Logistic Regression Lab

`colour-data.csv` : A data set with almost 4000 data points that we can try to learn with.

I'm being a little lazy here in saying "print score". If you are trying to make a decision about which model to use, you should not be using the accuracy score of the test set to make a choice for reasons I talked about last week. If we are trying to make a decision between using say RGB or LAB we should be comparing their *validation scores* using something like cross_val_score `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html`.

Tasks

Using the Jupyter Workbook `classlabstart` that has the following: Normal importing for numpy/pandas/matplotlib. Also includes lab2rgb, rgb2lab, a listing of basic colours and a method for displaying your results.

Do the following in the workbook

1. Start by getting the data: read the CSV with Pandas. Extract the X values (the R, G, B columns) into a NumPy array and normalise them to the 0-1 range (by dividing by 255: the tools we use will be looking for RGB values 0-1). Also extract the colour words as y values.
2. Check the shape of these arrays, are they as expected?
3. Partition into training sets and test sets. You will need to import train_test_split
4. Create a LogisticRegression model, call the model model_rgb You will need to import `LogisticRegression`.
5. Print the score for the model and then plot_predictions for the model.

First model done.

Note: An accuracy score of 0.5 is not necessarily bad here. It is not a binary problem, there are multiple classes available. If you just assigned colours randomly you would get a worse result, so our model is better than random! And some of its mistakes may be close (pink/red?).

You can use `classification_report` to see where the mistakes are happening

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

but I will talk more about this next week.

This approach implicitly assumes that distances in the input space make sense: distances between training X and new colour values are assumed to be comparable. Maybe we need to change our inputs in some way.

Possibly our inputs are wrong: the LAB colour space `https://en.wikipedia.org/wiki/Lab_color_space` is much more perceptually uniform. Let's convert the RGB colours we have been working with to LAB colours, and train on that. The skimage.color module has a function for the conversion we need.

We will want a *pipeline* here. A pipeline will allow us to build in any preprocessing as part of the model so we don't have to remember to make any conversions ourselves when making inferences.

```
from sklearn.pipeline import make_pipeline
```

We are also going to have a custom transformation, from rgb to lab. For this we need a *FunctionTransformer*

```
from sklearn.preprocessing import FunctionTransformer
```

and the function to convert a RGB matrix to a LAB matrix

```
def makelab(X):
    X = X.reshape(1,-1,3)
    lab = rgb2lab(X)
    return lab.reshape(-1,3)
```

Do the following in the workbook

1. We are going to create a pipeline model where the first step is a transformer that converts from RGB to LAB, and the second is a LogisticRegression classifier, exactly as before.

2. There is no built-in transformer that does the colour space conversion, but if you write a function that converts your X to LAB colours, you can create a FunctionTransformer to do the work. makelab above does the following.

   - Some Numpy reshaping will have to be done as part of the function you create. skimage.color assumes a 2D image of pixel colours.
   - `.reshape(1,-1,3)` should work
   - then convert to lab (rgb2lab)
   - Reshape back to original shape `.reshape(-1,3)`

3. Make a pipeline that first uses FunctionTransformer(makelab) and then creates a LogisticRegression model.

   ```
   model_lab = make_pipeline(
       FunctionTransformer(makelab),
       LogisticRegression()
       )
   ```

   This model now takes any input (in RGB), the first step is it converts it to LAB and then it performs LogisticRegression. `model_lab` can use `.fit`, `.score` etc. just fine

4. Same as above print score and the pictures

`LogisticRegression` has some hyperparameters that you can choose using k-fold cross validation to get a better model. Do LogisticRegression? to check which ones. C seems like an option to try different values. Try things like 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000. I will explain what *C* is in a later class.

What model performed best?

Examining the colour-data.csv the confidence column was not used. What happens to the models if we eliminate the ones with "poor" confidence? (this will affect the training and test data sets!). This would require creating a new dataframe.