**Analysis on Red Wine Quality Prediction**

* **Objective:**

In this blog, I am going to explain my analysis on the famous **“Red Wine Quality Prediction”** with the help of the Machine Learning Model in which I will implement multiple predictions and further I will save the best model according to the final result that I will get from different analysis. With the help of these analysis, I will be able to get the information regarding the actual quality of the poor, normal or excellent wine.

* **Let’s talk about Red Wine:**

****

The **Red wine** is a sort of wine made from dark-colored grape varieties. The actual color of the wine can range from serious violet, common of youthful wines, through to block red for mature wines and brown for more established red wines. The juice from most purple grapes is greenish-white, the red shading coming from anthocyan shades (additionally called anthocyanins) present in the skin of the grape; exemptions are the moderately uncommon teinturier varieties, which produce a red-hued juice. A significant part of the red-wine creation process consequently includes the extraction of shading and flavor parts from the grape skin. **Red wine** is a delicacy all throughout the planet.

* **Let’s talk about Vinho Verde Wine:**

** **

**Vinho Verde** (in a real sense 'green wine') alludes to Portuguese wine that began in the historic Minho province in the furthest north of the country. The present day **'Vinho Verde'** locale, initially assigned in 1908, incorporates the old Minho region in addition to nearby regions toward the south. In 1976, the old area was disintegrated.

**Vinho Verde** isn't a grape assortment, it is a DOC for the creation of wine. The name signifies "green wine," however interprets as "youthful wine", with wine being delivered three to a half year after the grapes are harvested. They may be red, white, or rosé, and they are normally burned-through before long bottling. A **Vinho Verde** can likewise be a shimmering, a Late Harvest or even Brandy. In its initial long stretches of creation, the slight effervesce of the wine came from malolactic fermentation taking place in the jug. In winemaking this is typically considered a wine fault but **Vinho Verde** makers found that shoppers loved the somewhat bubbly nature. In any case, the wines must be bundled in opaque bottles to stow away the unseemly turbidity and dregs that the "in-bottle MLF" delivered. Today, most **Vinho Verde** makers presently don't follow this training with the slight shimmer being added by artificial carbonation.

The locale is described by its numerous little producers, which numbered around 19,000 starting at 2014. A considerable lot of these producers utilized to train their plants high off the ground, up trees, wall, and even telephone poles so that they could cultivate vegetable crops underneath the plants that their families might use as a food source.

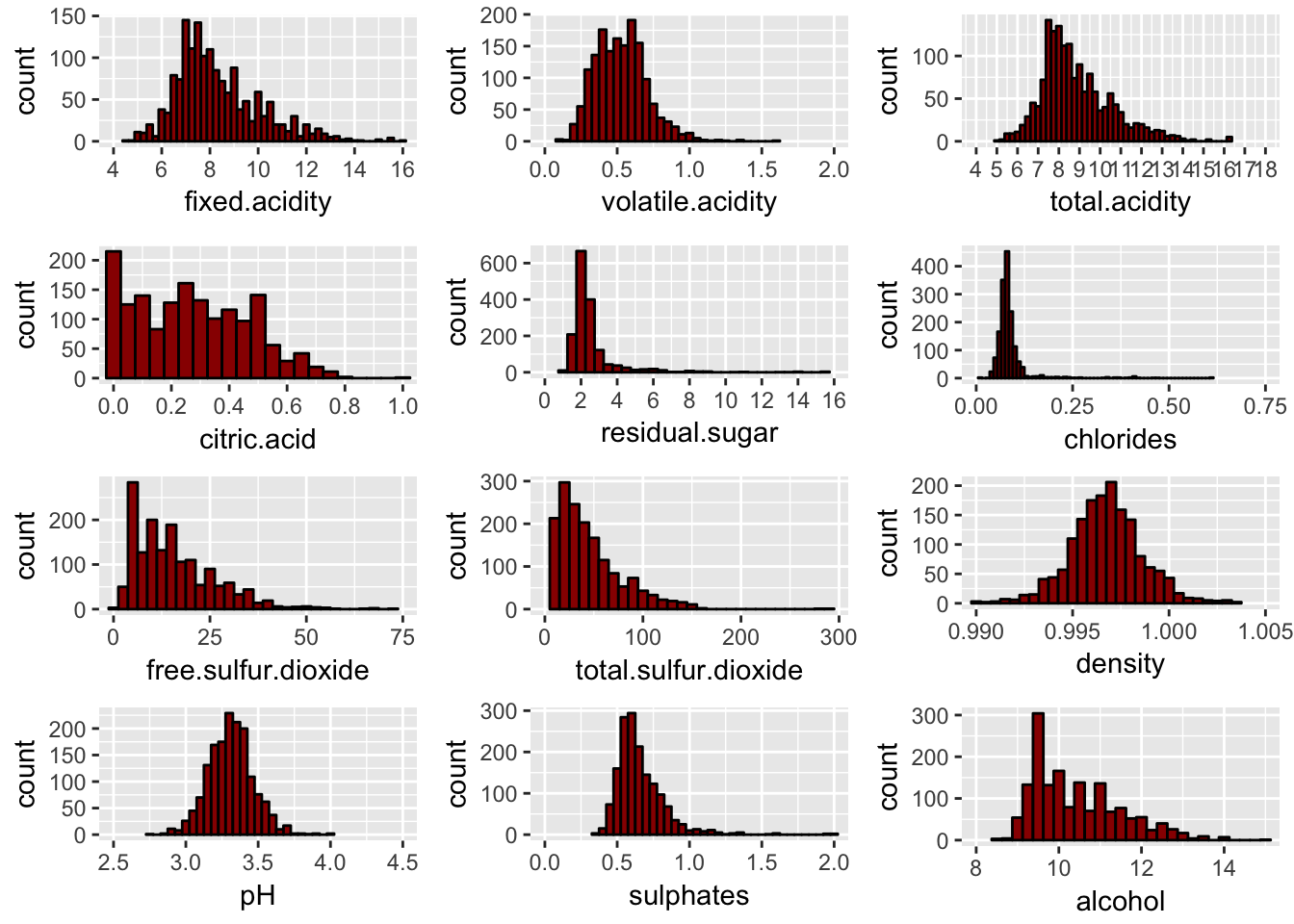
* **Problem Statement:**

The dataset is related to red and white variants of the Portuguese "Vinho Verde" wine. 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.).

This dataset can be viewed as classification task. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

* **Attribute Information:**

Input variables (based on physicochemical tests):  
01 - Fixed acidity  
02 - Volatile acidity  
03 - Citric acid  
04 - Residual sugar  
05 - Chlorides  
06 - Free sulfur dioxide  
07 - Total sulfur dioxide  
08 - Density  
09 - pH  
10 - Sulphates  
11 - Alcohol  
Output variable (based on sensory data):  
12 - Quality (score between 0 and 10)



What might be an interesting thing to do, is to set an arbitrary cutoff for your dependent variable (wine quality) at e.g. 7 or higher getting classified as 'good/1' and the remainder as 'not good/0'.

Let’s begin my Analysis on **“Red Wine Quality Prediction”** dataset. Below you will find the way to do analysis on **“Red Wine Quality Prediction”**  dataset via Python.

* **Importing all the Required Libraries:**

First, I will import all the required libraries to start the analysis on the given data.

***# Importing all the required libraries***

***import numpy as np***

***import pandas as pd***

***import matplot.pyplot as plt***

***import seaborn as sns***

***import filterwarinings(“ignore”)***

After importing all the required libraries, I will get data from csv and start doing my analysis on it. Further, I will check various factors such as datatype, null values (i.e. missing values) to get more efficient analysis. If required, I will create additional columns for further analysis.

* **Getting the Data:**

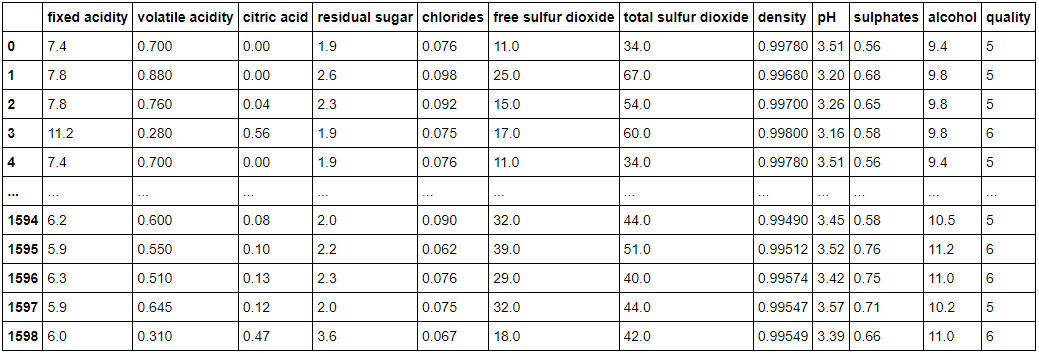
***# Loading the data***

red\_proj = pd.read\_csv('winequality-red.csv')

***# Print the red wine dataset***

red\_proj

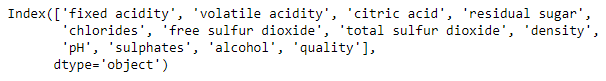
Using above command, we can see the total number of entries. It is always good to look on the data before we start our analysis. It is very helpful and easy to understand.



1599 rows × 12 columns

***# Getting total Column names***

red\_proj.columns



***# Get a total number of Rows & Columns***

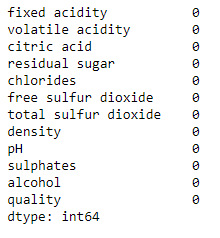
red\_proj.shape

(1599, 12)

From the above case, we can see that given data contains 1599 rows and 12 columns.

***# Checking for the Null values***

red\_proj.isnull().sum()



***# Let’s change the Quality to Categorical Data***

**for** i **in** range(0,1599):

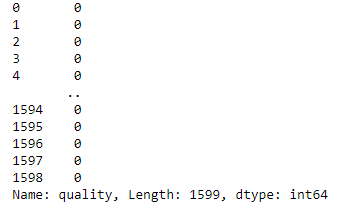
**if** red\_proj["quality"][i]>=7:

red\_proj["quality"][i]=1

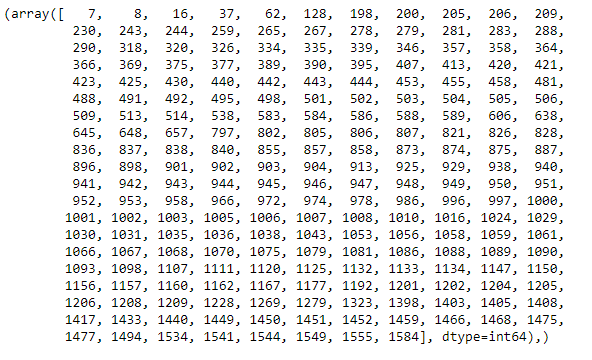
**else**:

red\_proj["quality"][i]=0

Red\_proj[“quality”]

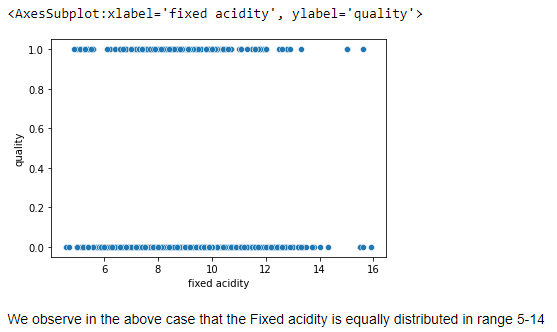


np.where(red\_proj["quality"]==1)

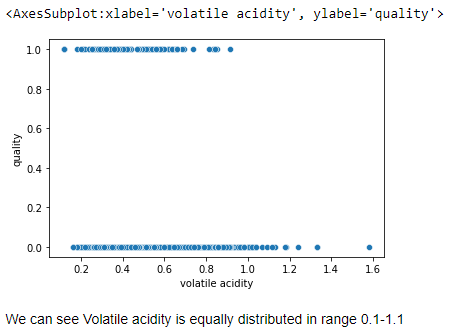


* **EDA**
* **Scatterplot:**

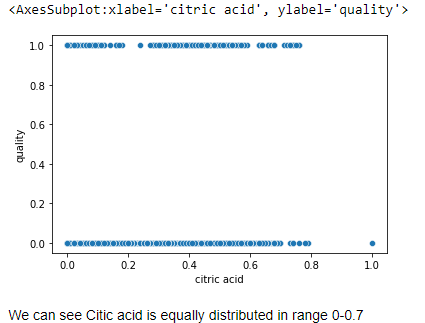
sns.scatterplot(x="fixed acidity", y="quality", data=red\_proj)



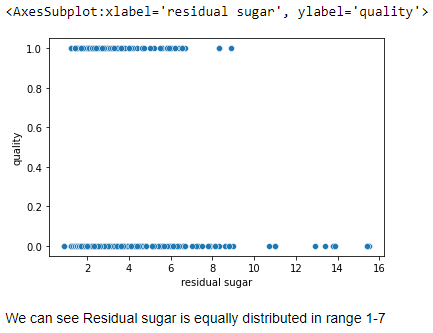
sns.scatterplot(x="volatile acidity", y="quality", data=red\_proj)



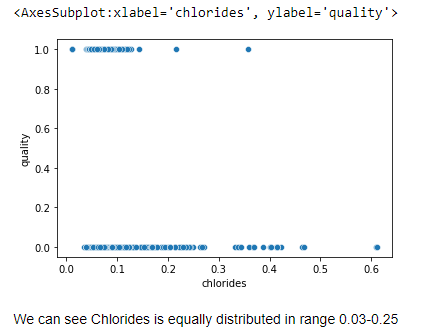
sns.scatterplot(x="citric acid" ,y="quality", data=red\_proj)



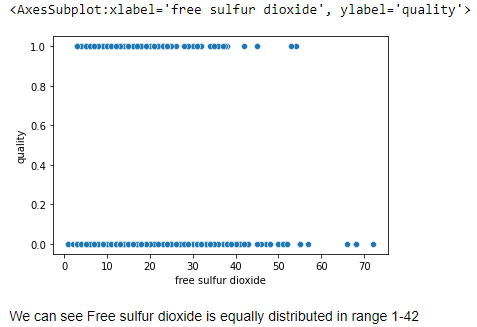
sns.scatterplot(x="residual sugar", y="quality", data=red\_proj)



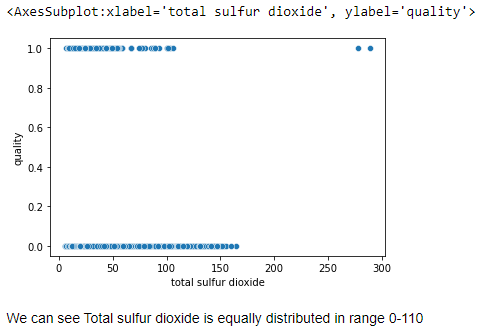
sns.scatterplot(x="chlorides" ,y="quality", data=red\_proj)



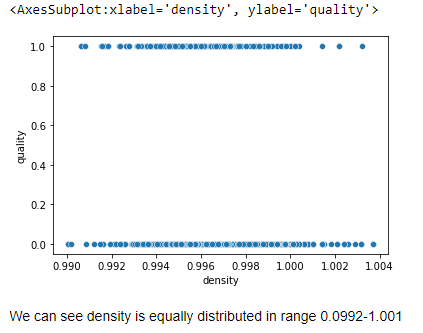
sns.scatterplot(x="free sulfur dioxide", y="quality", data=red\_proj)



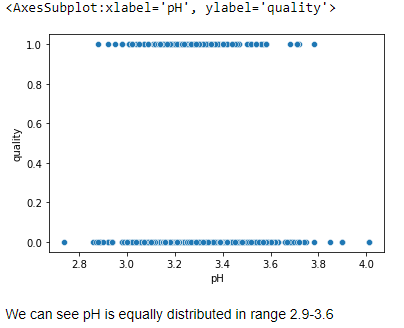
sns.scatterplot(x="total sulfur dioxide", y="quality" ,data=red\_proj)



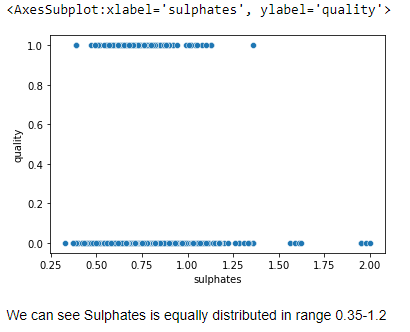
sns.scatterplot(x="density", y="quality", data=red\_proj)



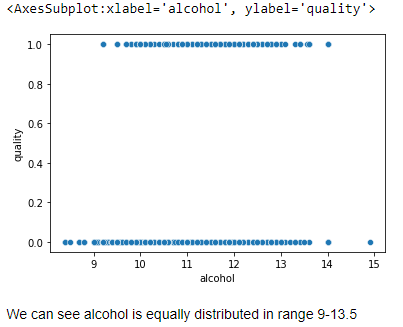
sns.scatterplot(x="pH", y="quality", data=red\_proj)



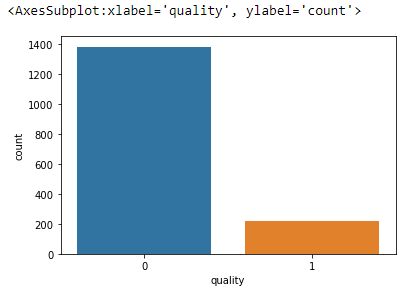
sns.scatterplot(x="sulphates", y="quality", data=red\_proj)



sns.scatterplot(x="alcohol", y="quality", data=red\_proj)

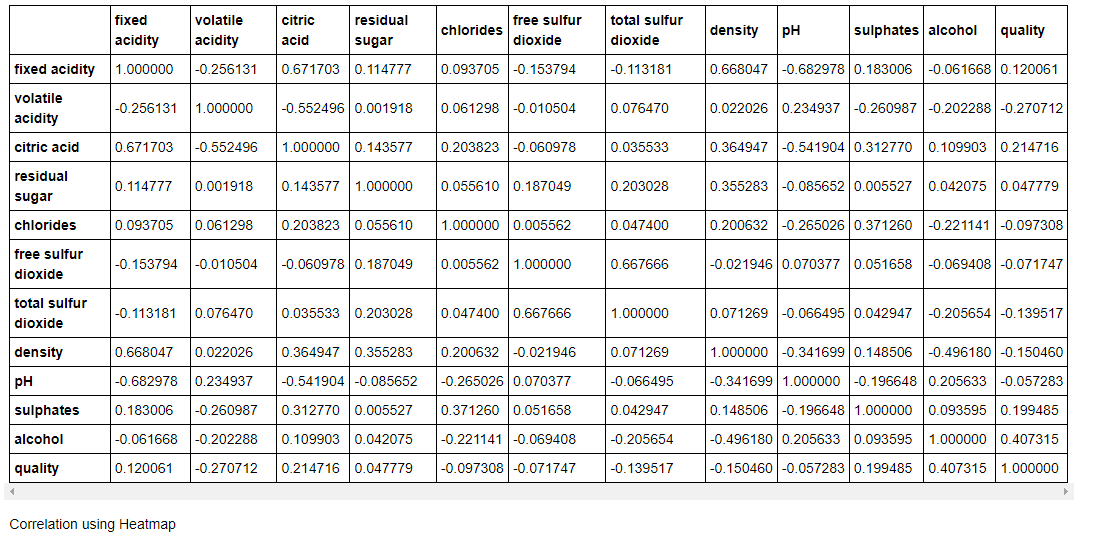


snscountplot(x =”quality”, data=red\_proj



* **Correlation:**

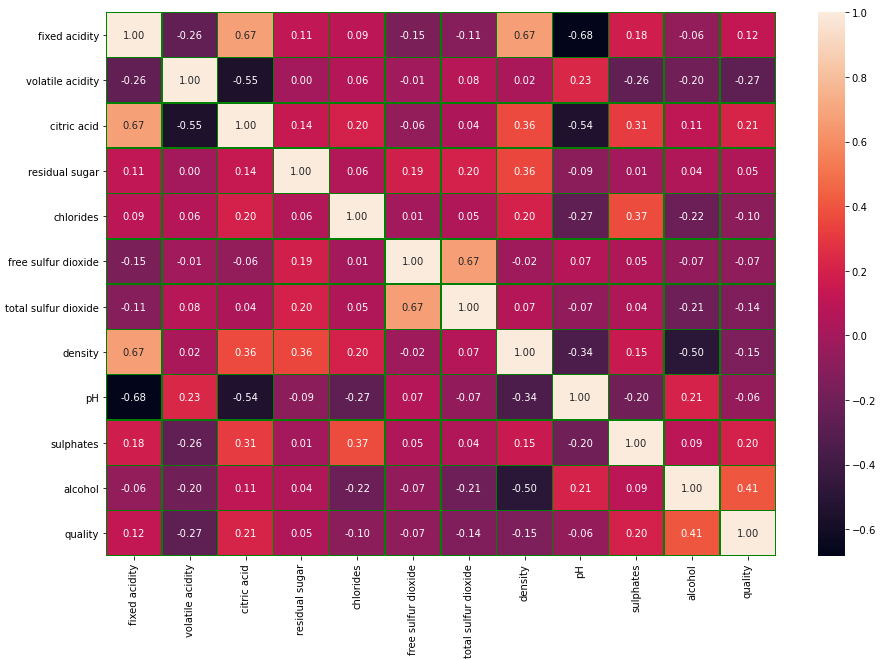
red\_proj.corr()



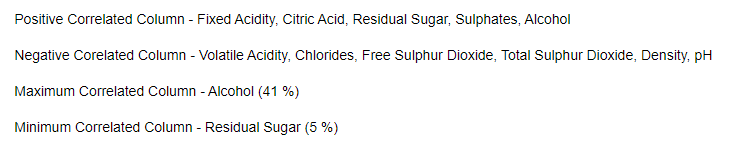
plt.figure(figsize=(15,10))

sns.heatmap(red\_proj.corr(), annot=**True**, linewidth=0.5, linecolor='green',

fmt='.2f')



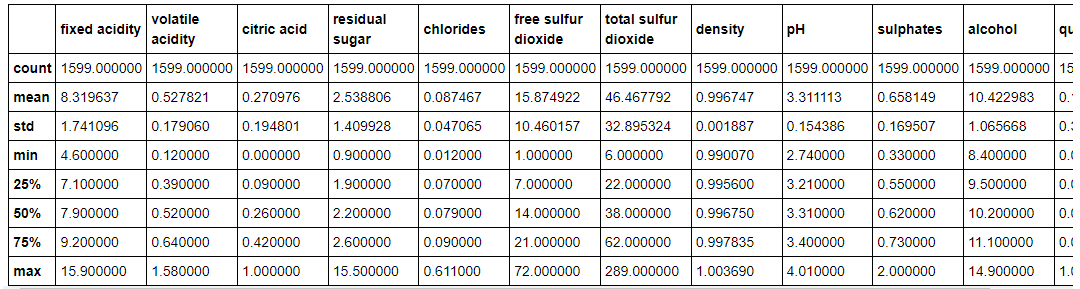
* **Outcome Of Correlation:**



* **Describing Dataset:**

***# Get data in statistics form.***

***red\_proj.describe()***



plt.figure(figsize=(18,24))

sns.heatmap(round(red\_proj.describe()[1:].transpose(),2), linewidth=1, annot=**True**, fmt='f')

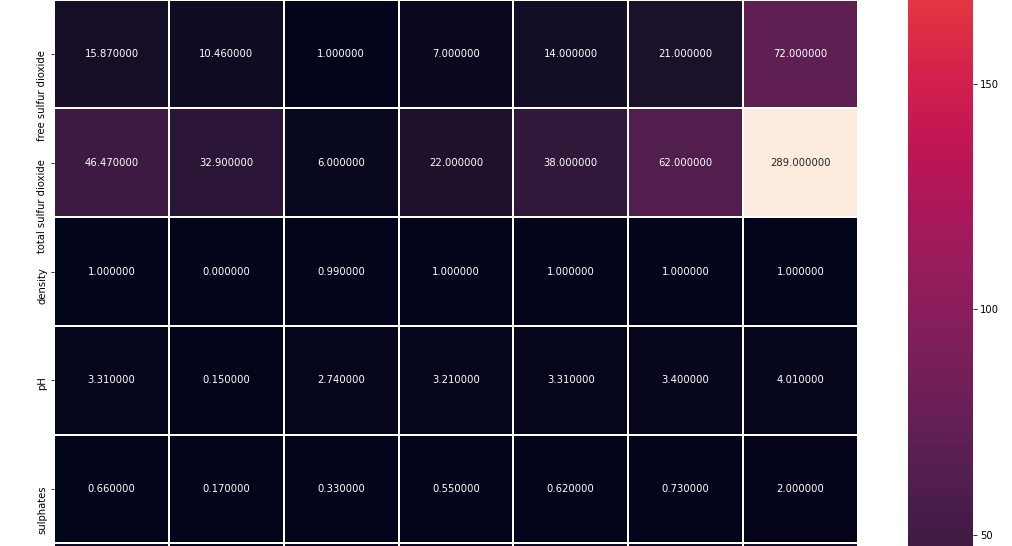
plt.xticks(fontsize=15)

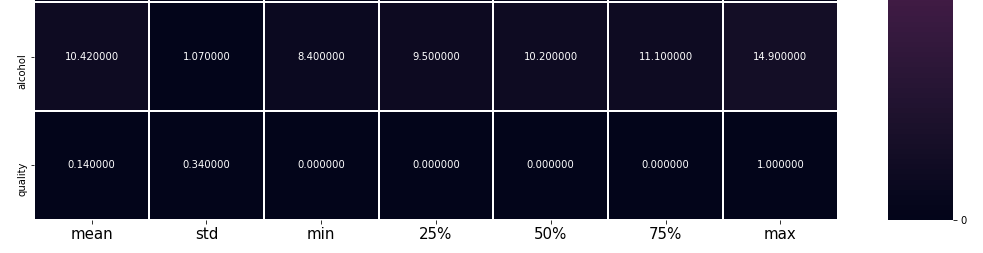
plt.yticks(fontsize=10)

plt.title("Variables Summary")

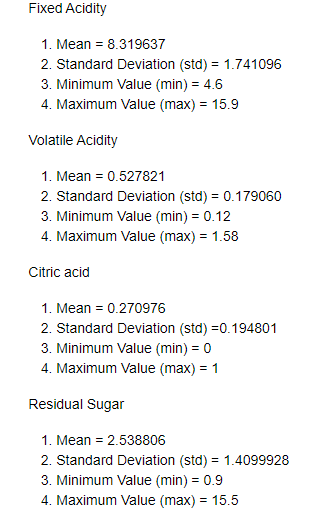
plt.show()

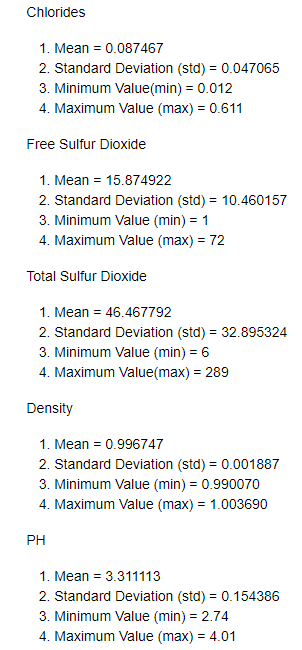


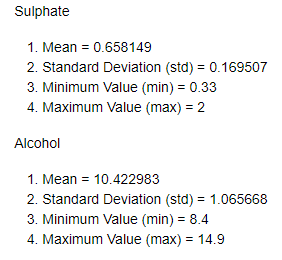




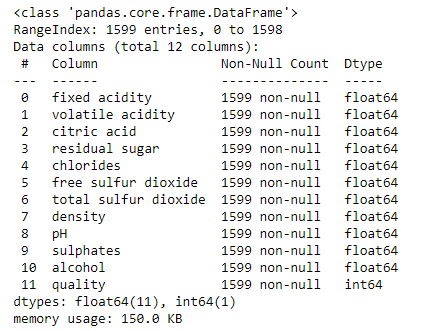
* **Dataset Outcome:**







red\_proj.info()



* **ANALYSING THE OUTLIERS:**

An outlier is an element of a data set that distinctly stands out from the rest of the data. In other words, outliers are those data points that lie outside the overall pattern of distribution. The easiest way to **detect outliers is to create a graph. Plots such as Box plots, Scatterplots and Histograms** **can help to detect outliers**. Alternatively, we can use **mean and standard deviation** to list out the outliers. Interquartile Range and Quartiles can also be used to detect outliers.

* **Outlier Checking:**

collist = red\_proj.columns.values

ncol = 30

nrows = 15

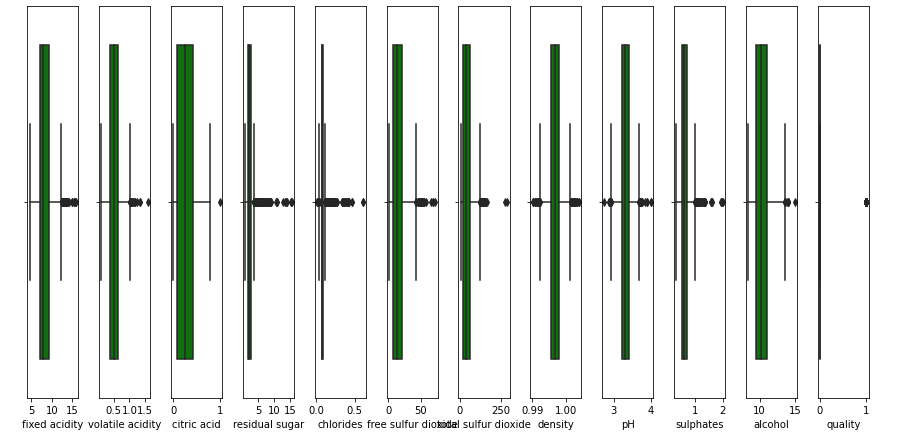
plt.figure(figsize=(ncol,3\*ncol))

**for** i **in** range(0,len(collist)):

plt.subplot(nrows,ncol,i+1)

sns.boxplot(red\_proj[collist[i]], color='green', orient='v')

plt.tight\_layout()

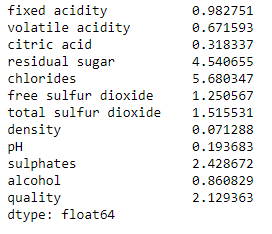


red\_proj["quality"].unique()



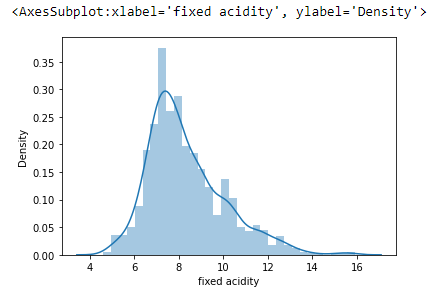
# Checking the skewness in the given dataset

red\_proj.skew()

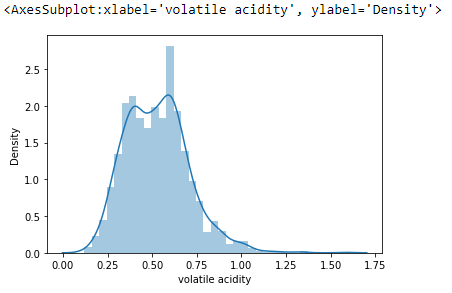


* **Normal Distribution Curve:**

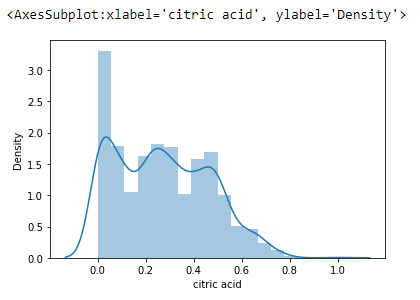
sns.distplot(red\_proj["fixed acidity"])



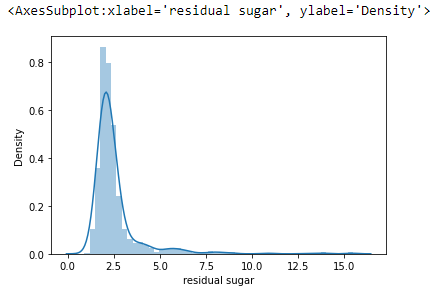
sns.distplot(red\_proj["volatile acidity"])



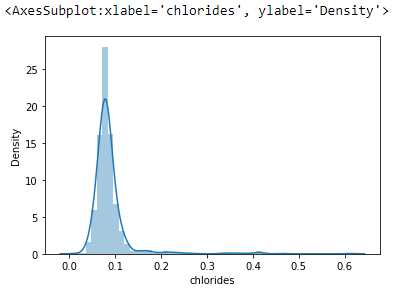
sns.distplot(red\_proj["citric acid"])



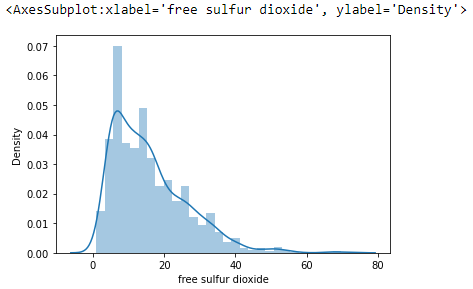
sns.distplot(red\_proj["residual sugar"])



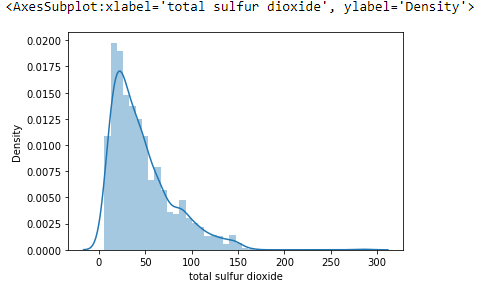
sns.distplot(red\_proj["chlorides"])



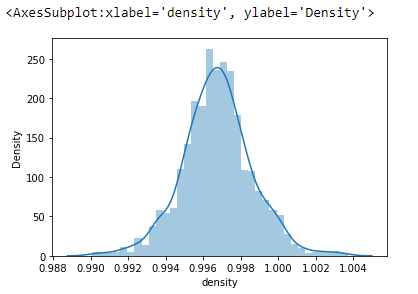
sns.distplot(red\_proj["free sulfur dioxide"])



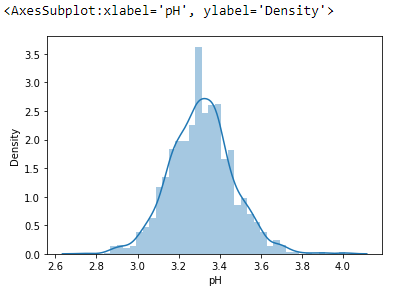
sns.distplot(red\_proj["total sulfur dioxide"])



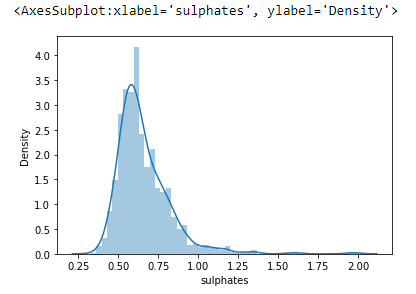
sns.distplot(red\_proj["density"])



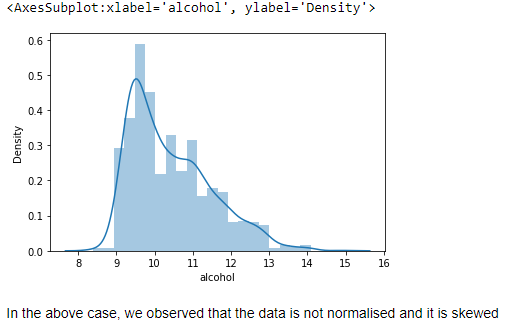
sns.distplot(red\_proj["pH"])



sns.distplot(red\_proj["sulphates"])

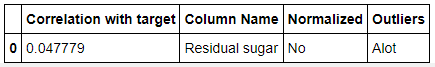


sns.distplot(red\_proj["alcohol"])



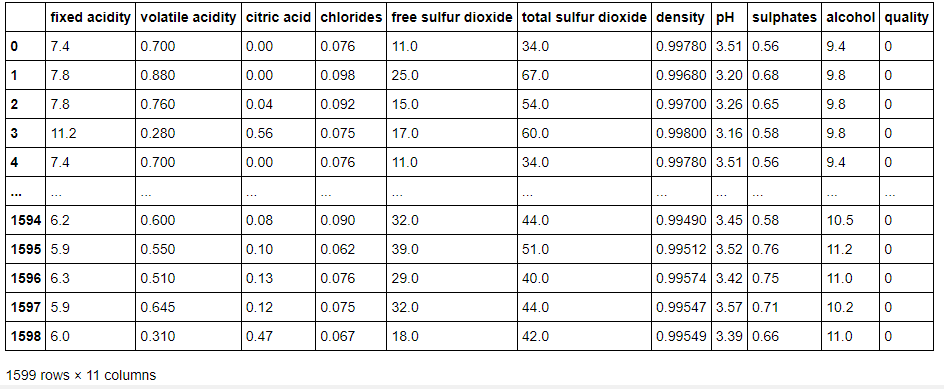
***# Deleting the unnecessary data from the given dataset***

delete = pd.DataFrame([['0.047779','Residual sugar','No','Alot']],columns=['Correlation with target','Column Name','Normalized','Outliers']) delete



red\_proj.drop('residual sugar', axis=1, inplace=**True**)

red\_proj

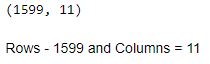


* **REMOVING THE OUTLIERS:**

**from** **scipy.stats** **import** zscore

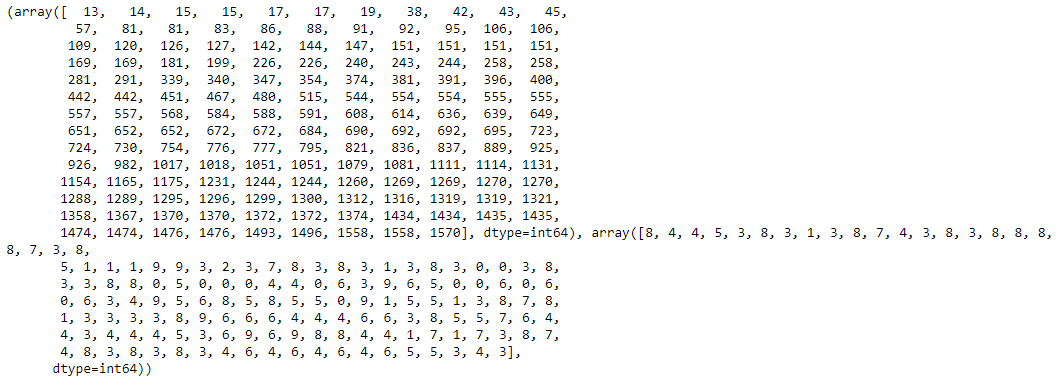
z = np.abs(zscore(red\_proj))

z.shape



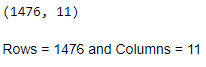
threshold = 3

print(np.where(z>3))



red\_proj\_new = red\_proj[(z<3).all(axis=1)]

red\_proj\_new.shape



loss\_percentage = (1599-1476)/1599\*100

print(loss\_percentage)

7.6923076923076925

* **Checking Multicolinearity:**

x = red\_proj\_new.iloc[:,:-1]

y = red\_proj\_new.iloc[:,-1]

**import** **scipy.stats** **as** **st**

**import** **statsmodels.api** **as** **sm**

**from** **statsmodels.stats.outliers\_influence** **import** variance\_inflation\_factor **as** viff

**def** calc\_vif(x1):

vif\_red = pd.DataFrame()

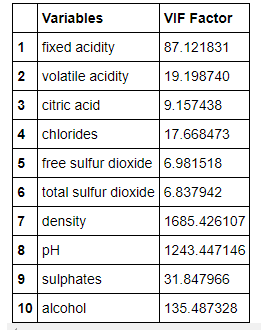
vif\_red["Variables"] = x1.columns

vif\_red['VIF Factor'] = [viff(x1.values,i) **for** i **in** range(x1.shape[1])]

vif\_red.index+=1

**return**(vif\_red)

calc\_vif(x)



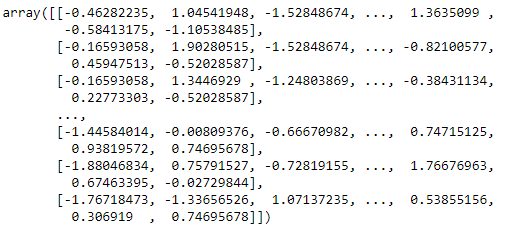
***# Dropping/removing the unnecessary data***

x.drop(["free sulfur dioxide"], axis=1, inplace=**True**)

**from** **sklearn.preprocessing** **import** power\_transform

x = power\_transform(x, method="yeo-johnson")

x

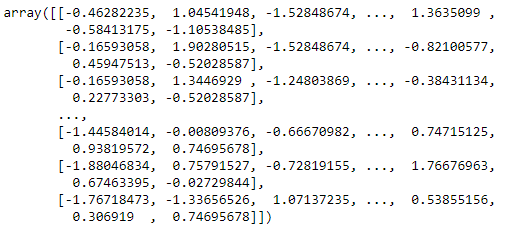


**from** **sklearn.preprocessing** **import** StandardScaler

stand\_scal = StandardScaler()

x = stand\_scal.fit\_transform(x)

x



* **MODEL TRAINING:**

In Machine Learning we create models to predict the outcome of certain events. To measure if the model is good enough, we can use a method called Train/Test.

**What is Train/Test?**

Train/Test is a method to measure the accuracy of your model. It is called Train/Test because you split the data set into two sets: a training set and a testing set.

* You train the model using the training set**.(Train the model means create the model.)**
* You test the model using the testing set.**(Test the model means test the accuracy of the model.)**
* **Classification Model:**

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.linear\_model** **import** LogisticRegression

lr=LogisticRegression()

**from** **sklearn.metrics** **import** accuracy\_score,confusion\_matrix,classification\_report,f1\_score

!pip install imblearn

**from** **imblearn.over\_sampling** **import** SMOTE

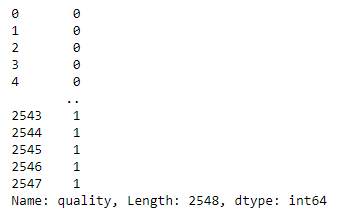
smt = SMOTE()

x,y = smt.fit\_resample(x,y)

y.value\_counts()



y



* **FINDING ACCURACY OF THE GIVEN DATASET:**

**Logistic Regression** is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

**for** i **in** range(0,1000):

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, random\_state=i, test\_size=0.20)

lr.fit(x\_train, y\_train)

pred\_train = lr.predict(x\_train)

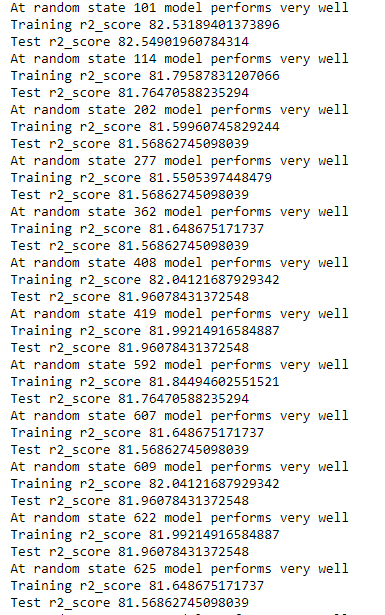
pred\_test = lr.predict(x\_test)

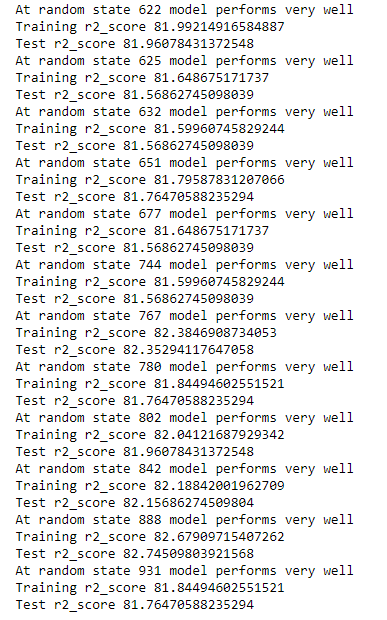
**if** round(accuracy\_score(y\_train, pred\_train)\*100,1)==round(accuracy\_score(y\_test, pred\_test)\*100,1):

print("At random state",i,"model performs very well")

print ("Training r2\_score", accuracy\_score(y\_train, pred\_train)\*100)

print("Test r2\_score", accuracy\_score(y\_test, pred\_test)\*100)





x\_train, x\_test, y\_train, y\_test=train\_test\_split(x, y, test\_size=0.20, random\_state=922)

**from** **sklearn.svm** **import** SVC

**from** **sklearn.naive\_bayes** **import** MultinomialNB

**from** **sklearn.tree** **import** DecisionTreeClassifier

**from** **sklearn.neighbors** **import** KNeighborsClassifier

**from** **sklearn.model\_selection** **import** GridSearchCV

svc = SVC()

dtc = DecisionTreeClassifier()

knn = KNeighborsClassifier()

nb = MultinomialNB()

param = {'kernel':['linear','rbf','poly','sigmoid']}

gd = GridSearchCV(svc,param,cv=5)

gd.fit(x,y)

print(gd.best\_params\_)



**GridSearchCV** implements a “fit” and a “score” method. It also implements **“score\_samples”, “predict”, “predict\_proba”, “decision\_function”, “transform” and “inverse\_transform”** if they are implemented in the estimator used.

param = {'criterion':['ginny','entropy'],'splitter':['best','random'],'max\_features':['none','auto','sqrt','log2']}

gd = GridSearchCV(dtc,param)

gd.fit(x,y) print(gd.best\_params\_)



param = {'n\_neighbors':[7,5,11],'weights':['uniform','distance'],'algorithm':['auto','bell\_tree','kd\_tree','brute']}

gd = GridSearchCV(knn,param)

gd.fit(x,y) print(gd.best\_params\_)



svc = SVC(kernel='rbf')

dtc = DecisionTreeClassifier(criterion='entropy', max\_features='log2', splitter='best')

knn = KNeighborsClassifier(n\_neighbors=5, weights='distance', algorithm='auto')

**def** fun(f):

f.fit(x\_train,y\_train)

pred=f.predict(x\_test)

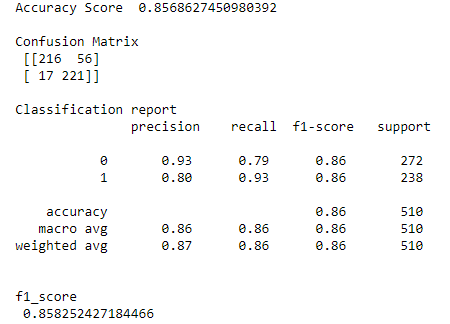
print("Accuracy Score ", accuracy\_score(y\_test,pred))

print("**\n**Confusion Matrix**\n**", confusion\_matrix(y\_test,pred))

print("**\n**Classification report**\n**", classification\_report(y\_test,pred))

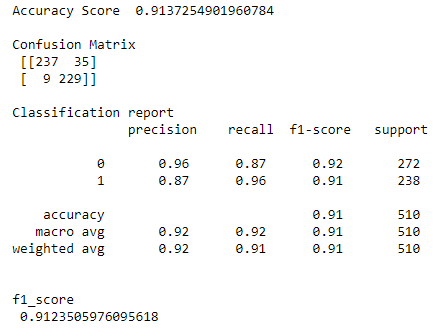
print("**\n**f1\_score**\n**", f1\_score(y\_test,pred))

fun(svc)



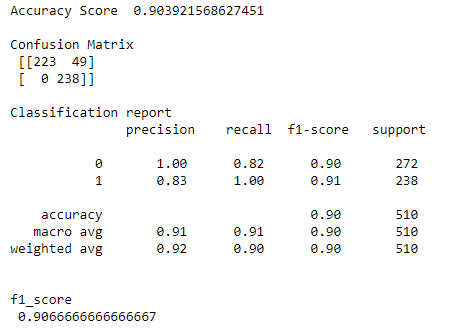
The **[Decision Trees](https://scikit-learn.org/stable/modules/tree.html" \l "tree)**is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve.

fun(dtc)



The **K** in the name of this classifier represents the **k nearest neighbors**, where **k is an integer** value specified by the user. Hence as the name suggests, this classifier implements learning based on the k nearest neighbors. The choice of the value of k is dependent on data.

fun(knn)



* **Cross Validation Score:**

**from** **sklearn.model\_selection** **import** cross\_val\_score,KFold

score = cross\_val\_score(SVC(kernel='rbf'),x,y,cv=5)

print(score)

print(score.mean())



score = cross\_val\_score(DecisionTreeClassifier(criterion='entropy',max\_features='log2',splitter='best'),x,y,cv=5)

print(score)

print(score.mean())



score = cross\_val\_score(KNeighborsClassifier(n\_neighbors=5,weights='distance',algorithm='auto'),x,y,cv=5)

print(score)

print(score.mean())



* **Ensemble Method:**

**Random Forest** is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

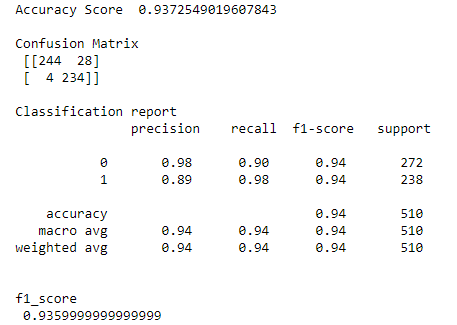
**from** **sklearn.ensemble** **import** RandomForestClassifier,AdaBoostClassifier,GradientBoostingClassifier

random\_for = RandomForestClassifier(n\_estimators=100,random\_state=23)

ad\_boost = AdaBoostClassifier()

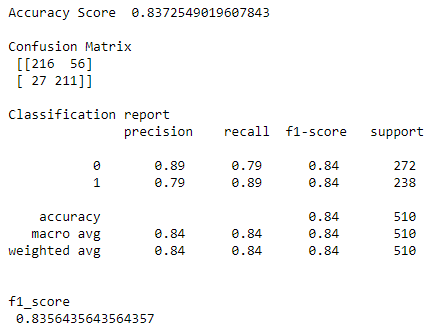
Gd\_boost = GradientBoostingClassifier(n\_estimators=200,learning\_rate=0.1)

fun(random\_for)



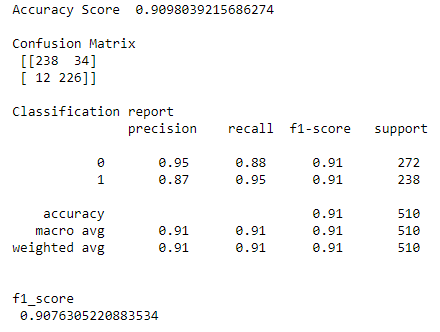
An **AdaBoost**  **classifier** is a meta-estimator that begins by fitting a **classifier** on the original dataset and then fits additional copies of the **classifier** on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent **classifiers** focus more on difficult cases.

fun(ad\_boost)



**Gradient boosting** is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

fun(Gd\_boost)



* **Voting Classifier:**

**from** **sklearn.ensemble** **import** VotingClassifier

estimator = []

estimator.append(('svc',SVC(kernel='rbf')))

estimator.append(('dtc',DecisionTreeClassifier(criterion='entropy',max\_features='log2',splitter='best')))

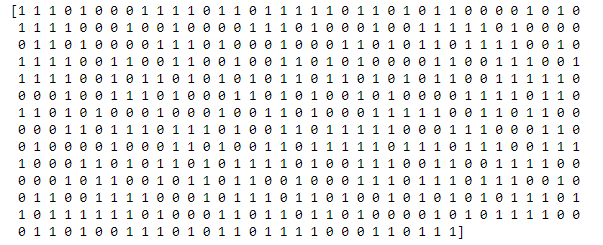
estimator.append(('knn',KNeighborsClassifier(n\_neighbors=5,weights='distance',algorithm='auto')))

vot\_soft = VotingClassifier(estimators=estimator,voting='hard')

vot\_soft.fit(x\_train,y\_train)

y\_pred=vot\_soft.predict(x\_test)

print(y\_pred)



print("Voting Score=", accuracy\_score(y\_test,y\_pred))



* **AUC ROC Curve:**

**from** **sklearn.metrics** **import** roc\_curve,auc

fpr,tpr,thresholds=roc\_curve(random\_for.predict(x\_test),y\_test)

roc\_auc = auc(fpr,tpr)

plt.figure()

plt.plot(fpr, tpr, color="red", lw=10, label="ROC Curve (area= **%0.2f**)" %roc\_auc)

plt.plot([0,1],[0,1],color="green", lw=15, linestyle="--")

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.05])

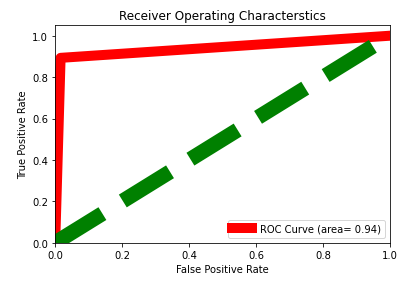
plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("Receiver Operating Characterstics")

plt.legend(loc='lower right')

plt.show()



* **Model Saving:**

**import** **pickle**

filename = 'output.pkl'

pickle.dump(random\_for, open(filename,'wb'))

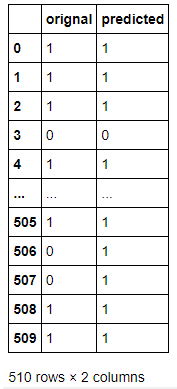
* **Conclusion:**

a = np.array(y\_test)

predicted = np.array(random\_for.predict(x\_test))

df\_com = pd.DataFrame({"orignal":a,"predicted":predicted},index=range(len(a)))

df\_com



The above observations show that the best model is **RANDOM FOREST CLASSIFIER** among all the other models because it has the highest accuracy score equals to **93.72%**