The primary file you will want to use is `find_interesting.py`. In it are located a number of different functions for producing rankings, as well as a few functions for printing rankings in a useful way. (There are also a few helper functions that you can ignore. In general, functions of the first type will have the first argument be `features=all_features`, and functions of the second type will have the first argument be `fn` or `ranking`. Any functions that don't have these as the first argument, you can generally ignore[1].)

There are a few examples of some calls you can make in the "main" section at the bottom of the file. You can un-comment any of them to have them run when you call `python find_interesting.py` (please use python v2.7). Here, I'll highlight a few and explain exactly what each does.

`top_20(novelty_ranking)`
Prints the names of the top 20 remixes in the novelty ranking.

`bottom_20(strangeness_ranking)`
Prints the names of the bottom 20 remixes in the strangeness ranking, with worst printed last.

`all_rankings(novelty_ranking)`
Prints the names of all the remixes next to their ranking number.

`all_rankings_norm(novelty_ranking)`
Prints the names of all the remixes next to their ranking number, normalized to [0,100].

`top_20(using_distance_to_original)`
Creates a ranking by calling `using_distance_to_original`, then prints the top 20 of that ranking.

`top_20(pca_and_call(fn=using_half_gmm_union, dim=5, k=20))`
Creates a ranking by performing PCA on the features, retaining 5 dimensions, then calls `using_half_gmm_union` with k=20.

`top_20(whiten_and_call(fn=using_distance_to_original))`
Creates a ranking by first whitening the features (0 mean, unit variance), then calling `using_distance_to_original`.

`top_20(ensemble_by_group(using_half_gmm, 5))`
Creates a ranking by splitting the features into the pitch, rhythm, and timbre groups, and calls `using_half_gmm` with k=5 on each of the three groups of features, then outputs the average ranking of those three.

`top_20(ensemble_by_group(pca_and_call, using_num_closer_than_original, 8))`
Splits the features into the groups of pitch, rhythm, and timbre. On each of those groups, performs PCA, retaining 8 dimensions, before creating a ranking with `using_num_closer_than_original`. Ensembles those three rankings together for the final output.

`top_20(ensemble_by_group(pca_and_call, using_half_gmm_originals, 3, 10))`
Same as above, but retains 3 dimensions, and calls `using_half_gmm_originals` with k=10.

---

[1] The exceptions to this rule are `ensemble_all_together`, which contains the final ensembling algorithm used for the writeup (please use version 1.2 if you want results consistent with mine), and `run`, which calls `ensemble_all_together` as many times as you ask in order to provide a more stable final ranking. Feel free to run these, but they may take a while as compared to the other functions. Because of this, I have serialized their output, which is loaded automatically for you into the variable name `novelty_ranking`.

I have included the code I used to find features from raw audio as well, though it may not work for you out of the box. The main file is `featurize.py`. Its titular function takes in a list of strings, each of which contains the release number you have used for each particular raw audio file. It adds a line to `features.csv` for each string.

- Each release number should have an entry in `bpms.txt`, `keys.txt`, and `releases.txt`.
- bpms can be found on sites like Beatport, and I found keys by hand (since Beatport's keys are occasionally incorrect).
- There should also be a symbolic link to each raw audio file in the relative filepath "`../anjuna_symlinks/<release number>`" (you can change this by finding it in each of the helper python scripts that calls an external binary).
- You should have aubio, Melodia, and sonic-annotator installed. All are free.
- Finally, please note that this project uses the convention for release numbers that original mixes are integers, remixes are floats, and `truncate(remix) == original mix`. The decimal portion of a remix number can be whatever you want, as long as it is different for all remixes of the same original mix.

There are a lot of steps involved in creating features using my code. Please note, however, that this is in no way necessary in order to appreciate the main purpose of the project, which you can do by simply relying on the features I have provided in `features.csv`.

Required libraries for manipulating features:
numpy
scipy
scikit-learn

Required libraries and binaries for finding features:
numpy
scipy
matplotlib
aubio
Melodia
sonic-annotator