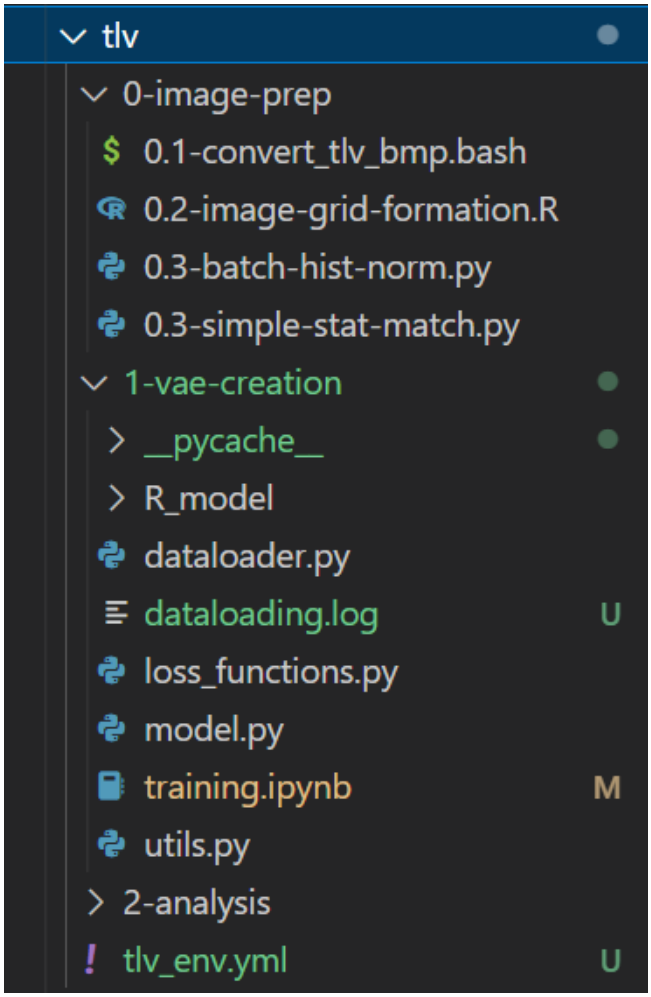
TLV repo code structure / organization:



- The entire "tlv" directory is found in the "projects" directory of the candescence_master repository. Other candescence projects (e.g. Guelph) are also found here.
- **Logic/Workflow**
    - Originally, the directories within tlv contained numbers that specified their order of execution and the general workflow. Since we abandoned R in favor of PyTorch, it got a little messy. "2-analysis", for example, is not used at all by the PyTorch model.
    - Now:
        1. Files in 0-image-prep are run first. 0.1 and 0.2 are there to convert the raw images received from the Berman Lab into the subdivided colony images - ultimately storing them in data/refined. These

datasets have largely already been created though, so these likely do not need to be run again unless new images are added.
    - The files titled 0.3-... are both equalization methods that create a new dataset where the images are ideally less varied in terms of background color. Right now simple-stat-match works best for this.
2. "1-vae-creation" contains all the building blocks, associated tools, and training files associated with the model. "_pycache_" and "R_model" can largely be ignored, with the latter containing the legacy files and components of the initial R model.
    a. Files within this directory are named according to their purpose, and often import from one another. These files are the main components of the current model and are explained in further detail below.

- **Main PyTorch model files**

    - **"Dataloader.py"**
        - This file contains the functions and classes related to a dataset's creation and loading into the model for training.
        - **transform**
            - The transforms that are applied to each colony image in order to get it in a format suitable for passing to the model. Always run are resizing (to 135 x 135), greyscaling, and ToTensor (pytorch models take tensors).
        - **CandidaDataset**
            - This is a custom dataset class that handles how the colonies are retrieved by the model. It has several different types of retrieval methods for either getting a filepath, colony, etc. CandidaDatasets are index-sensitive tuple lists (each index representing a colony), where each tuple able to be retrieved contains the image (in torch tensor form) and the one-hot encoded vector of the conditional variables used (also in torch tensor form). See **__getitem__** for how these tensors and one-hot encodings are generated and returned.
        - **create_dataloader**

            - This function creates the training, validation, and testing dataloaders (and associated datasets) which are used by PyTorch models during training. They are split up and divided according to the parameters desired by the user. The function also *saves* the datasets so that they can be referenced or used later.
        - **collect_categories**
            - Used in CandidaDataset to get the categories of a colony that ultimately make up its plate, which is then used to create the unique one-hot encoding of that plate.
    - **"Model.py"**
        - This file contains a lot of logging info (i.e. logger.[blah]), this is just for troubleshooting and does not affect the architecture.
        - **Encoder**
            - This class outlines the architecture of the encoder's layers, and the forward pass method through it. Takes as input a 135 x 135 image tensor (and its corresponding one-hot encoding for any conditional variables), and outputs the z/latent vector.
            - 3 convolutional layers (with 2 batchnorm layers in between), followed by a flattening layer, one-hot encoding concatenation, and linear layers (to derive sigma, mu, and z)
            - __init__ is used to define all the layers contained in the encoder, while the forward() method links them all together.
        - **Decoder**
            - Same concept as the Encoder class, but does the operations in reverse order (to go from z to a reconstructed image)
            - Has a conditional check that only concatenates the one-hot encoding of the colony if include_OH is true. Often you may not want to include conditional variables in the decoder.
        - **VAE**
            - Same concept as either the Encoder and Decoder.
            - __init__ initializes the VAE's encoder and decoder halves as instances of their respective classes. The forward() pass method links the two halves, and returns the output of the decoder.
    - **"Training.ipynb"**

        - This jupyter notebook is pretty well documented and commented. It essentially contains the entire data loading, model training, and figure generation / analysis pipeline.
        - By changing parameters in the arguments dictionary at the start of the file (experiment name, hyperparameters, file paths, etc.), you can completely customize your experimental run however you'd like. Then, just hit "run all" on the entire notebook.
    - **"Loss_functions.py"**
        - This short file simply contains the loss calculating functions used during training
        - Loss functions are named logically, and return either the KL, MSE, or KL + MSE (total) loss. MSE loss is calculated between x and x_hat (the reconstructed image), while KL loss (a measure of how well it fits to the distribution) uses the sigma and mu.
    - **"Utils.py"**
        - **reconstruction_compare()**
            - Takes a dataset, trained encoder and decoder, and an integer n. Creates a panel showing n reconstructed images from the dataset, alongside the original image.
        - **get_latent_vectors()**
            - Given the indices of images in a dataset, return a dataframe of only those images' filenames (one column), with additional columns representing the latent coordinates of those images after being passed through the trained encoder.
        - **dataframe_w_latentvecs()**
            - Merges the dataframe created by get_latent_vectors() with the appropriate corresponding data for each colony/plate found in the metadata table provided by Judy and Austin. Output is a big dataframe ("MASTER") in which each row is a colony, and there are columns for both metadata info (medium, day, geography, species, etc.) as well as the latent vector coordinates for the encoded colony.
        - **create_img_scatterplot()**
            - Uses either UMAP or t-SNE to reduce dimensionality of latent vector to 2. Plots resulting coordinates as a scatterplot, overlaying each point as an image of the original colony it represents (not the reconstructed image)

- **create_model_view_img_scatterplot()**
  - Not really used, but is supposed to do the same thing as create_img_scatterplot() but instead overlay the image of the colony as the model would see it (aka with some transformations like greyscaling, etc. applied)
- **create_annotated_scatterplot()**
  - Creates a scatterplot representing the latent space, but this time colors point by a metadata category of interest (e.g medium, day, etc.)
- **get_plate_id()**
  - From a colony filename/filepath, return the plate number (function says ID but it isn't really an ID in the same way as the term is used for one-hot encodings, as there are several plates with the same plate number - so it is not unique). Colonies from the plates with "Pwt" have no plate number, so are assigned the number of -1.
- **zoom_img_scatterplot()**
  - Given a set of (xmin, xmax, ymin, ymax), zoom in on a section of scatterplot (either a UMAP or t-SNE one) to get a better look at plot regions of interest. Can do image-overlay style or color-by-metadata style. Called "zoom", but essentially is just recreating a scatterplot for that specified bounding box.
- **get_growth_medium()**
  - From a colony filename/filepath, return the growth medium as a string. Fixes the inconsistencies in the names of some mediums, e.g. both 'spdr' and 'spider' become simply 'spider'.
- **get_day()**
  - From a colony filename/filepath, return the growth day/period of the colony (2 or 5).
- **get_one_hot_encoder()**
  - Get a one-hot encoder that is fitted to all the different plate IDs or possibilities (these are actually unique because they are made up of the plate's number + medium + day). This encoder can later be transformed with a plate number +

medium + day combo to give the unique plate one-hot encoding.

- **get_size_onehot_from_tensor()**
  - Given a colony's image tensor, return a one-hot encoding/vector of the colony's approximate size. This is done by binarizing the pixel values to 0 or 1 based on some threshold (in this case 0.5), and then finding the proportion/percentage of 1s in the image (which is an approximation of the size), and determining which size bin it fits into, and then getting a one-hot vector representing that bin.
- **get_intensity_onehot_from_tensor()**
  - Similar idea to get_size_onehot.., but tries to find the overall intensity of the colony and generate a one-hot encoding of it. Not used by default (transforming the entire dataset before use into one that normalizes the intensities of all images is more effective), and is commented out in dataloader.py.
- **create_loss_graph()**
  - Creates a graph of the training and validation loss over the training period/epochs.

- **NOTE:**
  - Many of these functions save their results (figures, dataframes, etc.) to the appropriate folders in data/refined, so they can be accessed later on if you keep track of which experiment they were a part of.

- **Setting up conda environment**
  1. Install miniconda3
  2. Create a virtual conda environment using the tlv_env.yml file found in the projects/tlv folder of the repo.
  3. Make sure to select the python installation found in this environment whenever prompted by VSCode or whichever interpreter.
  4. If installing new libraries or packages through the command line, make sure the conda environment has been activated first.