Challenge week 1

0. To complete the weekly programming challenges, you need to have the scikit-learn (sklearn) python machine learning library and its dependencies installed (instructions here: https://scikit-learn.org/stable/install.html). Use pyplot for visualization. Importing libraries looks like:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer

from sklearn.decomposition import PCA

...
```

The tools that you use in this challenge will be useful for all subsequent challenges. While working on these weekly challenges it is expected that you will have to search the scikit-learn online documentation for proper function usage – all the information you need will be there.

For this challenge you will use the “breast cancer” toy classification dataset that comes with scikit-learn. Load the data using:

```
dataset = load_breast_cancer() # load all the data

print(dataset.keys()) # lists the contents of the dataset

print(dataset.data.shape) # shape of the X matrix
```

This dataset includes: a **data** matrix where the rows are subjects and the columns are measured features; a **target** vector containing the binary classification of each subject (healthy or diagnosed with breast cancer); a **feature_names** array with the names of the features (columns of X).

<u>Note: all plots must have a descriptive title and axis labels. Leaving these out will result in deductions, as plots that are missing these are practically useless for communicating information.</u>

1. A useful way to quickly visualize a dataset is to use principal component analysis, which finds the axes of maximum variance in the data. Using sklearn PCA with two components, fit the PCA model to the **data** and project it onto the two components (`PCA(n_components=2).fit_transform(...)`), then plot this transformed data using a 2D scatterplot (`plt.plot(...)`). Note that this visualization doesn’t take the **target** into account (it is unsupervised).

2. When training a machine learning model we are often concerned with features that are highly correlated with each other, as they carry redundant information. Calculate the pearson correlation (`np.corrcoef(..., rowvar=False)` will calculate the *symmetric* correlation matrix, or `scipy.pearsonr(...)` also works) between each pair of features/columns, and plot a histogram with 50 bins of the pearson correlations (`plt.hist(..., bins=50)`). Hints: `np.tril(...,k=1)` will help remove the redundant correlation values and `np.flatten(...)` will convert a 2d array into a 1d array.

3. Plot a scatterplot (`plt.plot(...)`) of "mean concavity" vs "worst error", with points coloured by class label (**target**). Include a legend to indicate which color indicates which label. Hint: overlay two scatterplots, one for each set of points.

4. Create a violinplot (`plt.violinplot(..., showextrema=False)`) of the features with highest mean, highest median, highest variance (plot should have three vertical distributions). First calculate the mean (`np.mean(...)`), median (`np.median(...)`), and variance (`np.std(...)`), then create the violinplot of these three arrays.

5. Divide the data into a training and test set using `train_test_split(data, targets, test_size=0.2, shuffle=False)`. Train a logistic regression classifier (`LogisticRegression(..., max_iter=1e4)`, default parameters) on the training split and score on the test split. What is the out-of-sample classification accuracy (ie accuracy on the unseen test data)? How many iterations did it take for the model to converge (check the `.n_iter_` attribute of the trained model).

6. Create a pipeline model (`make_pipeline(...)`) that combines `StandardScaler()` followed by the same logistic regression model as in 5, and train/test this pipeline model on the same data splits as in 5. What is the out-of-sample classification accuracy now that the data is being standardized? How many iterations did it take for the model to converge (to access the attributes of the logistic regression model now that it's part of a pipeline, use `[your pipeline model].logisticregression`)? Notice the effect that scaling the data has.

7. Divide the data into training and test sets using `KFold(n_splits=5, shuffle=False)`, and train the pipeline model from 6 on each training split (you will re-train the model 5 times, once for each split). Create a scatterplot of the test accuracies for each split (x-axis is the data splits, y-axis is accuracy).

8. Use `LeaveOneOut()` and the same pipeline model as in 6. Create a scatterplot of the test accuracies for each split (the number of splits is now equal to the number of samples (rows).