INTRODUCTION TO DATA SCIENCE

PROJECT REPORT

Wine Quality Prediction

Submitted To: Dr. Subrat Dash, Dr. Sakthi Balan

Submitted By:

Arishta Jain	18ucs002
Prachi Singhal	18ucs056
Akshita Goyal	18ucs077
Kashish Jain	18ucs115

Objective

Our main objective is to predict the wine quality using machine learning through Python programming language. A large dataset is considered and wine quality is modelled to analyze the quality of wine through different parameters like fixed acidity, volatile acidity, pH, etc. All these parameters are analyzed through machine learning algorithms like KNN, Logistic Regression and SVM which will allow us to rate the wine as good, bad or medium.

Dataset Description

The dataset contains chemical description of 6497 Portuguese "Vinho Verde" wines. There are 4897 entries for white wine and 1600 entries for red wine.

The dataset (winequalityN.csv) was taken from Kaggle, provided by UCI Machine Learning Repository.

Plan of Working

1. **Importing the necessary libraries:** We have used several libraries for the purpose of this project - pandas, numpy, matplotlib, seaborn and sklearn.
2. **Understanding the data (Data description):** In this step, we gather information about the dataset - for example, size of table, type and name of attributes, etc.
3. **Bivariate data analysis (Data visualization):** We check how some important attributes vary with their quality scores which will help us to identify which attributes have a strong correlation with quality scores and which do not with the help of graphical methods.
4. **Removing missing values from the dataset (Data pre-processing):** We identify and remove all the missing values from the dataset.
5. **Binarization (Data pre - processing):** The dataset contains two types of wine: 'red' and 'white'. We apply binarization on the type attribute of the dataset.
6. **Normalization (Data pre - processing):** We normalize the dataset to change the value of numeric columns in the dataset to a common scale.
7. **Dropping duplicates (Data pre - processing):** We drop all the duplicate rows from the dataset.
8. **Gaining inferences from the dataset (Data analysis):** With the help of box plot, scatter plot and correlation matrix, we try to understand how various attributes are dependent on each other, and make inferences about the data.

9. **Categorizing wine into good, medium and bad:** Based on the quality score, we classify the wine as belonging to one of the three categories: good, medium, bad.
10. **Knowing the ML classification algorithms:** In this step, we explore various ML classification algorithms which we can use for the purpose of training.
11. **Splitting dataset into training and testing:** We split the dataset into testing set (25%) and training set (75%).
12. **Training the algorithms:** We use the chosen ML classification algorithms to predict the wine quality in the testing dataset.
13. **Analyzing performance of various algorithms:** We compare which algorithms lead to the best performance and derive the conclusion.

Dataset Description

Importing libraries:

a. Import pandas as pd: for using data frames to store the dataset and perform computations.
b. Import numpy as np: for efficient numerical computation
c. Import matplotlib.pyplot as plot: for MATLAB like plotting framework
d. Import seaborn as sb: for drawing attractive and informative statistical graphics.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plot
        !conda install -c anaconda seaborn -y
        import seaborn as sb
```

Reading dataset "winequalityN.csv":

```
In [2]: df=pd.read_csv("winequalityN.csv") #reading the dataset - winequalityN
```

```
In [3]: data=df.values
```

Finding the type of dataset:

```
In [4]: type(data) #finding the type of dataset
Out[4]: numpy.ndarray
```

Calculating the count of rows and columns:

```
In [5]: data.shape #calculating number of rows and columns

Out[5]: (6497, 13)
```

Data set consists of **6497 rows and 13 columns**.

Finding the attributes of the dataset:

```
In [6]: df.columns #attributes of dataset
Out[6]: Index(['type', 'fixed acidity', 'volatile acidity', 'citric acid',
               'residual sugar', 'chlorides', 'free sulfur dioxide',
               'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
               'quality'],
              dtype='object')
```

The ‘type’ attribute classifies the wine as either ‘white’ or ‘red’ wine. The next 11 attributes are various chemical properties of the wine. The 12th attribute is the quality score given to the wine by an expert.

Checking for any null attributes in the dataset:

```
In [7]: df.info() #checking if any attribute is null

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   type                  6497 non-null   object
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

All the attributes are non-null as shown.

Finding the data type of various attributes:

```
In [8]: df.dtypes #returns the dtypes in DataFrame

Out[8]: type                   object
        fixed acidity         float64
        volatile acidity      float64
        citric acid           float64
        residual sugar        float64
        chlorides             float64
        free sulfur dioxide   float64
        total sulfur dioxide  float64
        density               float64
        pH                    float64
        sulphates             float64
        alcohol               float64
        quality                 int64
        dtype: object
```

The ‘type’ attribute which classifies the wine as being ‘white’ or ‘red’ is a string object. The other 11 attributes representing the chemical properties of the wine are ‘float’ values and the ‘quality’ attribute takes ‘int’ value.

Displaying the initial rows of the dataset:

```
In [9]: df.head(10) #displays the first ten rows of the dataset
```

Out[9]:

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

Summarizing data set distribution:

```
In [10]: df.describe() #it summarizes the central tendency, dispersion and shape of the dataset's distribution, excluding NaN values.
```

Out[10]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
count	6487.000000	6489.000000	6494.000000	6495.000000	6495.000000	6497.000000	6497.000000	6497.000000	6488.000000	6493.000000	6497.000000	6497.(
mean	7.216579	0.339691	0.318722	5.444326	0.056042	30.525319	115.744574	0.994697	3.218395	0.531215	10.491801	5.(
std	1.296750	0.164649	0.145265	4.758125	0.035036	17.749400	56.521855	0.002999	0.160748	0.148814	1.192712	0.(
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.(
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	5.(
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.(
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000	11.300000	6.(
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.(

The result describes row count, mean, standard deviation, minimum value, 25% quartile, median (50% quartile), 75% quartile and maximum value for all the attributes.

The value ranges of different variables vary significantly and may require normalization in later phases of the analysis.

Note that the density has very less standard deviation (0.002999) which means that it is a much more stable component across the different samples.

We can also observe that since the mean and median are very similar, there are no significant outliers.

Grouping rows based on quality attribute:

The quality attribute takes values from 3-9. The code below gives the count of rows for each of these values. The result is displayed in descending order of frequency. NA values are excluded from the result.

```
In [11]: df['quality'].value_counts() #it groups the rows based on the value of quality attribute (in descending order)
Out[11]: 6    2836
         5    2138
         7    1079
         4     216
         8     193
         3      30
         9       5
         Name: quality, dtype: int64
```

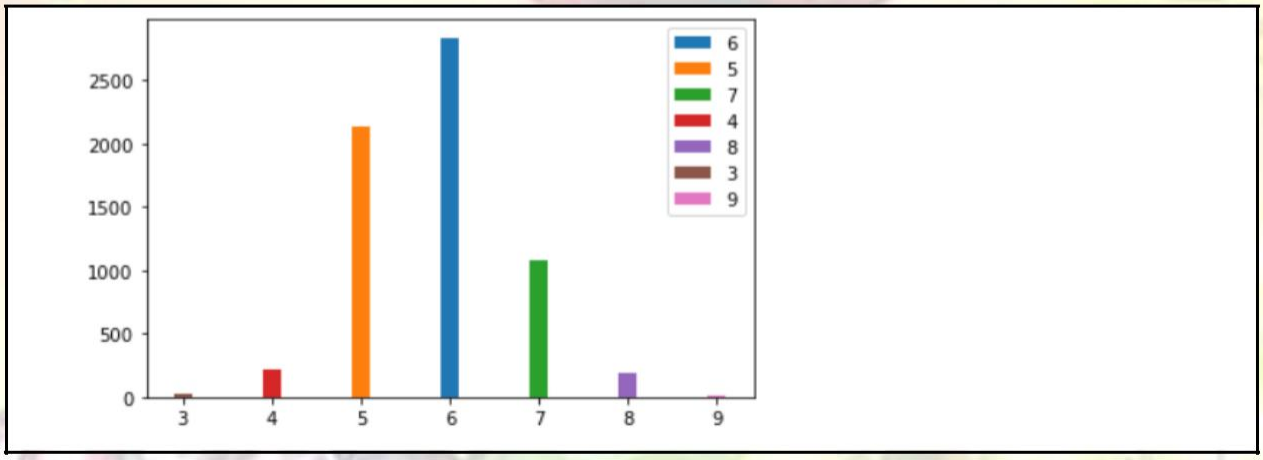
The code below gives the fraction of rows belonging to a particular quality score. The result is displayed in descending order of frequency. NA values are excluded from the result.

```
In [12]: df['quality'].value_counts(1) #it gives the percentage of rows belonging to a category (in descending order)
Out[12]: 6    0.436509
         5    0.329075
         7    0.166077
         4    0.033246
         8    0.029706
         3    0.004618
         9    0.000770
         Name: quality, dtype: float64
```

Data Visualization

Drawing bar graph representing various quality scores:

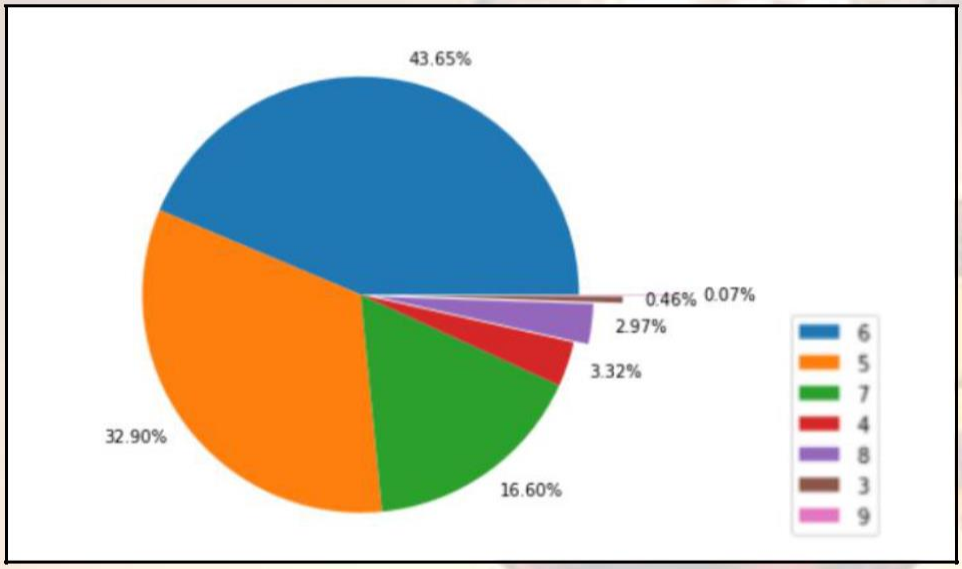
```
In [13]: ind1= np.array([0])
         count1= np.array([2836])
         ind2= np.array([0])
         count2= np.array([2138])
         ind3= np.array([0])
         count3= np.array([1079])
         ind4= np.array([0])
         count4= np.array([216])
         ind5= np.array([0])
         count5= np.array([193])
         ind6= np.array([0])
         count6= np.array([30])
         ind7= np.array([0])
         count7= np.array([5])
         plot.bar(ind1+6,count1,0.2,label='6')
         plot.bar(ind2+5,count2,0.2,label='5')
         plot.bar(ind3+7,count3,0.2,label='7')
         plot.bar(ind4+4,count4,0.2,label='4')
         plot.bar(ind5+8,count5,0.2,label='8')
         plot.bar(ind6+3,count6,0.2,label='3')
         plot.bar(ind7+9,count7,0.2,label='9')

         plot.legend()
         plot.show()
```



The minimum quality value is 3 and maximum is 8. The majority of wines have either 5 or 6 as their quality. About 1000 wines have their quality as 7, but the number is not as significant as 5 and 6. The rest are 3,4 or 8. The dataset majorly consists of medium quality wines (quality: 5 or 6). Bad quality wines are the least in number (quality: 3 or 4).

Drawing pie chart representing proportion of various quality scores:

```
In [14]: labels = ['43.65%','32.90%','16.60%','3.32%','2.97%','0.46%','0.07%']
         values = [2836,2138,1079,216,193,30,5]
         explodes=[0,0,0,0,0.1,0.3,0.7]
         plot.pie(values,explode=explodes,labels=labels,radius=1.5)
         plot.show()
```



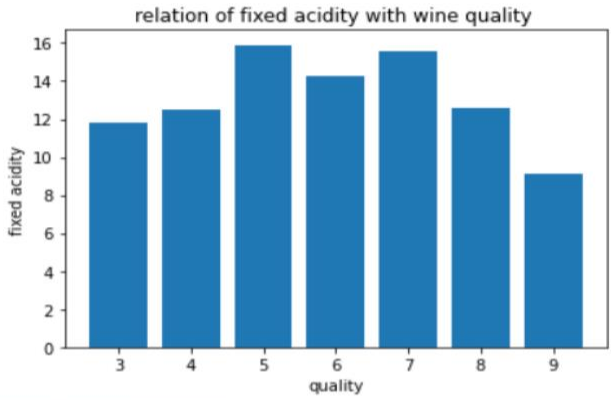
We can observe that medium quality wines (quality score: 5 or 6) makes the majority in the dataset, followed by good quality wines (quality score >=7).

Bivariate Data Analysis

It is the simplest form of analysis that determines the relationship between two variables. Here the purpose of analysis is to identify the relationship between quality scores and certain attributes. Some attributes show significant correlation with quality scores, while others don't. From bivariate data analysis we will be able to identify those parameters who have a significant impact on quality from the ones who do not.

Bar graph: Fixed acidity vs quality score:

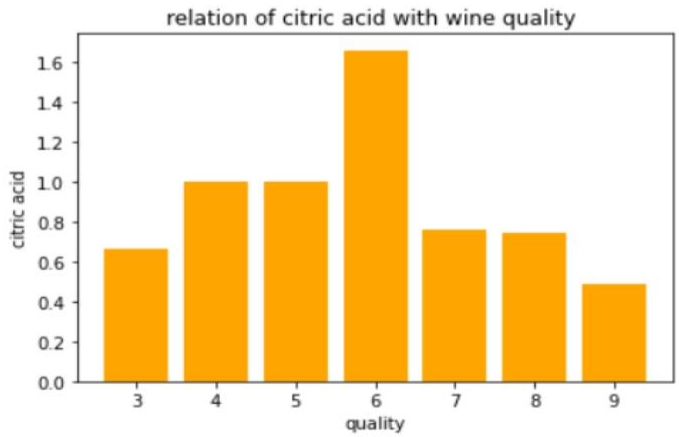
```
In [15]: #checking the variation of fixed acidity with different wine qualities
         plot.bar(df['quality'], df['fixed acidity'])
         plot.title('relation of fixed acidity with wine quality ')
         plot.xlabel('quality')
         plot.ylabel('fixed acidity')
         plot.show()
```



It can be observed that the value of fixed acidity does not vary a lot with quality, therefore we can say that they have a weak correlation.

Bar graph: Citric Acid vs quality score:

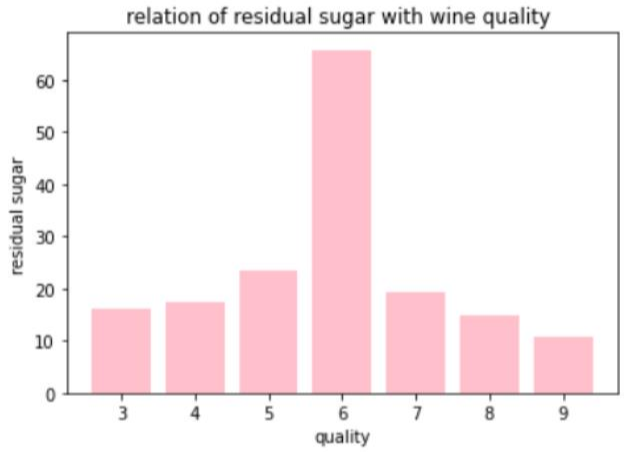
```
In [16]: #checking the variation of citric acid with different wine qualities
         plot.bar(df['quality'], df['citric acid'],color='orange')
         plot.title('relation of citric acid with wine quality ')
         plot.xlabel('quality')
         plot.ylabel('citric acid')
         plot.show()
```



We can observe that the graph almost remains constant for all quality scores, except for 6. Large quantities of citric acid is observed in medium quality wines, however it is less in both good and bad quality wines.

Bar graph: Residual sugar vs quality score:

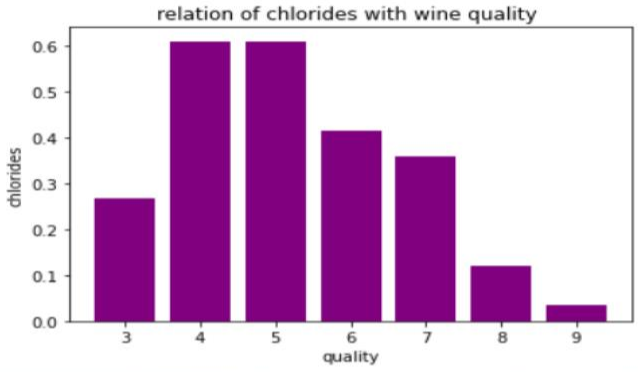
```
In [17]: #checking the variation of residual sugar with different wine qualities
         plot.bar(df['quality'], df['residual sugar'],color='pink')
         plot.title('relation of residual sugar with wine quality ')
         plot.xlabel('quality')
         plot.ylabel('residual sugar')
         plot.show()
```



It follows the same pattern as citric acid. The value of citric acid is at its peak in medium wines, and is less in both good and bad quality wines.

Bar graph: Chlorides vs quality score:

```
In [18]: #checking the variation of chlorides with different wine qualities
         plot.bar(df['quality'], df['chlorides'],color='purple')
         plot.title('relation of chlorides with wine quality ')
         plot.xlabel('quality')
         plot.ylabel('chlorides')
         plot.show()
```



It can be observed that if the quantity of chlorides is less, then the quality of the wine is poor.

Bar graph: Sulphates vs quality score:

```
In [19]: #checking the variation of suplhates with different wine qualities
         plot.bar(df['quality'], df['chlorides'],color='grey')
         plot.title('relation of sulphates with wine quality ')
         plot.xlabel('quality')
         plot.ylabel('sulphates')
         plot.show()
```

It can be observed that if the quantity of sulphates is less, then the quality of the wine is poor.

Data Pre-Processing

Data pre-processing refers to the steps applied to make the data more suitable for applying classification algorithms and getting inferences.

Removing NULL values from the dataset:

The code below gives the count of rows that contain NULL values for each of the attributes.

```
In [20]: df.isnull().sum()  #gives the count of rows for different attributes that have null values

Out[20]: type                    0
         fixed acidity          10
         volatile acidity        8
         citric acid             3
         residual sugar          2
         chlorides               2
         free sulfur dioxide     0
         total sulfur dioxide    0
         density                 0
         pH                      9
         sulphates               4
         alcohol                 0
         quality                 0
         dtype: int64
```

The attributes ‘fixed acidity’,’ volatile acidity’, ‘citric acid’, ‘residual sugar’, ‘chlorides’, ‘pH’ and ‘sulphates’ contain one or more NULL.

```
In [21]: df.dropna(subset=['fixed acidity'],how='any',inplace=True) #to remove all null values from fixed acidity
         df.dropna(subset=['volatile acidity'],how='any',inplace=True) #to remove all null values from volatile acidity
         df.dropna(subset=['citric acid'],how='any',inplace=True) #to remove all null values from citric acid
         df.dropna(subset=['residual sugar'],how='any',inplace=True) #to remove all null values from residual sugar
         df.dropna(subset=['chlorides'],how='any',inplace=True) #to remove all null values from clorides
         df.dropna(subset=['pH'],how='any',inplace=True) #to remove all null values from pH
         df.dropna(subset=['sulphates'],how='any',inplace=True) #to remove all null values from sulphates
```

Finally, we verify whether all the NULL values from these attributes have been removed.

```
In [22]: df.isnull().sum()  #gives the count of rows for different attributes that have null values

Out[22]: type                  0
         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
         dtype: int64
```

Binarization:

Binarization refers to changing the categorical attribute into binary variables. Since the 'type' attribute in our dataset is a categorial value, therefore, for further analysis, we convert it into 0 for 'red wine' and 1 for 'white wine'. It makes it easy for the classifier to read the data and also minimizes the memory usage. 'Type' attribute is modified as a result of binarization.

```
In [23]: data= df.copy()
         data.type=[1 if each=="white" else 0 for each in data.type]
         data
```

Out[23]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	1	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	1	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	1	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	1	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6491	0	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	6
6492	0	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
6494	0	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
6495	0	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
6496	0	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

6463 rows × 13 columns

Normalization:

It is the process of rescaling the numeric column values to a common scale i.e. from 0 to 1. Here normalization is required as the given data has different ranges of values for various attributes. This ensures that one attribute does not have dominance over others while calculating distances.

```
In [24]: def normalizeData(df):
             res = df.copy()
             for feature_name in df.columns:
                  max_value = df[feature_name].max()
                  min_value = df[feature_name].min()
                  res[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
             return res
         data1=normalizeData(data)
         data1
```

After normalization, our dataset is:

Out[24]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	1.0	0.264463	0.126667	0.216867	0.308282	0.059801	0.152778	0.377880	0.267785	0.217054	0.129213	0.115942	0.500000
1	1.0	0.206612	0.146667	0.204819	0.015337	0.066445	0.045139	0.290323	0.132832	0.449612	0.151685	0.217391	0.500000
2	1.0	0.355372	0.133333	0.240964	0.096626	0.068106	0.100694	0.209677	0.154039	0.418605	0.123596	0.304348	0.500000
3	1.0	0.280992	0.100000	0.192771	0.121166	0.081395	0.159722	0.414747	0.163678	0.364341	0.101124	0.275362	0.500000
4	1.0	0.280992	0.100000	0.192771	0.121166	0.081395	0.159722	0.414747	0.163678	0.364341	0.101124	0.275362	0.500000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6491	0.0	0.247934	0.360000	0.048193	0.019939	0.098007	0.093750	0.073733	0.181222	0.542636	0.337079	0.217391	0.500000
6492	0.0	0.198347	0.346667	0.048193	0.021472	0.134551	0.107639	0.087558	0.150183	0.565891	0.202247	0.362319	0.333333
6494	0.0	0.206612	0.286667	0.078313	0.026074	0.111296	0.097222	0.078341	0.166377	0.542636	0.297753	0.434783	0.500000
6495	0.0	0.173554	0.376667	0.072289	0.021472	0.109635	0.107639	0.087558	0.161172	0.658915	0.275281	0.318841	0.333333
6496	0.0	0.181818	0.153333	0.283133	0.046012	0.096346	0.059028	0.082949	0.161558	0.519380	0.247191	0.434783	0.500000

6463 rows × 13 columns

Dropping duplicate values from the table:

Dropping the duplicate values will lead to more accurate results. We can achieve it with the help of the code below.

```
In [27]: df.drop_duplicates(inplace=True)
```

Data analysis

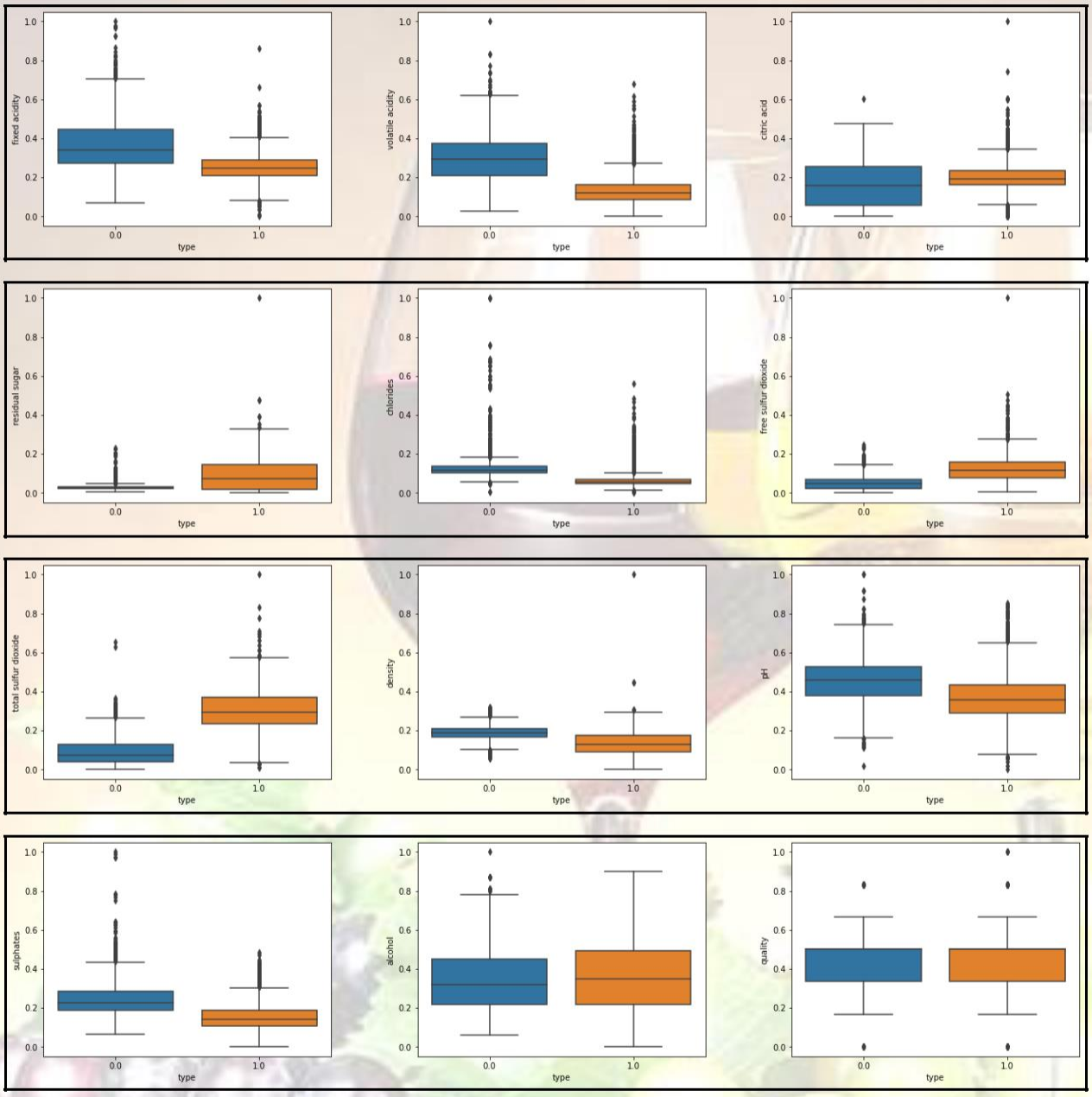
Box Plot:

Box plots are used to graphically depict the groups of numerical data through their quartiles and give a good indication of how the values are spread out. Box plots for 12 different attributes for both ‘red wine’ (type = 0) and ‘white wine’ (type = 1) are shown below.

```
In [25]: data=data1
```

```
In [26]: left = 0.1
         right = 3
         bottom = 0.1
         top = 1
         wspace = 0.3
         hspace = 0.2
         f, axes = plot.subplots(1, 3)
         plot.subplots_adjust(left, bottom, right, top, wspace, hspace)
         sb.boxplot('type', 'fixed acidity', data = data, ax=axes[0])
         sb.boxplot('type', 'volatile acidity', data = data, orient='v' , ax=axes[1])
         sb.boxplot('type', 'citric acid', data = data, orient='v' , ax=axes[2])
         f, axes = plot.subplots(1, 3)
         plot.subplots_adjust(left, bottom, right, top, wspace, hspace)
         sb.boxplot('type', 'residual sugar', data = data, orient='v' , ax=axes[0])
         sb.boxplot('type',  'chlorides', data = data, orient='v' , ax=axes[1])
         sb.boxplot('type', 'free sulfur dioxide', data = data, orient='v' , ax=axes[2])
         f, axes = plot.subplots(1, 3)
         plot.subplots_adjust(left, bottom, right, top, wspace, hspace)
         sb.boxplot('type',  'total sulfur dioxide', data = data, orient='v' , ax=axes[0])
         sb.boxplot('type', 'density', data = data, orient='v' , ax=axes[1])
         sb.boxplot('type', 'pH', data = data, orient='v' , ax=axes[2])

         f, axes = plot.subplots(1, 3)
         plot.subplots_adjust(left, bottom, right, top, wspace, hspace)
         sb.boxplot('type', 'sulphates', data = data, orient='v' , ax=axes[0])
         sb.boxplot('type','alcohol' , data = data, orient='v' , ax=axes[1])
         sb.boxplot('type',  'quality', data = data, orient='v' , ax=axes[2])
```

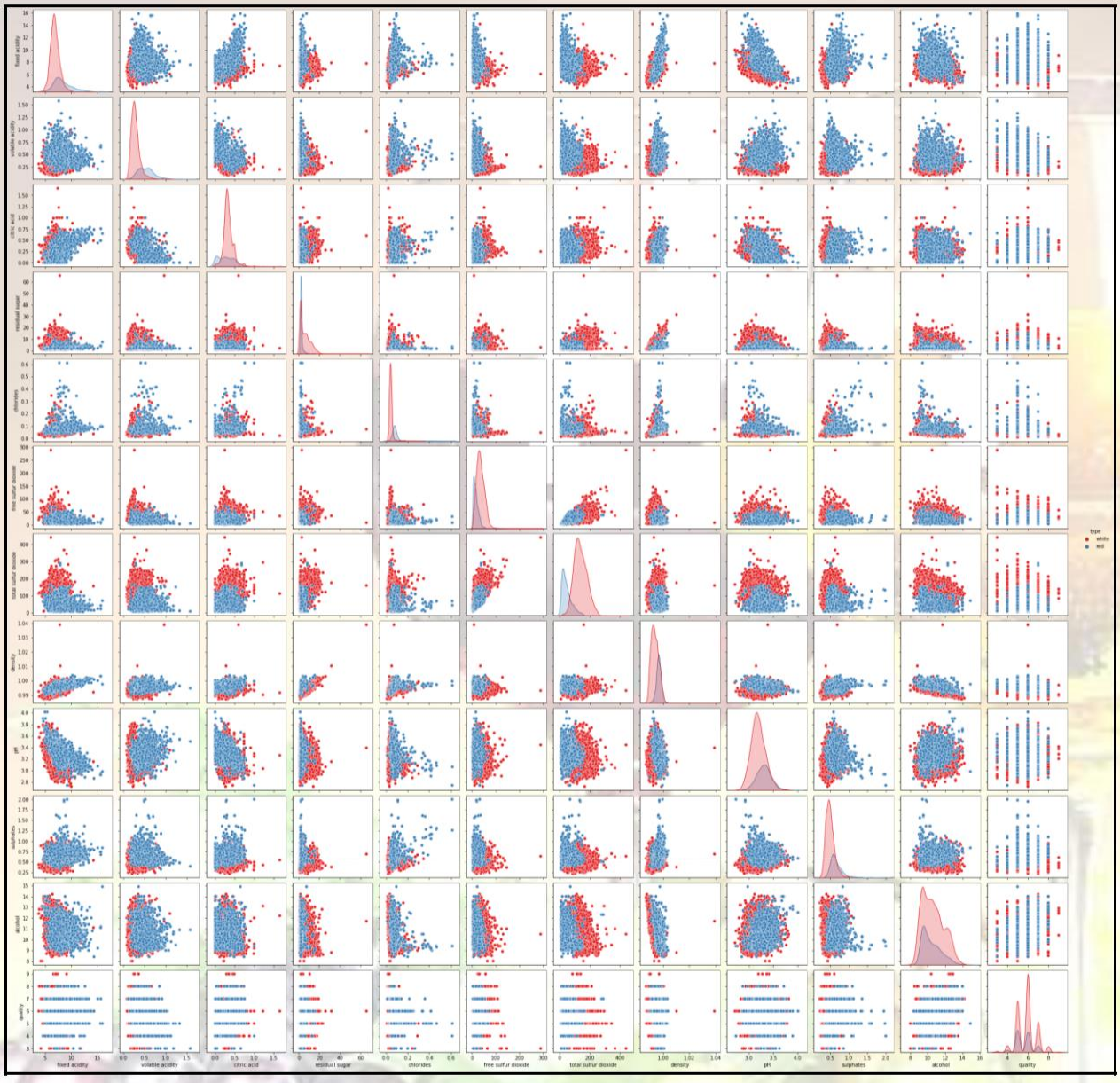
```
Out[26]: <AxesSubplot:xlabel='type', ylabel='quality'>
```



Pair plot

Pair plots are used to depict pairwise relationships in a dataset. This function creates a grid of axes such that each attribute is shared across the x axis across a single column and across the y axis across a single row.

```
In [28]: sb.pairplot(df, kind="scatter", hue="type", palette="Set1", diag_kind="auto",corner=False, dropna=False)
Out[28]: <seaborn.axisgrid.PairGrid at 0x7fc5f76e5a20>
```



Correlation Matrix

Correlation matrix is a covariance matrix in which the i-j position defines the correlation between the ith and jth parameter of the given dataset.

A negative correlation value implies an inverse relationship between the two variables and a positive correlation value implies a direct relationship between the two variables. If the correlation value is 0, then it means that the variables are unaffected by each other.

```
In [29]: corrMatrix=df.corr()
         plot.figure(figsize=(15,6))
         sb.heatmap(corrMatrix,annot=True)

Out[29]: <AxesSubplot:>
```

Observation:

Based on the data analysis, we can conclude that:

1. A negative correlation exists between the following pair of variables:
    a. Fixed acidity and pH
    b. Density and alcohol, etc.
2. A positive correlation exists between the following pair of variables:
    a. Density and fixed acidity
    b. Density and residual sugar
    c. Citric acid and fixed acidity, etc.

3. Sugar, pH and citric acid are factors that are not playing a big role in determining the quality of wine. But density, alcohol, total sulphur dioxide, etc are strongly related to the quality of the wine.

4. From the box plot, scatter plot and correlation matrix, we can conclude that alcohol and density are the most important factors for predicting wine quality, since they show a strong correlation with quality.

Label Encoding

Categorizing the wine into three categories (based on quality score):

The quality score for the wine ranges from 0 to 10. Here, we have defined three categories of wine based on their quality score.

a. Quality score (<=4): Bad quality
b. Quality score (5 or 6): Medium quality
c. Quality score (>=7): Good quality

```
In [30]: df['quality'] = df['quality'].map({3 : 'bad', 4 :'bad', 5: 'medium',
                                            6: 'medium', 7: 'good', 8: 'good' , 9: 'good'})
```

Count of wine in each category:

The below code gives the count of wine in each of the three categories - medium, good, bad, in decreasing order of their frequency.

```
In [31]: df['quality'].value_counts()

Out[31]: medium    4056
         good      1005
         bad        234
         Name: quality, dtype: int64
```

Applying label encoding:

The purpose of label encoding is to change the labelling of the data. After label encoding, the quality column now consists of values from 0 to 2.

a. Medium (quality = 2)
a. Good (quality = 1)
b. Bad (quality = 0)

```
In [32]: from sklearn.preprocessing import LabelEncoder

         le = LabelEncoder()

         df['quality'] = le.fit_transform(df['quality'])

         df['quality'].value_counts
```

```
Out[32]: <bound method IndexOpsMixin.value_counts of 0       2
         1       2
         2       2
         3       2
         6       2
                ..
         6490    2
         6491    2
         6492    2
         6495    2
         6496    2
         Name: quality, Length: 5295, dtype: int64>
```

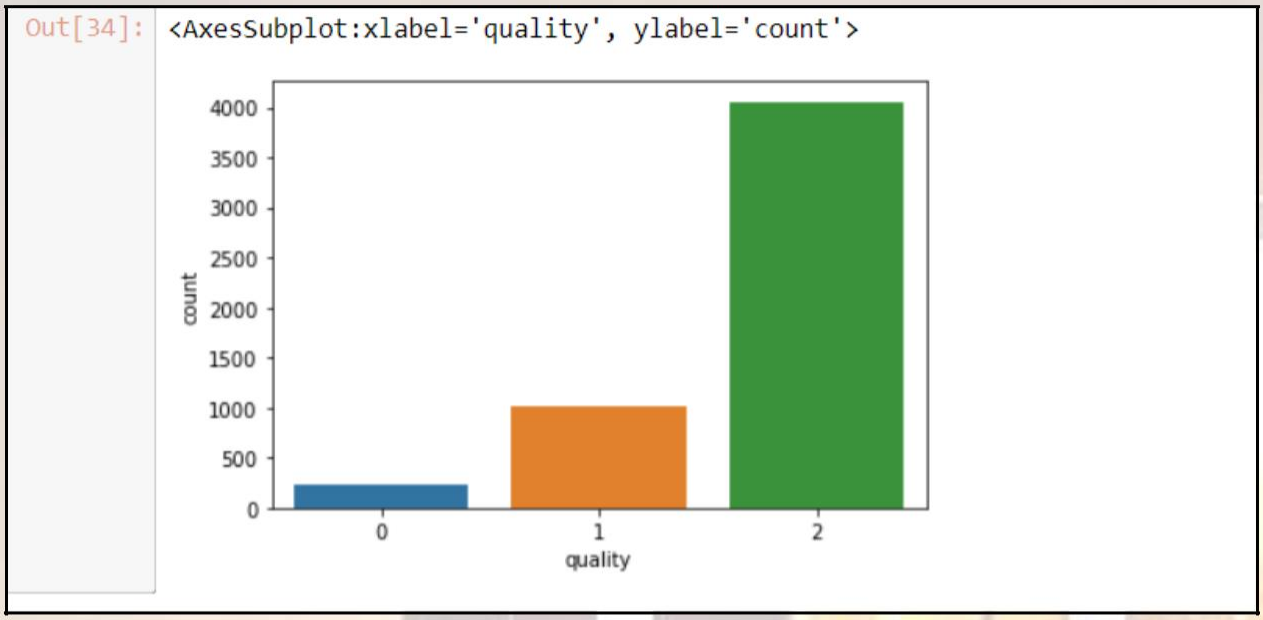
Displaying the table after label encoding:

```
In [33]: df.head(15)
```

Out[33]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.70	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	2
1	white	6.3	0.30	0.34	1.60	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	2
2	white	8.1	0.28	0.40	6.90	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	2
3	white	7.2	0.23	0.32	8.50	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	2
6	white	6.2	0.32	0.16	7.00	0.045	30.0	136.0	0.9949	3.18	0.47	9.6	2
9	white	8.1	0.22	0.43	1.50	0.044	28.0	129.0	0.9938	3.22	0.45	11.0	2
10	white	8.1	0.27	0.41	1.45	0.033	11.0	63.0	0.9908	2.99	0.56	12.0	2
11	white	8.6	0.23	0.40	4.20	0.035	17.0	109.0	0.9947	3.14	0.53	9.7	2
12	white	7.9	0.18	0.37	1.20	0.040	16.0	75.0	0.9920	3.18	0.63	10.8	2
13	white	6.6	0.16	0.40	1.50	0.044	48.0	143.0	0.9912	3.54	0.52	12.4	1
14	white	8.3	0.42	0.62	19.25	0.040	41.0	172.0	1.0002	2.98	0.67	9.7	2
15	white	6.6	0.17	0.38	1.50	0.032	28.0	112.0	0.9914	3.25	0.55	11.4	1
16	white	6.3	0.48	0.04	1.10	0.046	30.0	99.0	0.9928	3.24	0.36	9.6	2
18	white	7.4	0.34	0.42	1.10	0.033	17.0	171.0	0.9917	3.12	0.53	11.3	2
19	white	6.5	0.31	0.14	7.50	0.044	34.0	133.0	0.9955	3.22	0.50	9.5	2

```
In [34]: sb.countplot(df['quality'])
```



ML classification algorithms

1. Logistic Regression

Logistic Regression is a supervised classification algorithm. It is a regression model which predicts the probability that a given data entry belongs to the category numbered as "1".
It models the data using sigmoid function.

$$g(z) = \frac{1}{1+e^{-z}}$$

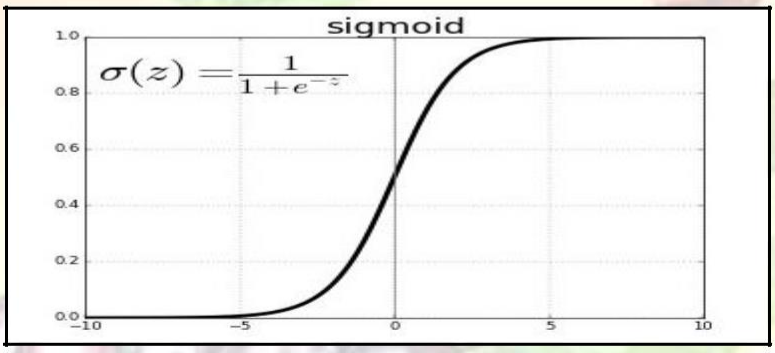


Fig.1. Plot of Sigmoid function

From the plot of sigmoid function, following predictions can be made-

- If the value of z goes to positive infinity then the predicted value of g(z) will become 1 and if it goes to negative infinity then the predicted value of g(z) will become 0.

- If the outcome of the sigmoid function is more than 0.5 then we classify that label as class 1 or positive class and if it is less than 0.5 then we can classify it to negative class or label as class 0.

2. K-Nearest Neighbours

KNN is a simple machine learning algorithm based on supervised learning techniques. The algorithm assumes the similarity between the new case and the available cases and then puts the new case in the category that is most similar to the available categories.

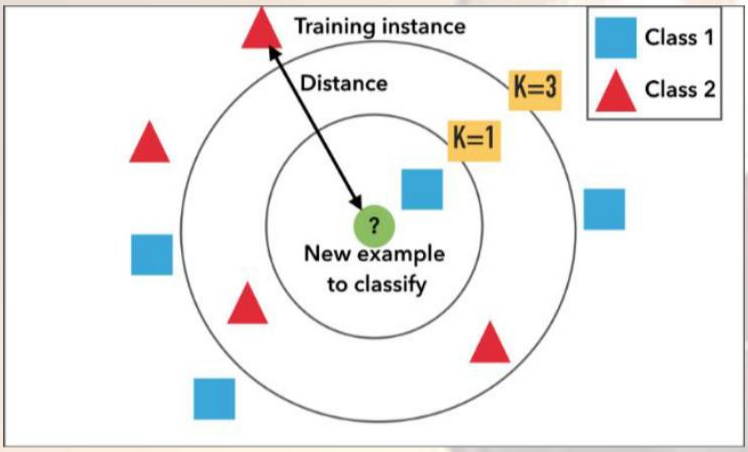


Fig.2. KNN Classifier

We should take care of some points while choosing the value of 'K' in the KNN algorithm.

- The most preferred value for K is 5.
- Very small values of K such that K=1, K=2 can be noisy and lead to the effects of outliers in the model.
- Large values of K are good but it may find some difficulties.

3. Support Vector Machine

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).

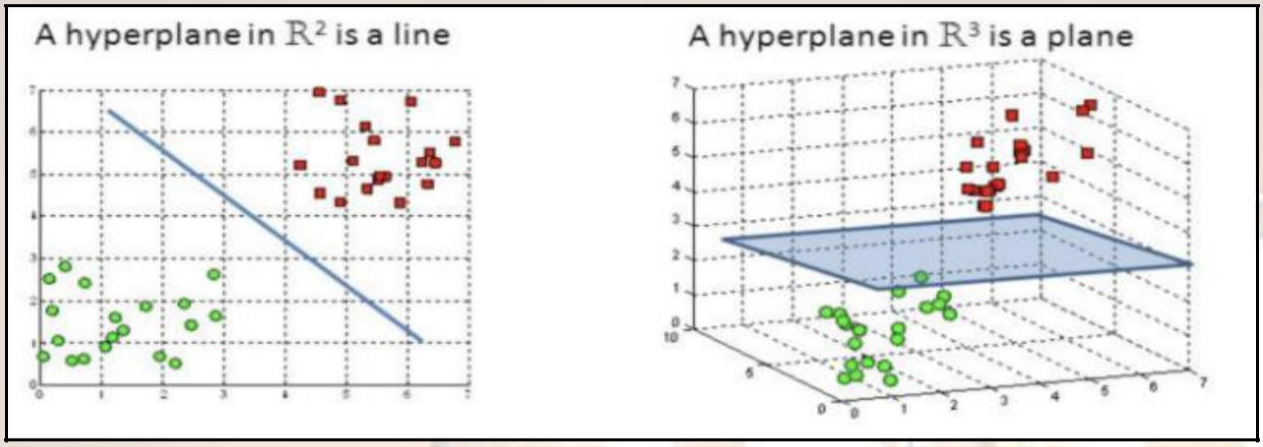


Fig.3. Hyperplanes in 2D and 3D featured space

Choosing the algorithms

We have chosen three classification algorithms for the purpose of the project:

1. **Logistic Regression**: We required an algorithm that gives us a value of 0,1 or 2 as the quality prediction for the wine. Logistic regression is used when the dependent variable takes numeric values. It is also used to understand the relationship between the dependent variable and one or more independent variables. Since, here we have many independent variables, logistic regression is appropriate to use.

2. **K nearest Neighbour**: This algorithm works on the assumption that things that appear in the proximity are similar. We wanted to categorize the wine quality as 0,1 or 2, and that would closely depend on various factors for the purpose of which we have used K nearest neighbour algorithm.

3. **Support Vector Machine**: SVM efficiently performs a non-linear classification using the kernel trick, implicitly mapping their inputs to a higher dimensional feature space. We use a simple SVM that is capable of discriminating between the three classes of wine - good, medium and bad according to the chemical attributes of the wine.

Splitting dataset

```
In [59]: X = data.drop(['quality'],axis=1)
         y = data['quality']
```

Splitting dataset into target and training set:

We split the dataset into two parts: training set (75%) and testing set (25%). The training set will be responsible for training the model and the testing set will help us know how well we have trained our model. We will apply the three ML classification algorithms - Logistic Regression, KNN and SVM to train our model.

```
In [61]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

Standardizing data for better accuracy:

```
In [62]: from sklearn.preprocessing import StandardScaler
         sc_X = StandardScaler()
         X_train = sc_X.fit_transform(X_train)
         X_test = sc_X.transform(X_test)
```

Model training and testing

Logistic Regression

```
In [63]: from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

```
In [64]: #LogisticRegression
         from sklearn.linear_model import LogisticRegression
         log_reg = LogisticRegression()
         log_reg.fit(X_train, y_train)
         pred_lr = log_reg.predict(X_test)
```

The accuracy score for logistic regression is:

```
In [65]: print("Accuracy Score: ",accuracy_score(y_test,pred_lr))

         Accuracy Score:  0.7719033232628398
```

The classification report gives us the summary of all the logistic regression data with precision, recall and f score.

```
In [66]: print("Classification Report: ")
         print(classification_report(y_test,pred_lr))

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.67      0.05      0.10        73
                    1       0.59      0.30      0.40       255
                    2       0.79      0.95      0.86       996

             accuracy                           0.77      1324
            macro avg       0.68      0.43      0.45      1324
         weighted avg       0.75      0.77      0.73      1324
```

The confusion matrix for logistic regression is:

```
In [67]: print("Confusion Matrix: ")
         confusion_matrix(y_test,pred_lr)

         Confusion Matrix:

Out[67]: array([[  4,   1,  68],
                [  0,  76, 179],
                [  2,  52, 942]])
```

K Nearest Neighbour Classifier Algorithm

```
In [68]: #KNeighborsClassifier
         from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier()
         knn.fit(X_train,y_train)
         pred_knn=knn.predict(X_test)
```

The accuracy score for KNN is:

```
In [69]: print("Accuracy Score: ",accuracy_score(y_test, pred_knn))

         Accuracy Score:  0.7711480362537765
```

The classification report gives us the summary of all the KNN data with precision, recall and f score.

In [70]:
```
print("Classification Report: ")
print(classification_report(y_test,pred_knn))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.41      0.10      0.16        73
           1       0.56      0.44      0.50       255
           2       0.81      0.90      0.86       996

    accuracy                           0.77      1324
   macro avg       0.60      0.48      0.50      1324
weighted avg       0.74      0.77      0.75      1324
```

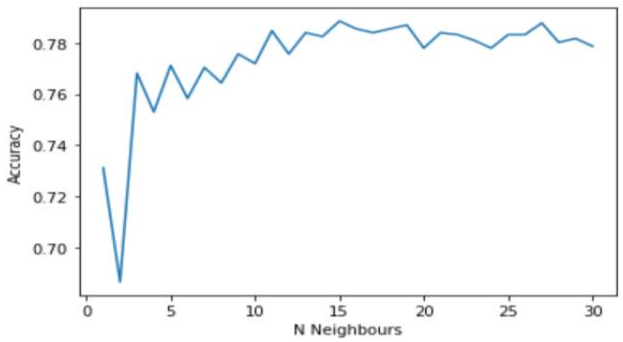
In [71]:
```
print("Confusion Matrix: ")
confusion_matrix(y_test,pred_knn)
```

```
Confusion Matrix:
```

Out[71]:
```
array([[  7,   3,  63],
       [  0, 113, 142],
       [ 10,  85, 901]])
```

In [72]:
```
accuracy_arr = []
for i in range(1,31):
    from sklearn.neighbors import KNeighborsClassifier
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train,y_train)
    pred_knn=knn.predict(X_test)
    accuracy_arr.append((i,accuracy_score(y_test, pred_knn)))
```

```
In [73]: import matplotlib.pyplot as plt
         x = list(range(0,31))
         x, y = zip(*accuracy_arr)
         plt.plot(x, y, label = "Accuracy vs Neighbor count")
         plt.xlabel("N Neighbours")
         plt.ylabel("Accuracy")

Out[73]: Text(0, 0.5, 'Accuracy')
```




```
In [74]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors = 26)
         knn.fit(X_train,y_train)
         pred_knn=knn.predict(X_test)

In [75]: print("Accuracy Score: ",accuracy_score(y_test, pred_knn))

         Accuracy Score:  0.7832326283987915
```

The classification report gives us the summary of all the KNN data with precision, recall and f score.

```
In [76]: print("Classification Report: ")
         print(classification_report(y_test,pred_knn))

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.00      0.00      0.00        73
                    1       0.64      0.36      0.46       255
                    2       0.80      0.95      0.87       996

             accuracy                           0.78      1324
            macro avg       0.48      0.44      0.44      1324
         weighted avg       0.73      0.78      0.74      1324
```

```
In [77]: print("Confusion Matrix: ")
         confusion_matrix(y_test,pred_knn)

         Confusion Matrix:

Out[77]: array([[  0,   0,  73],
                [  0,  93, 162],
                [  0,  52, 944]])
```

Support Vector Machine

```
In [78]: #SupportVectorMachine
         from sklearn.svm import SVC
         svc = SVC()
         svc.fit(X_train,y_train)
         pred_svc =svc.predict(X_test)
```

The accuracy score for SVM is:

```
In [79]: print("Accuracy Score: ",accuracy_score(y_test,pred_svc))

         Accuracy Score:  0.783987915407855
```

The classification report gives us the summary of all the SVM data with precision, recall and f score.

```
In [80]: print("Classification Report: ")
         print()
         print(classification_report(y_test,pred_svc))

         Classification Report:

                       precision    recall  f1-score   support

                    0       0.00      0.00      0.00        73
                    1       0.69      0.30      0.42       255
                    2       0.79      0.96      0.87       996

             accuracy                           0.78      1324
            macro avg       0.49      0.42      0.43      1324
         weighted avg       0.73      0.78      0.74      1324
```

The confusion matrix for SVM is:

```
In [81]: print("Confusion Matrix: ")
         confusion_matrix(y_test,pred_svc)

         Confusion Matrix:

Out[81]: array([[  0,   0,  73],
                [  0,  77, 178],
                [  0,  35, 961]])
```

Conclusion

1. In predicting the quality of wine, we do not need to account for all the features, rather only those features that have a significant impact on quality. Density and alcohol were very important for predicting the wine quality.

2. To make predictions about the wine quality, we trained three models- namely, Logistic Regression, KNN and Support Vector Machine model. By comparing the accuracy of the three models, we can conclude that SVM provides a marginal improvement over KNN. Also, both KNN and SVM give more accurate results than logistic regression.

Future Improvement

Good wine is a little more than a perfect combination of various chemical components. If more data can be collected on good, bad and medium quality wines, then the quality of analysis can be improved. Currently, in our dataset, maximum wines belong to the medium category, hence more number of wines belonging to the good and bad category will lead to improved results. We will be able to conclude if there exists a strong correlation between wine quality and the chemical attributes. Therefore, a larger dataset and exploring more ML classification algorithms will lead to improved results. Also, the results could be enhanced by studying more variables involved and their inter relationships.

References

[1] Data Set: https://www.kaggle.com/rajyellow46/wine-quality

[2] Official Pandas Documentation: https://pandas.pydata.org/docs/

[3] Official Seaborn Documentation: https://seaborn.pydata.org/

[4] For Machine Learning Algorithms: https://www.kaggle.com/, https://scikitlearn.org/stable/

[5] For resolving various errors: https://stackoverflow.com/

[6] For conceptual knowledge: Tom Michell, Machine Learning, McGraw Hill, Class Notes and Slides http://www.cs.cmu.edu/~tom/mlbook-chapter-slides.html