General Rules and Requirements for Final Report and Submission:

1. The final submission should contain your code; a file describing the changes you made — the functions/code parts that you implemented or changed; a report explaining your results.
2. Any intermediate results or even the failing experiments that are along with your process of analyzing the datasets could be useful.
3. Ensure that **nothing is hard-coded in your code**.
4. For each project, your codes and report should also support the specific requirements for that project and the input and output format that is given.

Project 1: Phenotypic Prediction from Transcriptomic Features

Overview of the intermediate results:

Almost all of the groups could beat the baseline. Some of you were able to use equivalence classes in a productive way to improve your validation score. **Defining at least one type of feature from equivalence classes that improves your intermediate results is a requirement for the final submission.**

One thing that you should note for the final results is that when you are using cross validation to provide some evaluation of your model, you shouldn't use **ANY** information or properties of the fold that you are evaluating the results on. For example if you are using a Decision Tree to select the best m features and then running a classifier on them, the decision tree and the classifier should **BOTH** be built only on the k-1 folds not used for validation (excluding that one fold that is used to evaluate the performance in that cross validation round) and not the whole training set. At the end, the model that you provide for testing should be built on all the train set.

Final Submission:

Multi-task learning (predicting more than one label at the same time) can reduce the complexity of the model by using more regularization and hence make the model more generalizable which in theory is expected to provide better prediction accuracy for each of the labels. However, it is not always what is happening in practice. For the final submission, we also provided you with one more label (**sequencing center**) in addition to the **population** that you modeled before. There are no baselines for this new label. The main purpose here is for you to use multi-task prediction approaches to predict both of the labels using one joint model and compare it with the individual classifier's performance for each of the labels.

The new csv label file which we uploaded in the same directory as before for project 1 (https://drive.google.com/open?id=1oag2NrGNfMMlmBTlnTmwaiQGzGvYyNcA) contains three columns :

- accession
- population
- sequencing_center

Final Submission Requirements:

1. An executable piece of code that gets the address to the directory containing all the test samples and returns the F1-score, and accuracy of the model on the test set.
   We should be able to test your model using the following command:
   ** <**executable> <address to the model dump> <address to the test samples root> <address to the test labels file>** **
   For example if your code is in python, the command would be
   ** **python predict_experimentLbl.py <address to the model dump> <address to the test samples root> <address to the test labels file>** **
   If you are using Jupyter notebook, at the end convert it to a python file that can be called in command line. But you can submit the jupyter notebook as well as the python file.
2. Your Model dump file (e.g. using pickle)
3. Make sure to have at least one feature type derived from equivalence classes.

4. In your report you should provide the cross validation results for each label using separate models and one multi-task model.

Project 3: Uncertainty Quantification

Overview of the intermediate results:

In the previous step, you designed different tasks and pipelines to predict and separate the rejected vs the accepted transcripts that have the required conditions. As we suggested to some of you, to get one step closer to the final required output, you could design a task to predict how far the mean of the bootstrapping distribution is from the true count of the transcript (predict the error). But note that the final purpose of all of these models at the end is to be used to update the bootstrapping counts and make them more accurate and closer to the truth.

One observation by all of you was that the number of transcripts in the truth file is not equal to the number of transcripts in bootstrapping file. **For the future analyses, assume that the true count for those transcripts that are not in the truth file is equal to 0.**

Final Submission Requirements:

1. Provide an executable file that takes a file containing the bootstrapping counts (with the format the same as the files given to you) as input and outputs a file with an updated count per each cell of the matrix.
    a. Notice that you are not given the truth file here. So the model that you apply to the counts to update them should be independent of the true transcript counts.
    b. The minimum requirement for the updated bootstrapping counts is to have fewer failing transcripts by the same definition as before.
    c. To be exact, we want the mean of the distribution of counts per each transcript to get closer to the true count.
2. You might use the distributions' mean and standard deviation as features to your model as well as each transcript individual properties and features derived from equivalence classes.
3. In your report, explain in details the statistical and/or machine learning tasks and trials that gave you the the final count correction model.

Project 4: Adding Salmon-based Modules to MultiQC
Final Submission Requirements:

1. Fix representation of the GC-bias model (if needed)
2. Add sequencing bias
3. Add heat maps

Bonus: Add visualization for effective length/transcript length

GC-bias: The three rows in the file represent the observed and expected GC bias assuming that the bias is actually low, medium or high, respectively. The weights are normalization values for all of them (you have to multiply the 3 rows with the weights to get actual values). The x-axis of your plots should be from 0-100.
We would like to see four plots. A plot each for the first, second and third row of all the samples ((observed*weight)/(expected*weight)). Hovering over the lines should display the sample name. The fourth should show the average of these three rows (multiplied by weight) across all the samples. I believe all of you have already done this part, or something similar, for the mid-project. Make the minor updates for the final submission.

Sequencing bias: The 5' and 3' observed and expected sequencing bias files are given. You can read in detail about these here: http://salmon.readthedocs.io/en/latest/file_formats.html#sequence-specific-bias-files. For this, we only need you to plot the marginalized properties for each base over all the samples for the 5' and 3' ends. Again, an additional plot should show the average bias across all the samples.

Heatmap: A heatmap to show the correlation of the average GC and sequencing bias between the samples. So there would be 3 heat maps (GC, 5', 3') of size nxn, where n is the number of total samples.

Bonus: The "quant.sf" file contains the effective transcript lengths and actual transcript lengths. We would like to see a single plot representing the distribution of the ratio of these two across the samples. Be creative with this part. Summarize the important information in a single plot as you see fit.

For final submission in addition to general requirements, make sure in your implementation you check for whether or not any bias file exists in the Salmon output are accurate and generic. Your code should fit in perfectly with the multiQC project. Depending on how the final submissions go, we would like to choose one project and commit it to the multiQC codebase, contributing to the Salmon module.

Project 5: Implementing Long Read Mapping Algorithms

Requirements for final submission:

1. Provide two separate executable files,
    a. one for building the containment hash and minhash indices on the reference file (the one that contains set of longer sequences)
    b. One for calling the query on the long reads (the file that contains set of shorter sequences)

    NOTE: No variable property of your index (e.g. kmer size or number of hash functions) should be hard-coded in your code and your query should extract these settings from your index.

    The executable files should be callable from command line. For index building you should provide all the different options for Bloom Filter and Min hash sketches as the input arguments in addition to the reference file address. The options can have default values though.

    The query exec file gets as input at least the address to the index file and the query file containing reads.
2. Generate simulated long reads using the same function provided in the containment hash repository.
3. Replicate the same plots as in figure 6 in the containment hash paper
4. Same plot as above with X axis being k-mer size.
5. Compare the space and running time of your implementation in C++ vs the one in python for some fixed values of number of hashes and kmer size.