Title: Wine Quality

2. Sources Created by: Paulo Cortez (Univ. Minho), Antonio Cerdeira, Fernando Almeida, Telmo Matos and Jose Reis (CVRVV) @ 2009

3. Modeling wine preferences by data mining from physicochemical properties. In the above reference, two datasets were created, using red and white wine samples. The inputs include objective tests (e.g. PH values) and the output is based on sensory data (median of at least 3 evaluations made by wine experts). Each expert graded the wine quality between 0 (very bad) and 10 (very excellent).

Several data mining methods were applied to model these datasets under a regression approach.

4. Relevant Information: The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine.

5. Number of Instances: red wine - 1599; white wine - 4898.

6. Number of Attributes: 11 + output attribute

Note: several of the attributes may be correlated, thus it makes sense to apply some sort of feature selection.

7. **Attribute information:**

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density

9 - pH

10 - sulphates

11 - alcohol Output variable (based on sensory data):

12 - quality (score between 0 and 10)

8. Missing Attribute Values: None

Disk

- Use the `genfromtxt` function to read in the `winequality-red.csv` file.
- Specify the keyword argument `delimiter=";"` so that the fields are parsed properly.
- Specify the keyword argument `skip_header=1` so that the header row is skipped.

```
wines = np.genfromtxt("winequality-red.csv", delimiter=";", skip_header=1)
```

`wines` will end up looking the same as if we read it into a list then converted it to an array of floats. NumPy will automatically pick a data type for the elements in an array based on their format.

Let's select the element at row `3` and column `4`. In the below code, we pass in the index `2` as the row index, and the index `3` as the column index. This retrieves the value from the fourth column of the third row:

```
wines[2,3]
2.2999999999999998
```

Since we're working with a 2-dimensional array in NumPy, we specify 2 indexes to retrieve an element. The first index is the row, or axis `1`, index, and the second index is the column, or axis `2`, index. Any element in `wines` can be retrieved using `2`indexes.
If we instead want to select the first three items from the fourth column, we can do it using a colon (`:`). A colon indicates that we want to select all the elements from the starting index up to but not including the ending index. This is also known as a slice:

```
wines[0:3,3]
```

```
array([ 1.9,  2.6,  2.3])
```

Just like with list slicing, it's possible to omit the `0` to just retrieve all the elements from the beginning up to element `3`:

```
wines[:3,3]
array([ 1.9,  2.6,  2.3])
```

`:`), with no starting or ending indices. The below code will select the entire fourth column:

```
wines[:,3]
array([ 1.9,  2.6,  2.3, ...,  2.3,  2. ,  3.6])
```

We selected an entire column above, but we can also extract an entire row:

```
wines[3,:]
array([ 11.2  ,   0.28 ,   0.56 ,   1.9  ,   0.075,  17.   ,  60.   ,
         0.998,   3.16 ,   0.58 ,   9.8  ,   6.   ])
```

If we take our indexing to the extreme, we can select the entire array using two colons to select all the rows and columns in `wines`. This is a great party trick, but doesn't have a lot of good applications:

```
wines[:,:]
```

We can also use indexing to assign values to certain elements in arrays. We can do this by assigning directly to the indexed value:

```
wines[1,5] = 10
```

We can do the same for slices. To overwrite an entire column, we can do this:

```
wines[:,10] = 50
```

The above code overwrites all the values in the eleventh column with `50`.

NumPy is a package for working with multidimensional arrays. One of the most common types of multidimensional arrays is the 1-dimensional array, or vector. As you may have noticed above, when we sliced `wines`, we retrieved a 1-dimensional array. A 1-dimensional array only needs a single index to retrieve an element. Each row and column in a 2-dimensional array is a 1-dimensional array. Just like a list of lists is analogous to a 2-dimensional array, a single list is analogous to a 1-dimensional array. If we slice wines and only retrieve the third row, we get a 1-dimensional array:

```
third_wine = wines[3,:]
```

Here's how `third_wine` looks:

```
11.200
0.280
0.560
1.900
0.075
17.000
60.000
```

```
0.998
3.160
0.580
9.800
6.000
```

We can retrieve individual elements from `third_wine` using a single index. The below code will display the second item in `third_wine`:

```
third_wine[1]
0.28000000000000003
```

Most NumPy functions that we've worked with, such as numpy.random.rand, can be used with multidimensional arrays. Here's how we'd use `numpy.random.rand` to generate a random vector:

```
np.random.rand(3)
array([ 0.88588862,  0.85693478,  0.19496774])
```

Previously, when we called `np.random.rand`, we passed in a shape for a 2-dimensional array, so the result was a 2-dimensional array. This time, we passed in a shape for a single dimensional array. The shape specifies the number of dimensions, and the size of the array in each dimension. A shape of `(10,10)` will be a 2-dimensional array with `10` rows and `10` columns. A shape of `(10,)` will be a 1-dimensional array with `10`elements.

This doesn't happen extremely often, but there are cases when you'll want to deal with arrays that have greater than dimensions. One way to think of this is as a list of lists of lists. Let's say we want to store the monthly earnings of a store, but we want to be able to quickly lookup the results for a quarter, and for a year. The earnings for one year might look like this:

```
[500, 505, 490, 810, 450, 678, 234, 897, 430, 560, 1023, 640]
```

```
[500, 505, 490, 810, 450, 678, 234, 897, 430, 560, 1023, 640
```

The store earned `$500` in January, `$505` in February, and so on. We can split up these earnings by quarter into a list of lists:

```
year_one = [
      [500,505,490],
      [810,450,678],
      [234,897,430],
      [560,1023,640]
]
```

We can retrieve the ea