Project Overview

STEPS

- Data Collection kaggle
- Feature Engineering
- Featue Scaling
- Model Creation
- Evolution of Model

Attribute Information:

Input variables (based on physicochemical tests):

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

1. quality (score between 0 and 10) Output variable (based on sensory data):

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as pt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('/Data/winequalityN.csv')
```

Exploratary Data Analysis

```
df.head()

    type  fixed acidity  volatile acidity  citric acid  residual sugar  
\
0  white            7.0              0.27         0.36            20.7   

1  white            6.3              0.30         0.34             1.6   

2  white            8.1              0.28         0.40             6.9   

3  white            7.2              0.23         0.32             8.5   

4  white            7.2              0.23         0.32             8.5   


   chlorides  free sulfur dioxide  total sulfur dioxide  density    pH  
\
0      0.045                 45.0                 170.0   1.0010  3.00
```

```
1      0.049                      14.0                       132.0   0.9940   3.30

2      0.050                      30.0                        97.0   0.9951   3.26

3      0.058                      47.0                       186.0   0.9956   3.19

4      0.058                      47.0                       186.0   0.9956   3.19


   sulphates  alcohol  quality
0       0.45      8.8        6
1       0.49      9.5        6
2       0.44     10.1        6
3       0.40      9.9        6
4       0.40      9.9        6
```

```
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
df['type']=label.fit_transform(df['type']) #white=1,red=0

## info about dataset
df.info()
```

```
<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   int32
 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), int32(1), int64(1)
memory usage: 634.6 KB
```

```
## size of the dataframe
df.shape
```

```
(6497, 13)
```

```
## duplicated rows
df.duplicated().any()
```

```
True
```

```
## droping druplicate
df.drop_duplicates(inplace=True)
```

```
df.shape ## shape of df after removing the dataset
```

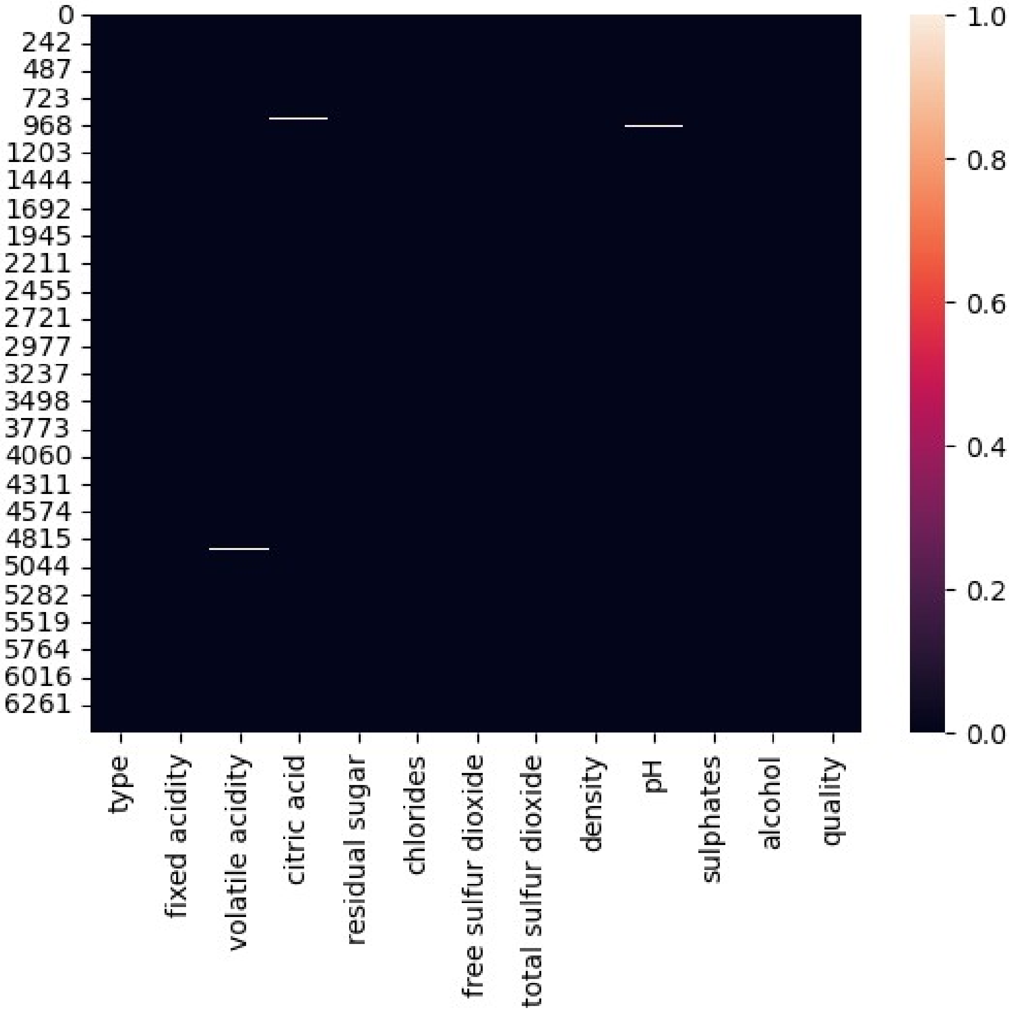
```
(5329, 13)
```

```
## Missing values
df.isnull().sum()
```

```
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
```

```
sns.heatmap(df.isnull())
```

```
<Axes: >
```



- columns are fixed acidity,volatile acidity,citric acide,residual sugar,chlorides,pH,sulphates are have the some of null values init

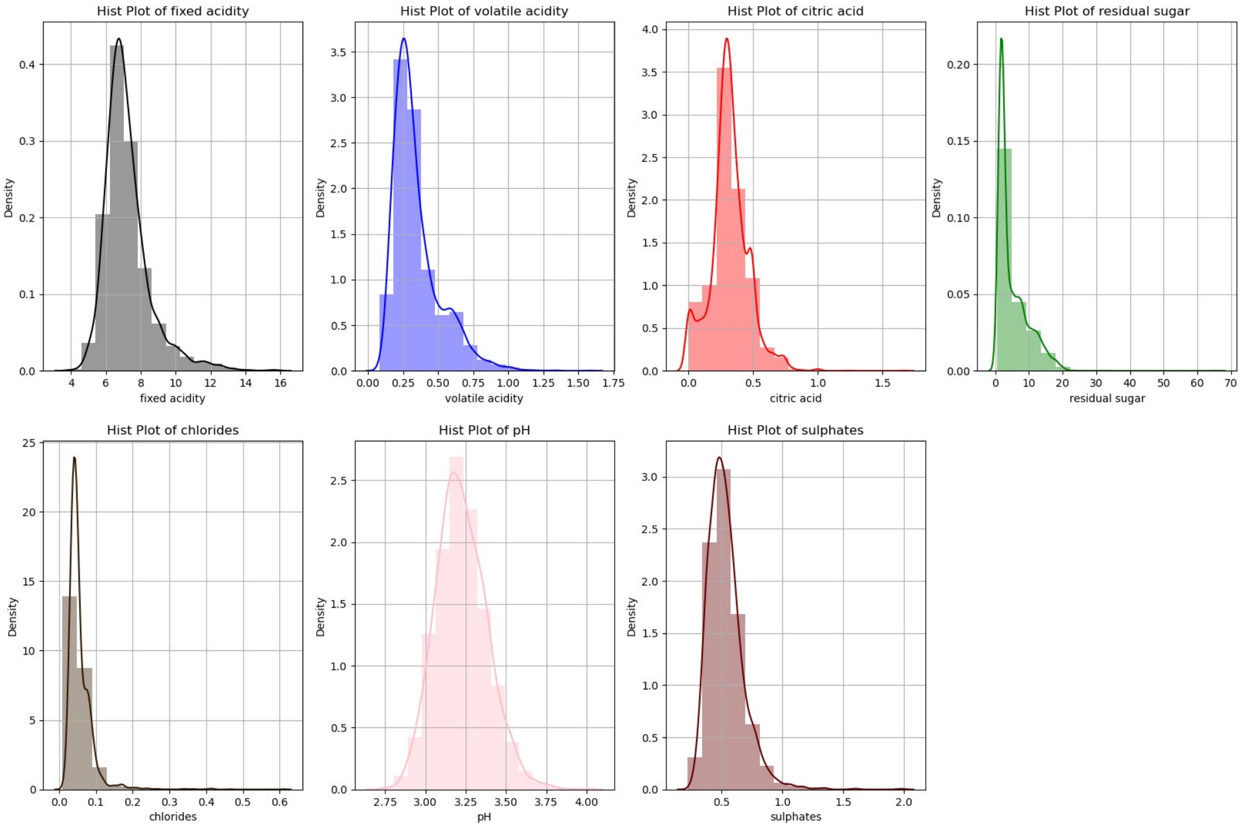
Handling Missing Values

```
## missing columns
columns_having_missing ={}
for column in df.columns:
    if df[column].isnull().sum() > 0:
        columns_having_missing[column] = df[column].isnull().sum()
print(columns_having_missing)

{'fixed acidity': 10, 'volatile acidity': 8, 'citric acid': 3,
'residual sugar': 2, 'chlorides': 2, 'pH': 9, 'sulphates': 4}
```

histplots before filling the miss values in columns

```
## histplots before filling the miss values in columns
pt.figure(figsize=(20, 20))
```

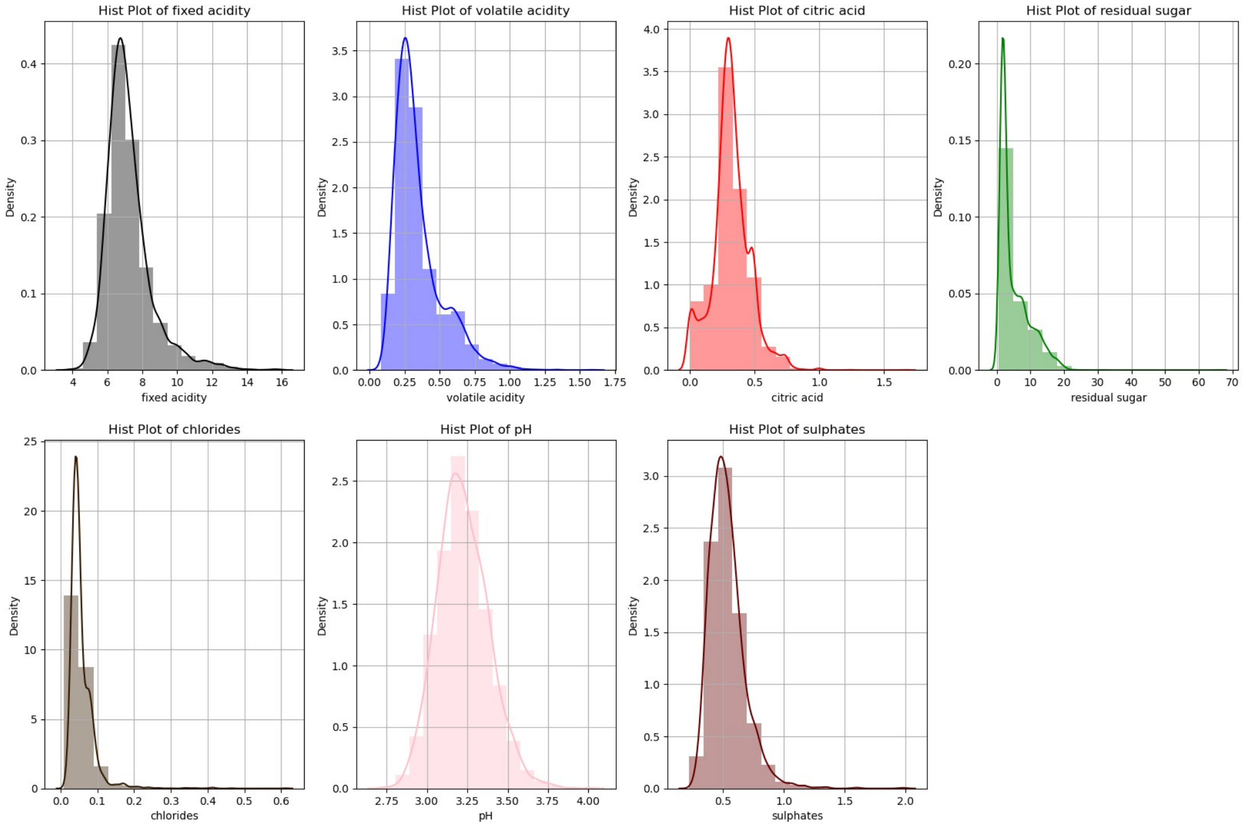
```
color = ['k','b','r','g','#331900','pink','#660000']
for i, feature in enumerate(list(columns_having_missing.keys())):
    pt.subplot(3,4 ,i + 1)
    sns.distplot(df[feature],color=color[i],bins=15)
    pt.grid(axis='both')
    pt.title(f'Hist Plot of {feature}')
```



```
##Filling the missing values with its column's mean
for feature in columns_having_missing.keys():
    df[feature].fillna(df[feature].mean(),inplace=True)
```

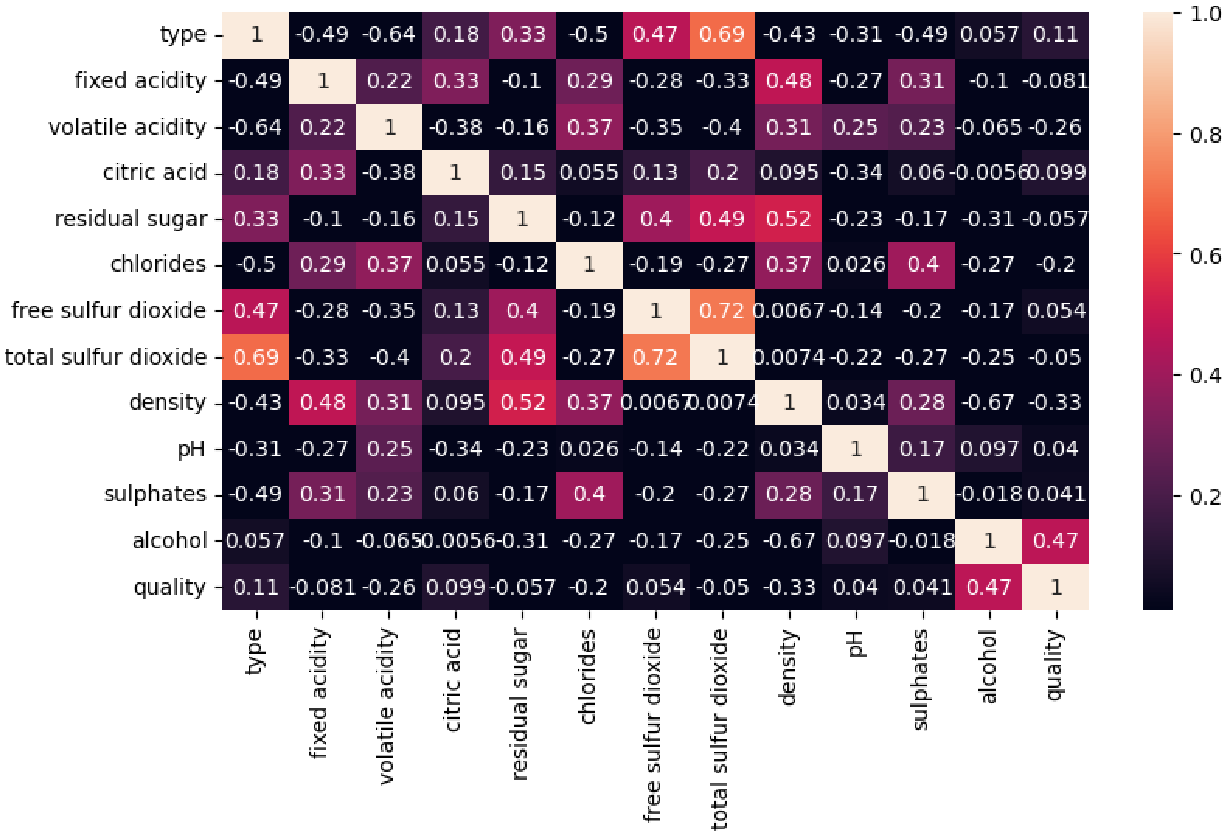
histplots after filling the miss values in columns

```
## seeing the distribution after filling the histplots
pt.figure(figsize=(20, 20))
color = ['k','b','r','g','#331900','pink','#660000']
for i, feature in enumerate(list(columns_having_missing.keys())):
    pt.subplot(3,4 ,i + 1)
    sns.distplot(df[feature],color=color[i],bins=15)
    pt.grid(axis='both')
    pt.title(f'Hist Plot of {feature}')
```



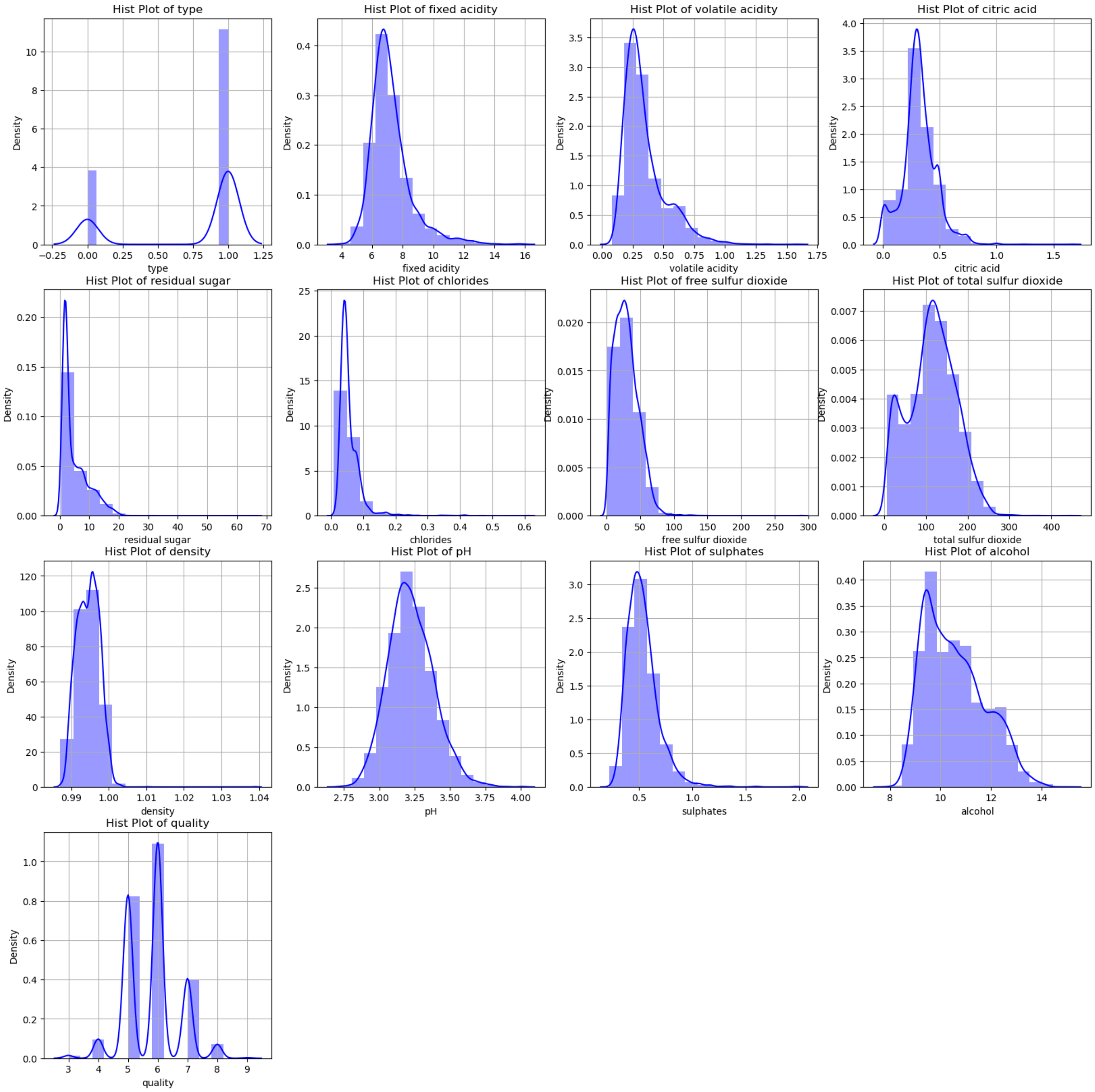
```
## corrlation matrxi
pt.figure(figsize=(9,5))
sns.heatmap(df.corr(),annot=True,vmin=0.01)
```

```
<Axes: >
```



	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
type	1	-0.49	-0.64	0.18	0.33	-0.5	0.47	0.69	-0.43	-0.31	-0.49	0.057	0.11
fixed acidity	-0.49	1	0.22	0.33	-0.1	0.29	-0.28	-0.33	0.48	-0.27	0.31	-0.1	-0.081
volatile acidity	-0.64	0.22	1	-0.38	-0.16	0.37	-0.35	-0.4	0.31	0.25	0.23	-0.065	-0.26
citric acid	0.18	0.33	-0.38	1	0.15	0.055	0.13	0.2	0.095	-0.34	0.06	-0.0056	0.099
residual sugar	0.33	-0.1	-0.16	0.15	1	-0.12	0.4	0.49	0.52	-0.23	-0.17	-0.31	-0.057
chlorides	-0.5	0.29	0.37	0.055	-0.12	1	-0.19	-0.27	0.37	0.026	0.4	-0.27	-0.2
free sulfur dioxide	0.47	-0.28	-0.35	0.13	0.4	-0.19	1	0.72	0.0067	-0.14	-0.2	-0.17	0.054
total sulfur dioxide	0.69	-0.33	-0.4	0.2	0.49	-0.27	0.72	1	0.0074	-0.22	-0.27	-0.25	-0.05
density	-0.43	0.48	0.31	0.095	0.52	0.37	0.0067	0.0074	1	0.034	0.28	-0.67	-0.33
pH	-0.31	-0.27	0.25	-0.34	-0.23	0.026	-0.14	-0.22	0.034	1	0.17	0.097	0.04
sulphates	-0.49	0.31	0.23	0.06	-0.17	0.4	-0.2	-0.27	0.28	0.17	1	-0.018	0.041
alcohol	0.057	-0.1	-0.065	0.0056	-0.31	-0.27	-0.17	-0.25	-0.67	0.097	-0.018	1	0.47
quality	0.11	-0.081	-0.26	0.099	-0.057	-0.2	0.054	-0.05	-0.33	0.04	0.041	0.47	1

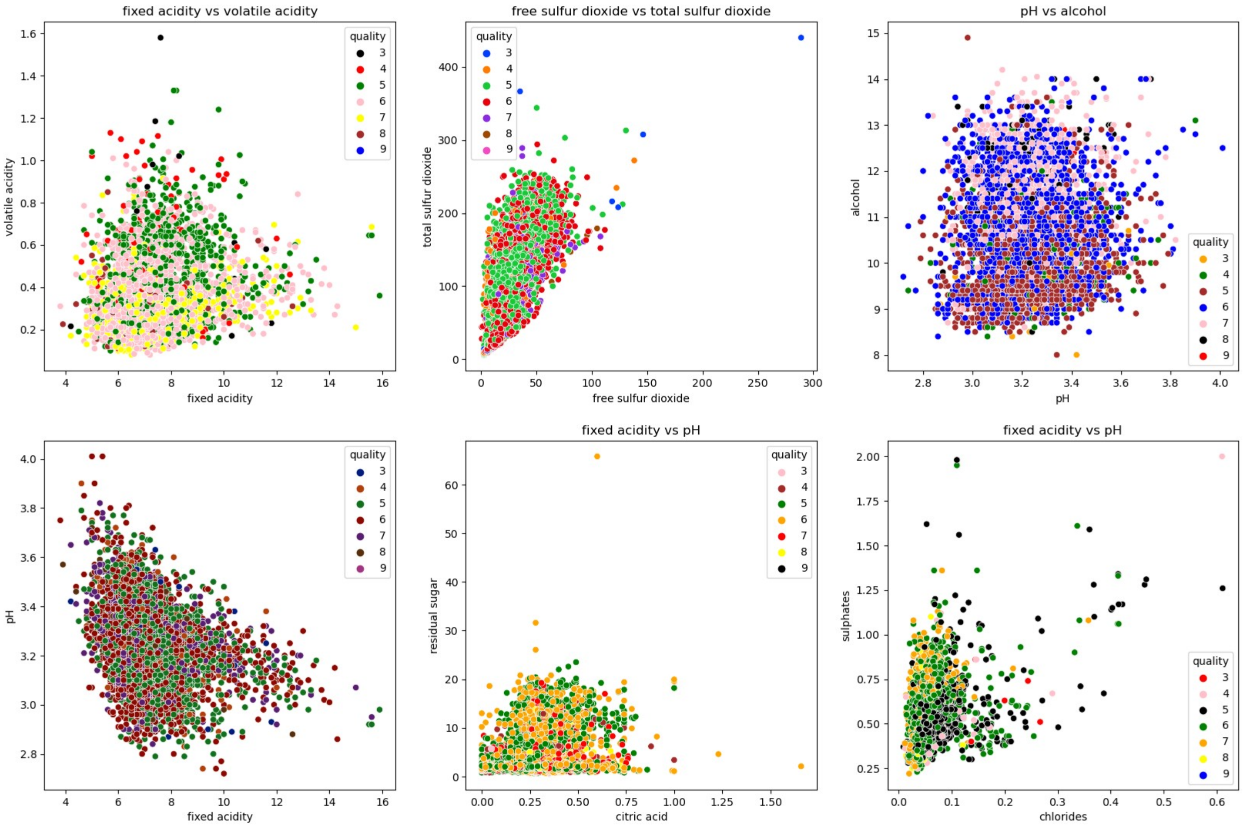
```python
## Univaraity analysis
pt.figure(figsize=(20, 20))
for i, feature in enumerate(df.columns):
    pt.subplot(4,4 ,i + 1)
    sns.distplot(df[feature],color='blue',bins=15)
    pt.grid(axis='both')
    pt.title(f'Hist Plot of {feature}')
```


Hist Plot of type
Hist Plot of fixed acidity
Hist Plot of volatile acidity
Hist Plot of citric acid
Hist Plot of residual sugar
Hist Plot of chlorides
Hist Plot of free sulfur dioxide
Hist Plot of total sulfur dioxide
Hist Plot of density
Hist Plot of pH
Hist Plot of sulphates
Hist Plot of alcohol
Hist Plot of quality

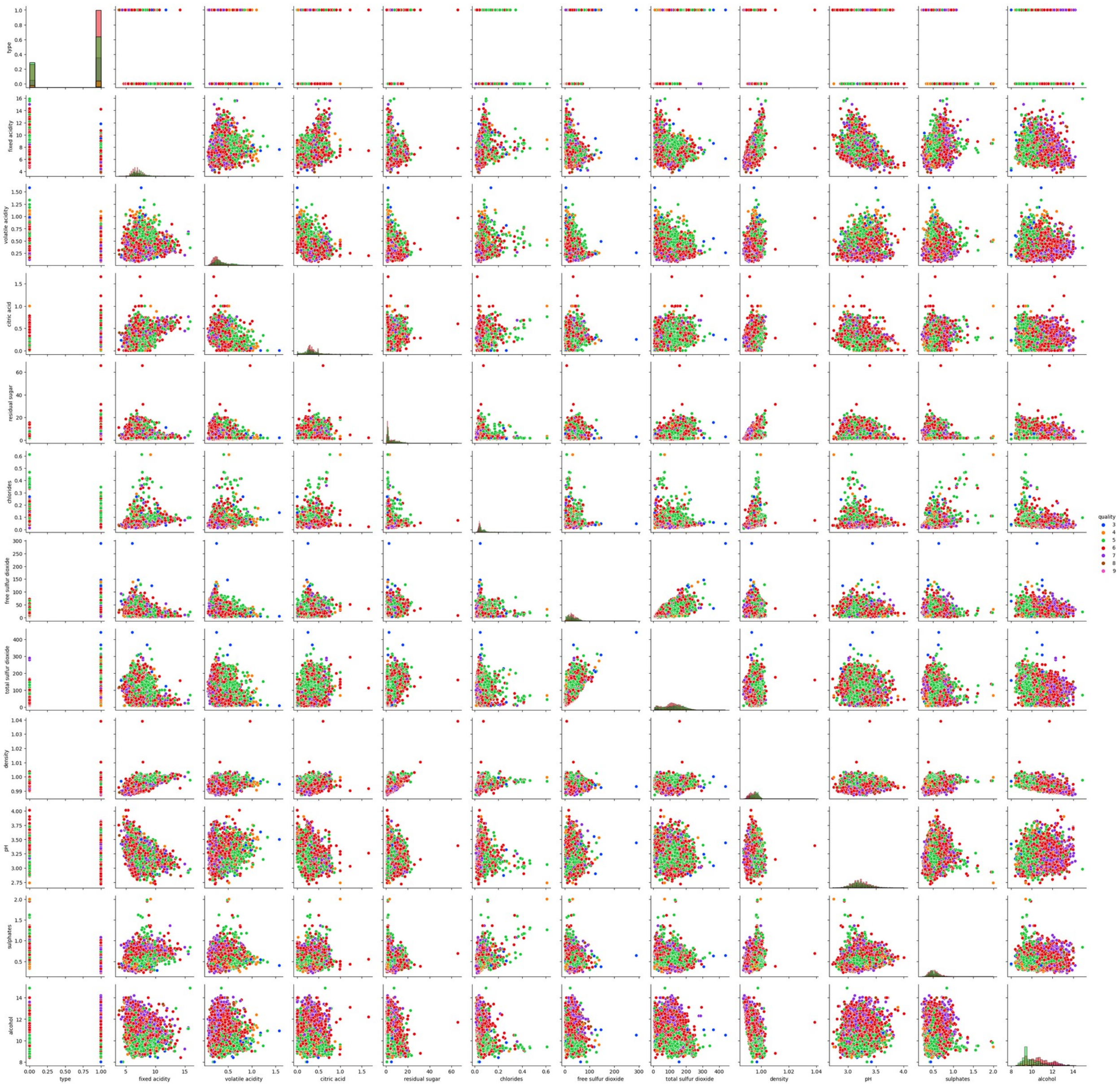
```
df.head()

   type  fixed acidity  volatile acidity  citric acid  residual sugar
\
0     1            7.0              0.27         0.36            20.7

1     1            6.3              0.30         0.34             1.6

2     1            8.1              0.28         0.40             6.9

3     1            7.2              0.23         0.32             8.5

6     1            6.2              0.32         0.16             7.0
```

```
   chlorides  free sulfur dioxide  total sulfur dioxide  density    pH
\
0      0.045                 45.0                 170.0   1.0010  3.00

1      0.049                 14.0                 132.0   0.9940  3.30

2      0.050                 30.0                  97.0   0.9951  3.26

3      0.058                 47.0                 186.0   0.9956  3.19

6      0.045                 30.0                 136.0   0.9949  3.18


   sulphates  alcohol  quality
0       0.45      8.8        6
1       0.49      9.5        6
2       0.44     10.1        6
3       0.40      9.9        6
6       0.47      9.6        6
```

```
## Bivariate Analysis
pt.figure(figsize=(20,20))
pt.title('Bivariat Analysis')
pt.subplot(3,3,1)
sns.scatterplot(df,x='fixed acidity',y='volatile
acidity',color='g',hue='quality',palette=['black','red','green','pink'
,'yellow','brown','blue'])
pt.title('fixed acidity vs volatile acidity')

pt.subplot(3,3,2)
sns.scatterplot(df,x='free sulfur dioxide',y='total sulfur
dioxide',hue='quality',palette='bright')
pt.title('free sulfur dioxide vs total sulfur dioxide')

pt.subplot(3,3,3)
sns.scatterplot(df,x='pH',y='alcohol',hue='quality',palette=['orange',
'green','brown','blue','pink','black','red'])
pt.title('pH vs alcohol')

pt.subplot(3,3,4)
sns.scatterplot(df,x='fixed
acidity',y='pH',hue='quality',palette='dark')


pt.subplot(3,3,5)
sns.scatterplot(df,x='citric acid',y='residual
sugar',hue='quality',palette=['pink','brown','green','orange','red','y
ellow','black'])
pt.title('fixed acidity vs pH')
pt.subplot(3,3,6)
sns.scatterplot(df,x='chlorides',y='sulphates',hue='quality',palette=[
'red','pink','black','green','orange','yellow','blue'])
pt.title('fixed acidity vs pH')
pt.show()
```

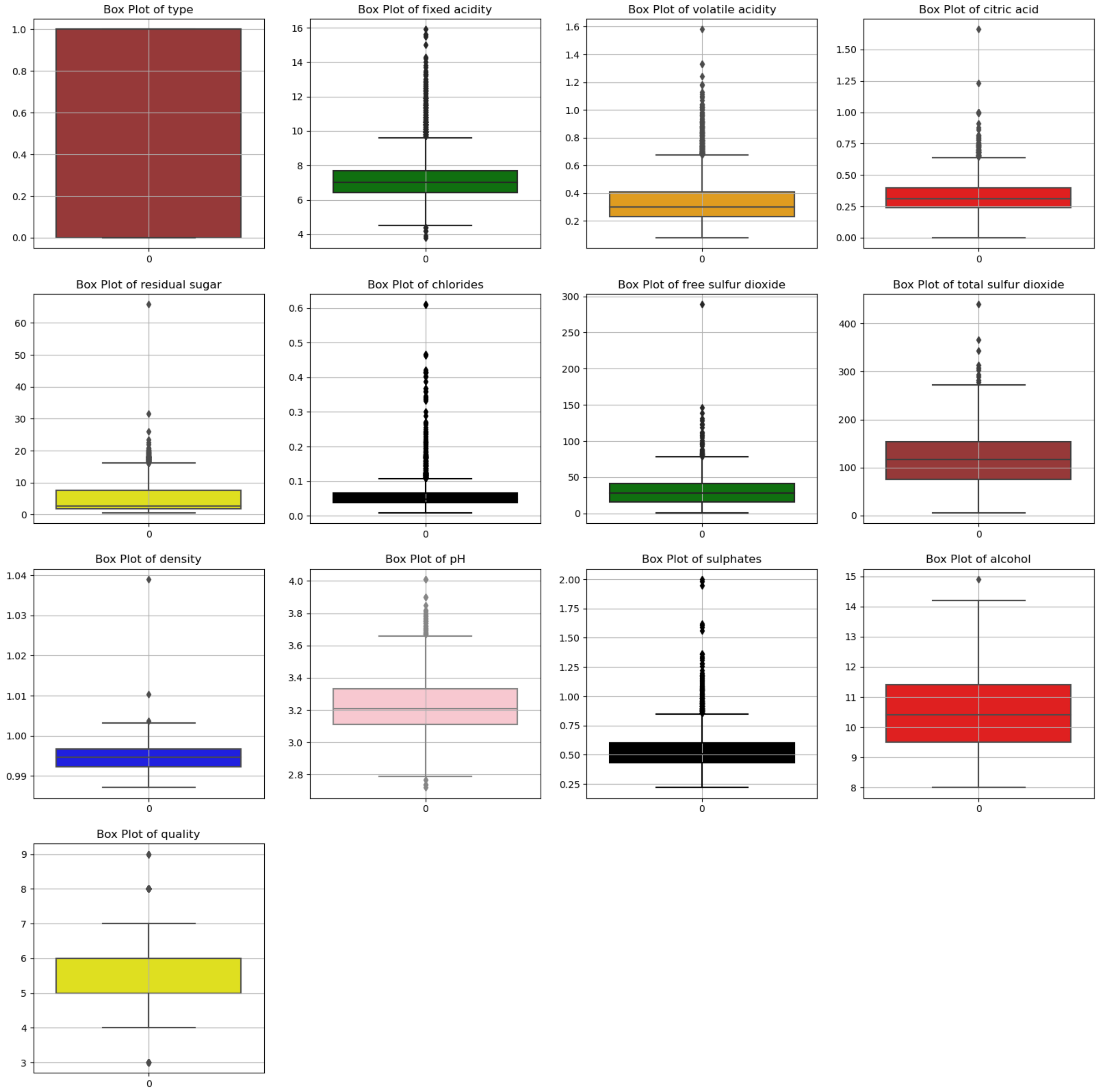


```
g = sns.PairGrid(df, hue="quality",palette='bright')
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
g.add_legend()

<seaborn.axisgrid.PairGrid at 0x19c0fabf290>
```



Outliers

```
pt.figure(figsize=(20, 20))
colors =
['pink','brown','green','orange','red','yellow','black','green','brown
','blue','pink','black','red','yellow']
for i, feature in enumerate(df.columns):
    pt.subplot(4,4 ,i + 1)
    sns.boxplot(df[feature],color=colors[i+1])
    pt.grid(axis='both')
    pt.title(f'Box Plot of {feature}')
```



Mapping the taget output classes as Low,Medium,High Quality

Target column have 3,4,5,6,7,8,9 are output classes Generaizing output classes

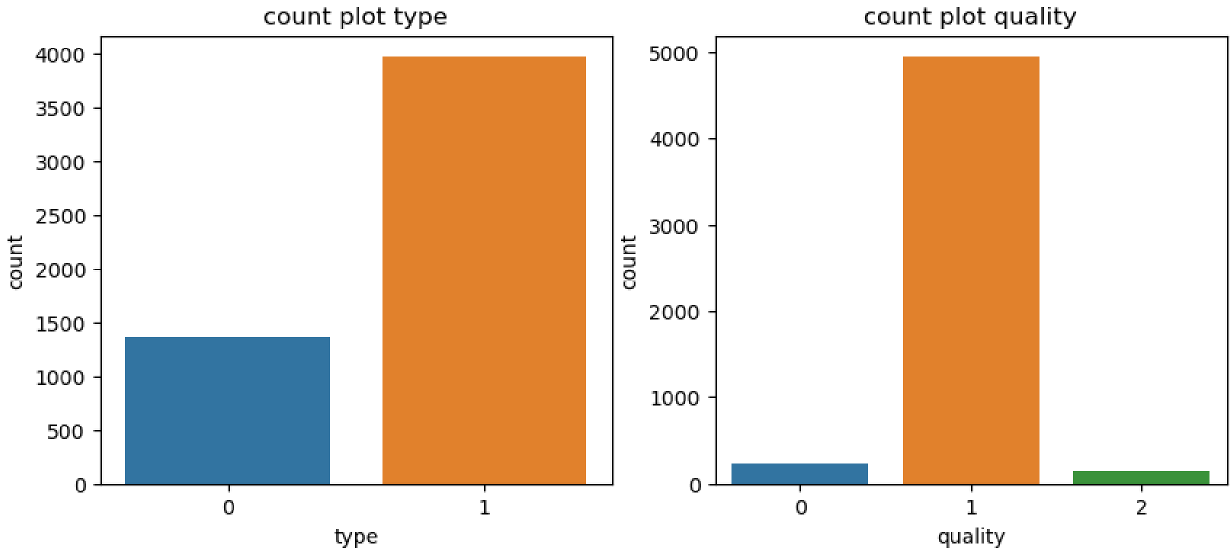
- (3,4) --> LOW Quality
- (5,6,7) --> MEDIUM Quality
- (8,9) --> HIGH Quality

```
### Mapping to output classes
df['quality'] = 
df['quality'].map({3:'Low',4:'Low',5:'Medium',6:'Medium',7:'Medium',8:
'High',9:'High'})

df['quality'] = df['quality'].map({'Low':0,'Medium':1,"High":2})
```

```
df.quality.unique()
```

```
array([1, 2, 0], dtype=int64)
```

```
## categorical feature count plot
pt.figure(figsize=(10,4))
for i,feature in enumerate(df[['type','quality']].columns):
    pt.subplot(1,2,i+1)
    sns.countplot(df,x=feature)
    pt.title('count plot '+feature)
```



```
##checking the data is whether is imbalanced dataset
print('Low',len(df[df['quality']==0]))
print('Medium',len(df[df['quality']==1]))
print('High',len(df[df['quality']==2]))
```

```
Low 236
Medium 4939
High 154
```

```
X = df.drop('quality',axis=1)
y = df.quality
```

```
print(X.shape)
print(y.shape)
```

```
(5329, 12)
(5329,)
```

Feature Importance

```
#### Feature Importance
from sklearn.ensemble import ExtraTreesClassifier
```

```
feature_imp= ExtraTreesClassifier()
feature_imp.fit(X,y)
scores = feature_imp.feature_importances_
pd.DataFrame({'Feature': X.columns,'Feature_Importance':
(scores*100)}).sort_values(by = 'Feature_Importance', ascending =
True)
```

```
                 Feature  Feature_Importance
0                   type            1.024119
8                density            8.073902
5              chlorides            8.266900
3            citric acid            8.410160
1          fixed acidity            8.463813
9                     pH            8.499229
4         residual sugar            8.573696
10             sulphates            8.792134
7   total sulfur dioxide            8.861663
11               alcohol            9.442263
2       volatile acidity           10.767609
6    free sulfur dioxide           10.824512
```

Converting the Imbalanced dataset into Balanced dataset

```
# transform the dataset in to balanced formet
from imblearn.over_sampling import SMOTE
oversample = SMOTE(k_neighbors=4)
X, y = oversample.fit_resample(X, y)

print(X.shape)
print(y.shape)
```

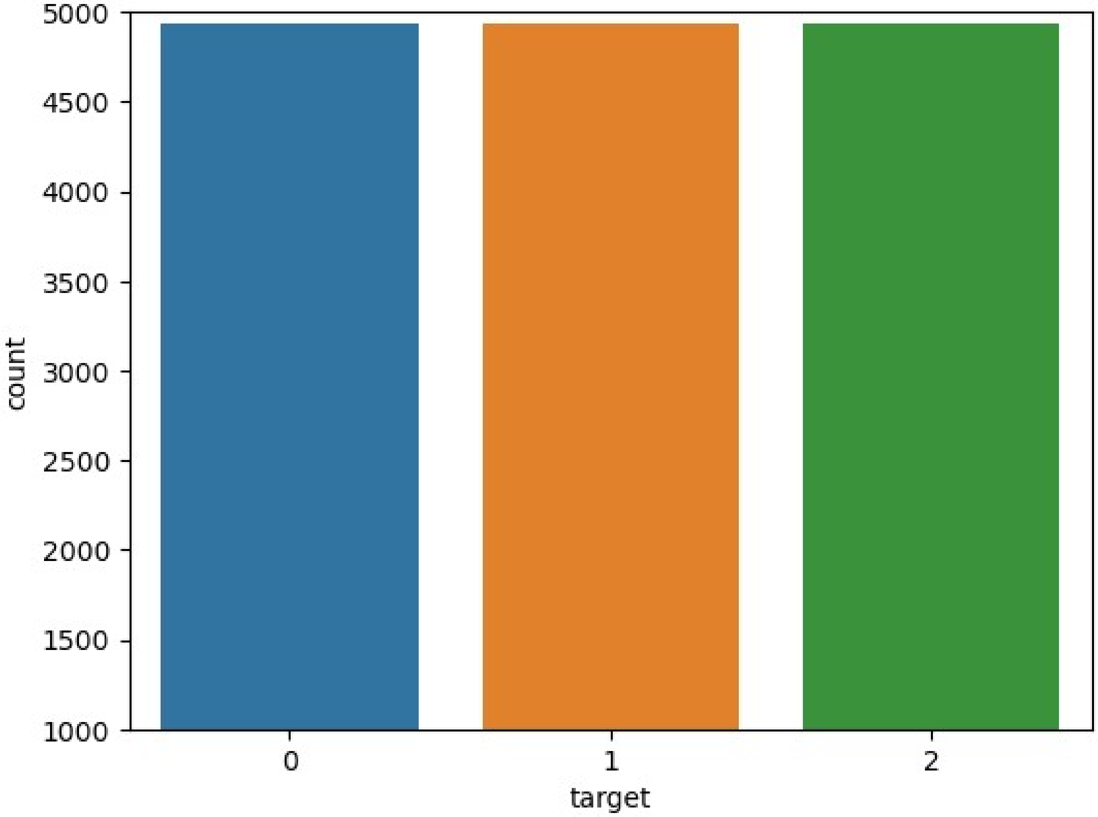
```
(14817, 12)
(14817,)
```

```
y.value_counts()
```

```
1    4939
2    4939
0    4939
Name: quality, dtype: int64
```

```
### The target classes in balanced formet
y_df = pd.DataFrame(np.array(y),columns=['target'])
sns.countplot(y_df,x='target')
pt.ylim([1000,5000])
```

```
(1000.0, 5000.0)
```



```
df.head()
```

```
   type  fixed acidity  volatile acidity  citric acid  residual sugar
\
0     1            7.0              0.27         0.36            20.7

1     1            6.3              0.30         0.34             1.6

2     1            8.1              0.28         0.40             6.9

3     1            7.2              0.23         0.32             8.5

6     1            6.2              0.32         0.16             7.0


   chlorides  free sulfur dioxide  total sulfur dioxide  density    pH
\
0      0.045                 45.0                 170.0   1.0010  3.00

1      0.049                 14.0                 132.0   0.9940  3.30

2      0.050                 30.0                  97.0   0.9951  3.26

3      0.058                 47.0                 186.0   0.9956  3.19
```

```
6      0.045                30.0                   136.0   0.9949  3.18


   sulphates  alcohol  quality
0       0.45      8.8        1
1       0.49      9.5        1
2       0.44     10.1        1
3       0.40      9.9        1
6       0.47      9.6        1

### Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.2,random_state=42)
```

Feature Scaling

Model Training

```
#### Model selection
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

models = {'Logistic Regression':LogisticRegression(),
          'Support Vector Machine':SVC(),
          'DecsionTree':DecisionTreeClassifier(),
          'RadomForestClassier':RandomForestClassifier(),
         'AdaBosster':AdaBoostClassifier(),
          'XGBboost':XGBClassifier()}

for i in range(len(models)):
    model =list(models.values())[i]
    model.fit(X_train,y_train)
    print(list(models.keys())[i]+' score:
',model.score(X_test,y_test))
    cros_score = cross_val_score(model,X_train,y_train,cv=5)
    print(list(models.keys())[i]+' Cross_Val :',list(cros_score))
    print('mean : ',np.mean(cros_score))
    print('__'*40)

Logistic Regression score:  0.6835357624831309
Logistic Regression Cross_Val : [0.6596372838464782,
0.6625896246309574, 0.6748207507380852, 0.6620253164556962,
0.6662447257383967]
mean :  0.6650635402819227
________________________________________________________________________________
```

```
__________
Support Vector Machine score:  0.8488529014844804
Support Vector Machine Cross_Val : [0.8304512863770561,
0.8253901307465205, 0.8253901307465205, 0.8227848101265823,
0.8265822784810126]
mean :  0.8261197272955384
______________________________________________________________________
__________
DecsionTree score:  0.9024966261808367
DecsionTree Cross_Val : [0.8857022353437368, 0.8975115984816533,
0.8810628426824125, 0.8662447257383966, 0.870042194092827]
mean :  0.8801127192678052
______________________________________________________________________
__________
RadomForestClassier score:  0.9622132253711201
RadomForestClassier Cross_Val : [0.9603542808941375,
0.9502319696330662, 0.956980177140447, 0.9531645569620253,
0.9514767932489452]
mean :  0.9544415555757242
______________________________________________________________________
__________
AdaBosster score:  0.6835357624831309
AdaBosster Cross_Val : [0.6992830029523408, 0.6840995360607338,
0.6811471952762548, 0.6734177215189874, 0.6649789029535865]
mean :  0.6805852717523807
______________________________________________________________________
__________
XGBboost score:  0.9723346828609987
XGBboost Cross_Val : [0.972163644032054, 0.9734289329396879,
0.9730071699704765, 0.9729957805907173, 0.970042194092827]
mean :  0.9723275443251527
______________________________________________________________________
__________
```

Here RandomForest,XGBbooster are giving more score compare to other models so I am taking XGBbooster as my final model and doing hyperparameter tuning on it

HYPER PARAMETER TUNNIG

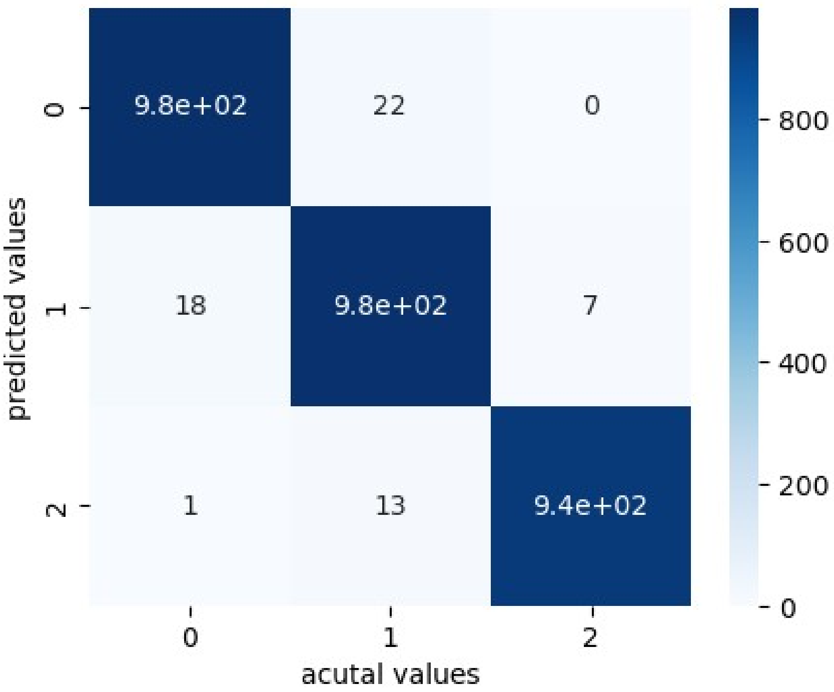
XGBooster Tunning

```
Xgb = {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.5],
        'max_depth': [3, 5, 7],
        'min_child_weight': [1, 3, 5],
}

from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
dic = {}
```

```
grid = GridSearchCV(XGBClassifier(),param_grid=Xgb,cv=5)
grid.fit(X_train,y_train)
dic['xgb'] = grid.best_params_

xgb = XGBClassifier(learning_rate=0.5,max_depth= 7,min_child_weight=
1,n_estimators=200)
xgb.fit(X_train,y_train)
xgb.score(X_test,y_test)

0.9794197031039136

## freecodecamp.org is the providing cources with certificates

###Test Data
y_pred = xgb.predict(X_test)
```

Model Evolution Metrics

```
### Perfomence metrics
from sklearn.metrics import
f1_score,classification_report,accuracy_score,recall_score,precision_s
core,confusion_matrix,auc
print('accuracy: ',accuracy_score(y_pred,y_test))
print('recall: ',recall_score(y_pred,y_test,average=None))
print('precision: ',precision_score(y_pred,y_test,average=None))
print('classification report: ',classification_report(y_pred,y_test))

accuracy:  0.9794197031039136
recall:  [0.97810945 0.97507478 0.98535565]
precision:  [0.98103792 0.96544916 0.99262381]
classification report:                precision    recall  f1-score
support

           0       0.98      0.98      0.98      1005
           1       0.97      0.98      0.97      1003
           2       0.99      0.99      0.99       956

    accuracy                           0.98      2964
   macro avg       0.98      0.98      0.98      2964
weighted avg       0.98      0.98      0.98      2964


##Heatmap
pt.figure(figsize=(5,4))
sns.heatmap(confusion_matrix(y_pred,y_test),annot=True,cmap='Blues')
pt.xlabel('acutal values')
pt.ylabel('predicted values')
pt.show()
```



```
accuracy_dataframe3 = pd.DataFrame({"y_test": y_test, "y_pred":
y_pred})

print( 'Acutally points are incorrectly classified',
sum(accuracy_dataframe3['y_test']-
accuracy_dataframe3['y_pred']),'points')

Acutally points are incorrectly classified -4 points
```

pickle file

```
###pickle file
import pickle
with open('model_pkl', 'wb') as files:
    pickle.dump(model, files)

with open('model_pkl' , 'rb') as f:
    model = pickle.load(f)

model.predict([[1,7.0,0.270,0.36,20.7,0.045,45.0,170.0,1.00100,3.00,0.
450000,8.8]])

array([1], dtype=int64)
```