Using Python for Research Homework: Week 3, Case Study 3

In this case study, we will analyze a dataset consisting of an assortment of wines classified as "high quality" and "low quality" and will use k-Nearest Neighbors classification to determine whether or not other information about the wine helps us correctly guess whether a new wine will be of high quality.

In [1]:
```
# DO NOT EDIT
import numpy as np, random, scipy.stats as ss

def majority_vote_fast(votes):
    mode, count = ss.mstats.mode(votes)
    return mode

def distance(p1, p2):
    return np.sqrt(np.sum(np.power(p2 - p1, 2)))

def find_nearest_neighbors(p, points, k=5):
    distances = np.zeros(points.shape[0])
    for i in range(len(distances)):
        distances[i] = distance(p, points[i])
    ind = np.argsort(distances)
    return ind[:k]

def knn_predict(p, points, outcomes, k=5):
    ind = find_nearest_neighbors(p, points, k)
    return majority_vote_fast(outcomes[ind])[0]
```

Exercise 1

Our first step is to import the dataset.

Instructions

- Read in the data as a pandas dataframe using `pd.read_csv`. The data can be found at https://courses.edx.org/asset-v1:HarvardX+PH526x+2T2019+type@asset+block@wine.csv

In [2]:
```
import pandas as pd

data = pd.read_csv('asset.csv', index_col=0)

data
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	q
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	q
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	
...	...	...	...	...	...	...	...	...	...	...	...	
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	

6497 rows × 14 columns

Exercise 2

Next, we will inspect the dataset and perform some mild data cleaning.

Instructions

- In order to get all numeric data, we will change the `color` column to an `is_red` column.
  - If `color == 'red'`, we will encode a `1` for `is_red`
  - If `color == 'white'`, we will encode a `0` for `is_red`
- Create this new column, `is_red`.
- Drop the `color`, `quality`, and `high_quality` columns as we will be predict the quality of wine using numeric data in a later exercise
- Store this all numeric data in a pandas dataframe called `numeric_data`

In [3]:

```
# write your code here!

data["is_red"] = ""

is_red = []

for row in data['color']:
    if row == 'red':
        is_red.append(1)
    else:
        is_red.append(0)

data['is_red'] = is_red

numeric_data = data.drop(['color', 'quality', 'high_quality'], 1)


#numeric_data

#numeric_data.sum()
```

Exercise 3

We want to ensure that each variable contributes equally to the kNN classifier, so we will need to scale the data by subtracting the mean of each variable (column) and dividing each variable (column) by its standard deviation. Then, we will use principal components to take a linear snapshot of the data from several different angles, with each snapshot ordered by how well it aligns with variation in the data. In this exercise, we will scale the numeric data and extract the first two principal components.

Instructions

- Scale the data using the `sklearn.preprocessing` function `scale()` on `numeric_data`.
- Convert this to a `pandas` dataframe, and store as `numeric_data`.
  - Include the numeric variable names using the parameter `columns = numeric_data.columns`.
- Use the `sklearn.decomposition` module `PCA()` and store it as `pca`.
- Use the `fit_transform()` function to extract the first two principal components from the data, and store them as `principal_components`.
- *Note*: You may get a `DataConversionWarning`, but you can safely ignore it

In [13]:
```python
import sklearn.preprocessing
scaled_data = sklearn.preprocessing.scale(numeric_data)
numeric_data = pd.DataFrame(scaled_data, columns = numeric_data.columns)

# Junk code
# n_samples == 0
# n_samples == count
# n_components == min(n_samples, n_features)
# pca = sklearn.decomposition.PCA(numeric_data, n_components == 2)
# principal_components = pca.fit_transform(numeric_data)
# pca = sklearn.decomposition.PCA()
# pca
#principal_components = pca.fit_transform(numeric_data)
#pca

#print(numeric_data)

import sklearn.decomposition


pca = sklearn.decomposition.PCA()
principal_components = pca.fit_transform(numeric_data)

#principal_components
principal_components.shape
#pca
```

Out[13]:
```
(6497, 12)
```

Exercise 4

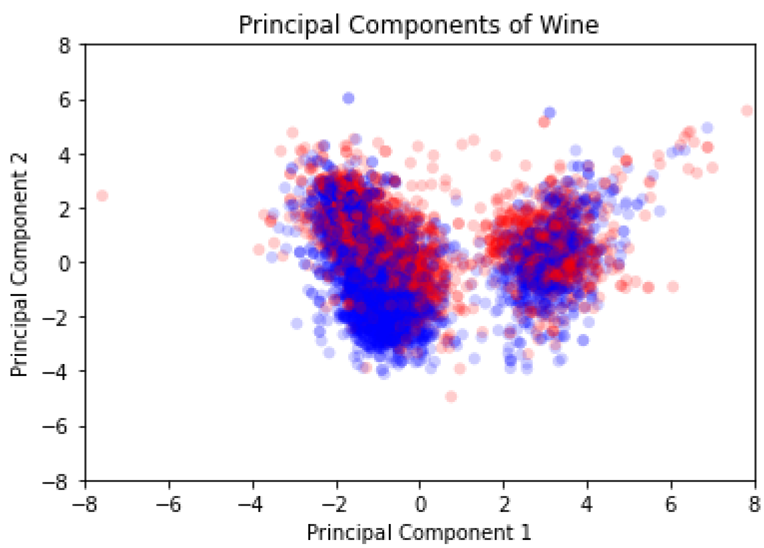
In this exercise, we will plot the first two principal components of the covariates in the dataset. The high and low quality wines will be colored using red and blue, respectively.

Instructions

- The first two principal components can be accessed using `principal_components[:,0]` and `principal_components[:,1]`. Store these as `x` and `y` respectively, and make a scatter plot of these first two principal components.
- How well are the two groups of wines separated by the first two principal components?

In [5]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from matplotlib.backends.backend_pdf import PdfPages
observation_colormap = ListedColormap(['red', 'blue'])
x = principal_components[:,0]
y = principal_components[:,1]

plt.title("Principal Components of Wine")
plt.scatter(x, y, alpha = 0.2,
    c = data['high_quality'], cmap = observation_colormap, edgecolors = 'none')
plt.xlim(-8, 8); plt.ylim(-8, 8)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



Exercise 5

In this exercise, we will create a function that calculates the accuracy between predictions and outcomes.

Instructions

- Create a function `accuracy(predictions, outcomes)` that takes two lists of the same size as arguments and returns a single number, which is the percentage of elements that are equal for the two lists.
- Use accuracy to compare the percentage of similar elements in the `x` and `y` `numpy` arrays defined below.
- Print your answer.

In [26]:
```
import numpy as np
```

```python
np.random.seed(1) # do not change

x = np.random.randint(0, 2, 1000)
y = np.random.randint(0 ,2, 1000)


## Junk code
# def accuracy(predictions, outcomes):
#     count = 0
#     for prediction in predictions:
#             for outcome in outcomes:
#                 if prediction == outcome:
#                     count =+ 1
#     accuracy = (count / len(x))*100
#     return accuracy

def accuracy(predictions, outcomes):
    return 100 * np.sum(predictions == outcomes) / len(outcomes)
accuracy(x,y)

#accuracy(x,y)
```

Out[26]:

```
array([False, False,  True, False, False, False, False,  True,  True,
        True, False, False, False,  True, False, False, False,  True,
       False, False, False,  True, False,  True,  True,  True,  True,
       False,  True, False,  True,  True,  True, False, False, False,
       False,  True,  True,  True, False,  True, False, False, False,
        True, False,  True, False, False,  True,  True, False,  True,
        True, False,  True, False, False,  True, False, False,  True,
       False,  True,  True, False, False, False, False,  True,  True,
        True,  True,  True, False, False, False, False, False, False,
       False, False, False,  True, False,  True,  True, False, False,
       False, False,  True,  True, False, False, False, False,  True,
       False,  True, False, False,  True,  True,  True,  True, False,
       False,  True, False,  True,  True, False, False, False,  True,
       False, False, False, False,  True, False, False, False, False,
       False, False, False, False, False,  True, False,  True,  True,
       False, False,  True, False, False, False, False, False,  True,
        True,  True,  True,  True, False,  True, False, False,  True,
       False,  True, False, False, False,  True, False, False, False,
       False, False,  True, False,  True, False, False, False, False,
       False,  True,  True,  True, False,  True,  True,  True, False,
       False, False,  True, False, False, False,  True, False, False,
       False,  True,  True,  True, False,  True, False,  True,  True,
       False,  True,  True,  True, False, False, False, False,  True,
       False, False,  True,  True, False,  True,  True, False,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True, False,  True,  True,  True, False,
        True,  True,  True, False, False,  True, False, False, False,
        True, False, False, False,  True,  True, False,  True,  True,
        True,  True, False, False, False, False, False,  True,  True,
       False,  True,  True, False, False,  True,  True, False, False,
        True,  True, False, False, False,  True, False,  True, False,
        True, False, False,  True,  True,  True,  True,  True,  True,
       False,  True, False, False,  True,  True, False, False,  True,
       False,  True, False,  True,  True,  True,  True,  True,  True,
       False,  True,  True, False,  True,  True,  True,  True,  True,
        True, False,  True,  True, False, False,  True, False,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True, False, False, False, False,
        True, False,  True, False,  True, False, False,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True, False, False, False, False,  True,  True, False,  True,
```

```
False, False,  True, False, False,  True, False, False, False,
False, False,  True, False, False,  True, False,  True, False,
False,  True, False,  True, False,  True,  True, False, False,
 True,  True,  True, False,  True, False, False,  True,  True,
 True, False, False,  True,  True,  True,  True,  True, False,
False, False,  True,  True,  True, False, False,  True,  True,
 True,  True, False,  True,  True, False,  True,  True, False,
False,  True,  True, False, False,  True, False,  True, False,
False, False,  True, False, False, False, False,  True, False,
False,  True,  True,  True, False, False,  True, False,  True,
 True,  True, False,  True,  True, False,  True,  True,  True,
 True,  True,  True,  True,  True, False,  True,  True, False,
False, False,  True,  True, False, False, False, False, False,
False,  True,  True, False,  True, False,  True,  True,  True,
False,  True,  True, False, False,  True,  True,  True,  True,
False,  True, False, False,  True,  True,  True, False, False,
 True,  True,  True, False,  True, False,  True,  True,  True,
 True, False,  True,  True,  True,  True,  True,  True,  True,
False,  True, False, False,  True, False,  True, False, False,
False,  True,  True,  True, False,  True, False,  True, False,
 True,  True, False,  True,  True,  True,  True,  True, False,
False, False, False, False,  True, False,  True,  True, False,
 True,  True,  True, False, False,  True,  True, False, False,
 True, False, False, False, False,  True,  True,  True, False,
 True,  True, False, False,  True,  True,  True,  True,  True,
 True,  True, False, False,  True,  True, False,  True, False,
False,  True,  True,  True,  True, False, False, False,  True,
False, False, False, False,  True,  True, False, False,  True,
False, False, False,  True, False, False,  True,  True, False,
False,  True,  True,  True,  True, False,  True, False, False,
 True,  True,  True,  True, False,  True,  True,  True, False,
False, False, False, False, False,  True, False, False, False,
 True, False, False,  True,  True, False,  True, False, False,
False,  True,  True, False,  True,  True, False, False, False,
False,  True, False,  True,  True, False, False,  True, False,
False, False,  True,  True,  True, False,  True, False,  True,
 True,  True,  True, False, False,  True, False,  True, False,
False, False, False, False, False, False, False,  True, False,
 True,  True,  True, False,  True,  True,  True,  True, False,
 True, False, False,  True,  True,  True, False,  True,  True,
False,  True,  True,  True, False, False,  True, False,  True,
 True,  True,  True, False,  True, False,  True,  True,  True,
False,  True, False,  True,  True, False, False,  True,  True,
 True, False,  True, False,  True,  True,  True, False, False,
 True, False,  True, False, False, False, False, False,  True,
False,  True,  True,  True, False, False,  True, False, False,
 True, False, False, False,  True,  True,  True, False,  True,
 True, False,  True,  True,  True,  True, False, False, False,
 True, False, False,  True,  True, False,  True, False,  True,
 True, False,  True, False, False,  True,  True,  True,  True,
 True,  True, False,  True, False,  True,  True, False,  True,
False, False, False,  True,  True, False, False,  True, False,
 True, False,  True,  True,  True, False, False, False,  True,
False,  True, False,  True,  True, False, False,  True, False,
 True, False, False,  True,  True,  True, False,  True,  True,
 True, False, False,  True,  True, False,  True,  True, False,
False,  True,  True, False, False, False, False,  True, False,
False,  True, False,  True,  True, False, False, False, False,
 True, False, False,  True, False, False,  True, False, False,
 True, False,  True,  True, False,  True,  True,  True, False,
 True, False, False,  True, False, False,  True, False,  True,
False,  True, False,  True,  True, False, False, False, False,
False, False, False,  True, False,  True,  True,  True,  True,
False,  True,  True, False, False,  True,  True,  True, False,
 True,  True, False,  True,  True,  True, False,  True,  True,
```

```
        True,  True,  True, False, False,  True,  True, False, False,
        True, False,  True, False,  True, False, False,  True,  True,
       False, False,  True, False,  True, False, False,  True, False,
        True, False, False,  True,  True, False,  True,  True,  True,
        True, False,  True, False,  True,  True, False,  True,  True,
        True])
```

Exercise 6

The dataset remains stored as data. Because most wines in the dataset are classified as low quality, one very simple classification rule is to predict that all wines are of low quality. In this exercise, we determine the accuracy of this simple rule.

Instructions

- Use `accuracy()` to calculate how many wines in the dataset are of low quality. Do this by using 0 as the first argument, and `data["high_quality"]` as the second argument.
- Print your result.

In [27]:
```
accuracy(0,data['high_quality'])
```

Out[27]:
```
36.69385870401724
```

Exercise 7

In this exercise, we will use the kNN classifier from `scikit-learn` to predict the quality of wines in our dataset.

Instructions

- Use `knn.predict(numeric_data)` to predict which wines are high and low quality and store the result as `library_predictions` .
- Use `accuracy` to find the accuracy of your predictions, using `library_predictions` as the first argument and `data["high_quality"]` as the second argument.
- Print your answer. Is this prediction better than the simple classifier in Exercise 6?

In [31]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(numeric_data, data['high_quality'])
# Enter your code here!

library_predictions = knn.predict(numeric_data)

accuracy(library_predictions, data['high_quality'])
```

Out[31]:
```
84.14652916730799
```

Exercise 8

Unlike the `scikit-learn` function, our homemade kNN classifier does not take any shortcuts in calculating which neighbors are closest to each observation, so it is likely too slow to carry out on the whole dataset. In this exercise, we will select a subset of our data to use in our homemade kNN classifier.

Instructions

- Fix the random generator using `random.seed(123)` , and select 10 rows from the dataset using `random.sample(range(n_rows), 10)` . Store this selection as `selection` .

In [32]:
```
n_rows = data.shape[0]
# Enter your code here.

random.seed(123)
selection = random.sample(range(n_rows),10)
selection
```

Out[32]:
```
[428, 2192, 714, 6299, 3336, 2183, 882, 312, 3105, 4392]
```

Exercise 9

We are now ready to use our homemade kNN classifier and compare the accuracy of our results to the baseline.

Instructions

- For each predictor `p` in `predictors[selection]` , use `knn_predict(p, predictors[training_indices,:], outcomes[training_indices], k=5)` to predict the quality of each wine in the prediction set, and store these predictions as a np.array called `my_predictions` . Note that knn_predict is already defined as in the Case 3 videos.
- Using the `accuracy` function, compare these results to the selected rows from the `high_quality` variable in data using `my_predictions` as the first argument and `data.high_quality.iloc[selection]` as the second argument. Store these results as `percentage` .
- Print your answer.

In [36]:
```
predictors = np.array(numeric_data)
training_indices = [i for i in range(len(predictors)) if i not in selection]
outcomes = np.array(data["high_quality"])

#my_predictions = # Enter your code here!
#percentage = # Enter your code here!

for p in predictors[selection]:
    my_predictions = knn_predict(p, predictors[training_indices,:], outcomes[training_i
percentage = accuracy(my_predictions, data.high_quality.iloc[selection])

percentage
```

Out[36]:
```
60.0
```

In [ ]:
```
```