Visualizing and predicting the wine type based on wine characteristics

Data Set: Wine characteristics and quality (Red and White)

Project Objective: Learning a model to classify the type of wine.

Code Link:
https://github.com/Chhavi41/Red-White-wine-Classification-model/blob/main/ids.ipynb

Team Members:

18ucc041	Anirudh Singh
18ucs192	Chhavi Batra
18ucs007	Dhruv Pandya
18ucs060	Vatsal Mishra

Dataset Description:

In this project, we are working on a wine characteristic and quality dataset, link given below:
https://archive.ics.uci.edu/ml/datasets/Wine+Quality
The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. All the attributes are of datatype float.

Attribute Description:

- Fixed Acidity - This attribute **has distinct characteristics for each type of wine.**

- Volatile Acidity - This attribute is a measure of the wine's gaseous acids that contributes to the smell and taste of vinegar in wine.

- Citric Acid - **This attribute is most commonly used as an acid supplement during the fermentation process to help winemakers boost the acidity of their wine especially grapes grown in warmer climates.**

- Residual Sugar - This attribute measures the sweetness in the wine.

- Chlorides - Tells the content of chlorides in the wine, these have a key role in a potential salty taste of the wine.

- Free Sulfur Dioxide - Tells the amount of free sulfur dioxide in the wine.

- Total Sulfur Dioxide - Total Sulfur Dioxide presents the portion of SO2 that is free in the wine plus the portion that is bound to other chemicals in the wine such as aldehydes, pigments, or sugars.

- Density - This attribute describes the measurement of the density of the wine.

- pH - This attribute describes the pH of the wine(ranges from 3 to 4).

- Sulfates - Wines with more color (i.e., red wines) tend to need fewer sulfites than clear wines (i.e., white wines)

- Alcohol Output Variable (based on sensory data) - This attribute tells the alcohol content in the wine.

- Quality - Graded the wine on a scale that ranges from 0 (very bad) to 10 (excellent).

WORKING WITH THE DATASET

Plan of working:

The plan for approaching this dataset shall be as follows:

1. First, we shall import the required libraries. Here we would need numpy, then we would need pandas which has many inbuilt functions to help to perform various kinds of slices and selections. Matplotlib and seaborn will be used for visualizations and sklearn has important libraries related to applying algorithms.
2. After we have the dataset, we shall check if there are any missing values. If they are, then our task would be to eliminate them.
3. Then we analyze the variations in the attributes of both the datasets and eliminate those features that behave in a similar way for both datasets `white_wine_data_raw` and `red_wine_data_raw.`
4. After cleaning the data, we shall first see all the columns, and first, try to gain inference from the continuous data and think what all could be done.
5. After that, we shall start the visualization. The aim is to gain inferences from the dataset. The graphs we shall plot shall help us see various trends as to what factors does our target variable actually depends on and how other values relate among themselves.
6. In this step, we perform standardization of data to ensure uniformity in data and divide our data into train and test sets.
7. Then after this, we shall think about choosing the correct algorithms.
8. Finally, when we would have the model predictions, we shall look into their details and see how well our algorithms have performed and end this project by looking at the confusion matrix and other related things.

Problem Statement:

Train a model using the Wine characteristics and quality training data set, and then test the model against the testing dataset.

Importing Libraries:

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
```

Data Preprocessing Steps:

Reading the dataframe set, since columns are separated by ";"(semicolon) so we set delimiter = ';' and use the **`read_csv`** function to get the dataframe.

```
[ ]  red_wine_data_raw = pd.read_csv("winequality-red.csv",delimiter = ';')
     white_wine_data_raw = pd.read_csv("winequality-white.csv",delimiter = ';')
```

```
[ ] red_wine_data_raw[:10]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5

```
[ ] white_wine_data_raw[:10]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
5	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
6	6.2	0.32	0.16	7.0	0.045	30.0	136.0	0.9949	3.18	0.47	9.6	6
7	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
8	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
9	8.1	0.22	0.43	1.5	0.044	28.0	129.0	0.9938	3.22	0.45	11.0	6

The number of rows in the dataframe `white_wine_data_raw` is 4898 and that in `red_wine_data_raw` is 1599 as shown below, observed using the `count()` function on the dataframe.

```
[ ]  print(white_wine_data_raw.count())
```

```
fixed acidity           4898
volatile acidity        4898
citric acid             4898
residual sugar          4898
chlorides               4898
free sulfur dioxide     4898
total sulfur dioxide    4898
density                 4898
pH                      4898
sulphates               4898
alcohol                 4898
quality                 4898
dtype: int64
```

```
[ ]  print(red_wine_data_raw.count())
```

```
fixed acidity           1599
volatile acidity        1599
citric acid             1599
residual sugar          1599
chlorides               1599
free sulfur dioxide     1599
total sulfur dioxide    1599
density                 1599
pH                      1599
sulphates               1599
alcohol                 1599
quality                 1599
dtype: int64
```

Now, we have added a column ‘Type’ to both dataframe and assign the value 1/0(White/Red) accordingly.

```
[9]  white_wine_data_raw['Type'] = 1
     red_wine_data_raw['Type'] = 0
```

Checking for the null values

For cleaning the data, we first check for the null values, if any present in the data.

```
[ ] white_wine_data_raw.isnull().sum()
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Type                    0
dtype: int64
```

```
[ ] red_wine_data_raw.isnull().sum()
```

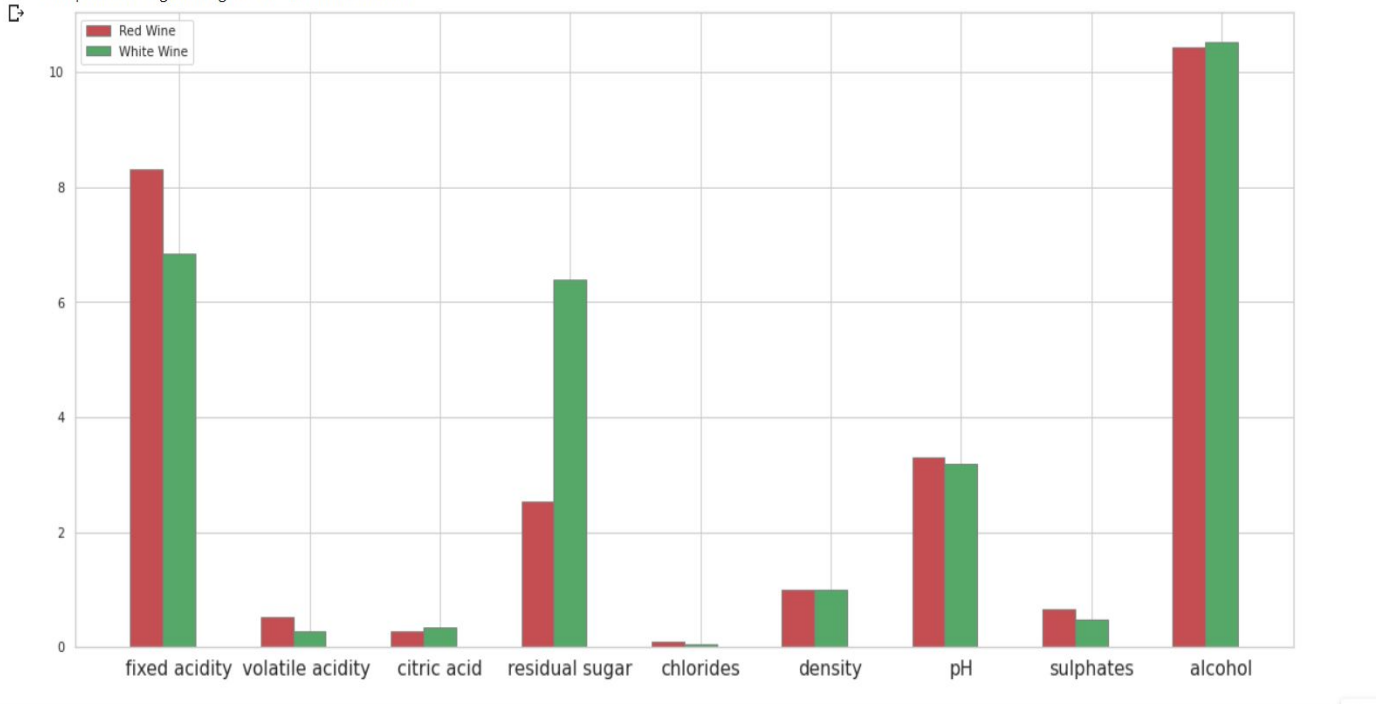
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Type                    0
dtype: int64
```

As we see above, there are no null values present in the dataset.

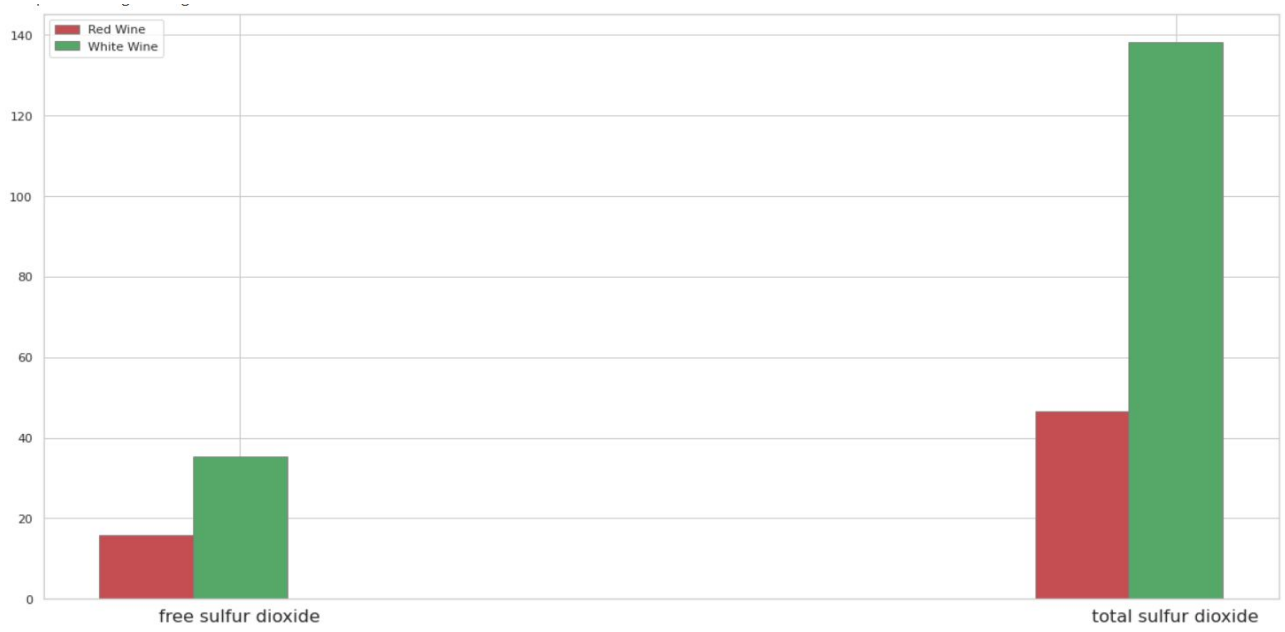
Analyzing the variation of various features in both type of wines

```
[ ] import numpy as np
    barWidth = 0.25
    fig = plt.subplots(figsize =(20, 9))
    red_m = red_wine_data_raw.mean()
    red_m = red_m.drop(labels=['free sulfur dioxide','total sulfur dioxide','Type','quality'])
    white_m = white_wine_data_raw.mean()
    white_m = white_m.drop(labels=['free sulfur dioxide','total sulfur dioxide','Type','quality'])
    br1 = np.arange(len(red_m))
    br2 = [x + barWidth for x in br1]
    p1 = plt.bar(br1, red_m, color ='r', width = barWidth,
            edgecolor ='grey', label ='RED WINE')
    p2 = plt.bar(br2, white_m, color ='g', width = barWidth,
            edgecolor ='grey', label ='WHITE WINE')
    plt.xticks([r + barWidth for r in range(len(red_m))],
               ['fixed acidity', 'volatile acidity ', 'citric acid', 'residual sugar', 'chlorides','density','pH','sulphates','alcohol'])
    plt.xticks(fontsize=16)
    plt.legend((p1, p2), ('Red Wine', 'White Wine'))
```

```
<matplotlib.legend.Legend at 0x7f38a4fc7da0>
```



```
[ ]  barWidth = 0.1
     fig = plt.subplots(figsize =(20, 10))
     red_m1 = red_wine_data_raw.mean()
     red_m = red_m1[5:7]
     white_m1 = white_wine_data_raw.mean()
     white_m = white_m1[5:7]
     br1 = np.arange(len(red_m))
     br2 = [x + barWidth for x in br1]
     p1 = plt.bar(br1, red_m, color ='r', width = barWidth,
             edgecolor ='grey', label ='RED WINE')
     p2 = plt.bar(br2, white_m, color ='g', width = barWidth,
             edgecolor ='grey', label ='WHITE WINE')
     plt.xticks([r + barWidth for r in range(len(red_m))],
                ['free sulfur dioxide','total sulfur dioxide'])
     plt.xticks(fontsize=16)
     plt.legend((p1, p2), ('Red Wine', 'White Wine'))
```



As we see above, there is a major difference between mean values of various attributes like free **`sulphur dioxide`**, **`total sulphur dioxide`**, **`fixed acidity`**, **`residual sugar`**, and **`volatile acidity`**. Thus we aim to use these attributes to train a model that can identify the type of wine whether white wine or red wine.

We use `plt.subplots()` function that returns the axes object, where the `pyplot` package is imported as `plt` from `matplotlib` library. So we make a boxplot using `sns.boxplot( white_wine_data_raw['alcohol'] ,orient='vertical')` function, where the `seaborn` library is imported as `sns`. Similarly, we plot a boxplot for `red_wine_data_raw['alcohol'],` and for the density columns of both the dataframes.

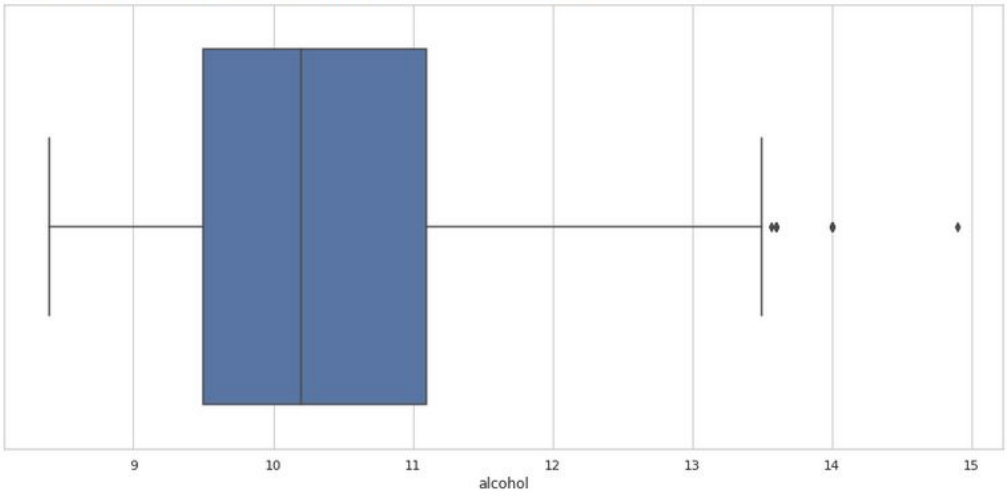
Observation: We observe that both alcohol and density have nearly the same statistical distribution of values for both `red_wine_data_raw` and `white_wine_data_raw` dataframes, so both `alcohol` and `density` can be dropped from the dataframes, which helps in reducing the complexity of the classification model.

Observing the Box Plots of alcohol distribution for both datasets

Red wine : alcohol

```
[ ] ax=plt.subplots(figsize=(15,7))
    ax = sns.boxplot(x=red_wine_data_raw['alcohol'], orient = 'vertical')
```

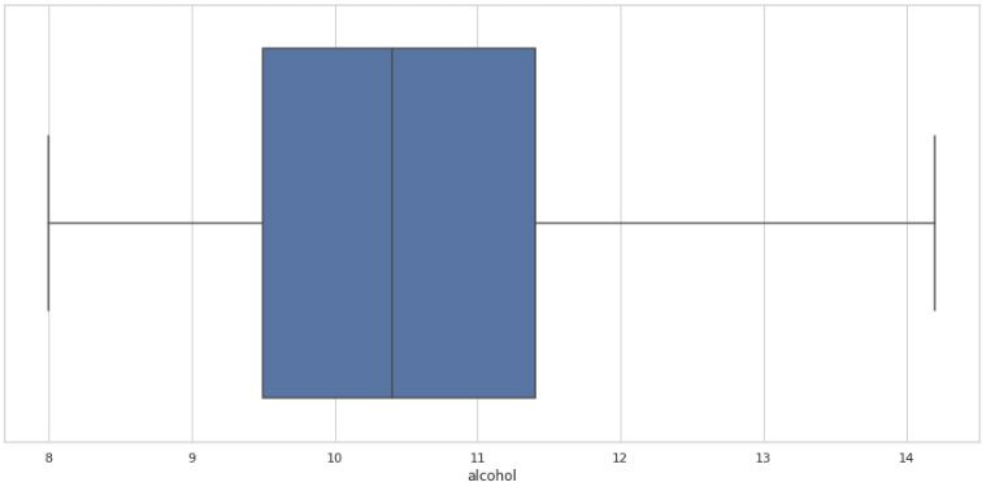
```
/usr/local/lib/python3.6/dist-packages/seaborn/_core.py:1303: UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



White wine: alcohol

```
[ ] ax=plt.subplots(figsize=(15,7))
    ax = sns.boxplot(x=white_wine_data_raw['alcohol'], orient = 'vertical')
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_core.py:1303: UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```

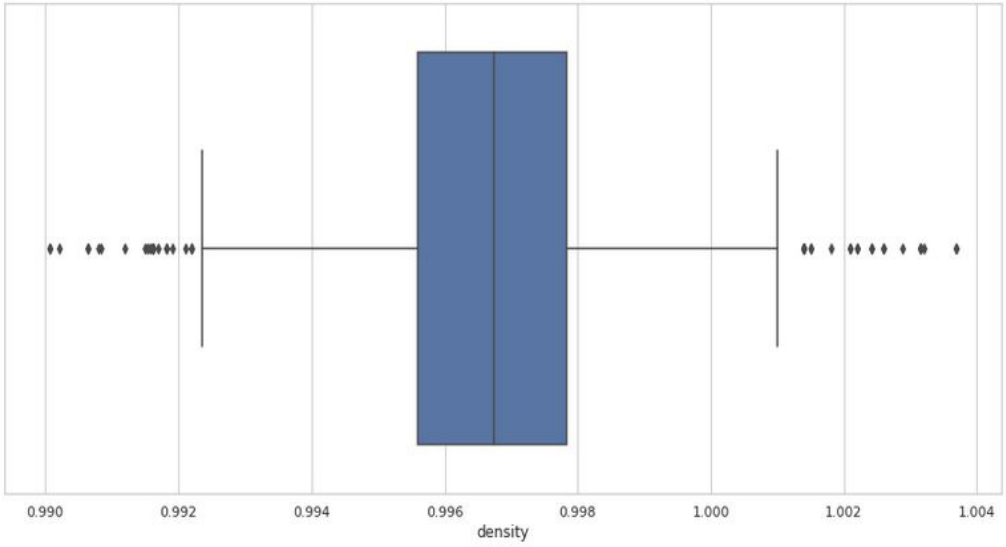


Observing the Box Plots of density distribution for both datasets

Red wine: density

```
[ ] ax=plt.subplots(figsize=(15,7))
    ax = sns.boxplot(x=red_wine_data_raw['density'], orient = 'vertical')
```

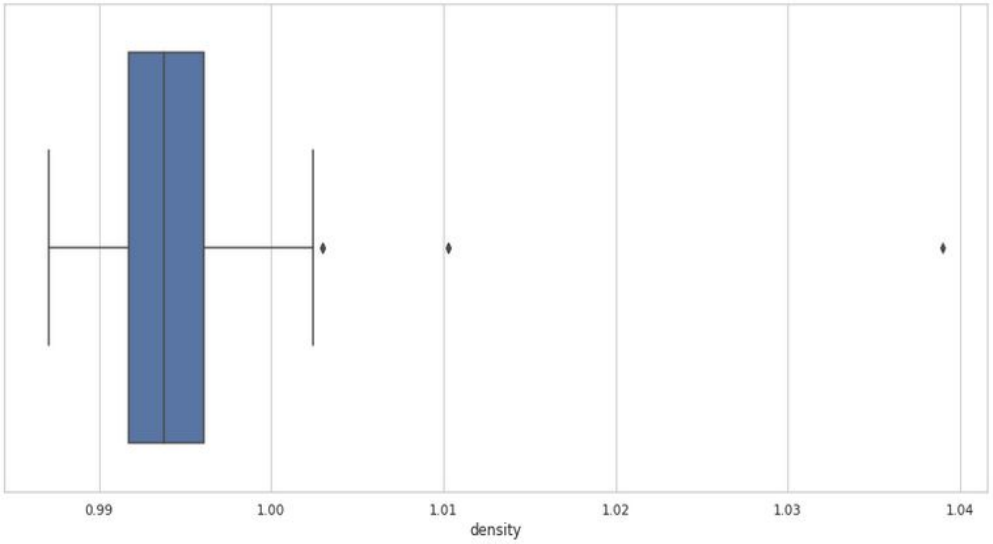
```
/usr/local/lib/python3.6/dist-packages/seaborn/_core.py:1303: UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



White wine: density

```
[ ] ax=plt.subplots(figsize=(15,7))
    ax = sns.boxplot(x=white_wine_data_raw['density'], orient = 'vertical')
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_core.py:1303: UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```

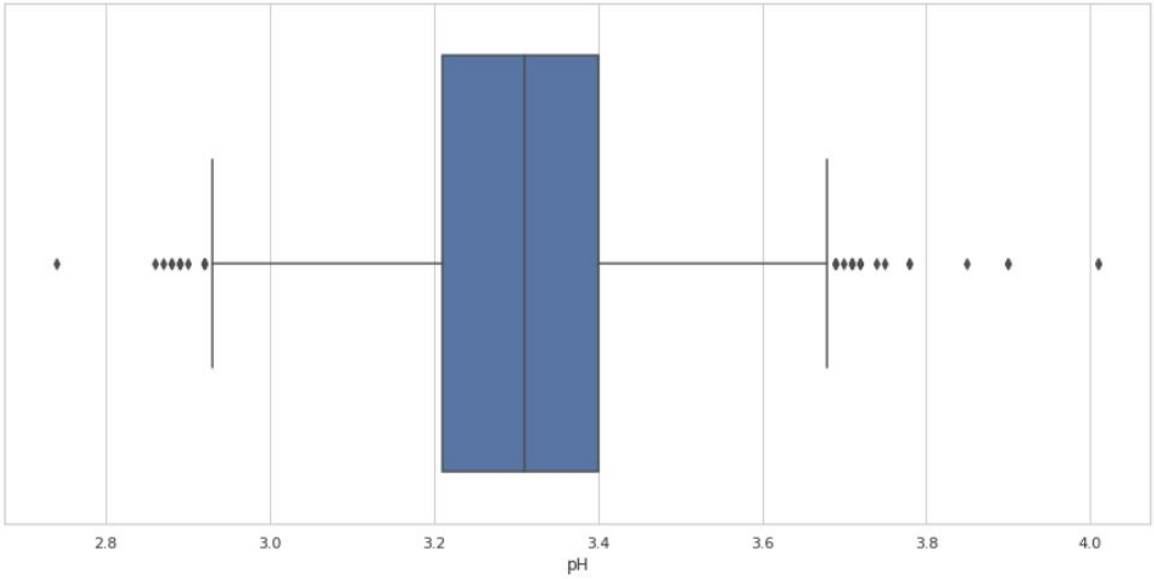


Observing the Box Plots of pH distribution for both datasets

Red wine: pH

```
sns.set_theme(style="whitegrid")
ax=plt.subplots(figsize=(15,7))
ax = sns.boxplot(x=red_wine_data_raw['pH'], orient = 'vertical')
```

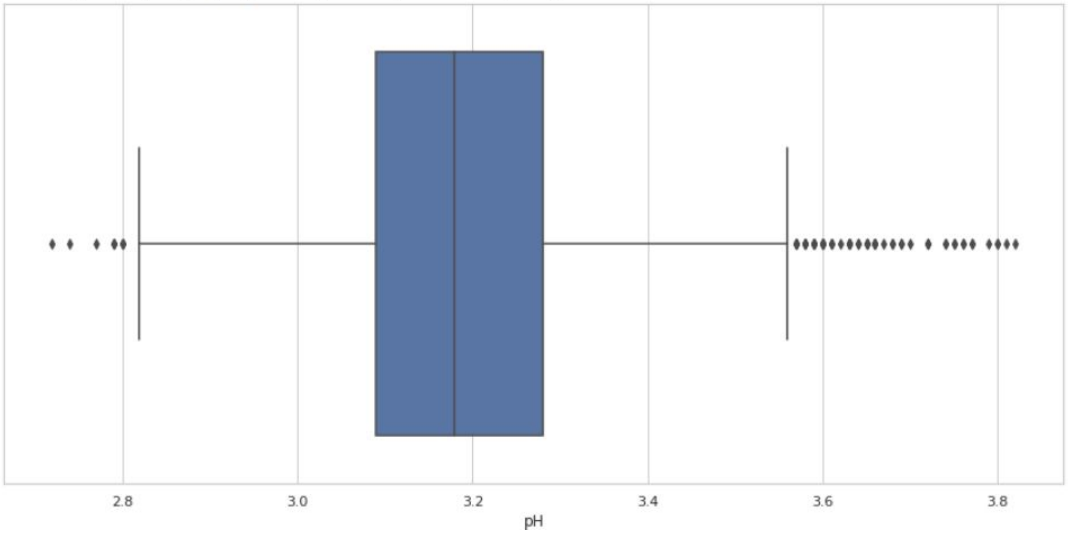
```
/usr/local/lib/python3.6/dist-packages/seaborn/_core.py:1303: UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



White wine: pH

```
ax=plt.subplots(figsize=(15,7))
ax = sns.boxplot(x=white_wine_data_raw['pH'], orient = 'vertical')
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_core.py:1303: UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



From the above box plots of the pH values, we observe that the difference in the values for pH are significant enough to consider them as an important parameter. Thus the pH column is not dropped from the datasets.

Combining both Datasets

Merging both red and white wine dataframe

```
merged_df=pd.concat([white_wine_data_raw,red_wine_data_raw])
merged_df[:10]
```

Now, we will drop the **`alcohol`**, **`density`**, and **`quality`** columns.

```
#Dropping alchohol,density and quality columns from the dataframe
merged_df=merged_df.drop(['alcohol','density','quality'],axis=1)
merged_df
```

Now the final dataset after removing alcohol, density, and quality columns from both datasets, and merging the red_wine_data_raw and white_wine_data_raw is as follows:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	pH	sulphates	Type
0	7.0	0.270	0.36	20.7	0.045	45.0	170.0	3.00	0.45	1
1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	3.30	0.49	1
2	8.1	0.280	0.40	6.9	0.050	30.0	97.0	3.26	0.44	1
3	7.2	0.230	0.32	8.5	0.058	47.0	186.0	3.19	0.40	1
4	7.2	0.230	0.32	8.5	0.058	47.0	186.0	3.19	0.40	1
...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	3.45	0.58	0
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	3.52	0.76	0
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	3.42	0.75	0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	3.57	0.71	0
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	3.39	0.66	0

6497 rows × 10 columns

Standardizing the data

What is the standardization of data?

Data standardization is the process of rescaling one or more attributes so that they have a mean value of 0 and a standard deviation of 1. Standardization assumes that your data has a Gaussian (bell curve) distribution.

Why perform data standardization?

Data standardization makes sure that data is internally consistent; i.e, each data type has the same content and format. Standardized values are useful for tracking data that isn't easy to compare otherwise. Thus standardization ensures uniformity.
For standardization of the data we'll use Scikit-learn library. As can be seen in the image below the, we have imported StandardScaler class from the preprocessing package of the Scikit-learn library.

```
from sklearn.preprocessing import StandardScaler
x=merged_df.drop(['Type'],axis=1)
y=merged_df['Type']
scaled_x=StandardScaler().fit_transform(x)
scaled_x
```

```
array([[-0.16608919, -0.42318303,  0.28468605, ...,  0.9599756 ,
        -1.35904886, -0.54617826],
       [-0.70607349, -0.24094936,  0.14704613, ...,  0.28761754,
         0.50691489, -0.27735097],
       [ 0.68245757, -0.36243847,  0.55996589, ..., -0.33165963,
         0.25811972, -0.61338508],
       ...,
       [-0.70607349,  1.03468634, -1.29817304, ..., -1.34019672,
         1.25330039,  1.47002637],
       [-1.01463595,  1.85473786, -1.366993  , ..., -1.26942219,
         2.18628226,  1.20119908],
       [-0.93749534, -0.1802048 ,  1.04170561, ..., -1.30480946,
         1.06670401,  0.86516498]])
```

As shown above we use `StandarScaler` to standardize the data. The result is stored in the variable named scaled_x which is stored in the memory as an array.

Applying Algorithms

Before deciding our algorithm, we have to split our dataset in train and test split. We use `train_test_split` to split the dataset into training and testing datasets. The division is done as per the default value i.e 25% of the dataset is taken as a testing set and 75% as a training set. The training set will help training the model and the testing set will help to analyze how well we have trained.

```
[553] from sklearn.model_selection import train_test_split
      X_train,X_test,y_train,y_test=train_test_split(scaled_x,y,random_state=0)
```

GridSearch Function:

```
[ ]  def GridSearch(cls,param,X_train=X_train,y_train=y_train):
         from sklearn.model_selection import GridSearchCV
         grid=GridSearchCV(cls,param_grid=param,cv=5)
         grid.fit(X_train,y_train)
         return grid
```

Why is Grid-search used?

Grid-search is used to find the optimal hyperparameters of a model which results in the most 'accurate' predictions.
A model **hyperparameter** is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyperparameter has to be set before the learning process begins.

k-Nearest Neighbors (kNN):

k-Nearest Neighbors is a supervised learning algorithm. It relies on the labelled input data to make predictions according to the requirement of the type of learning that is needed to be performed. KNN can be used to perform both classification or regression tasks.

According to our problem statement, we are trying to make a machine learning algorithm which classifies the classes into either the red-wine type or white-wine type.

The KNN algorithm tries to group different data on the basis of its neighbor data points, and it assumes that similar things exist in close proximity. In our project, we aim to classify whether the wine is red or white and this would closely depend on many factors and hence knn is a reasonable choice.

```
[ ]
    from sklearn.neighbors import KNeighborsClassifier
    knn=KNeighborsClassifier()
    param={'n_neighbors':[2,4,5,6,8,10,15,20,40,60,100]}
    knn_grid=GridSearch(knn,param)
    print(knn_grid.best_estimator_)
    print(knn_grid.best_params_)
    print(knn_grid.best_index_)
    knn_grid.score(X_test,y_test)
```

Output:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=15, p=2,
                     weights='uniform')
{'n_neighbors': 15}
6
0.9913846153846154
```

We have used the `GridSearch(knn,param)` function in the above code block, which runs for the various values of k as shown above in the variable `param`, and returns the most optimal value of k for which the model gives the best score.

Result:

The `GridSearch(knn,param)` returns the optimal value of k as k=15 and the score of the model is 0.9913, i.e the accuracy of the model is 99.13%, which is pretty great accurate.
From the result our choice of the classifier is justified.

Confusion matrix plot:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
y_pred_knn = knn_grid.predict(X_test)
c_m = confusion_matrix(y_test, y_pred_knn)
print('Confusion matrix plot')
sns.heatmap(confusion_matrix(y_test,y_pred_knn),annot=True, fmt="d" )
plt.xlabel('Predicted ')
plt.ylabel('True')
plt.show()
print("Classfication report")
print(classification_report(y_test, y_pred_knn))
print("Accuracy score")
print(accuracy_score(y_test,y_pred_knn))
```

Output:

Confusion matrix plot

Classfication report

	precision	recall	f1-score	support
0	0.98	0.98	0.98	395
1	0.99	0.99	0.99	1230
accuracy			0.99	1625
macro avg	0.99	0.99	0.99	1625
weighted avg	0.99	0.99	0.99	1625

Accuracy score

0.9913846153846154

In the above confusion matrix, it can be seen that for the class labelled as 0, the our model predicts 388 values correctly and only 7 values incorrectly. Similarly for the class labelled as 1, the correctly predicted values are 1223 and here also, only 7 values are predicted incorrectly by the model.

The output also contains information about the macro average which is the weighted mean average per label and weighted average which is calculated by averaging the support weighted mean per label.

This gives us a jist of the K Nearest Neighbour Algorithm Data. We can see the confusion matrix along with precision, recall, and f score.

References:

https://archive.ics.uci.edu/ml/datasets/Wine+Quality

https://medium.com/@erikgreenj/k-neighbors-classifier-with-gridsearchcv-basics-3c445ddeb657

https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e

https://stackoverflow.com/

https://www.geeksforgeeks.org/

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html