Final Python Project - A Classification Project

This project counts as your final exam in Python.

In this project we want to predict the quality of a white wine based NOT on its region, grape or production year but rather on some chemical characteristics. The sweetness of a wine comes from sugar whereas the tartness comes from acidic components. White wines have much lower levels of tannins than red wine so this is not included as a feature here.

Suppose you have a good wine palette and only want to select wines which have a quality rating of 7 or higher on a scale of 1 to 10. So we want to turn this into a binary classification problem where we predict whether the wine is acceptable (1=True) or not acceptable (0=False) to purchase. The given data set provides several chemical features (see description below) and a quality rating between 1 and 10.

Once we have cleaned the data, separated it into training and test sets and then scaled it, we compare the 3 classification algorithms Logistic Regression, ANN and kNN for predicting whether a wine is acceptable or not based upon our criteria of a rating of 7 or higher.

Before fitting the data we need to put it in a data frame with the desired target value (0 or 1) and then we need to clean the data. To do this we complete the following steps.

1. We first determine if there are any data instances which have a feature value of NaN. If there are, delete these instances unless there are too many. If the majority of NaN's occur within a single feature, drop that feature.
2. Data instances which are *outliers* may skew the outcome of our algorithms so we want to delete such data, if possible. By an outlier we mean a data instance which has a feature value that is far from the mean. For example, for the residual sugar feature the mean is around 6.4 with a standard deviation of around 5. However, the maximum value occurring for residual is almost 66 so we may consider this data instance an outlier; note that its distance from the mean (|6.4-66|=61.6) is around 12 times its standard deviation so clearly something is wrong. The criteria we use to identify data entries as outliers is somewhat arbitrary but typically one requires the difference of the feature value from the feature mean to be less than some multiple of the standard deviation. In our case we will assume that points which are more than 5 standard deviations from the mean are outliers. So our criteria for determining if a data instance with feature value x is an outlier is | x - mean | > 5 times the standard deviation where mean and standard deviation are for the feature.

However, we do not want to delete too much data. If a feature has a large number of outliers then it is probably better to just not use that feature. In our data set we won't have to delete too many data instances so don't worry about deleting a feature. It is important to realize that in some applications, such as climate, outliers may be important and should not be deleted.

Dataset - White wine quality

This is a dataset which is used to predict the quality of a white wine based upon several features. The data is in the file 'white_wine_quality.csv'. However the data is NOT separated by a comma but rather by a semi-colon. This can be handled by the usual pandas command for reading a csv file except that you need to add the argument **sep=';'**

The features are:

1. fixed acidity (tartaric acid - g / dm^3): most acids involved with wine are fixed or nonvolatile (do not evaporate readily)
2. volatile acidity (acetic acid - g / dm^3): the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste
3. citric acid (g / dm^3): found in small quantities, citric acid can add 'freshness' and flavor to wines
4. residual sugar (g / dm^3): the amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/liter and wines with greater than 45 grams/liter are considered sweet
5. chlorides (sodium chloride - g / dm^3): the amount of salt in the wine
6. free sulfur dioxide (mg / dm^3): prevents microbial growth and the oxidation of wine

7. total sulfur dioxide (mg / dm^3): in low concentrations, SO2 is mostly undetectable in wine, but at free SO2 concentrations over 50 ppm, SO2 becomes evident in the nose and taste of wine
8. density (g / cm^3): the density of wine is close to that of water depending on the percent alcohol and sugar content
9. pH: describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the pH scale
10. sulphates (potassium sulphate - g / dm3): a wine additive which can contribute to sulfur dioxide gas (S02) levels, wich acts as an antimicrobial and antioxidant
11. alcohol (% by volume): the percent alcohol content of the wine

The target/outcome in the data file is a number between 0 and 10 indicating the perceived **quality** of the wine. This data set may not contain wines from each quality rating.

What to do --

Step 0 Import libraries

Step 1 Load data and create a dataframe. Add a column to the data frame which gives the target value of 1 if the quality rating is 7 or higher and 0 otherwise; this answers the question "Is the wine acceptable?" based on our given criteria. Print out enough lines in your data frame to show that this column is correct. Hint: To create the extra column you can use the **.map()** method as we did many times with the iris data set or **.apply()** as we did in the diabetes dataset in Week 13

Step 2 Determine how many data instances are in the file; add a formatted print statement giving this value. Check to make sure there are no values containing NaN; if there are, delete those data entries.

Step 3 Decide how many different quality classes (based on the quality rating of 1 to 10) this data is divided into and how many wines are in each class. What quality rating does the majority of the wines in the dataset have? Give a frequency plot (using, for example, Seaborn's **countplot**) for each quality class and give the actual numbers in each quality rating. (The **groupby** method for a dataframe may be useful.)

Step 4 We now want to delete each data entry which has an outlier for any feature based upon the criteria we gave (i.e., the difference of a feature value from the feature mean is > five times the standard deviation). For each feature starting with 'fixed acidity' and going in feature order, loop through the data entries and see if the value is an outlier; if it is, delete that data entry. For each feature print out how many data instances you deleted. Remember that NumPy has built-in functions to determine the mean and standard deviation (**np.std(f)**); here **f** is a NumPy array containing feature information. How many data instances are remaining in the data frame after the data entries containing outliers are eliminated?

Step 5 Put the data into the correct form for the classifiers. Remember that the target array y is just 0 or 1 (acceptable or not). To get the feature data you can either use `vstack` for all 11 features or an easier way is to create a new data frame where you drop the columns that do not contain the feature data and then create a 2D NumPy array from this using, for example, `X = df_mod.values` where `df_mod` is your modified dataframe. Either way, X should be dimensioned by the number of data entries at the end of Step 4 by 11 features.

Step 6 Split the data into training and test sets with an 80-20 split using scikit-learn's function `train_test_learn` . Print out the number of data instances in each set.

Step 7 Scale the data as we have done previously using scikit-learn's `StandardScaler`

Step 8 Apply scikit-learn's logistic regression algorithm to solve this binary classification problem. Be sure to set the solver using `lbfgs` . Print out the percent of accuracy for predicting the test set.

Step 9 Apply scikit-learn's ANN algorithm to solve this binary classification problem. Print out the percent of accuracy for predicting the test set.

Step 10 Apply scikit-learn's kNN algorithm to solve this binary classification problem for values of k=1 to k=5. Determine which choice of k gives the best result. Print out the percent of accuracy for predicting the test set.

Step 11 Compare results from the 3 algorithms.

Extra Credit Apply 1 or more additional algorithms from scikit-learn's library to this binary classification problem and compare with the 3 algorithms in steps 8-10.

```python
import numpy as np
import matplotlib.pyplot as plt
from pandas import DataFrame
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
import pandas as pd
import seaborn as sns
df = pd.read_csv('white_wine_quality.csv', sep=';')
```

```python
# Step 1 - create dataframe
print(df)
```

```
4894           6.6             0.32         0.36            8.0      0.047
4895           6.5             0.24         0.19            1.2      0.041
4896           5.5             0.29         0.30            1.1      0.022
4897           6.0             0.21         0.38            0.8      0.020

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    45.0                 170.0  1.00100  3.00       0.45
1                    14.0                 132.0  0.99400  3.30       0.49
2                    30.0                  97.0  0.99510  3.26       0.44
3                    47.0                 186.0  0.99560  3.19       0.40
4                    47.0                 186.0  0.99560  3.19       0.40
...                   ...                   ...      ...   ...        ...
4893                 24.0                  92.0  0.99114  3.27       0.50
4894                 57.0                 168.0  0.99490  3.15       0.46
4895                 30.0                 111.0  0.99254  2.99       0.46
4896                 20.0                 110.0  0.98869  3.34       0.38
4897                 22.0                  98.0  0.98941  3.26       0.32

      alcohol  quality
0         8.8        6
1         9.5        6
2        10.1        6
3         9.9        6
4         9.9        6
...       ...      ...
4893     11.2        6
4894      9.6        5
4895      9.4        6
4896     12.8        7
4897     11.8        6
```

```python
# Step 1 continued.  Is the wine acceptable? 1 = True, 0 = False
df['acceptable'] = df['quality'].apply(lambda x: 1 if x >= 7 else 0)

print(df)
```

```
4894           6.6             0.32         0.36            8.0      0.047
4895           6.5             0.24         0.19            1.2      0.041
```

```
4                 47.0                186.0  0.99560  3.19       0.40
...                ...                  ...      ...   ...        ...
4893              24.0                 92.0  0.99114  3.27       0.50
4894              57.0                168.0  0.99490  3.15       0.46
4895              30.0                111.0  0.99254  2.99       0.46
4896              20.0                110.0  0.98869  3.34       0.38
4897              22.0                 98.0  0.98941  3.26       0.32

      alcohol  quality  acceptable
0         8.8        6           0
1         9.5        6           0
2        10.1        6           0
3         9.9        6           0
4         9.9        6           0
...       ...      ...         ...
4893     11.2        6           0
4894      9.6        5           0
4895      9.4        6           0
4896     12.8        7           1
4897     11.8        6           0
```

```python
# Step 2. Nan values?
print("Number of Nan values:", df.isnull().sum().sum())

df = df.dropna()

numi = len(df)

print(numi)
```

```
Number of Nan values: 0
4898
```

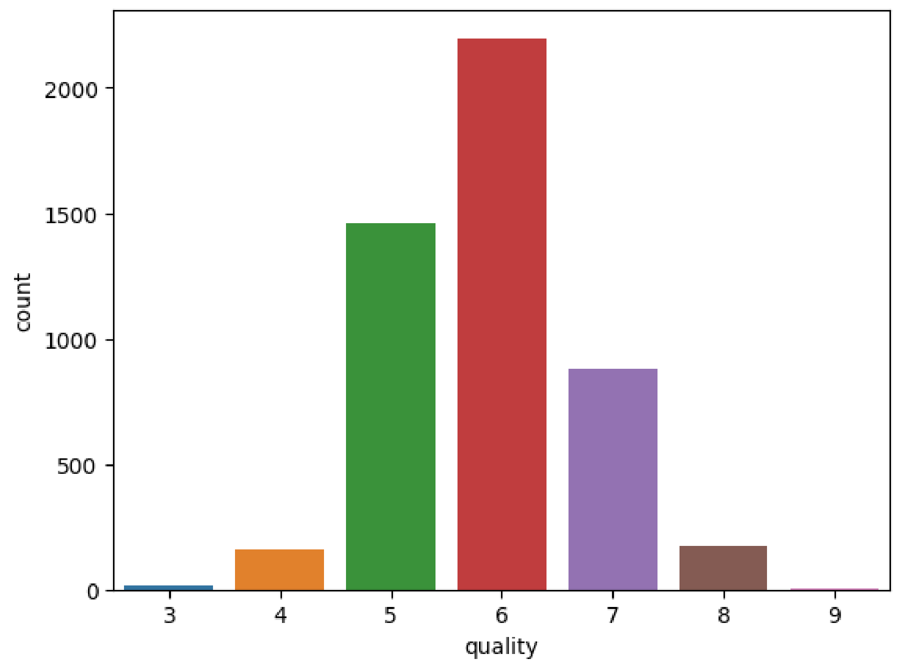
```python
# Step 3. Quality groupings and countplot
num_wines = df.groupby(['quality']).size().reset_index(name='counts')
print(num_wines)

modeq = num_wines.loc[num_wines['counts'].idxmax(), 'quality']
print("The mode quality rating is:", modeq)

sns.countplot(data=df, x='quality')
```

```
   quality  counts
0        3      20
1        4     163
2        5    1457
3        6    2198
4        7     880
5        8     175
6        9       5
The mode quality rating is: 6
```

```
<AxesSubplot:xlabel='quality', ylabel='count'>
```



```python
# Step 4. For each feature determine if any data instance is an outlier;
# if it is delete that data instance

 # Example follows for one feature
f = df['fixed acidity'].values

mean = np.mean(f)
std = np.std(f)
count = 0
factor = 5
for i in range(0,len(f)):
    z = ( f[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)

v = df['volatile acidity'].values

mean = np.mean(v)
std = np.std(v)
count = 0
factor = 5
for i in range(0,len(v)):
    z = ( v[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)

c = df['citric acid'].values

mean = np.mean(c)
std = np.std(c)
count = 0
factor = 5
for i in range(0,len(c)):
    z = ( c[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)

r = df['residual sugar'].values

mean = np.mean(r)
std = np.std(r)
count = 0
factor = 5
for i in range(0,len(r)):
    z = ( r[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)
```

```
8.705105871848145
5.860769568232091
number of data instances dropped is  2
6.21758267510449
5.67196082288019
6.267184661670337
7.209622406421397
6.4655926079337185
5.3247469169192705
6.812806513894635
8.152060151172464
5.0271349975241995
number of data instances dropped is  9
10.954598384631716
5.5012317050890545
5.4186049372171965
5.5012317050890545
5.5012317050890545
5.5012317050890545
7.4016473661418
5.5012317050890545
number of data instances dropped is  8
11.712597104042679
number of data instances dropped is  1
```

```
ch = df['chlorides'].values

mean = np.mean(ch)
std = np.std(ch)
count = 0
factor = 5
for i in range(0,len(ch)):
    z = ( ch[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)

fsd = df['free sulfur dioxide'].values

mean = np.mean(fsd)
std = np.std(fsd)
count = 0
factor = 5
for i in range(0,len(fsd)):
    z = ( fsd[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)

tsd = df['total sulfur dioxide'].values

mean = np.mean(tsd)
std = np.std(tsd)
count = 0
factor = 5
for i in range(0,len(tsd)):
    z = ( tsd[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)

d = df['density'].values

mean = np.mean(d)
std = np.std(d)
count = 0
factor = 5
for i in range(0,len(d)):
    z = ( d[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)
```

```
ph = df['pH'].values

mean = np.mean(ph)
std = np.std(ph)
count = 0
factor = 5
for i in range(0,len(ph)):
    z = ( ph[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)

s = df['sulphates'].values

mean = np.mean(s)
std = np.std(s)
count = 0
factor = 5
for i in range(0,len(s)):
    z = ( s[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)

aa = df['alcohol'].values

mean = np.mean(aa)
std = np.std(aa)
count = 0
factor = 5
for i in range(0,len(aa)):
    z = ( aa[i] - mean ) / std
    if z > factor:
        print(z)
        count = count + 1
        df = df.drop([i])
print("number of data instances dropped is ", count)
```

```
number of data instances dropped is  0
5.167319535096534
number of data instances dropped is  1
number of data instances dropped is  0
```

```
# Step 5. get data into correct form
```

```
df['acceptable'] = np.where(df['quality'] >= 7, 1, 0)

# Drop columns not containing feature data
df_mod = df.drop(['quality', 'acceptable'], axis=1)

# Convert modified dataframe to 2D numpy array
X = df_mod.values

# Get target array
y = df['acceptable'].values
```

```
# Step 6. Split data in test and trial sets with 80-20 split
from sklearn.model_selection import train_test_split

(X_train, X_test, target_train, target_test ) = train_test_split(df, y, test_size = .2)
```

```
# Step 7. Scale the data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train= sc.fit_transform(X_train)
X_test= sc.transform(X_test)
```

```
#  Step 8. Logistic Regression
Lr = LogisticRegression(solver ='lbfgs')
Lr.fit(X_train, target_train)

lr_predict = Lr.predict(X_test)

accuracy = (target_test == lr_predict).mean()
print("Logistic Regression Accuracy:", accuracy*100)
```

```
Logistic Regression Accuracy: 100.0
```

```
# Step 9. Create ANN classifier and train
clf = MLPClassifier(solver = 'lbfgs', activation='logistic')
clf.fit(X_train, target_train)
clf_predict = clf.predict(X_test)

accuracy = (target_test == clf_predict).mean()
print("ANN Accuracy:", accuracy*100)
```

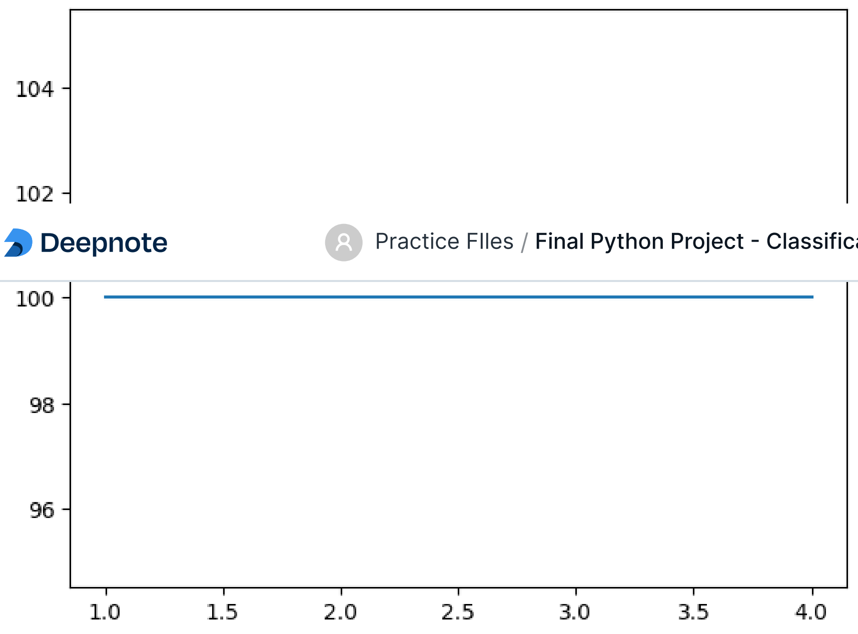
```
ANN Accuracy: 100.0
```

```
# Step 10. Create kNN classifier and see what value of k is best
from sklearn.neighbors import KNeighborsClassifier
k = np.arange(1,5)

accuracy=[]

for i in k:
    knn = KNeighborsClassifier (n_neighbors = i)
    knn.fit (X_train,target_train)
    accuracy.append(100 * knn.score(X_test, target_test))
print("kNN Accuracy:", accuracy)
plt.plot(k, accuracy)
print("k=1 gives absolute accuracy (100% accuracte)")
```

```
kNN Accuracy: [100.0, 100.0, 100.0, 100.0]
k=1 gives absolute accuracy (100% accuracte)
```



Compare your results

```
ı longer process on this problem, other binary classification problem might favor the K Nearest Neighbors Algorithm.")
```

```
the same accuracy of 100%. The Logistic Regression Algorthim and ANN Algortihm are similar in coding time and execution time making them mo
```

```
# Extra credit.
# Apply 1 or more additional algorithms from scikit-learn's library to this binary classification problem
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(random_state=0)

dtc.fit(X_train, target_train)

dtc_predict = dtc.predict(X_test)

accuracy =(target_test == dtc_predict).mean()
print("Decision Tree Classifier Accuracy:", accuracy*100)
print("Here using the Decision Tree Clasifier algorithm from scikit-learn's library, we get absolute accuracy in our pr
```

```
Decision Tree Classifier Accuracy: 100.0
Here using the Decision Tree Clasifier algorithm from scikit-learn's library, we get absolute accuracy in our predictions showing that it w
```