hw07: Preprocessing and Visualization

Due: See Labs Schedule document
Please also make sure you've read the Labs Plan document for information about how you'll work on this assignment, how you can ask questions, and how it will be graded.

What you'll learn in this assignment

In this assignment, you'll implement some basic data preprocessing functionality, and compare 1-NN performance for a few approaches. You'll then explore a bit of visualization and use that to make more conclusions about the dataset.

Related Code

From your hw06 solution, copy into your hw07 file your OneNNClassifier class, as well as the accuracyOfActualVsPredicted and findNearestHOF functions.

---

1) **Standardization on a DataFrame**

Recall the formula for standardization:

$$x_{new} = \frac{x - \mu_x}{\sigma_x}$$

Write a function called `standardize` that takes a DataFrame and a list of columns (attribute names) as arguments. It should compute the above formula on every value in every specified column in the DataFrame, all at once. It should mutate the DataFrame accordingly.

To do this, note that the same types of operators and methods that we learned about on Series objects can be applied to DataFrame objects. For example, study the behavior of the provided operationsOnDataFrames function. Using that function as an example, implement the body of the standardize function in just one or two lines of code.

When you're done, you should be able to run the testStandardize function to standardize only certain columns (Ash, Alcalinity of Ash, and Magnesium in this example). The output should be:

```
Before standardization, first 5 rows:
   Malic Acid   Ash  Alcalinity of Ash  Magnesium  Total Phenols
0        1.87  2.38               12.0        102           3.30
1        1.71  2.31               16.2        117           3.15
2        4.12  2.38               19.5         89           1.80
3        0.90  1.71               16.0         86           1.95
4        1.01  1.70               15.0         78           2.98

After standardization, first 5 rows:
   Malic Acid       Ash  Alcalinity of Ash  Magnesium  Total Phenols
0        1.87  0.049147          -2.244288   0.158126           3.30
1        1.71 -0.206007          -0.986639   1.208363           3.15
```

```
2        4.12  0.049147           0.001514  -0.752080           1.80
3        0.90 -2.393042          -1.046527  -0.962128           1.95
4        1.01 -2.429493          -1.345967  -1.522254           2.98

Means are approx 0:
Ash                 -8.183217e-16
Alcalinity of Ash   -6.860929e-17
Magnesium           -3.991813e-17
dtype: float64

Stds are approx 1:
Ash                  1.0
Alcalinity of Ash    1.0
Magnesium            1.0
dtype: float64
```

---

2) **Normalization on a DataFrame**

Recall the formula for normalization:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Write a function called normalize that takes a DataFrame and a list of columns, and mutates the DataFrame's specified columns according to the above formula. Again, do this by operating on the entire DataFrame at once – this should be a one- or two-line function.

When you're done, you should be able to run testNormalize to normalize only certain columns, and get:

```
Before normalization, first 5 rows:
   Malic Acid   Ash  Alcalinity of Ash  Magnesium  Total Phenols
0        1.87  2.38               12.0        102           3.30
1        1.71  2.31               16.2        117           3.15
2        4.12  2.38               19.5         89           1.80
3        0.90  1.71               16.0         86           1.95
4        1.01  1.70               15.0         78           2.98

After normalization, first 5 rows:
   Malic Acid       Ash  Alcalinity of Ash  Magnesium  Total Phenols
0        1.87  0.545455           0.072165   0.347826           3.30
1        1.71  0.508021           0.288660   0.510870           3.15
2        4.12  0.545455           0.458763   0.206522           1.80
3        0.90  0.187166           0.278351   0.173913           1.95
4        1.01  0.181818           0.226804   0.086957           2.98

Maxes are 1:
Ash                  1.0
Alcalinity of Ash    1.0
Magnesium            1.0
dtype: float64

Mins are 0:
```

```
Ash                  0.0
Alcalinity of Ash    0.0
Magnesium            0.0
dtype: float64
```

---

3) **Comparison of Preprocessing Approaches**

Write a function called comparePreprocessing that prints the mean accuracy in 10-fold cross validation for the OneNNClassifier on the wine dataset, for each of the following:

- The original, unaltered dataset
- The normalized dataset
- The standardized dataset

You will probably find it useful to use the `copy` method on your DataFrame object, so that you can have one separate copy on which to do normalization, and another copy on which to do standardization.

Discuss the following as comments in your file. I'll read these as I determine your grade. You may like to use a multi-line comment, which starts and ends with three single quotes ''' in Python.

a) Report your results from comparePreprocessing, along with a few sentences explaining why these results occurred.
b) Read the wine.names file. It reports accuracy obtained using a few different techniques. Note the result for 1-NN. What is z-transformed data?
c) Note the wine.names file's mention of leave-one-out in the reported results. Explain in a couple sentences what leave-one-out is. Based on this, also explain why the "1NN" results reported in wine.names are slightly higher than the results you obtained here.

---

4) **Introduction to Visualization**

Highly-dimensional datasets (i.e. datasets with lots of attributes) like this one can be difficult to visualize. For this reason, many data visualization techniques are available. We're not going to spend a lot of time on visualization in this course, but we'll hit a few key ideas here. Some of this discussion is adapted from:
https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-with-seaborn.html.
You can check that site for more details if you'd like, and you might also just Google around. All the official documentation is also available online, with many options possible.

Setting Up

For visualization, we'll play around just a bit with the module called seaborn. It's built on top of another visualization module called matplotlib. To get access to seaborn, add this to your code:

```
import seaborn as sns
```

Why "sns"? Eh… apparently it's some silly joke that now lots of people do:

https://stackoverflow.com/questions/41499857/seaborn-why-import-as-sns
So we'll just stick with this "standard".

Let's just play around a bit at the shell. Make sure you've loaded your code so that you have access to all the functions you need, and seaborn is imported. Next, let's prepare our data like this:

```
>>> fullDF, inputCols, outputCol = readData()
>>> standardize(fullDF, inputCols)
```

Histogram and Distribution Estimate

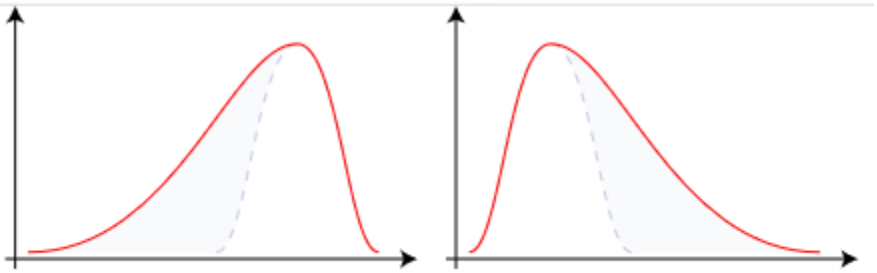
Alright, let's make our first visualization. You can observe a histogram of the distribution with displot. For example, for the Malic Acid attribute:

```
>>> sns.displot(fullDF.loc[:, 'Malic Acid'])
```

When you run this in Spyder, the plot is generated and ready to be viewed in the upper-right pane. You will likely need to click on the tab labeled "Plots" at the bottom of the upper-right pane first, and then you can select this plot. Also note that you can right-click on a plot to delete it, save it, etc.

Briefly, a histogram is a bar graph. Each bar represents values within a certain range on the x-axis. The height of the bar represents the number of values in the corresponding range.

Note the following figure:



So you can see from the displot that Malic Acid is positively skewed. Try this too:

```
>>> fullDF.loc[:, 'Malic Acid'].skew()
```

a) Make a displot and find the skew for Alcohol. Write its precise skew measure as a comment in your code, and indicate whether it is positively or negatively skewed.

Histogram and Distribution Estimate for Two Attributes

You can combine the distribution estimates of two attributes using jointplot. Try this:

```
>>> sns.jointplot(x='Malic Acid', y='Alcohol', data=fullDF.loc[:,
['Malic Acid', 'Alcohol']], kind='kde')
```

Note:

- The first argument specifies the x-axis text, the second is the y-axis text. Note, then, that this order must match what is given in the .loc command, or else the text won't match the data.
- kind='kde' means we're using the "kernel density estimator" method to determine the estimated distributions.
- Individual distributions are shown at the top (Malic Acid) and right side (Alcohol).
- The middle area combines the two distributions.

b) By observing a jointplot, approximately what combination of values is most likely for the Ash and Magnesium attributes? Try to be accurate within 0.5 or so. (And of course we're still using all standardized values here.) Write your answer as a comment in your file.
NOTE: I am not asking for the correlation between Ash and Magnesium. I want the two values (one for Ash, one for Magnesium) that together are the most likely combination of values.

Pair Plots

It is surprisingly easy to make a series of plots, one for each pair of attributes, showing the relationships between the pair in the dataset. Specifically, we can make a series of scatterplots, in which each row in the DataFrame is presented as a single point in 2-dimensional space, corresponding to the pair of attributes under consideration. Try this:

```
>>> sns.pairplot(fullDF, hue=outputCol)
```

That might take a little bit of time to execute. It takes about 20 seconds or so on the machine in my office (a machine that was brand new and high-powered in November 2017), so if your machine is less powerful, it may take longer. Watch for the red square at the top-right of the shell pane – when it turns grey, your code is done running. If it's red, it's still running.

The plot has so much information on it that it's probably too small to read. I recommend you right-click on it and save it as an image, then open up the resulting .png file in an image viewer (like Paint 3D in Windows) so you can zoom in on parts of it.

Note:

- The hue parameter sets the color of the dots according to the 'Class' attribute (the value of outputCol in the wine dataset).
- The graphs in the diagonal show curves derived from histograms, showing the frequency of each classification (by color) for that attribute.
- Note that the upper-right "triangle" of the output is the same as the lower-left, except that the axes are switched.
- There's a legend for the colors in the center right of the image.

Talk through what you see here just a bit with your partner, thinking about what the different graphs mean.

As a comment in your file, answer the following questions by looking at the pair plot:

c) If Proline has a ***positive value***, which classification is most likely?

d) Suppose you dropped most input columns from your dataset, keeping only Diluted and Proline. Clearly 1-NN performance would suffer. Would you expect performance to drop a lot, or only some? Why? (Your answer must explain why to receive credit.)

e) Answer question (d) again, but for Nonflavanoid Phenols and Ash.

f) Write a function called testSubsets() that checks your answers for parts (d) and (e). Specifically, read in the data and standardize it. Make a 1-NN classifier with inputDF containing only Diluted and Proline, and see how well it performs. Write your result as a comment in your code. Then make a second 1-NN classifier with inputDF containing only Nonflavanoid Phenols and Ash. Write your result as a comment in your code. Do your experimental results match your hypotheses in (d) and (e) based on the pair plot?

There is so much more visualization you can do! I'd encourage you to read up on this more if it interests you. But this gives you an idea of the kinds of things that are possible.

Preparing for Grading

When you've completed this assignment, please be prepared to run the test07 function. You should get:

```
0    1.699109
1    0.689041
Name: Alcohol, dtype: float64
0    0.881579
1    0.665789
Name: Alcohol, dtype: float64
Good dataset, accuracy: 0.8764705882352942
Bad dataset, accuracy: 0.4879084967320261
```

Your output text may vary for the last two lines, but the numbers themselves should match.

Please also think about what questions or ideas you might like to discuss now that the assignment is completed.