There were too many different files so instead of going through each one, I am putting all my notes into a doc.

**Main.py**

You have several imported scripts that are unused. This will slow your processing.

line 37: No need for the if \_\_name\_\_ == '\_\_main\_\_'. That is only necessary if this is a module that can be used as a main or import.

Because of that though and the fact that there is so little here, you may consider simply adding this code to data and use the

if \_\_name\_\_ == '\_\_main\_\_' in that script to call the main functions up. Everything not in if \_\_name\_\_ == '\_\_main\_\_' can be used

if the script is imported. Alternatively, as I had suggested before, instead of placing this code into data, you could add the code

to create a gui here. That would be the best option if you have the time.

line 39: I don't see where you diferentiate the training data vs the testing data. It is always a good idea to differentiate the two

so that anyone that looks at your script will be able to determine what does what. The data (as opposed to data\_test) should

be separated into around 2/3 training and 1/3 testing data randomly. This allows you to know if your model is training well or not

and if your model is over-trained or not. Over-training the model is just as crippling if not more so than a model that is not

trained enough. In other words, to know if the model is working at least somewhat properly, you much have training and testing on

data from within the same session. Then data\_test should be completely test data with no training. But there is no distinction from

this code. It looks like it is all treated the same.

Also, comments, where be they at.

**Data.py**

colors\_dict and tastant\_colors\_dict are the exact same thing and you do not use colors\_dict after copying it to

tastant\_colors\_dict. So having both is wasted code. Eliminated one of them.

\* lines indicates lines before I did anything.

line 113: There is no reason to define a function in a defined function when that function is the only defined

funcion and is going to be run every time the original function is run. In fact, you can reduce it even further

by puting the one line function into the if statement as a true/false

line 90: For this function, putting elif instead of just if wil ensure you only get a single error notice instead

of potentially 3 errors. For this to work though, you need to move the "if not any" statement to be first.

line 89: You use @staticmethod but still use instance data??????? That defeats the purpose of using @staticmethod.

Line 107 is good.

You have a lot of defined functions and some of them can probably be combined since they do very little. The

defined functions are used to clean the code up and keep variables to a minimum/delete variables after a function

is run. This has an added effect of speeding up the program. By separating out functions so much I think you may be

making more work than there needs to be.

**Funcs.py**

\* lines indicates lines before I did anything.

If a function is only going to be used once or in a single script, as many of these seem to be, it should just be placed

inside the script that uses it. The benefit of using a functions script is that you can call the script from many programs

and use the same function each time without having to rewrite the function. Alex fell into this trap as well where he had

scripts calling scripts calling more scripts (script-ception). Simple is always better as is fewer scripts.

ln174 "if not files\_events:" should be "elif not files\_events:" This will ensure it only comes up if files\_trace does work.

line 304: My opinion is that this gui should go in the main script. The main script should be the glue that holds everything

together and runs through each step. Once again, ideally you do not have script-ception.

**models.py**

\* lines indicates lines before I did anything.

Once again, you have imported functions that are not used.

As far as immutable vs mutable. That will really depend on how well the model performs from one trial to the next.

However, I suspect that a single immutable model will not work because too many variables will change between

sessions/animals. The immutable would be a better model, however, if it is reliable because each instance of the

model will have different parameters that it uses to predict tastants and this adds in variability that is not

ideal.

Many learning models are not suited for categorical data, not sure what type of model SVM() is, but it is interesting

that this model doesn't do categorical. That shouldn't be too much of an issue though because you convert to numerical

and then convert back before displaying the data.

Functions that you only call once don't really need to be defined especially if they are only authenticating data.

line 174: You "from sklearn.svm import SVC". All imports should be done in the beginning for consistency.

As for the model, it looks reasonable, but since I do not know this model, I would not be able to help with specifics.

However, as I mentioned in the main script, you need to train and test (evaluate) on data from the same session to

get best results. When you have a model that is trained properly, then you use it to test accuracy on new sessions.