CSM-423

Name: IMMIDICHETTY REDDY SWETHAK

Reg No: 12008336

Section: K20UP

Roll No: RK20PMA03

Group: B.Tech(CSE)

MACHINE LEARNING ANALYSIS REPORT

My Dataset

Wine Quality Prediction

INTRODUCTION

The aim of this project is to predict the quality of wine on a scale of 0–10 given a set of features as inputs. The dataset used is Wine Quality Data set from UCI Machine Learning Repository. Input variables are fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulphur dioxide, total sulphur dioxide, density, pH, sulphates, alcohol. And the output variable is quality (score between 0 and 10).We are dealing only with red wine. We have quality being one of these values: [3, 4, 5, 6, 7, 8]. The higher the value the better the quality. In this project we will treat each class of the wine separately and their aim is to be able and find decision boundaries that work well for new unseen data. These are the classifiers.

CONTENT

Dataset/Source: Kaggle https://www.kaggle.com/uciml/red-wine-qua lity-cortez-et-al-2009

Structured/Unstructured data: Structured Data in CSV format.

Dataset Description: The two datasets are related to red wine of the Portuguese "Vinho Verde" wine. For more details, consult: [Web Link] or the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

1)fixed acidity

2) volatile acidity

3) citric acid

4) residual sugar

5) chlorides

6)free sulfur dioxide

7)total sulfur dioxide

8)density

9)pH

10) sulphates

11) alcohol Output variable (based on sensory data)

12)quality (score between 0 and 10)

DOMAIN

Wineries and vineyards can use wine quality prediction models to monitor and control the quality of their wines during the production process. By analyzing various chemical and sensory attributes, winemakers can make informed decisions about blending, aging, and bottling to achieve desired wine quality standards.

In real-life applications, wine quality prediction offers valuable insights to both professionals in the wine industry and consumers. It can enhance the winemaking process, improve customer experiences, and enable more informed decisions about wine purchases and investments. Additionally, it contributes to the overall appreciation and enjoyment of wine by providing data-driven guidance and recommendations.

DETAILS

This datasets is related to red variants of the Portuguese "Vinho Verde" wine. For more details, consult the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

Why?

Quality Control in Winemaking: Wine quality prediction models help winemakers ensure that the wines they produce meet specific quality standards. This is crucial for maintaining the reputation of the winery and producing consistent, high-quality wines.

Wine Industry Competitiveness: In a highly competitive wine industry, having the ability to predict and control wine quality gives wineries a competitive edge. It allows them to differentiate their products in a crowded market.

Consumer Guidance: Wine quality prediction benefits consumers by providing them with recommendations and guidance for selecting wines that align with their preferences. This enhances the overall wine-buying experience.

Question/Plans

- Import necessary libraries
- Displaying the basic statistical details of the data.
- EDA
    - Histogram
    - Pairplot
    - Scatterplot
    - Heatmap and correlation
- Dataset Processing
    - Split the data
- Building Models
    - Logistic Regression
    - k-nearest neighbors algorithm
    - Gradient Boosting

Acknowledgements

This dataset is also available from the UCI machine learning repository, https://archive.ics.uci.edu/ml/datasets/wine+quality , I just shared it to kaggle for convenience. (I am mistaken and the public license type disallowed me from doing so, I will take this down at first request. I am not the owner of this dataset.

Import necessary libraries:

Pandas is a useful library in data handling.

Numpy library used for working with arrays.

Seaborn/Matplotlib are used for data visualisation purpose.

Sklearn – This module contains multiple libraries having pre-implemented functions to perform tasks from data preprocessing to model development and evaluation.

statsmodels: regression, linear models, time series analysis, extensions to topics also covered by scipy.stats.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as st
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score

import warnings
warnings.filterwarnings('ignore')
```

Displaying the dataset:

```
wine = pd.read_csv("winequality-red.csv")
df = wine.copy()
df.head(n = 10).style.background_gradient(cmap = "Purples_r")
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.400000	0.700000	0.000000	1.900000	0.076000	11.000000	34.000000	0.997800	3.510000	0.560000	9.400000	5
1	7.800000	0.880000	0.000000	2.600000	0.098000	25.000000	67.000000	0.996800	3.200000	0.680000	9.800000	5
2	7.800000	0.760000	0.040000	2.300000	0.092000	15.000000	54.000000	0.997000	3.260000	0.650000	9.800000	5
3	11.200000	0.280000	0.560000	1.900000	0.075000	17.000000	60.000000	0.998000	3.160000	0.580000	9.800000	6
4	7.400000	0.700000	0.000000	1.900000	0.076000	11.000000	34.000000	0.997800	3.510000	0.560000	9.400000	5
5	7.400000	0.660000	0.000000	1.800000	0.075000	13.000000	40.000000	0.997800	3.510000	0.560000	9.400000	5
6	7.900000	0.600000	0.060000	1.600000	0.069000	15.000000	59.000000	0.996400	3.300000	0.460000	9.400000	5
7	7.300000	0.650000	0.000000	1.200000	0.065000	15.000000	21.000000	0.994600	3.390000	0.470000	10.000000	7
8	7.800000	0.580000	0.020000	2.000000	0.073000	9.000000	18.000000	0.996800	3.360000	0.570000	9.500000	7
9	7.500000	0.500000	0.360000	6.100000	0.071000	17.000000	102.000000	0.997800	3.350000	0.800000	10.500000	5

Shape,size and describe of the dataset:

```
In [4]:  1 df.shape
Out[4]: (1599, 12)
```

Information

The dataset consists of **1599 rows and 12 columns.**

Finding the “dtypes” of columns in the dataset:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Brief information

It turns out that the dataset does not have **null values.** The dataset consists of **1599 rows and 12 columns.** The data type of all variables are **numeric.**

Numeric features:

```
1 df.describe().T.style.background_gradient(cmap = "magma")
```

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1599.000000	8.319637	1.741096	4.600000	7.100000	7.900000	9.200000	15.900000
volatile acidity	1599.000000	0.527821	0.179060	0.120000	0.390000	0.520000	0.640000	1.580000
citric acid	1599.000000	0.270976	0.194801	0.000000	0.090000	0.260000	0.420000	1.000000
residual sugar	1599.000000	2.538806	1.409928	0.900000	1.900000	2.200000	2.600000	15.500000
chlorides	1599.000000	0.087467	0.047065	0.012000	0.070000	0.079000	0.090000	0.611000
free sulfur dioxide	1599.000000	15.874922	10.460157	1.000000	7.000000	14.000000	21.000000	72.000000
total sulfur dioxide	1599.000000	46.467792	32.895324	6.000000	22.000000	38.000000	62.000000	289.000000
density	1599.000000	0.996747	0.001887	0.990070	0.995600	0.996750	0.997835	1.003690
pH	1599.000000	3.311113	0.154386	2.740000	3.210000	3.310000	3.400000	4.010000
sulphates	1599.000000	0.658149	0.169507	0.330000	0.550000	0.620000	0.730000	2.000000
alcohol	1599.000000	10.422983	1.065668	8.400000	9.500000	10.200000	11.100000	14.900000
quality	1599.000000	5.636023	0.807569	3.000000	5.000000	6.000000	6.000000	8.000000

What can we see from this statistic?

- The average value of fixed acidity is **8.31** , the highest value is **15.9**
- The average value of volatile acidity is **0.52** , the highest value is **1.58**
- The average value of citric acid is **0.27** , the highest value is **1**
- The average value of residual sugar is **2.53,** the highest value is **15.5**
- The average value of chlorides is **0.08** , the highest value is **0.61**
- The average value of free sulfur dioxide is **15.87** , the highest value is **72**
- The average value of total sulfur dioxide is **46.46** , the highest value is **289**
- The average value of density is **0.99** , the highest value is **1**
- The average value of pH is **3.31** , the highest value is **4.01**
- The average value of sulphates is **0.65** , the highest value is **2**
- The average value of alcohol is **10.42** , the highest value is **14.90**
- The average value of quality is **5.63** , the highest value is **8**

Checking the null values:

```
In [9]:  1 df.isnull().sum()

Out[9]: 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
        quality_cat             0
        dtype: int64
```

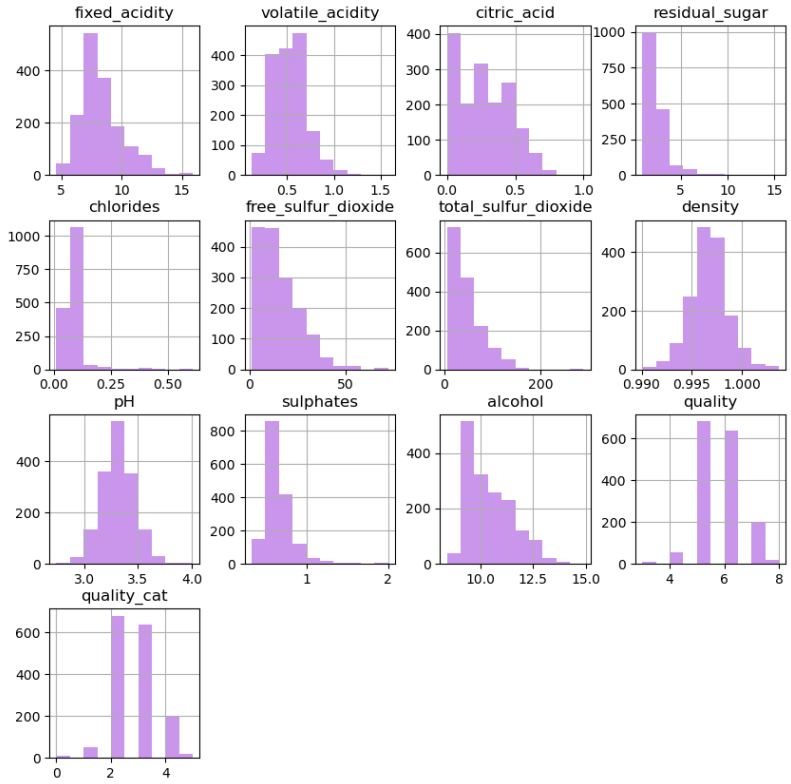
Total null values

There are **not any null values** in the dataset. **Total "null" values** in the dataset is **zero**

EDA:

EDA is an approach to analysing the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations. Now let's check the number of null values in the dataset columns wise.

HISTOGRAM:



1. Most of the values of the "fixed_acidity" variable are in the range of 7 - 8;

2. Most of the values of the "volatile_acidity" variable are in the range of 0.4 - 0.7;

3. Most values of the "citric_acid" variable are in the range of 0.0 - 0.1;

4. Most of the values of the "residual_sugar" variable are in the range of 1 - 2.5;

5. Most of the values of the "chlorides" variable are in the range of 0.085 - 0.15;

6. Most values of the "free_sulfur_dioxide" variable are in the range 0 - 15;

7. Most values of the "total_sulfur_dioxide" variable are in the range 0 - 30;

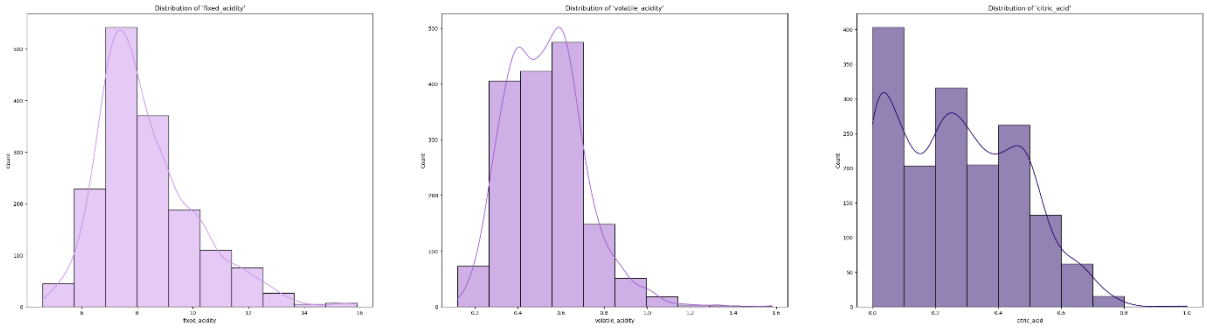
8. Most of the values of the "density" variable are in the range of 0.996 - 0.998;

9. Most of the values of the "pH" variable are in the range of 3.2 - 3.4;

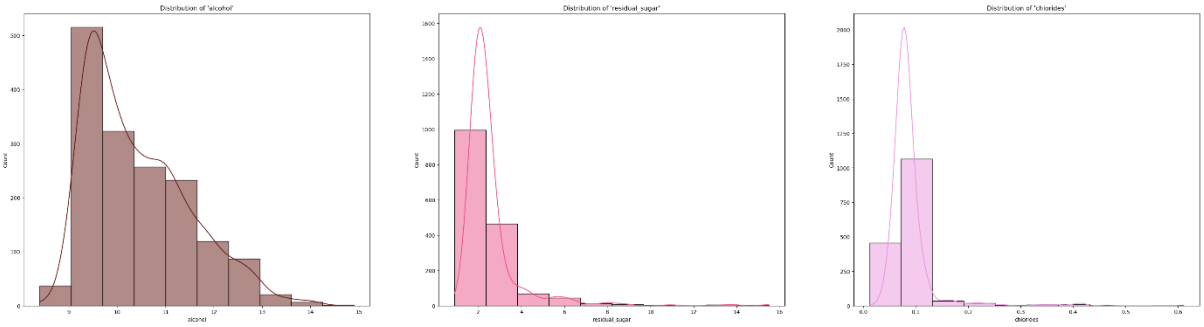
10. Most of the values of the "sulphates" variable are in the range of 0.50 - 0.75;

11. Most of the values of the "alcohol" variable are in the range of 9 - 10;

12. Most values of the "quality" variable are 5 and 6.

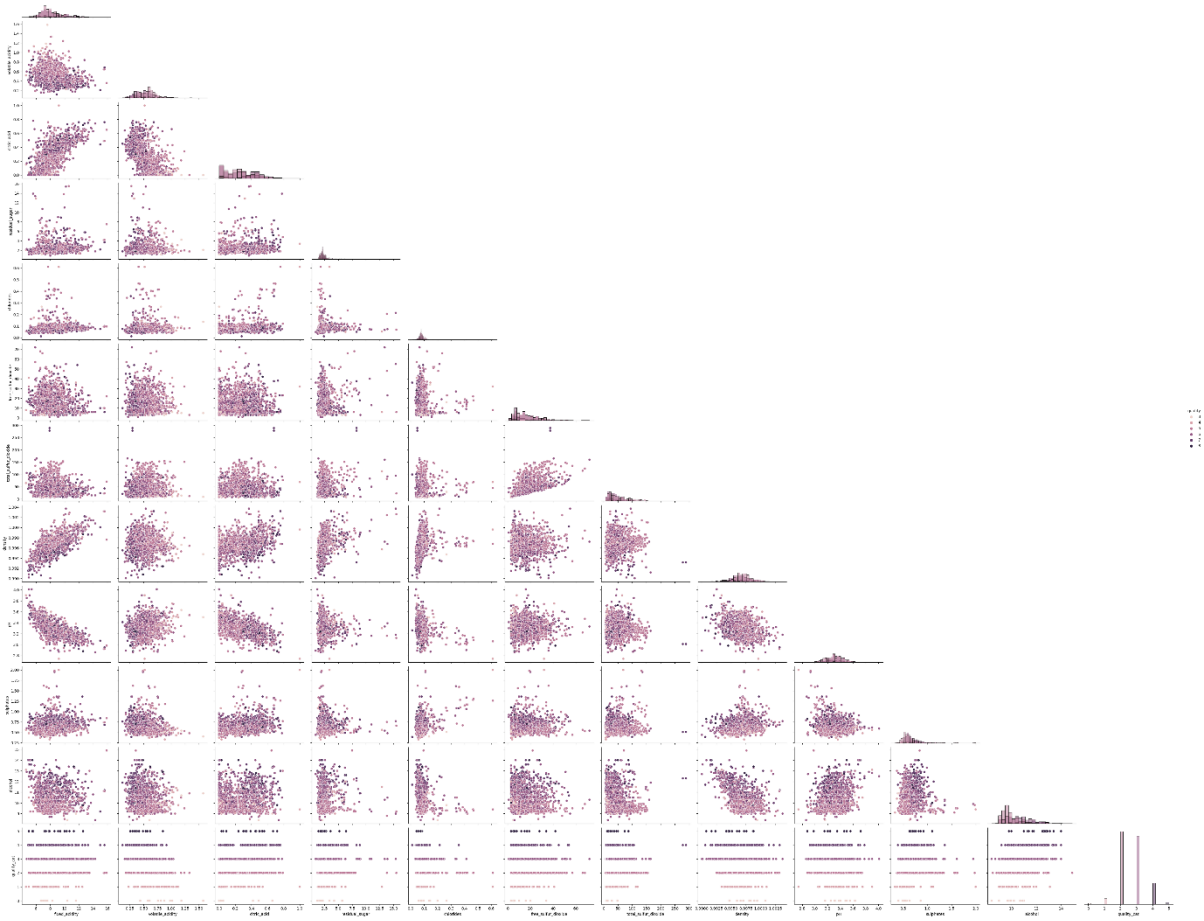


Analyzing the graphs here, it turns out that the values of the variable 'fixed_acidity' are relatively normally distributed (but a bit left skewed). But there are two peaks in the distributions of other 'volatile_acidity' and 'citric_acid' variables.



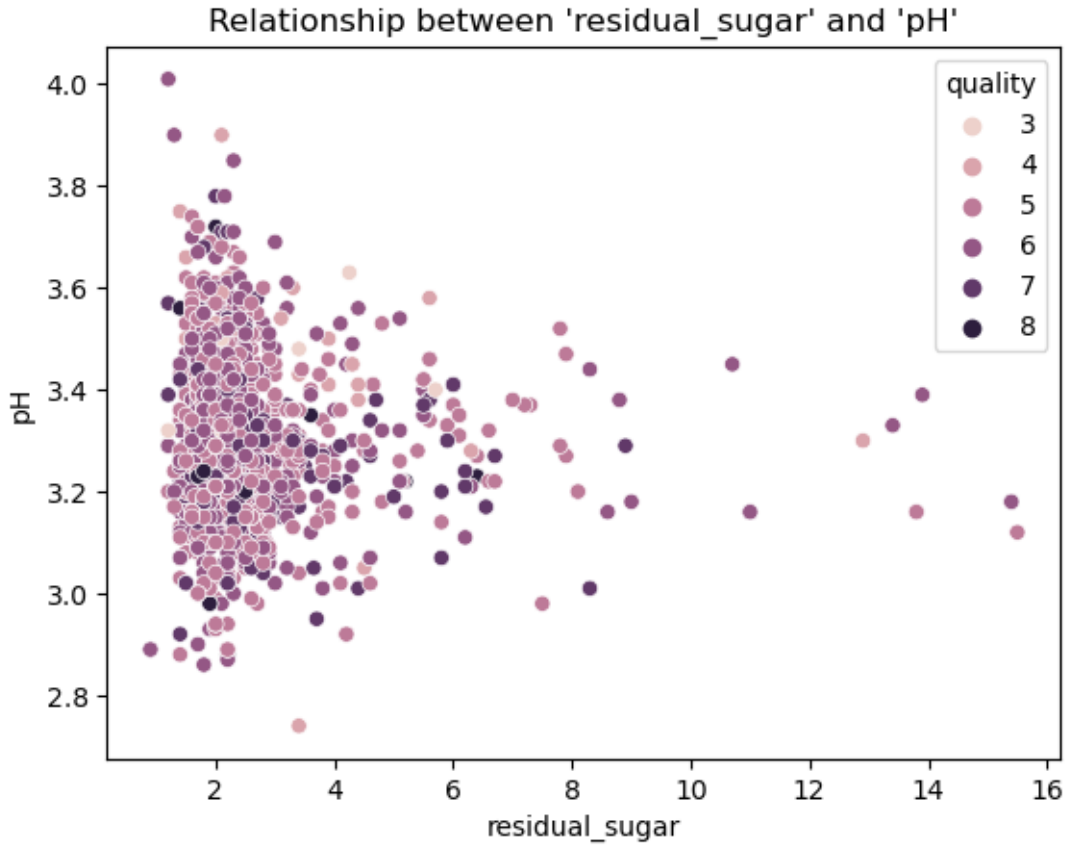
Analyzing the graphs here, it turns out that the distributions of these variables are not normal.

PAIRPLOT:

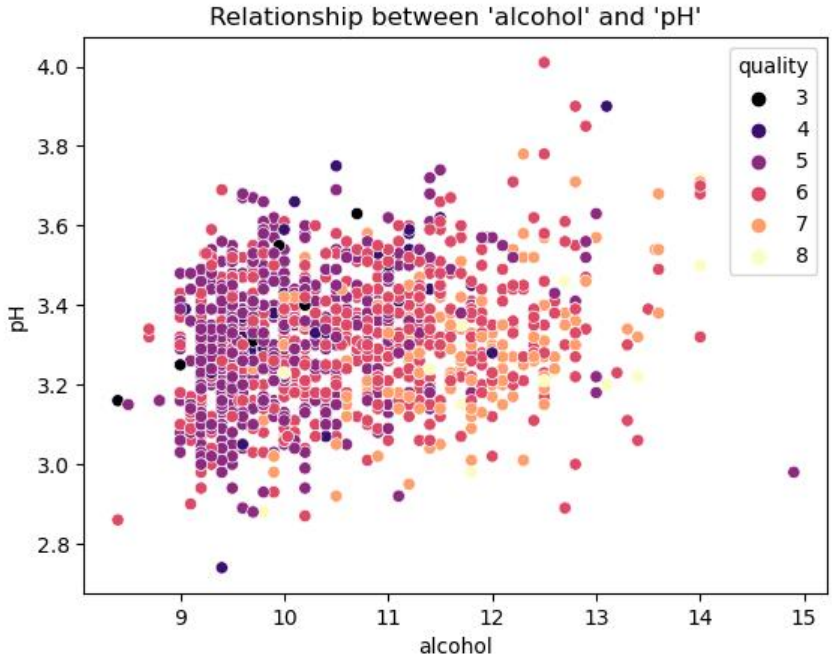


We see here that there is correlation between some variables.And this is what we don't want. This problem is called 'multicollinearity’.

SCATTERPLOT:



As it can be seen there is not correlation between 'residual_sugar' and 'pH' variables.



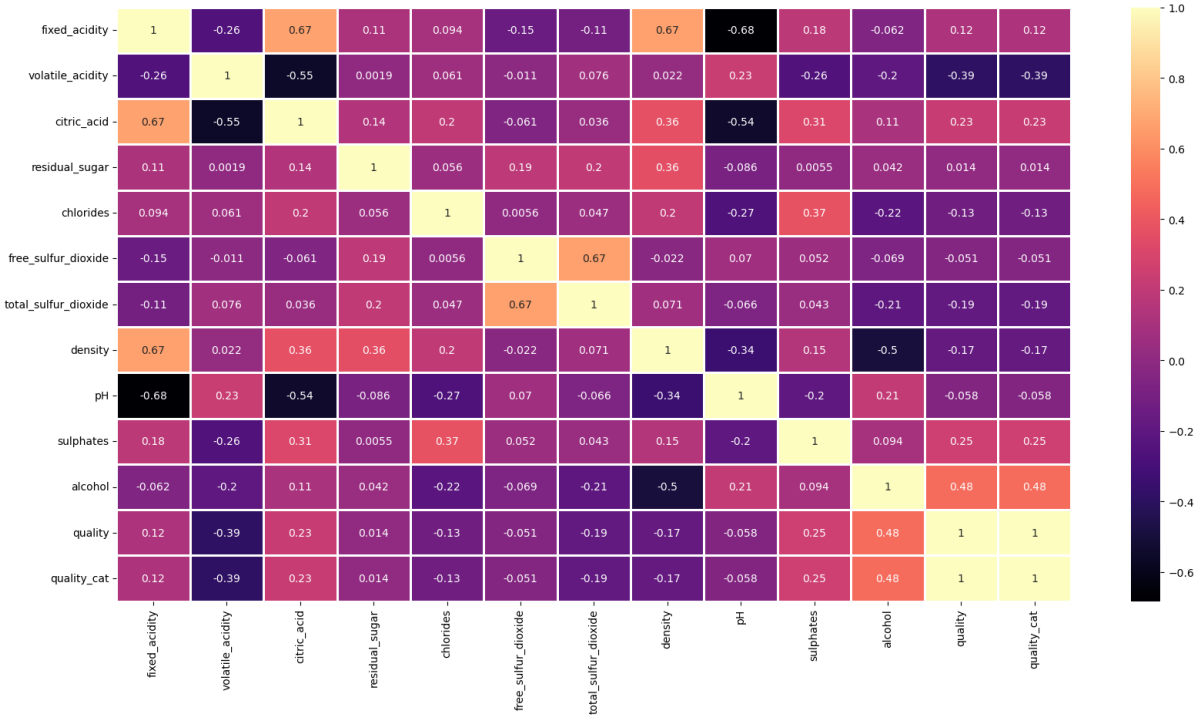
As it can be seen there is not correlation between 'alcohol' and 'pH' variables.

HEATMAP AND CORRELATION:

Two or more variables considered to be related, in a statistical context, if their values change so that as the value of one variable increases or decreases so does the value of the other variable (although it may be in the opposite direction). For example, for the two variables "hours worked" and "income earned" there is a relationship between the two if the increase in hours worked is associated with an increase in income earned. If we consider the two variables "price" and "purchasing power", as the price of goods increases a person's ability to buy these goods decreases (assuming a constant income).

Correlation is a statistical measure (expressed as a number) that describes the size and direction of a relationship between two or more variables. A correlation between variables, however, does not automatically mean that the change in one variable is the cause of the change in the values of the other variable.

Causation indicates that one event is the result of the occurrence of the other event; i.e. there is a causal relationship between the two events. This is also referred to as cause and effect.



There is 'multicollinearity' problem

Here we see that there is relatively high (0.67, positive) correlation between 'free sulfur dioxide' and 'total_sulfur_dioxide' variables. There is relatively high (-0.68, negative) correlation between "pH" and "fixed_acidity" variables. And there is about 0.5 correlation between some of other variables. That's why we must consider when build Machine Learning models.

```
           alcohol   density
alcohol    1.00000 -0.49618
density   -0.49618  1.00000


                fixed_acidity        pH
fixed_acidity         1.000000 -0.682978
pH                   -0.682978  1.000000


              citric_acid        pH
citric_acid      1.000000 -0.541904
pH              -0.541904  1.000000


                fixed_acidity   density
fixed_acidity         1.000000  0.668047
density               0.668047  1.000000


                      free_sulfur_dioxide  total_sulfur_dioxide
free_sulfur_dioxide              1.000000              0.667666
total_sulfur_dioxide             0.667666              1.000000
```

In the code blocks below we will look at the Pearson correlation coefficient between some variables

DATASET PREPROCESSING:

In this dataset quality range is between 3 and 8

We will divide quality range into two parts:

- High quality wine: 6 - 8

- Low quality wine: 3 - 5

In [25]:
```
df["quality"] = np.where(df["quality"] > 5, 1, 0)
df["quality"]
```

Out[25]:
```
0       0
1       0
2       0
3       1
4       0
       ..
1594    0
1595    1
1596    1
1597    0
1598    1
Name: quality, Length: 1599, dtype: int32
```

Look at Dataset (with changed 'quality' variable)

Select Dependent and Independent Variables

In [27]:
```
# we select dependent variable (label)
y = df["quality"]

# we select independent variable
x = df.drop("quality", axis = 1)
```

Split Dataset into Train and Test Sets

In [28]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.25,
                                                    shuffle = True,
                                                    random_state = 1)
```

Standardization:

Standardizing the features around the center and 0 with a standard deviation of 1 is important when we compare measurements that have different units. Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bais. For example, a variable that ranges between 0 and 1000 will outweigh a variable that ranges between 0 and 1. Using these variables without standardization will give the variable with the larger range weight of 1000 in the analysis. Transforming the data to comparable scales can prevent this problem. Typical data standardization procedures equalize the range and/or data variability.

But we will not use 'StandardScaler', because our dataset is not normally distributed. We will use 'MinMaxScaler' for normalizing this dataset. It transforms features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between 0 and 1.

```
print(y_train.head())
print(y_train.shape)
print("_____")
print(y_test.head())
print(y_test.shape)
```

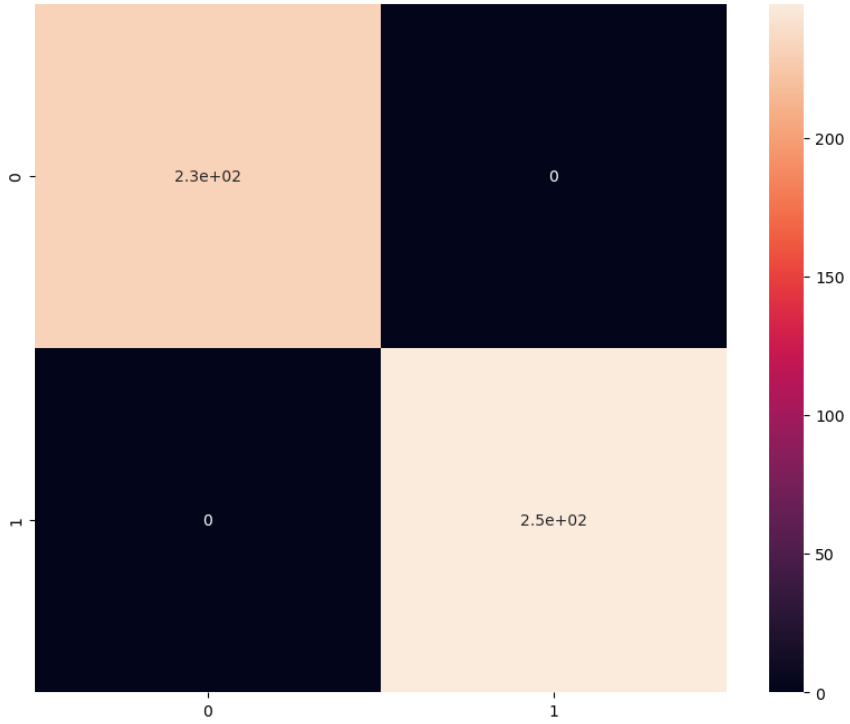
```
1144    0
73      0
446     0
399     0
647     0
Name: quality, dtype: int32
(1199,)
_____
75      0
1283    1
408     1
1281    1
1118    1
Name: quality, dtype: int32
(400,)
```

BUILDING CLASSIFICATION MODELS:

Logistic Regression:

In statistics, the logistic model (or logit model) is a statistical model that models the probability of an event taking place by having the log-odds for the event be a linear combination of one or more independent variables. In regression analysis, logistic regression[1] (or logit regression) is estimating the parameters of a logistic model (the coefficients in the linear combination). Formally, in binary logistic regression there is a single binary dependent variable, coded by an indicator variable, where the two values are labeled "0" and "1", while the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling;[2] the function that converts log-odds to probability is the logistic function, hence the name. The unit of measurement for the log-odds scale is called a logit, from logistic unit, hence the alternative names.

Confusion matrix:



```
from sklearn.metrics import classification_report
target_names = ['Bad', 'Good']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

         Bad       1.00      1.00      1.00       232
        Good       1.00      1.00      1.00       248

    accuracy                           1.00       480
   macro avg       1.00      1.00      1.00       480
weighted avg       1.00      1.00      1.00       480
```

k-nearest neighbors algorithm:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). **If k = 1, then the object is simply assigned to the class of that single nearest neighbor.**

In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. T**his can be thought of as the training set for the algorithm, though no explicit training step is required.**

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

Searching the best hyperparametres with GridSearchCV method

In [124]:
```
knn_params = {"n_neighbors": np.arange(2, 50),
              "weights": ["uniform", "distance"],
              "leaf_size": [25, 30, 25]}

knn_cv_model = GridSearchCV(knn, knn_params, cv = 10)
knn_cv_model.fit(x_train, y_train)
```

Out[124]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'leaf_size': [25, 30, 25],
                         'n_neighbors': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),
                         'weights': ['uniform', 'distance']})
```

Get Best Parameters of KNN Model

In [125]:
```
print("Best score for train set: " + str(knn_cv_model.best_score_))

print("_____________________________________________")

print("best K value: " + str(knn_cv_model.best_params_["n_neighbors"]),
     "\nbest weights: " + knn_cv_model.best_params_["weights"],
     "\nbest leaf size: " + str(knn_cv_model.best_params_["leaf_size"]))
```

```
Best score for train set: 0.9776705276705275
_____________________________________________
best K value: 23
best weights: distance
best leaf size: 25
```

Build KNN Model with Best Parameters

In [126]:
```
knn_model = KNeighborsClassifier(n_neighbors = knn_cv_model.best_params_["n_neighbors"],
                                 leaf_size = knn_cv_model.best_params_["leaf_size"],
                                 weights = knn_cv_model.best_params_["weights"])

knn_model.fit(x_train, y_train)
```

Out[126]:
```
KNeighborsClassifier(leaf_size=25, n_neighbors=23, weights='distance')
```

Accuracy Score of KNN Model on Test set

In [127]:
```
y_pred = knn_model.predict(x_test)
accuracy_score(y_test, y_pred)
```

Out[127]:
```
0.95625
```

Classification Report of KNN Model

In [128]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.94      0.95       232
           1       0.94      0.98      0.96       248

    accuracy                           0.96       480
   macro avg       0.96      0.96      0.96       480
weighted avg       0.96      0.96      0.96       480
```

Gradient Boosting:

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest. A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

Searching the best hyperparametres with GridSearchCV method

In [130]:
```
gbm_params = {"learning_rate": [0.005, 0.008, 0.1, 0.15],
              "n_estimators": [80, 100, 150, 200],
              "max_depth": [2, 3, 4],
              "min_samples_split": [2, 3, 4]}

gbm_cv_model = GridSearchCV(gbm, gbm_params, cv = 10, n_jobs = -1)
gbm_cv_model.fit(x_train, y_train)
```

Out[130]:
```
GridSearchCV(cv=10, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.005, 0.008, 0.1, 0.15],
                         'max_depth': [2, 3, 4], 'min_samples_split': [2, 3, 4],
                         'n_estimators': [80, 100, 150, 200]})
```

Get Best Parameters of GBM Model

In [131]:
```
print("Best score for train set: " + str(gbm_cv_model.best_score_))

print("_____________________________________________________")

print("best learning_rate value: " + str(gbm_cv_model.best_params_["learning_rate"]),
     "\nbest n_estimators value: " + str(gbm_cv_model.best_params_["n_estimators"]),
     "\nbest max_depth value: " + str(gbm_cv_model.best_params_["max_depth"]),
     "\nbest min_samples_split value: " + str(gbm_cv_model.best_params_["min_samples_split"]))
```

```
Best score for train set: 1.0
_____________________________________________________
best learning_rate value: 0.005
best n_estimators value: 80
best max_depth value: 2
best min_samples_split value: 2
```

Build GBM Model with Best Parameters

In [132]:
```
gbm = GradientBoostingClassifier(learning_rate = gbm_cv_model.best_params_["learning_rate"],
                                 max_depth = gbm_cv_model.best_params_["max_depth"],
                                 n_estimators = gbm_cv_model.best_params_["n_estimators"],
                                 min_samples_split = gbm_cv_model.best_params_["min_samples_split"])
gbm_model = gbm.fit(x_train, y_train)
```

Accuracy Score of GBM Model on Test set

In [133]:
```
y_pred = gbm_model.predict(x_test)
accuracy_score(y_test, y_pred)
```

Out[133]:
```
1.0
```

Conclusion:

Based on the Histogram plotted we come to an conclusion that not all input features are essential and affect the data, for example from the bar plot against quality and residual sugar we see that as the quality increases residual sugar is moderate and does not have change drastically. So this feature is not so essential as compared to others like alcohol and citric acid, so we can drop this feature while feature selection.

For classifying the wine quality, we have implemented multiple algorithms, namely

1) Logistic Regression

2) k-nearest neighbors algorithm

3) Gradient Boosting

We were able to achieve maximum accuracy using Logistic Regression of 1.0. K-nearest neighbors algorithm giving an accuracy of 0.95 . Gradient Boosting has an accuracy of 1.0 .

Reference:

Dataset is from Kaggle
Link: kaggle_wine_quality