AS sim
ORDER BY sim DESC
RETURN u2.name AS Neighbor, sim AS Similarity

MATCH(u1:users)-[x:RATED]->(s:songs)
WITH u1, algo.similarity.asVector(songs, x.rating) AS u1Vector
MATCH(u2:users)-[x:RATED]->(s:songs) WHERE u2 <> u1

WITH u1, u2, u1Vector, algo.similarity.asVector(songs, x2.rating) AS u2Vector
WHEREsize(apoc.coll.intersection([v in u1Vector | v.category], [v in u2Vector | v.category])) > 4

WITH u1, u2, algo.similarity.pearson(u1Vector, u2Vector, {vectorType: "maps"}) AS similarity
ORDER BY similarity DESC
LIMIT 4

MATCH (u2)-[r:RATED]->(s:songs) WHERE NOT EXISTS( (u1)-[:RATED]->(m) )
RETURN m.songID, SUM( similarity * r.userRating) AS score
ORDER BY score DESC LIMIT 25
```